Colóquio 19°
Brasileiro de Matemática

INTRODUCTION TO METHODS OF PARALLEL OPTIMIZATION

Yair Censor
Stavros Zenios

YAIR CENSOR (Univ. of Haifa, Israel)
STRAVOS A. ZENIOS (Univ. of Pennsylvania, USA)

ISBN
85-244-0074-9

Conselho Nacional de Desenvolvimento Científico e Tecnológico

INSTITUTO DE MATEMÁTICA PURA E APLICADA
Estrada Dona Castorina, 110 - J. Botânico
CEP: 22460.320 - Rio de Janeiro-RJ

To Erga, Aviv, Nitzan and Keren --- Y.C.

To Christiana, Efy and Elena --- S.A.Z.

Preface

Problems of mathematical optimization are encountered in many diverse areas of the natural and social sciences and engineering where real-world applications are found. Developments in the vast field of optimization are, to a great extend, motivated by these applications and have drawn, over the years, both from mathematics and from computer science. Mathematics has provided the foundations for the design and analysis of optimization algorithms. Computer science provides the tools for the design of data-structures, and the translation of the mathematical algorithms into numerical procedures that are implementable on a computer. Efficient and robust implementations of optimization algorithms are crucial when one deals with the solution of large-scale, real-world applications. Recent technological innovations, with the introduction of parallel computer architectures, are having a significant impact on every area of scientific computing where large-scale problems are attacked. In these Lecture Notes we introduce parallel computing ideas and techniques into both optimization theory and into some numerical algorithms for large-scale optimization problems.

We examine a broad family of algorithms for constrained optimization. When viewed from the proper perspective these algorithms satisfy the design characteristics of "good" parallel algorithms. A basic introduction to parallel computers — what they are, how to measure their performance, how to design and implement parallel algorithms — is given. This core knowledge on parallel computers is then linked with the theoretical algorithms. The combined mathematical algorithms and parallel computing techniques are brought together to bear on the solution of several important applications: image reconstruction from projections, matrix balancing, network optimiza-

tion, nonlinear stochastic programming and financial planning.

We believe that the value of bringing together applications, mathematical algorithms and parallel computing techniques, extends beyond the successful solution of the specific problems at hand. Namely, it introduces the reader to the complete process from the modeling of a problem through the design of solution algorithms, and to the art and science of parallel computations. It is not possible anymore to study these three disciplinary efforts — modelling, mathematics of algorithms and parallel computing — in isolation from each other. The successful solution of real-world problems in scientific computing is the result of coordinated efforts across all three fronts.

These Lecture Notes are prepared especially for an "Advanced Short Course" delivered at the *19th Colloquium of the Brazilian Mathematical Society* at the Instituto de Mathematica Pura e Aplicada (**IMPA**) in Rio de Janeiro, Brazil, in July 1993. As such we had to meet **IMPA's** deadline for publication. Having raced against a deadline is our excuse for many omissions, oversights, errors and other imperfections that remain in this manuscript. We are currently working on extending and refining these Lecture Notes, and would, therefore, greatly appreciate comments from readers who would take the time and make the effort to draw our attention to matters that need to be corrected, such as references to other works that we may have missed, other relevant material that should be included, errors or inaccuracies, etc. We will be happy to acknowledge such help in the next version of this work.

Yair Censor, Haifa.

Stavros A. Zenios, Philadelphia.

April, 1993.

Acknowledgements

We thank the organizers of the *19th Brazilian Mathematical Colloquium* for giving us the opportunity to present our work in this form. The material presented here is based largely on our own published works. We express our appreciation to our past and present collaborators from whom we learned a great deal.

Particular thanks for many useful discussions and support are extended to Marty Altschuler, Dimitri Bertsekas, Dan Butnariu, Alvaro De Pierro, Jonathan Eckstein, Tommy Elfving, Gabor Herman, Dan Gordon, Alfredo Iusem, Elizabeth Jessup, Arnold Lent, Robert Lewitt, Jill Mesirov, John Mulvey, Soren Nielsen, Mustafa Pinar and Michael Schneider. Much of the work of Yair Censor in this area was done in collaboration with the Medical Image Processing Group (MIPG) at the Department of Radiology, University of Pennsylvania.

This work was supported by grants from the National Institutes of Health (HL–28438), the Air-Force Office of Scientific Research (AFOSR-91–0168) and the National Science Foundation (CCR–9104042 and SES–9100216). Computing resources were made available though the North-East Parallel Architectures Center (NPAC) at Syracuse University, the Army High-Performance Computing Research Center (AHPCRC) at the University of Minnesota, Thinking Machines Corporation, Cray Research Inc., and Argonne National Laboratory.

"This Book was followed by a full-size publication of the book: Y. Censor and S.A. Zenios, "Parallel Optimization: Theory, Algorithms, and Applications", Oxford University Press, New York, NY, USA, 1997 (576pp., ISBN 0-19-510062-X)."

Contents

Chapter 1

Introduction

Computing power improved by a factor of one million in the period 1955–1990, and it is expected to improve by that much more just within the next decade. How can this accelerated improvement be sustained? The answer is found in novel computer architectures: multiple, semi-autonomous processors coordinating for the solution of a single problem. The late 1980's have witnessed the advent of *massive parallelism*: the number of processors that are brought to bear on a single problem could be several thousands, or even millions. This rapid technological development is expected to transform in a significant way the exact sciences, the social sciences and all branches of engineering. The American Academy of Arts and Sciences devotes the Winter 1992 issue of its Journal, *DÆDALUS*, to an analysis of the coming "New Era of Computation".

Massive parallelism is improving by a quantum leap the size and complexity of models that can be represented on and solved by a computer. These Lecture Notes give an introduction to methods of parallel computing for

optimization problems. The Notes take firstmost an algorithmic approach. Specific iterative algorithms are presented, their mathematical convergence is established and their suitability for parallel computations is discussed. The algorithms, in turn, are strongly influenced by the structure of several real-world applications. Some significant problems from several important areas of application are discussed in detail too. Those are drawn from such diverse areas as operations research and image reconstruction from projections in diagnostic medicine. Other areas where similar approaches and techniques are used are also mentioned. Implementations of the algorithms on specific parallel architectures are given extensive treatment. Alternative approaches to the parallel implementation of an algorithm are discussed, and benchmark results on various computer architectures are summarized. Those results highlight the effectiveness of alternative implementations on different architectures.

Several of the algorithms and their underlying theory have a long history that dates back to the 1930's. At that time the issue of *computations* — that is, implementation of the algorithm as a computer program — was irrelevant. Here we present the algorithms not just with a view towards computations, but more importantly, towards parallel computations. It is when viewed within the context of parallel computing that these algorithms are realized to posses some important features: They are well-suited for implementation on a wide range of parallel architectures, with as few as two processors or as many as millions.

1.1 How Does Parallelism Affect Computations

Mathematical theory and computational techniques always went hand-in-hand in the field of optimization. This has been especially true in the context of large-scale applications. For small-scale problems the correctness of a theoretical algorithm is usually sufficient to guarantee that the underlying problem can be solved successfully. For large-scale problems theoretical correctness does not suffice. Can the algorithm be translated into a computer code that does not exceed the memory limits of available computers? Will the algorithm work when there are (small) numerical errors in the execution of its steps, or will the errors be amplified? Does the algorithm require an inordinate number of operations in order to arrive to the proximity of a solution, and hence will it exhaust the user's computer budget and patience?

These issues are important whether the algorithm is implemented on a uniprocessor or on a multiprocessor, parallel, computer. However, additional considerations become important when a parallel computer is used for the implementation. For example, an algorithm that requires a large number of operations to reach a solution could be more efficient than an algorithm that requires fewer operations, if the steps of the former can be executed concurrently. The following additional issues need to be addressed when designing an algorithm for parallel implementation:

Partitioning problem: Partition the operations of the algorithm into sets of operations that can be executed concurrently. These modules of the algorithm are called *tasks*.

Scheduling problem: Assign each task to one or more processors for simultaneous execution. This problem appears, at first glance, to be an

issue for the implementor of the algorithm and not for the mathematician who is designing the algorithm.

However, parallel computers come in a variety of architectures. At one extreme we have machines with few (four, eight, or so) but extremely powerful processing units. The CRAY Y-MP (Cray Research) and the IBM 3090-600 (International Business Machines) are examples of two currently popular models at this end of the spectrum. At the other extreme we have computers with thousands of simple processing elements. The Connection Machine CM–2 (Thinking Machines Corporation), the MassPar and the DAP (Active Memory Technologies) are examples of currently popular models of this category. Scheduling the tasks of an algorithm for concurrent execution is a problem for the algorithm implementor. But the scheduling problem can not be solved satisfactorily unless the underlying theoretical algorithm is flexible. If the algorithm can be structured into as many independent tasks as the number of available processors then the scheduling problem can be resolved. The role of the algorithm designer is therefore important in this respect.

Synchronization problem: Specify an order of execution of the tasks, and instances during the execution of the algorithm where information must be exchanged among the tasks, in order to ensure the correct progress of the iterations according to the algorithm. This problem can again be viewed as one of computer implementation. But algorithms whose convergence is guaranteed without excessive requirements for synchronization are likely to be more efficient when implemented in

parallel.

While the structure of the mathematical algorithm is our focus when addressing the aforementioned design issues, the structure of the underlying application can also play a crucial role in parallel computations. It may be possible to decompose the problem into domains that can be solved independently. For example, in image reconstruction from projections we may consider the decomposition of a discretized cross-section of the image into sub-sections. If the sub-sections are properly chosen the image could be reconstructed from these smaller sections.

This is the background on which these Lecture Notes are being developed. It is not enough to just know what a parallel computer is. And for a mathematician, it is not enough to "develop parallel algorithms". In order to successfully attack a significant real-world problem we have to carefully study the modelling process, the appropriate parallel computing architecture, and develop algorithms which are suitable for both.

It is important to stress that the design of an algorithm should not be linked to the specifics of a parallel computer architecture. Such an approach would render the algorithms obsolete once the architecture changes. Given also the diversity of existing parallel architectures this approach would find limited applicability. The major thesis of these Lecture Notes is that algorithms should be flexible enough to facilitate their implementation on a range of architectures. Indeed, we present algorithms for several optimization problems that — when viewed from the proper perspective — are flexible enough to be implementable on a broad spectrum of parallel machines. The rest of this introduction makes these ideas more precise. Examining the general structure of the iterative algorithms, we understand their suitabil-

ity for parallel implementation, as well as their flexibility for adaptation to different architectures.

Details of the implementation of a parallel algorithm is also an important problem, but one which is bound to be linked closely with the computer architecture. We discuss implementation issues too. To the extend possible, implementations are discussed in a way that is linked to a class of machines, not to any specific hardware model.

1.2 A Classification of Iterative Algorithms

We are dealing with applications that can be modeled by a system of linear or nonlinear equations or inequalities, i.e., a system of the form

$$f_i(x) * 0, \quad i = 1, 2, \ldots, I, \tag{1.1}$$

where $x \in \Re^J$, the J-dimensional Euclidean space, $f_i : \Re^J \to \Re$, and $*$ stand for equality signs, inequality signs or a mixture of such. The Lecture Notes are focused on well-defined applications, models and algorithms, but this generic description allow us to classify a broader set of iterative procedures and discuss their suitability for parallel computations.

Solving a *feasibility problem* (1.1) may be attempted if the system is *feasible*, i.e., $\{x \in \Re^J \mid f_i(x) * 0, \quad i = 1, 2, \ldots, I\} \neq \emptyset$, or even if it is infeasible. The mathematical model may also be set up as an *optimization problem* with an objective function $f_0 : \Re^J \to \Re$ imposed and (1.1) serving as constraints,

$$\begin{cases} \text{opt} \quad f_0(x) \\ \text{subject to} \quad f_i(x) * 0, \ \ i = 1, 2, \ldots, I, \end{cases} \tag{1.2}$$

with or without additional constraints of the form $x \in Q$, where $Q \subseteq \Re^J$ is a given subset, describing, for example, box constraints, and *opt* stands for either *min* or *max*. Specific examples of such problems are described, as they arise in our fields of applications, in the next chapters and references to others are given.

Special-purpose iterative algorithms designed to solve these problems may employ iterations which use in each step a single row of the constraints system (1.1) or a group of rows. A *row-action iteration* has the functional form

$$x^{k+1} \doteq R_{i(k)}(x^k, f_{i(k)}), \tag{1.3}$$

where $i(k)$ is the *control index*, $1 \leq i(k) \leq I$, specifying the row which is acted upon by the algorithmic operator $R_{i(k)}$. The algorithmic operator generates, in a specified manner, the new iterate x^{k+1} from the current iterate x^k, and from the information contained in $f_{i(k)}$. $R_{i(k)}$ may depend on additional parameters which vary from iteration to iteration such as relaxation parameters, weights, tolerances, etc.

The system (1.1) may be decomposed into M groups of constraints (called *blocks*) by choosing integers $\{m_t\}_{t=0}^M$ such that

$$0 = m_0 < m_1 < \cdots < m_{M-1} < m_M = I, \tag{1.4}$$

and defining for each $t, 1 \leq t \leq M$, the subset

$$I_t \doteq \{m_{t-1} + 1,\ m_{t-1} + 2, \ldots, m_t\}, \tag{1.5}$$

yielding the partition

$$\{1, 2, \ldots,\ I\} = I_1 \cup I_2 \cup \cdots \cup I_M. \tag{1.6}$$

A *block-iteration* has then the functional form

$$x^{k+1} \doteq B_{t(k)}(x^k, \{f_i\}_{i \in I_{t(k)}}), \tag{1.7}$$

where $t(k)$ is the control index, $1 \leq t(k) \leq M$, specifying the block which is used when the algorithmic operator $B_{t(k)}$ generates x^{k+1} from x^k and from the information contained in all those rows of (1.1) whose indices belong to $I_{t(k)}$. Again, additional parameters may be included in each $B_{t(k)}$. The special-purpose iterative algorithms that we consider in our classification scheme may address any problem of the form (1.1) or (1.2) and may be classified as having one of the following four basic structures.

1. **Sequential Algorithms.** For this class of algorithms a control sequence $\{i(k)\}_{k=0}^{\infty}$ is defined and the algorithm performs, in a strictly sequential manner, row-action iterations, from an appropriate initial point until a stopping rule applies, according to (1.3).

2. **Simultaneous Algorithms.** Algorithms in this class first execute simultaneously row-action iterations on all rows

$$x^{k+1,i} \doteq R_i(x^k, f_i), \qquad i = 1, 2, \ldots I, \tag{1.8}$$

using the same current iterate x^k, and the next iterate x^{k+1} is then generated from all intermediate ones $x^{k+1,i}$ by an additional operation

$$x^{k+1} \doteq S(\{x^{k+1,i}\}_{i=1}^{I}). \tag{1.9}$$

Here S and R_i are algorithmic operators, the R_i's being all of the row-action type.

3. **Sequential Block-Iterative Algorithms.** Here the system (1.1) is decomposed into fixed blocks according to (1.6) and a control sequence $\{t(k)\}_{k=0}^{\infty}$ over the set $\{1, 2, \ldots, M\}$ is defined. The algorithm performs sequentially, according to the control sequence, block iterations of the form (1.7).

4. **Simultaneous Block-Iterative Algorithms.** In this class, block iterations are first performed using the same current iterate x^k, on all blocks simultaneously

$$x^{k+1,t} \doteq B_t(x^k, \{f_i\}_{i \in I_t}), \qquad t = 1, 2, \ldots, M, \tag{1.10}$$

and the next iterate x^{k+1} is then generated from the intermediate ones $x^{k+1,t}$ by

$$x^{k+1} \doteq S(\{x^{k+1,t}\}_{t=1}^{M}). \tag{1.11}$$

Here S and B_t are algorithmic operators, the B_t's being of the block-iteration type (1.7).

An important generalization of block-iterative algorithms is that of *variable block-iterative* algorithms. In this kind of algorithms block iterations are performed according to formula (1.7). The blocks $I_{t(k)}$, however, are not determined a priori according to (1.6) and kept fixed, but are rather free to be nonempty subsets

$$\emptyset \neq I_{t(k)} \subseteq I. \tag{1.12}$$

This means that as the iterations proceed both the sizes of the blocks and the specific assignments of constraints to blocks may change in a much more

general manner than the scheme with fixed block sizes. Of course, convergence theory of such an algorithm could still impose certain restrictions on how the sets $I_{t(k)}$ may be reconstructed.

Although not formulated so before, many special-purpose iterative algorithms that were proposed and implemented over the years can be identified according to this classification. As we proceed through these Lecture Notes, the algorithmic structures will become clearer with the presentation of specific algorithms.

1.3 Parallel Computations with Iterative Algorithms

We examine now two approaches for introducing parallelism in both the *sequential* and the *simultaneous* algorithms. We address, in particular, the problem of task partitioning, that is, identifying operations of the algorithm that can be executed concurrently.

A sequential row-action algorithm uses information contained in a single row $f_{i(k)}$ and the current iterate x^k, in order to generate the next iterate x^{k+1}. The control index $i(k)$ specifies the order in which rows are selected. Is it possible to select rows in such a way that two (or more) rows can be operated upon simultaneously, without altering the mathematical structure of the algorithm? This can be achieved if at every iteration only different components of the updated vector are changed. The algorithmic operator $R_{i(k)}$ is a vector-to-vector mapping. If $R_{i(k)}$ and $R_{i(k+1)}$ use different components of x^k to operate upon, and leave the others unchanged, then the operations can proceed concurrently. Identifying row-indices $i(k)$ such that the oper-

ator can be applied concurrently on such rows depends on the structure of the family $\{f_i(x)\}_{i=1}^I$. We will see that several important problems have a structure that allows us to identify such row-indices. The parallel algorithm is, with this approach, mathematically identical to the serial algorithm.

The parallelism in the simultaneous algorithms is obvious. The iterates (1.8) can be executed concurrently using up to I parallel processors. The step (1.9) is a synchronization step where the processors must cooperate, and exchange information contained in the I vectors $\{x^{k+1,i}\}_{i=1}^I$ in order to compute the next iterate x^{k+1}.

The block algorithms — sequential or simultaneous — lead to parallel computations in exactly the same two ways outlined above. However, the introduction of blocks permits more flexibility in handling the task scheduling problem. The blocks can be chosen in a way that ensures that all processors receive tasks of the same computational difficulty. Hence, all processors complete their work in (more or less) the same amount of time. Delays while processors wait on each other are minimized. The specification of block-size will typically depend on the computer architecture. For massively parallel systems it is preferable to create a very large number of small blocks. Fewer blocks are needed for systems with a small number of processors.

1.4 Organization of the Lecture Notes

Chapter 2 introduces the notion of a *generalized distance*. It uses then generalized distances to define *generalized projections* onto convex sets and particularly onto hyperplanes. Several properties of generalized distances and projections are established. This chapter lays the theoretical foundations

from which the algorithms of subsequent chapters are developed.

Chapter 3 defines the convex feasibility problem and introduces iterative algorithms for its solution. Methods based on successive orthogonal projections, cyclic subgradient projections, Bregman's nonorthogonal projections are presented together with some well-known algorithms of Cimmino, Agmon-Motzkin-Schoenberg, Oettli, Eremin and others. Convergence results are established and some of the relationships of the algorithms to each other are pointed out.

Chapter 4 defines the problem of optimizing functions that belong to a special class — known as *Bregman* functions — over linear equality, inequality and interval constraints. Bregman's general *row-action* iterative algorithm is presented and its convergence established. This chapter then develops as special cases several well-known algorithms such as Kaczmarz's method for norm- minimization over equality constraints, Hildreth's method for norm minimization over inequality constraints, and a method called ART4 for norm-minimization over interval constraints. Specializations for entropy optimization are also presented.

The algorithms of chapter 4 are applicable only to Bregman functions. Linear programming problems can not be solved directly with these methods. Chapter 5 presents a proximal minimization algorithm that facilitates the solution of linear programming problems using nonlinear perturbations based on generalized distances. The algorithms of chapter 4 are then applicable, and some special cases are presented.

Chapter 6 discussed three distinct areas of applications that give rise to optimization problems such as those discussed earlier. One application is the problem of *matrix balancing*, that is, of adjusting the entries of a

matrix to satisfy certain consistency requirements. Such problems arise in economics, transportation, telecommunications, regional planning and so on. The application areas are briefly reviewed, and the chapter shows how certain approaches to modeling these problems lead to optimization problems. Specific algorithms, derived as special cases from the results of chapter 4 are presented. The second application is the problem of *image reconstruction from projections.* Such problems appear in diagnostic medicine, nondestructive material testing, seismic tomography and other areas. The chapter shows how certain approaches to modeling these problems lead to convex feasibility or optimization problems. The final application is from the area of operations research, and, in particular, for the problem of optimizing a multi-period problem in the face of uncertainty in the data. A *stochastic network programming* formulation of this problem is presented and its application to portfolio optimization for financial planning is briefly discussed, together with other related applications. An algorithm for solving this problem, drawing from the results of chapters 4 and 5 is developed.

Chapter 7 develops the link between the theoretical algorithms of chapters 3 – 5, and the applications of chapter 6, with parallel computing. An introduction to parallel computations is first given. This is followed by discussions on the parallel implementation of algorithms for solving the three major application areas of matrix balancing, image reconstruction and stochastic network programming. Experiences with the computational efficiency of these algorithms, when implemented in parallel, are presented.

Chapter 2

Generalized Distances and Generalized Projections

2.1 Introduction

A *generalized distance* is a real-valued nonnegative function of two vector variables $D(x, y)$, defined in a specific manner, which may be used to "measure" the "distance" between x and y in some generalized sense. When defining generalized distances it is customary to allow nonsymmetry (i.e., $D(x, y) \neq D(y, x)$) and not to insist on the triangle inequality which a traditional distance function must obey. Because of the lack of symmetry such distances are also referred to sometimes as "directed distances". Under certain conditions, one can use a generalized distance to define *projections onto convex sets*. This is done by minimizing $D(x, y)$ over all x in the set, for some fixed y whose projection onto the set is sought. In this chapter we present a family of generalized distances and their associated generalized projections that

serve as basic tools in the subsequent developments in optimization theory. The material in this chapter is taken from Bregman's pioneering paper [31] and from Censor and Lent [47], De Pierro and Iusem [188], Censor et al. [49], and Teboulle [219]. Much work has been done on generalized distances and their applications during recent years. Some (but by no means all) representative sources are: Csiszár [59], Eckstein [80], Chen and Teboulle [53] and Byrne [35].

2.2 Bregman Functions and Generalized Projections

Let S be a nonempty, open, convex set, such that the closure $\overline{S} \subseteq \Lambda$, where Λ is the domain of a function $f : \Lambda \subseteq \Re^n \to \Re$. Assume that $f(x)$ has continuous first partial derivatives at every $x \in S$, and denote by $\nabla f(x)$ its gradient at x.

From $f(x)$, construct the function $D_f(x,y)$, $D_f : \overline{S} \times S \subseteq \Re^{2n} \to \Re$, by

$$D(x,y) \equiv D_f(x,y) \doteq f(x) - f(y) - \langle \nabla f(y), x - y \rangle. \tag{2.1}$$

$D_f(x,y)$ may be interpreted as the difference $f(x) - h(x)$ where $h(z)$ represents the tangent hyperplane to the epigraph of f at the point $(y, f(y))$ in $\Re^{n+1}$.

We adopt the following notation for the *partial level sets of $D_f(x,y)$*. For any $\alpha \in \Re$,

$$\begin{cases} L_1(y,\alpha) \doteq \{x \in \overline{S} \mid D(x,y) \leq \alpha\}, \\ L_2(x,\alpha) \doteq \{y \in S \mid D(x,y) \leq \alpha\}. \end{cases} \tag{2.2}$$

The reason for collecting the conditions (i)-(vi) in the following definition under one heading is that these are precisely the conditions which are

needed to ensure the applicability of certain iterative methods for linearly constrained, equality, inequality and interval convex programming problems, when the function $D(x, y)$ has the form (2.1).

Definition 2.2.1 (Bregman Functions.) *A function $f : \Lambda \subseteq \Re^n \to \Re$ is called a Bregman function if there exists a nonempty, open, convex set S, such that $\overline{S} \subseteq \Lambda$ and the following hold:*

(i) $f(x)$ has continuous first partial derivatives at every $x \in S$;

(ii) $f(x)$ is strictly convex on $\overline{S}$;

(iii) $f(x)$ is continuous on $\overline{S}$;

(iv) for every $\alpha \in \Re$, the partial level sets $L_1(y, \alpha)$ and $L_2(x, \alpha)$ are bounded for every $y \in S$, for every $x \in \overline{S}$, respectively;

(v) if $\lim_{k\to\infty} y^k = y^* \in \overline{S}$, then $\lim_{k\to\infty} D_f(y^*, y^k) = 0$;

(vi) if $\lim_{k\to\infty} D_f(x^k, y^k) = 0$, $\lim_{k\to\infty} y^k = y^* \in \overline{S}$, and $\{x^k\}$ is bounded, then $\lim_{k\to\infty} x^k = y^*$.

We denote the family of Bregman functions by $\mathcal{B}$, refer to the set S as the *zone* of the function f, and write $f \in \mathcal{B}(S)$.

Lemma 2.2.1 *For every $f \in \mathcal{B}(S)$, $D(x, y) \geq 0$ and $D(x, y) = 0$, if and only if $x = y$.*

Proof. This is a basic result in convex analysis which says that the assertion of this lemma is equivalent to the strict convexity of f (Definition 2.2.1 (ii)), when f is differentiable (Definition 2.2.1(i)). See, e.g., Bazaraa and Shetty [19, Theorem 3.3.3]. □

Definition 2.2.2 (Generalized Projections.) *Given $\Omega \subseteq \Re^n$ and $y \in S$, a point $x^* \in \Omega \cap \overline{S}$ for which*

$$\min_{z \in \Omega \cap \overline{S}} D(z, y) = D(x^*, y) \tag{2.3}$$

(i.e., the minimum exists and the equality holds) is denoted by $P_\Omega(y)$ and is called a generalized projection (or simply, a projection) of the point y onto the set Ω.

The next lemma guarantees the existence and uniqueness of generalized projections.

Lemma 2.2.2 *If $f \in \mathcal{B}(S)$, then for any closed convex set $\Omega \subseteq \Re^n$, such that $\Omega \cap \overline{S} \neq \emptyset$, and for any $y \in S$, there exists a unique generalized projection $x^* \equiv P_\Omega(y)$.*

Proof. For any $w \in \Omega \cap \overline{S}$, the set

$$B \doteq \{z \in \overline{S} \mid D(z, y) \leq D(w, y)\} \tag{2.4}$$

is bounded [because of the boundedness of the partial level sets, Definition 2.2.1(iv)] and closed [because $D(z, y)$ is continuous in z on $\overline{S}$, by Definition 2.2.1(iii)]. Therefore, the nonempty (since $w \in B \cap \Omega$) set $T \doteq (\Omega \cap \overline{S}) \cap B$ is bounded; and, since the intersection of closed sets is closed, T is also closed, hence compact. Consequently, $D(z, y)$ as a continuous function of z takes its infimum on the compact set T at a point $x^* \in T$. For every $z \in \Omega \cap \overline{S}$ which lies outside B, we have $D(w, y) < D(z, y)$. Hence, x^* satisfies (2.3) and is a generalized projection of y onto Ω.

To show uniqueness, suppose that there are two points $u, v \in \Omega \cap \overline{S}$, such that $u \neq v$ and

$$D(u, y) = D(v, y) = \min_{z \in \Omega \cap \overline{S}} D(z, y),$$

for some $y \in S$. Then, $\frac{1}{2}(u+v) \in \Omega \cap \overline{S}$, because of the convexity of $\Omega \cap \overline{S}$. We recall that the closure of a convex set is convex (see, e.g., Rockafellar [196, Theorem 6.2]), and the intersection of convex sets is convex. But, since f is strictly convex,

$$\begin{aligned} D(\tfrac{1}{2}(u+v), y) &= f(\tfrac{1}{2}(u+v)) - f(y) - \langle \nabla f(y), \tfrac{1}{2}(u+v) - y \rangle \\ &< \tfrac{1}{2} f(u) + \tfrac{1}{2} f(v) - \tfrac{1}{2} f(y) - \tfrac{1}{2} f(y) \\ &\quad -\tfrac{1}{2}\langle \nabla f(y), u - y \rangle - \tfrac{1}{2}\langle \nabla f(y), v - y \rangle \\ &= \tfrac{1}{2}[D(u,y) + D(v,y)] = \min_{z \in \Omega \cap \overline{S}} D(z,y), \end{aligned}$$

which is a contradiction. □

Example 2.2.1 The function

$$f(x) = \tfrac{1}{2} \|x\|^2 \tag{2.5}$$

is a Bregman function, with

$$\Lambda = S = \overline{S} = \Re^n, \tag{2.6}$$

and $D(x,y)$ is then, according to (2.1),

$$D(x,y) = \tfrac{1}{2} \|x - y\|^2 . \tag{2.7}$$

The generalized projection of any point y onto any closed, convex set Ω is, in this case, the usual orthogonal projection.

Example 2.2.2 The "$x \log x$" (Shannon's) *entropy functional*, ent x, maps the nonnegative orthant $\Re^n_+$ of the n-dimensional Euclidean space $\Re^n$ into $\Re$ according to

$$ent\ x \doteq -\sum_{j=1}^{n} x_j \log x_j . \tag{2.8}$$

Here "log" denotes the natural logarithms and, by definition, $0 \log 0 \doteq 0$.

The next lemma shows that this is indeed a Bregman function and the D_f-function is given by eq. (2.10).

Lemma 2.2.3 *The function (-ent x) is a Bregman function with* $\Lambda = \Re^n_+$ *and zone* S_e *defined by*

$$S_e \doteq \{x \in \Re^n \mid x_j > 0, \quad \forall j = 1, 2, \ldots, n\}. \tag{2.9}$$

Proof. (i) and (ii) of Definition 2.2.1 are simple. Property (iii) is valid due to the convention $0 \log 0 = 0$. For the function -ent x we have that

$$D(x, y) = \sum_{j=1}^{n} x_j \log(x_j/y_j) - 1) + \sum_{j=1}^{n} y_j, \tag{2.10}$$

which is the well-known *Kullback-Liebler cross entropy function* from statistics. Fixing y let any component of x go to $+\infty$. Then $D(x, y) \to +\infty$ as well and so, for any $\alpha \in \Re$, $L_1(y, \alpha)$ is bounded. A similar argument shows that $L_2(x, \alpha)$ is bounded, proving (iv). Property (v) also follows from (2.10).

Assume now that the premises of property (vi) are satisfied. It is sufficient to show that any convergent subsequence of $\{x^k\}$ converges to y^*. Consider a general term of (2.10), namely $t : \Re_+ \times (\Re_+ \setminus \{0\}) \to \Re$ defined by

$$t(x, y) \doteq x(\log(x/y) - 1) + y, \tag{2.11}$$

$x \geq 0$ and $y > 0$. For any fixed y,

$$t(x, y) \geq 0, \tag{2.12}$$

for all $x \geq 0$, and

$$t(x, y) = 0 \quad \text{if and only if} \quad x = y. \tag{2.13}$$

Now consider a convergent subsequence $\{x^{k_s}\}$ of $\{x^k\}$ and assume that $x^{k_s} \rightarrow \overline{x}$, as $s \rightarrow \infty$.

$$D(x^{k_s}, y^{k_s}) = \sum_{j=1}^{n} t(x_j^{k_s}, y_j^{k_s}) \longrightarrow 0, \quad as \quad s \rightarrow \infty. \tag{2.14}$$

From (2.12) it follows, that for each j,

$$t(x_j^{k_s}, y_j^{k_s}) \longrightarrow 0, \quad as \quad s \rightarrow \infty. \tag{2.15}$$

Noting that $x_j^{k_s} \rightarrow \overline{x}_j$ and $y_j^{k_s} \rightarrow y_j^*$, consider two cases. If $y_j^* > 0$, then (2.13) and (2.15) imply that $y_j^* = \overline{x}_j$. If $y_j^* = 0$, then (2.11) and (2.15) imply that $\overline{x}_j = 0$, and again $y_j^* = \overline{x}_j$. Hence $y^* = \overline{x}$. □

Example 2.2.3 Denote the Cartesian product of n intervals (a, b) by $(a, b)^n$, let $S = (-1, 1)^n$, and define

$$f(x) \doteq -\sum_{j=1}^{n} \sqrt{1 - x_j^2}. \tag{2.16}$$

Then it is straightforward to confirm that f meets conditions (i)-(iv) of Definition 2.2.1. By suitable scaling and translation of the argument, f can be transformed so that its zone is the Cartesian product of a collection of n arbitrary open bounded intervals.

Example 2.2.4 If f is a Bregman function, then $f(x) + \langle c, x \rangle + \beta$, for any $c \in \Re^n$ and $\beta \in \Re$, also meets the conditions and produces exactly the same D_f-function, and therefore must also be a Bregman function. More examples can be found in Teboulle [219].

2.3 Generalized Projections onto Hyperplanes

A key role in the iterative projection methods for linearly constrained convex programming discussed in Chapter 4 is played by generalized projections on hyperplanes. Therefore we take a closer look at these projections now. A *hyperplane* is a set of the form

$$H = \{x \in \Re^n \mid \langle a, x \rangle = b\}, \tag{2.17}$$

where $a \in \Re^n$ and $b \in \Re$ are given. To facilitate our presentation, we introduce the following definition.

Definition 2.3.1 (Zone Consistency.) *(i) A function $f \in \mathcal{B}(S)$ with zone S is said to have the zone consistency property with respect to the hyperplane H if, for every $y \in S$, we have $P_H(y) \in S$. That is, the function has the property that the generalized projection of any point $y \in S$ onto the hyperplane H remains in S.*

(ii) $f \in \mathcal{B}(S)$ with zone S is strongly zone consistent with respect to the hyperplane H if it is zone consistent with respect to H as well as with respect to every other hyperplane H' which parallels H and lies between y and H.

The following lemma characterized generalized projections onto hyperplanes.

Lemma 2.3.1 *Let $f \in \mathcal{B}(S)$, $H = \{x \mid \langle a, x \rangle = b\}$, and assume that f is zone consistent with respect to H. For any given $y \in S$, the system*

$$\nabla f(x^*) = \nabla f(y) + \lambda a, \tag{2.18}$$

$$\langle a, x^* \rangle = b \tag{2.19}$$

determines uniquely the point x^* which is the generalized projection of y onto H. For a fixed representation of H (i.e., for fixed $a \in \Re^n$ and $b \in \Re$), the system also determines uniquely the real number λ.

Remark 2.3.1 *For some fixed $a \in \Re^n$ and $b \in \Re$ representing the hyperplane H, the λ obtained from the system (2.18)-(2.19) is called the generalized projection parameter associated with the generalized projection of y onto H.*

Proof of Lemma 2.3.1. Consider the constrained minimization problem (2.3) with $\Omega = H$. Due to the zone consistency assumption, we look for a minimum within the open set S, so that we can write the Lagrangian of this problem as

$$L(z, y, \lambda) = f(z) - f(y) - \langle \nabla f(y), z - y \rangle - \lambda(\langle a, z \rangle - b), \tag{2.20}$$

and the necessary conditions for a point to be the generalized projection of y onto H are then

$$\nabla_z L(z, y, \lambda) = 0, \quad \nabla_\lambda L(z, y, \lambda) = 0, \tag{2.21}$$

from which (2.18)-(2.19) follow. To show the uniqueness of x^* and the parameter λ, for given a and b, assume that $x^{**} \in \Re^n$ and the parameter μ also solve (2.18)-(2.19), i.e.,

$$\nabla f(x^{**}) = \nabla f(y) + \mu a, \tag{2.22}$$

$$\langle a, x^{**} \rangle = b, \tag{2.23}$$

and that $x^* \neq x^{**}$. Then, multiplying (2.18) through by $x^{**} - x^*$ and using (2.19) and (2.13), we get

$$\langle \nabla f(x^*), x^{**} - x^* \rangle = \langle \nabla f(y), x^{**} - x^* \rangle. \tag{2.24}$$

Since $x^* \neq x^{**}$ we know, from (2.1) and Lemma 2.2.1, that

$$f(x^{**}) - f(x^*) > \langle \nabla f(x^*), x^{**} - x^* \rangle, \tag{2.25}$$

thus, combining (2.24) and (2.25),

$$f(x^{**}) - f(x^*) > \langle \nabla f(y), x^{**} - x^* \rangle. \tag{2.26}$$

Using similar arguments, after first multiplying (2.22) through by $x^* - x^{**}$, we obtain

$$f(x^*) - f(x^{**}) > \langle \nabla f(y), x^* - x^{**} \rangle. \tag{2.27}$$

The contradiction $0 > 0$ is then obtained by adding up (2.26) and (2.27). Consequently $x^* = x^{**}$ which implies also that $\lambda = \mu$. □

Remark 2.3.2 It is also important to note, and easy to check, that, if (2.18) is considered alone and λ is prespecified, then for given a and y, x^* is uniquely determined.

What is fairly obvious in the case of orthogonal projections needs verification here. The next result is a statement about the signs of the parameters associated with generalized projections onto hyperplanes.

Lemma 2.3.2 *Let $H = \{x \mid \langle a, x \rangle = b\}$ and $y \in S$. For any $f \in \mathcal{B}(S)$ which is zone consistent with respect to H, we have for the parameter λ associated with the generalized projection of y onto H, in some particular representation (i.e., $a \neq 0$ and b given) of H, that*

$$\lambda(b - \langle a, y \rangle) > 0, \qquad \text{if } y \notin H, \tag{2.28}$$

$$\lambda = 0, \qquad \text{if } y \in H. \tag{2.29}$$

Proof. From (2.18), (2.19), we get, if $x^* \neq y$, that

$$\langle \nabla f(x^*) - \nabla f(y), x^* - y \rangle = \lambda \langle a, x^* - y \rangle;$$

but

$$\langle \nabla f(x^*) - \nabla f(y), x^* - y \rangle = D(x^*, y) + D(y, x^*),$$

and also

$$\langle a, x^* - y \rangle = b - \langle a, y \rangle.$$

Therefore, if $y \notin H$,

$$\lambda(b - \langle a, y \rangle) = D(x^*, y) + D(y, x^*), \tag{2.30}$$

and the result follows from Lemma 2.2.1. If $\langle a, y \rangle = b$, then $x^* = y$ is the unique generalized projection of y onto H; thus, $\lambda = 0$ □

Assume that orthogonal projections are performed successively onto two given parallel hyperplanes (i.e., having the same normal vector a) starting from some initial point. In this case the final point is the same point that we would reach by orthogonally projecting the initial point onto the second hyperplane. The same thing happens with generalized projections, as the next lemma shows.

Lemma 2.3.3 *Let H_1 and H_2 be two parallel hyperplanes in $\Re^n$ with representations*

$$H_1 = \{x \mid \langle a, x \rangle = b_1\}, \quad H_2 = \{x \mid \langle a, x \rangle = b_2\},$$

and let $y \in S$. Then, for any $f \in \mathcal{B}(S)$ which is zone consistent with respect to both hyperplanes,

$$P_{H_2}(P_{H_1}(y)) = P_{H_2}(y), \tag{2.31}$$

and the associated projection parameters obey the equation

$$\lambda_1 + \hat{\lambda}_2 = \lambda_2, \tag{2.32}$$

where λ_i, $i = 1, 2$, *is the parameter associated with projecting* y *onto* H_i, *and* $\hat{\lambda}_2$ *is the parameter associated with projecting* $P_{H_1}(y)$ *onto* H_2.

Proof. From Lemma 2.3.1 we have

$$\nabla f(x^1) = \nabla f(y) + \lambda_1 a, \quad \langle a, x^1 \rangle = b_1, \tag{2.33}$$

$$\nabla f(x^2) = \nabla f(y) + \lambda_2 a, \quad \langle a, x^2 \rangle = b_2, \tag{2.34}$$

$$\nabla f(\hat{x}^2) = \nabla f(x^1) + \hat{\lambda}_2 a, \quad \langle a, \hat{x}^2 \rangle = b_2, \tag{2.35}$$

describing the projections of y onto H_1, of y onto H_2, and of $x^1 = P_{H_1}(y)$ onto H_2, respectively. Now, substitute (2.33) into (2.35) to obtain

$$\nabla f(\hat{x}^2) = \nabla f(y) + (\lambda_1 + \hat{\lambda}_2) a, \quad \langle a, \hat{x}^2 \rangle = b_2. \tag{2.36}$$

The uniqueness of projection implies that $x^2 = \hat{x}^2$ and that $\lambda_1 + \hat{\lambda}_2 = \lambda_2$. □

Lemma 2.3.4 *If the conditions of Lemma 2.3.3 hold, with*

$$H_1 = \{x \mid \langle a, x \rangle = \gamma\}, \quad H_2 = \{x \mid \langle a, x \rangle = \delta\},$$

then $\Gamma \leq \Delta$, *if and only if* $\gamma \leq \delta$, *where* Γ *and* Δ *are the parameters associated with the projections of* y *onto* H_1 *and* H_2, *as represented, respectively.*

Proof. Observe that

$$\nabla f(z^1) = \nabla f(y) + \Gamma a, \quad \langle a, z^1 \rangle = \gamma, \tag{2.37}$$

$$\nabla f(z^2) = \nabla f(y) + \Delta a, \quad \langle a, z^2 \rangle = \delta. \tag{2.38}$$

Subtracting and then taking the inner product with $z^2 - z^1$ gives

$$\langle \nabla f(z^2) - \nabla f(z^1), z^2 - z^1 \rangle = (\Delta - \Gamma)(\delta - \gamma), \tag{2.39}$$

and the left-hand side equals $D(z^1, z^2) + D(z^2, z^1)$, which is nonnegative by Lemma 2.2.1. □

2.4 Characterization of a Family of Bregman Functions

The six conditions of Definition 2.2.1 are burdensome and also are not all directly formulated for the function f. Some of them are related to f through the level sets or through properties of the D_f-functions. Therefore it is useful to provide more easily checkable conditions that would guarantee that $f \in \mathcal{B}(S)$. In this subsection we present such conditions for functions defined on all $\Re^n$ derived by De Pierro and Iusem [188].

Theorem 2.4.1 *Let $f : \Re^n \to \Re$ be a function which satisfies the following two properties:*

(I) f is twice continuously differentiable and strictly convex;

(II) $\lim_{\|x\| \to \infty} [f(x) / \| x \|] = \infty$.

Then $f \in \mathcal{B}(\Re^n)$.

The proof of Theorem 2.4.1 requires four lemmas. Before presenting them, observe that, in Definition 2.1.1, conditions (v) and (vi) hold trivially in the interior of the zone, as a consequence of condition (i), and need to be checked only on the boundary of S. So, when considering functions defined on all $\Re^n$, those conditions are trivially satisfied. Also, in this case,

condition (ii) is a consequence of (I), which also implies conditions (i) and (iii) of Definition 2.1.1. The only difficulty lies in condition (iv).

Consider a function $g : \Re^n \setminus \{0\} \to \Re$ which has the following properties:

(III) g is twice continuously differentiable;

(IV) $\lim_{\|x\| \to \infty} g(x) = \infty$;

(V) the function $h : \Re^n \to \Re$ defined by $h(x) \doteq g(x) \, \| x \|$, if $x \neq 0$, and $h(0) \doteq 0$, is strictly convex.

For $x \neq 0$, define the function $\varphi_x : \Re_{++} \to \Re$ as

$$\varphi_x(\lambda) \doteq \langle x, \nabla f(\lambda x) \rangle, \tag{2.40}$$

where $\Re_{++}$ is the set of positive real numbers. Because of (III), φ_x is continuously differentiable. The sequence of four lemmas now follows.

Lemma 2.4.1 *For any* $x \neq 0$, $\varphi_x(1) \leq 0$ *implies* $\varphi_x(\lambda) \leq 0$, *for all* $\lambda \in (0, 1]$.

Proof. Assume that there exists a $\lambda_0 \in (0, 1)$ such that

$$\varphi_x(\lambda_0) > 0. \tag{2.41}$$

Let $\lambda_1 \doteq \inf\{\lambda \mid \lambda > \lambda_0, \ \varphi_x(\lambda) = 0\}$, so that

$$\varphi_x(\lambda) > 0, \qquad \text{for } \lambda \in (\lambda_0, \lambda_1). \tag{2.42}$$

Since φ_x is continuous, λ_1 is well-defined and

$$\varphi_x(\lambda_1) = 0; \tag{2.43}$$

also, the derivative

$$\varphi_x'(\lambda_1) \leq 0, \tag{2.44}$$

because otherwise, $\varphi_x(\lambda) < \varphi_x(\lambda_1)$, for $\lambda \in (\lambda_1 - \epsilon, \lambda_1)$, for some positive ϵ, in contradiction with (2.42). From (2.43), (2.44), we have

$$0 \geq 2\lambda_1 \varphi_x\ (\lambda_1) + \lambda_1^2 \varphi_x'(\lambda_1) = 2\lambda_1 \langle x, \nabla g(\lambda_1 x)\rangle + \lambda_1^2 \langle x, \nabla^2 g(\lambda_1 x) x\rangle$$

$$= 2\langle y, \nabla g(y)\rangle + \langle y, \nabla^2 g(y) y\rangle, \quad \text{with } y \doteq \lambda_1 x, \tag{2.45}$$

where $\nabla^2 g$ is the Hessian matrix of g. From condition (V), we have

$$0 < \langle y, \nabla^2 h(y) y\rangle = 2\langle y, \nabla g(y)\rangle + \langle y, \nabla^2 g(y) y\rangle,$$

in contradiction with (2.45). □

Lemma 2.4.2 *For every $x \neq 0$, there exists $\mu > 1$ such that $\varphi_x(\mu) > 0$.*

Proof. Otherwise, $0 \geq \varphi_x(\mu) = \langle x, \nabla g(\mu x)\rangle$, for all $\mu > 1$. Then, by the mean-value theorem, $g(\mu x) \leq g(x)$ would hold for all $\mu > 1$, in contradiction with (IV). □

Lemma 2.4.3 *The set $V \doteq \{x \in \Re^n \setminus \{0\} \mid \langle x, \nabla g(x)\rangle \leq 0\} = \{x \in \Re^n \setminus \{0\} \mid \varphi_x(1) \leq 0\}$, is a bounded set.*

Proof. Suppose that there exists a sequence $\{x^k\}_{k=1}^{\infty} \subseteq V$ such that $\lim_{k\to\infty} \| x^k \| = \infty$. Let x^* be a cluster point of $\{x^k / \| x^k \|\}$. Take any $\mu \geq 1$. Given $\epsilon > 0$, take k such that $\| (x^k / \| x^k \|) - x^* \| < \epsilon/\mu$ and $\| x^k \| > \mu$. Let $\lambda \doteq \mu / \| x^k \| < 1$. So,

$$\| \lambda x^k - \mu x^* \| < \epsilon. \tag{2.46}$$

By Lemma 2.4.1, $\lambda x^k \in V$, and $\mu \geq 1$, implies $\| \lambda x^k \| \geq 1$. Because of continuity of φ_x, the set $\hat{V} \doteq V \cap \{x \in \Re^n \mid \| x \| \geq 1\}$ is closed, and therefore (2.46) implies that $\mu x^* \in \hat{V} \subset V$. So, $\varphi_{x^*}(\mu) \leq 0$, for all $\mu \geq 1$, in contradiction with Lemma 2.4.2. □

Lemma 2.4.4 *If f satisfies conditions (I) and (II) of Theorem 2.4.1, then*

(i) $\lim_{\|x\|\to\infty}(f(x) - \langle a, x\rangle) = \infty$, *for all* $a \in \Re^n$; *and*

(ii) $\lim_{\|x\|\to\infty}[\langle (x-a), \nabla f(x)\rangle - f(x)] = \infty$, *for all* $a \in \Re^n$.

Proof. (i) This part is an exercise in Ortega and Rheinbolt [184, p.110]. Using (II), take M such that $f(x)/\parallel x \parallel \geq 2 \parallel a \parallel$, for $\parallel x \parallel > M$. Then, $\parallel a \parallel \parallel x \parallel \leq f(x) - \parallel a \parallel \parallel x \parallel \leq f(x) - \langle a, x\rangle$, and the left-hand side tends to infinity as $\parallel x \parallel \to \infty$, if $a \neq 0$. If $a = 0$, (i) follows directly from (II).

(ii) For an arbitrary $\rho > 0$, consider the function

$$\overline{f}(x) \doteq f(x+a) + \rho.$$

Clearly, $\overline{f}$ satisfies (I) and (II). So, $g(x) \doteq \overline{f}(x)/\parallel x \parallel$ satisfies (II), (IV), (V), and, from Lemma 2.4.3, the set $V = \{x \in \Re^n \setminus \{0\} \mid \langle x, \nabla g(x)\rangle \leq 0\}$ is bounded. Take M such that, for $\parallel y \parallel > M$, $\langle y, \nabla g(y)\rangle \geq 0$. Then, $0 \leq \langle y, \nabla g(y)\rangle = (1/\parallel y \parallel)[\langle y, \nabla \overline{f}(y)\rangle - \overline{f}(y)]$ implies that $0 \leq \langle y, \nabla \overline{f}(y)\rangle - \overline{f}(y)$, for $\parallel y \parallel > M$.

Letting $x \doteq y + a$, we obtain that, for $\parallel x \parallel \geq M + \parallel a \parallel$,

$$0 \leq \langle (x-a), \nabla f(x)\rangle - f(x) - \rho \;\Rightarrow\; \rho \leq \langle (x-a), \nabla f(x)\rangle - f(x).$$

Since ρ is arbitrary, (ii) holds. □

Proof of Theorem 2.4.1. As noted before, only condition (iv) in Definition 2.2.1 has to be checked. We have

$$L_1(y, \alpha) = \{x \mid f(x) - \langle \nabla f(y), x\rangle \leq \alpha + f(y) - \langle \nabla f(y), y\rangle\}.$$

Applying Lemma 2.4.4(i) with $a \doteq \nabla f(y)$, yields

$$\lim_{\|x\|\to\infty}[f(x) - \langle \nabla f(y), x\rangle] = \infty,$$

thus $L_1(y,\alpha)$ is bounded for all $y \in \Re^n$. For

$$L_2(x,\alpha) = \{y \mid \langle (y-x), \nabla f(y)\rangle - f(y) \leq \alpha - f(x)\},$$

apply Lemma 2.4.4(ii) with $a \doteq x$, and conclude that

$$\lim_{\|y\|\to\infty} [\langle (y-x), \nabla f(y)\rangle - f(y)] = \infty.$$

So, $L_2(y,\alpha)$ is bounded for all $x \in \Re^n$. □

Condition (II) is not necessary for a function to belong to $\mathcal{B}$, even for twice continuously differentiable functions defined in all $\Re^n$, as the following example, with $n = 1$, shows:

$$f(x) \doteq \begin{cases} \frac{1}{2}(x^2 - 4x + 3), & \text{if } x \leq 1, \\ -\log x, & \text{if } x \geq 1. \end{cases}$$

It is straightforward to verify that f is a twice continuously differentiable Bregman function; but, $\lim_{x\to+\infty}[f(x)/x] = 0$.

2.5 Characterization of Generalized Projections

The characterization of generalized projections given below depends on the following basic result which we repeatedly use in the sequel.

Theorem 2.5.1 *Let $f \in \mathcal{B}(S)$ and let $\Omega \subseteq \Re^n$ be a closed convex set such that $\Omega \cap \overline{S} \neq \emptyset$. Assume that $y \in S$, implies $P_\Omega(y) \in S$. Let $z \in \Omega \cap \overline{S}$, then for any $y \in S$ the inequality*

$$D_f(P_\Omega(y), y) \leq D_f(z,y) - D_f(z, P_\Omega(y)), \tag{2.47}$$

holds.

Proof. This is a specialization of Bregman [31, Lemma 1] for the case of D_f-functions. Define the function

$$G(u) \doteq D_f(u, y) - D_f(u, P_\Omega(y)). \tag{2.48}$$

Expanding G(u) according to (2.1) shows that it is an affine function of the form $\langle u, a\rangle + b$, where the vector a and the real b are independent of u, thus G(u) is convex. For any λ, $0 \leq \lambda \leq 1$, we denote $u_\lambda \doteq \lambda z + (1-\lambda)P_\Omega(y)$ and obtain, due to the convexity of G(u), that

$$D_f(u_\lambda, y) - D_f(u_\lambda, P_\Omega(y)) \leq \lambda[D_f(z, y) - D_f(z, P_\Omega(y))] + (1-\lambda)D_f(P_\Omega(y), y). \tag{2.49}$$

This leads, for $\lambda > 0$, to

$$\begin{aligned} D_f(z, y) - D_f(z, P_\Omega(y) - D_f(P_\Omega(y), y) \;&\geq\; (1/\lambda)[D_f(u_\lambda, y) - D_f(P_\Omega(y), y)] \\ &-\; (1/\lambda)D_f(u_\lambda, P_\Omega(y)). \end{aligned} \tag{2.50}$$

The first term on the right-hand-side of (2.50) is nonnegative because of (2.3). The second term tends to zero as $\lambda \to 0$ because straightforward calculation shows that

$$\nabla_x D_f(x, P_\Omega(y)) \mid_{x=P_\Omega(y)} = 0, \tag{2.51}$$

using (2.1) and Definition 2.2.1(i). □

From this, the following characterization of generalized projections is obtained.

Theorem 2.5.2 *Under the assumptions of Theorem 2.5.1, for any $x \in S$, $x^* = P_\Omega(x)$ if an only if*

$$\langle u - x^*, \nabla f(x) - \nabla f(x^*)\rangle \leq 0, \quad \forall u \in \Omega \cap \overline{S}. \tag{2.52}$$

Proof. From the definition of D_f, we verify that

$$D_f(v,x) = D_f(u,x) - D_f(u,v) + \langle \nabla f(x) - \nabla f(v), u - v \rangle. \tag{2.53}$$

If $v = x^* = P_\Omega(x)$, then (2.52) follows from (2.47) and (2.53). In the other direction, if (2.52) holds, then for some $x^* \in S$, (2.53) reduces, because of the nonnegativity of D_f, to

$$D_f(x^*,x) \leq D_f(u,x), \quad \forall u \in \Omega \cap \overline{S}, \tag{2.54}$$

which means that $x^* = P_\Omega(x)$. □

This was proved in a different way by Teboulle in [219, Theorem 2.1]. For the special case f(x) = $\frac{1}{2} \| x \|^2$, with $S = \Re^n$, Theorem 2.5.2 reduces to the well-known result about orthogonal projections, see, e.g., Kinderlehrer and Stampacchia [142, Theorem 2.3 in Chapter I].

Chapter 3

Iterative Methods for the Convex Feasibility Problem

3.1 Introduction

The *convex feasibility problem* (**CFP**, for short) is to find a point in the nonempty intersection $Q \doteq \cap_{i=1}^{m} Q_i \neq \emptyset$ of a finite family of closed convex sets $Q_i \subseteq \Re^n$, $i = 1, 2, \ldots, m$, in the n-dimensional Euclidean space $\Re^n$. This fundamental problem has many applications in mathematics as well as in other fields, including those treated later in these Lecture Notes. When the sets are given in the form

$$Q_i = \{x \in \Re^n \mid g_i(x) \leq 0, \ g_i \ \text{is a convex function}\}, \tag{3.1}$$

then we deal with the problem of solving a system of inequalities with convex functions, of which the linear case is an important special case.

In this chapter we present several iterative methods for the convex feasibility problem, aiming at a description of the methods along with some of

the apparent connections between them. Section 3.2 contains a brief review of the following methods.

(i) The method of successive orthogonal projections (**SOP**) of Gubin, Polyak and Raik, [105] also known in recent literature on image recovery as the method of projections onto convex sets (**POCS**), see, e.g., Youla [230], Sezan and Stark [209].

(ii) The cyclic subgradient projections method (**CSP**) of Eremin [85], Raik [194] and Censor and Lent [48].

(iii) The interior points algorithm (**IP**) and the (δ, η)-algorithm of Aharoni, Berman and Censor [4].

(iv) The general schemes of Oettli [183] and of Eremin [84, 85].

The Block-Iterative Projections (**BIP**) method of Aharoni and Censor [5], see also Flam and Zowe [92] and Butnariu and Censor [33], discussed in Section 3.3, generalizes the **SOP** method by allowing a great flexibility, in each iterative step, in choosing the subset of constraints sets $\{Q_i\}$ with respect to which the iteration is performed.

Replacing orthogonal projections, in the **SOP** method, by generalized one gives rise to Bregman's sequential algorithm for the convex feasibility problem, presented in Section 3.4.

In contrast with the constructive approach taken here, the reader will find existence results for systems of (mainly strict) inequalities involving convex functions in Fan et al. [91], and in [196, Section 21].

Many of the methods obey a specific control sequence and employ relaxation parameters. A *control sequence* $\{i(k)\}_{k=0}^{\infty}$ is a sequence of indices according to which individual sets Q_i may be chosen for the execution of an iterative algorithm. Here are some important controls.

1. *Cyclic control.* $i(k) = k(\text{mod } m) + 1$, where m is the number of sets in the convex feasibility problem.

2. *Almost cyclic control.* $i(k)\}_{k=0}^{\infty}$ *is almost cyclic on* $\{1, 2, \cdots, m\}$ if $1 \leq i(k) \leq m$, for all $k \geq 0$, and there exists an integer C such that, for all $k \geq 0$, $\{1, 2, \cdots, m\} \subseteq \{i(k+1), i(k+2), \cdots, i(k+C)\}$.

   Almost cyclic controls are less restrictive than the cyclic control and therefore add an important option as to how the application of a method to a particular problem may be carried out. This flexibility could be very important in devising parallel implementations of the algorithms. A cyclic control is almost cyclic with $C = m$.

3. *Repetitive control.* The control $\{i(k)\}_{k=0}^{\infty}$ is called repetitive on $\{1, 2, \cdots, m\}$ if $1 \leq i(k) \leq m$, for all $k \geq 0$, and for every $1 \leq l \leq m$ and every $k \geq 0$, there exists a $k' > k$ such that $i(k') = l$.

4. *Remotest set control.* This is obtained by determining $i(k)$ such that:

$$d(x^k, Q_{i(k)}) = \max\{d(x^k, Q_i) \mid i = 1, 2, \cdots, m\} \tag{3.2}$$

   where $d(x^k, Q_i)$ is the Euclidean distance between x^k and the set Q_i.

Other controls, i.e., the *approximately remotest set control*, the *most violated constraint control*, and the *threshold control*, are mentioned in Section 4.3 below.

The sequence $\{\lambda_k\}_{k=0}^{\infty}$ of *relaxation parameters*, appearing in several methods, allows, loosely speaking, to overdo or underdo the move prescribed in an iterative step. Relaxation parameters add an extra degree of freedom to the way a method might actually be implemented and have important practical consequences.

3.2 A Review of Some Methods

3.2.1 The Method of Successive Orthogonal Projections

A well-known iterative algorithm for solving the convex feasibility problem is the *method of successive orthogonal projections* (**SOP**, for short) of Gubin, Polyak and Raik [105]. Starting from an arbitrary point, the method generates a sequence $\{x^k\}_{k=0}^{\infty}$ that converges to a point in Q, by performing successive, possibly relaxed, orthogonal projections onto the individual convex sets Q_i.

Algorithm 3.2.1 The SOP Method.

Initialization: $x^0 \in \Re^n$ arbitrary.

Iterative Step:

$$x^{k+1} = x^k + \lambda_k(P_{Q_{i(k)}}(x^k) - x^k), \tag{3.3}$$

where $P_{Q_{i(k)}}(x^k)$ stands for the orthogonal projection of x^k onto the set $Q_{i(k)}$ and $\{\lambda_k\}_{k=0}^{\infty}$ are relaxation parameters confined to $\eta \leq \lambda_k \leq 2 - \eta$, for all $k \geq 0$, with some $\eta > 0$.

Gubin et al. [105] proved convergence of the **SOP** method under cyclic control and their proof carries over easily to the almost cyclic case. They also proved convergence for the remotest set control.

The **SOP** method is particularly useful when the projections onto the individual sets are easily calculated. In general, however, application of this method would require at each iterative step the solution of a subsidiary

minimization problem associated with the projection onto the current set, namely,

$$\min_{y \in Q_{i(k)}} \| x^k - y \|, \tag{3.4}$$

where $\| \cdot \|$ stands for the Euclidean norm. An instance of easy enough projections is the linear case, where Q_i are half-spaces in $\Re^n$, in which the **SOP** method coincides with the relaxation method of Agmon [3] and Motzkin and Schoenberg [165].

For the convex feasibility problem with sets of the form (3.1) the iterative step of the **SOP** method requires that a move will be made in a direction determined by $\nabla g_{i(k)}(x^{k+1})$, the gradient of $g_{i(k)}$ calculated at the *next*, and momentarily not yet known, iterate x^{k+1}. Methods which circumvent this difficulty in various ways are described below.

3.2.2 The Cyclic Subgradient Projections Method

The cyclic subgradient projections method (**CSP**, for short) differs from the **SOP** method since it requires at each iterative step a move in the direction of the gradient (or subgradient) calculated at the *current* available iterate, i.e., $\nabla g_{i(k)}(x^k)$. In this way some of the computational difficulties associated with the **SOP** method are circumvented. The **CSP** method presented by Censor and Lent [48] is as follows.

Algorithm 3.2.2 CSP Method.

Initialization: $x^0 \in \Re^n$ arbitrary.

Iterative Step:

$$x^{k+1} = x^k - \lambda_k \frac{g^+_{i(k)}(x^k)}{\| t^k \|^2} t^k, \tag{3.5}$$

where $g_i^+(x) \doteq \max\{0, g_i(x)\}$, $t^k \in \partial g_{i(k)}^+(x^k)$ is a subgradient (see, for example, [196, page 214] of $g_{i(k)}^+$ at the point x^k and the relaxation parameters $\{\lambda_k\}_{k=0}^{\infty}$ are confined to the interval $\epsilon_1 \leq \lambda_k \leq 2 - \epsilon_2$, for all $k \geq 0$, with some $\epsilon_1, \epsilon_2 > 0$.

Control: $\{i(k)\}_{k=0}^{\infty}$ is almost cyclic on $\{1, 2, \ldots, m\}$.

Remotest-set-controlled, or other noncyclically controlled methods with the same iterative step can be derived from the schemes of Oettli and of Eremin described below. The **CSP** method is actually a slight generalization of the method of Eremin [85] and of Raik [194]. An extension of the **CSP** method to the reverse convex feasibility problem was proposed and experimented with, though convergence was not proven, by Censor et al. [45].

3.2.3 An "Interior Points" Algorithm

An orthogonal projection of a point x onto a set Q amounts to an orthogonal projection of x onto the particular hyperplane H which separates x from Q, and supports Q at the closest point to x. In view of the simplicity of an orthogonal projection onto a hyperplane, it is natural to ask whether one could use other separating supporting hyperplanes instead of the particular hyperplane through the closest point to x. Aside from theoretical interest, this approach leads to algorithms useful in practice, provided that the computational load of finding such other hyperplanes favorably competes with the work involved in performing orthogonal projections onto the given sets.

With these thoughts in mind, a general framework for the design of algorithms for the convex feasibility problem is given by the, so-called,

(δ, η) – *algorithm.* With a given point $x \in \Re^n$ and a given closed and convex set $Q \subseteq \Re^n$ we associate a set $A_Q(x)$ in the following manner. Choose $0 \leq \delta \leq 1$ and $0 \leq \eta$ and let $B \doteq B(x, \delta d(x, Q))$ be the ball centered at x with radius $\delta d(x, Q)$, where $d(x, Q)$ is the Euclidean distance between x and Q. For $x \notin Q$ denote by $\mathcal{H}_{x,Q}$ the set of all hyperplanes which separate (see, for example, [196, page 95]) B from Q.

Define:

$$A_Q(x) \doteq \begin{cases} \{x\}, & \text{if } x \in Q, \\ \{x + \lambda(P_H(x) - x) \mid H \in \mathcal{H}_{x,Q}, \ \eta \leq \lambda \leq 2 - \eta\}, & \text{if } x \notin Q. \end{cases} \tag{3.6}$$

Algorithm 3.2.3 The (δ, η)-Algorithm.

Initialization: $x^0 \in \Re^n$ arbitrary.

Iterative Step:

$$x^{k+1} \in A_{Q_{i(k)}}(x^k), \tag{3.7}$$

Control: $\{i(k)\}_{k=0}^{\infty}$ is a repetitive control sequence (see Section 3.1 above).

The next convergence theorem is proved in Aharoni et al. [4].

Theorem 3.2.1 *A sequence $\{x^k\}_{k=0}^{\infty}$ generated by a (δ, η)- algorithm with $\delta > 0$ and $\eta > 0$ converges to a point in $Q = \cap_{i=1}^{m} Q_i$.*

Observation. For $\delta = 1$ and any $\eta > 0$ the (δ, η)-algorithm coincides with the **SOP** method because then all hyperplanes in $\mathcal{H}_{x,Q}$ are also supporting Q at the closest point to x.

Another realization of the (δ, η)-algorithm is the *interior points* (IP) *algorithm* of [4]. Here is a concrete version of this algorithm for the convex feasibility problem with sets of the form (3.1). A prerequisite of this algorithm is the availability of interior points in the individual sets, thus the sets Q_i have to be solid sets (i.e., have nonempty interiors).

Algorithm 3.2.4 Algorithm IP.

Initialization:

Step 1. Locate interior points y^i such that $g_i(y^i) < 0, \quad i = 1, 2, \ldots, m$. Choose an arbitrary $x^0 \in \Re^n$.

Iterative Step:

Step 2. Given x^k, pick $Q_{i(k)}$ by $i(k) = k(\text{mod } m) + 1$.

Step 3. If $g_{i(k)}(x^k) \leq 0$, set $x^{k+1} = x^k$ and return to **Step 2.**

Step 4. If $g_{i(k)}(x^k) > 0$, define $z(\theta) \doteq \theta y^{i(k)} + (1 - \theta)x^k$, where $0 \leq \theta \leq 1$.

Step 5. Solve the single equation $g_{i(k)}(z(\theta)) = 0$, denote by θ_k the smallest value of θ for which $z(\theta)$ solves this equation, and set $z^k = z(\theta_k)$.

Step 6. Calculate a subgradient $t_k \in \partial g_{i(k)}(z^k)$.

Step 7. Let

$$x^{k+1} = x^k + \lambda_k \frac{\langle t^k, z^k \rangle - \langle t^k, x^k \rangle}{\| t^k \|^2} t^k, \tag{3.8}$$

where $0 < \eta \leq \lambda_k \leq 2 - \eta$ are relaxation parameters, and return to **Step 2.**

3.2.4 The Schemes of Oettli and Eremin

The methods of Oettli [183] and of Eremin [84, 86] are algorithmic schemes for the solution of the convex feasibility problem with sets of the form (3.1).

Algorithm 3.2.5 Oettli's Scheme.

Initialization: $x^0 \in \Re^n$ is arbitrary.

Iterative Step:

$$x^{k+1} = x^k - \lambda_k \frac{\varphi(x^k)}{\| t^k \|^2} t^k, \quad \text{if } x^k \notin Q. \tag{3.9}$$

Here $Q = \cap_{i=1}^m Q_i$ and $\varphi : \Re^n \to \Re$ is defined by

$$\varphi(x) \doteq p(g_1^+(x), g_2^+(x), \ldots, g_m^+(x)),$$

where $g_i^+(x) \doteq \max\{0, g_i(x)\}$. In the definition of the auxiliary function φ, we are free to choose any *monotonic norm* p: $\Re^m \to \Re$, (see, for example, [184, page 52]. $t^k \in \partial\varphi(x^k)$ is any subgradient of φ at x^k and the relaxation parameters $\lambda_k \in (0, 2)$ are such that $\sum_{k=0}^{\infty} \lambda_k(2 - \lambda_k) = +\infty$.

Control: The control of the method is determined by the choice of the monotonic norm p.

Various choices of the monotonic norm p in Oettli's scheme give rise to different concrete algorithms. Choosing p to be the l_∞-norm in $\Re^m$ yields the remotest-set-controlled method of subgradient projections for solving

the convex feasibility problem. However, we believe (but are unable to verify) that there is no monotonic norm p that upon substitution into Oettli's scheme yields the **CSP** method, Algorithm 3.2.2.

The Scheme of Eremin resembles that of Oettli but is not quite identical to it.

Algorithm 3.2.6 Eremin's Scheme.

Initialization: $x^0 \in \Re^n$ is arbitrary.

Iterative Step:

$$x^{k+1} = \begin{cases} x^k, & \text{if } d(x^k) \leq 0, \\ x^k - \lambda_k \frac{d(x^k)}{\|e^k\|^2} e^k, & \text{otherwise,} \end{cases} \tag{3.10}$$

where $d : \Re^n \to \Re$ is a suitably chosen continuous convex function and $e^k = e(x^k)$ with $e : \Re^n \to \Re^n$ a suitably chosen mapping.

Relaxation Parameters: $\{\lambda_k\}_{k=0}^{\infty}$ are confined to $\epsilon_1 \leq \lambda_k \leq 2 - \epsilon_2$, for all $k \geq 0$, with some $\epsilon_1, \epsilon_2 > 0$.

Control: Determined by the choice of the functions d and e.

We do not go here into any further details and direct the reader to Eremin's papers for the precise conditions on the functions d and e involved in the scheme. Various specific methods may be derived from Eremin's algorithmic scheme through a proper choice of the functions d and e. In particular, the choice $d(x) \doteq \max_{1 \leq i \leq m} g_i(x)$ and $e^k \doteq \nabla g_{i(k)}(x^k)$ with

$i(k) \in \{i \mid g_i(x^k) = d(x^k)\}$, again gives rive to a remotest-set-controlled gradient projections method, and is, therefore, equivalent to Oettli's method with the l_∞-norm. Other choices of d and e, which are equivalent to choices of the l_2-norm and the weighted l_1-norm in Oettli's scheme, are given in [86, 87] To the best of our knowledge it is not possible to choose d and e such that the **CSP** method will be obtained, thus, the cyclic case has to be studied separately.

3.3 The Block-Iterative Projections (BIP) Algorithm

The **SOP** method (Algorithm 3.2.1) as well as its generalization in the (δ, η)—*Algorithm* (Algorithm 3.2.3) are sequential algorithms. A fully simultaneous iterative projections method is obtained by projecting the current iterate x^k separately and concurrently onto each and every set Q_i, $i = 1, 2, \cdots, m$, and then taking the next iterate x^{k+1} to be a convex combination of all the projections $P_{Q_i}(x^k)$. This idea was first suggested for the case of linear equations by Cimmino [55], see also in Gastinel's book [98].

The Block-Iterative Projections (**BIP**) method is an algorithmic scheme which encompasses both **SOP** and the fully simultaneous Cimmino idea. Its added novelty lies in the fact that it allows the processing of blocks (i.e., groups of sets Q_i) which need not be fixed in advance, but may rather change dynamically throughout iterations. The number of blocks, their sizes, and the assignment of sets Q_i to blocks may all vary, provided that the weights attached to the sets are not allowed to fade out, i.e., they have to fulfill a technical condition given in Theorem 3.3.1 below.

Let $I \doteq \{1, 2, \ldots, m\}$, and let $\{Q_i \mid i \in I\}$ be a finite family of closed convex sets. The intersection $Q = \cap\{Q_i \mid i \in I\}$ is again assumed to be nonempty. Denoting the nonnegative ray of the real numbers by $\Re_+$, a mapping $w : I \to \Re_+$ is called a *weight vector* if $\sum_{i \in I} w(i) = 1$.

A sequence $\{w^k \mid k = 0, 1, 2, \ldots\}$ of weight vectors is called *fair* if for every $i \in I$ there exist infinitely many values of k for which $w^k(i) > 0$. Given a weight vector w, we define the convex combination $P_w(x) \doteq \sum_{i \in I} w(i) P_i(x)$ where $P_i(x)$ is the orthogonal projection of x onto the set Q_i. The general scheme for block-iterative projections is as follows:

Algorithm 3.3.1 The BIP Algorithm.

Initialization: $x^0 \in \Re^n$ is arbitrary.

Iterative Step:

$$x^{k+1} = x^k + \lambda_k [P_{w^k}(x^k) - x^k], \tag{3.11}$$

where $\{w^k\}$ is a fair sequence of weight vectors and $\{\lambda_k\}$ is a sequence of user-determined *relaxation parameters.*

The special case where the weight vectors are given by $w^k = e^{i(k)}$, with $e^t \in \Re^n$ being the tth standard basis unit vector (having one in its tth coordinate and zeros elsewhere), gives rise to a *row-action* method (see Section 1.2, and Chapter 4, below). The **BIP** algorithm then coincides with the **SOP** method (Algorithm 3.2.1), and $\{i(k)\}$ is a *control sequence* of the algorithm. For example, a cyclic control sequence dictates $i(k) = k(\text{mod } m) + 1$.

At the other extreme, choosing any sequence of weight vectors $\{w^k\}$ with $w^k(i) \neq 0$, for all $k = 0, 1, 2, \ldots$, and all $i \in I$, leads to a fully simultaneous

Cimmino-type algorithm in which *all* sets $\{Q_i\}$ are being acted upon in every iterative step; see [8, 39, 133].

A block-iterative version with fixed blocks may be obtained by partitioning the indices of I as $I = I_1 \cup I_2 \cup \cdots \cup I_M$ into M blocks and using weight vectors of the form

$$w^k = \Sigma_{i \in I_{t(k)}} w^k(i) e^i,$$

where $\{t(k)\}$ is a control sequence over the set $\{1, 2, \ldots, M\}$ of block indices. In this case, if one considers the linear inequalities problem with

$$Q_i \doteq \{x \in \Re^n \mid \langle a^i, x \rangle \leq b_i\}, \tag{3.12}$$

for every $i \in I$, where $a^i \in \Re^n$, $b_i \in \Re$, then a block version of the relaxation method of Agmon, Motzkin, and Schoenberg (**AMS**) [3, 165, 153] is obtained. Proposed in [42] and referred to as "block-AMS" in [41] and as "block-Cimmino" in [42], this method has the following form.

Algorithm 3.3.2 The Block-AMS or Block-Cimmino Algorithm.

Initialization: $x^0 \in \Re^n$ is arbitrary.

Iterative step:

$$x^{k+1} = x^k + \lambda_k \left[\sum_{i \in I_{t(k)}} w^k(i) c_i(x^k) a^i \right], \tag{3.13}$$

where $\{t(k)\}$ is a cyclic or almost cyclic control sequence on $\{1, 2, \ldots, M\}$ and $c_i(x^k)$ is defined by

$$c_i(x^k) \doteq \min \left(0, \ \frac{b_i - \langle a^i, x^k \rangle}{\| a^i \|^2} \right). \tag{3.14}$$

The generality of the definition of a fair sequence of weight vectors permits also variable block sizes and/or variable block assignments to be used. This turns out to provide a necessary mathematical justification for some of the block-iterative algorithms used heuristically in the field of image reconstruction from projections, see [118, 117].

3.3.1 Convergence of the BIP Algorithm

The following notation will be used. For any $J \subseteq I$ and any weight vector w, define $w(J) \doteq \sum_{j \in J} w(j)$. For any $B \subseteq \Re^n$ denote by $J(B)$ the set of indices of those sets Q_i which do not meet B, i.e. $J(B) \doteq \{i \in I \mid B \cap Q_i = \emptyset\}$. For a singleton $B = \{x\}$ write $J(\{x\}) = J(x) = \{i \in I \mid x \notin Q_i\}$.

$B(x, \rho) \doteq \{y \in \Re^n \mid \| x - y \| \leq \rho\}$ is the ball with radius ρ centered at $x \in \Re^n$. Finally, define $x + \lambda[P_i(x) - x] \doteq P_{i,\lambda}(x)$ and $x + \lambda[P_w(x) - x] \doteq P_{w,\lambda}(x)$, where $\lambda \in \Re$ and w is a weight vector.

Proposition 3.3.1 *If $x \in \Re^n$, then for every $y \in Q_i$ and every $\lambda \in [\tau_1, 2 - \tau_2]$ with some fixed $\tau_1, \tau_2 > 0$,*

$$\| P_{i,\lambda}(x) - y \| \leq \| x - y \| . \tag{3.15}$$

Moreover, if $x \notin Q_i$, then the inequality is strict.

Proof. The following holds

$$\begin{aligned} \| P_{i,\lambda}(x) - y \|^2 &= \| x - y \|^2 + \lambda^2 \| P_i(x) - x \|^2 + 2\lambda \langle x - y, P_i(x) - x \rangle \\ &= \| x - y \|^2 + \lambda(\lambda - 2) \| P_i(x) - x \|^2 + 2\lambda \langle P_i(x) - x, P_i(x) - y \rangle \\ &\leq \| x - y \|^2 - \tau_1 \tau_2 \| P_i(x) - x \|^2 \leq \| x - y \|^2 . \end{aligned} \tag{3.16}$$

This follows from the fact that $x - P_i(x)$ supports Q_i at $P_i(x)$, which provides that $\langle P_i(x) - x, P_i(x) - y \rangle \leq 0$, for every $y \in Q_i$. See, e.g., [105, Lemma 1]. The last inequality of (3.16) is strict if $x \notin Q_i$. □

Proposition 3.3.2 *Let $q \in Q = \cap\{Q_i \mid i \in I\}$, and $\lambda \in [\tau_1, 2 - \tau_2]$, with $\tau_1, \tau_2 > 0$ be fixed, and let w be any weight vector. Then for every $x \in \Re^n$*

$$\| P_{w,\lambda}(x) - q \| \leq \| x - q \| . \tag{3.17}$$

Proof. Obviously, $P_{w,\lambda}(x) = \sum_{i \in I} w(i) P_{i,\lambda}(x)$, and repeated use of Proposition 3.3.1 with $y = q$ shows that $P_{i,\lambda}(x) \in B = B(q, \| x - q \|)$. The convex combination must, therefore, also be in B. □

Proposition 3.3.3 *Let $u \in \Re^n$, $J = J(u)$, $\lambda \in [\tau_1, 2 - \tau_2]$ with $\tau_1, \tau_2 > 0$ fixed, and let w be any weight vector. Then for every $r > 0$ there exists a real nonnegative γ such that if $\| x \| \leq r$ then*

$$\| P_{w,\lambda}(x) - u \| \leq \| x - u \| + \gamma w(J). \tag{3.18}$$

Proof. Define

$$\gamma_1 \doteq \max\{\| P_{j,\lambda}(x) - u \| \mid \| x \| \leq r, \ \ j \in J, \ \ \lambda \in [\tau_1, 2 - \tau_2]\}$$

and $\gamma \doteq \max\{\gamma_1, \ r + \| u \|\}$. Then, by Proposition 3.3.1, $\| P_{j,\lambda}(x) - u \| \leq \| x - u \|$ whenever $j \notin J$, implying that

$$\begin{aligned} \| P_{w,\lambda}(x) - u \| &= \| \textstyle\sum_{j \in J} w(j) [P_{j,\lambda}(x) - u] + \sum_{j \notin J} w(j) [P_{j,\lambda}(x) - u] \| \\ &\leq w(J)\gamma_1 + (1 - w(J)) \| x - u \| \\ &= \| x - u \| + w(J)(\gamma_1 - \| x - u \|) \\ &\leq \| x - u \| + \gamma w(J). \end{aligned}$$

□

Proposition 3.3.4 *Let $q \in Q$, and let $B \subseteq \Re^n$ be any compact set, $J \doteq J(B)$; Let $\lambda \in [\tau_1, 2-\tau_2]$ with $\tau_1, \tau_2 > 0$ fixed; and let w be any weight vector. Then there exists an $\alpha > 0$ such that, for every $x \in B$,*

$$\| P_{w,\lambda}(x) - q \| \leq \| x - q \| - \alpha w(J). \tag{3.19}$$

Proof. Define $\alpha \doteq \min\{(\| x - q \| - \| P_{j,\lambda}(x) - q \|) \mid x \in B,\ j \in J\}$. From Proposition 3.3.1 and the compactness of B, it follows that $\alpha > 0$. From the definition of α we have that $\| P_{j,\lambda}(x) - q \| \leq \| x - q \| - \alpha$, for every $x \in B$ and every $j \in J$. Using this and the fact that if $j \notin J$ then $\| P_{j,\lambda}(x) - q \| \leq \| x - q \|$, we get

$$\| P_{w,\lambda}(x) - q \| = \| \sum_{j \in J} w(j) [P_{j,\lambda}(x) - q] + \sum_{j \notin J} w(j) [P_{j,\lambda}(x) - q] \|$$

$$\leq w(J)(\| x - q \| - \alpha) + [1 - w(J)] \| x - q \| = \| x - q \| - \alpha w(J). \quad \square$$

To formulate the convergence theorem we use some additional notation. For a given fair sequence $\{w^k\}$ of weight vectors we write $F(\{w^k\}) \doteq \{i \in I \mid \sum_{k=0}^{\infty} w^k(i) = +\infty\}$ and define the set $\hat{Q} \doteq \cap\{Q_i \mid i \in F(\{w^k\})\}$ with the convention that if $F(\{w^k\}) = \emptyset$ then $\hat{Q} = \Re^n$.

Theorem 3.3.1 *If $Q \neq \emptyset$, if $\{w^k\}$ is any fair sequence of weight vectors, and if $\{\lambda_k\}$ is any sequence of relaxation parameters for which $\lambda_k \in [\tau_1, 2-\tau_2]$ for all $k = 0, 1, 2, \cdots$, where $\tau_1, \tau_2 > 0$, then any sequence $\{x^k\}$ generated by the* **BIP** *Algorithm (Algorithm 3.3.1) converges to a point $x^* \in \hat{Q}$.*

Proof. Any sequence $\{x^k\}$ generated by the **BIP** Algorithm is *Fejer-monotone* with respect to Q, i.e., for every $k = 0, 1, 2, \cdots$, $\| x^{k+1} - q \| \leq \| x^k - q \|$ for any $q \in Q$. This follows from Proposition 3.3.2 and, in

turn, implies that $\{x^k\}$ is bounded. Next we show that $\{x^k\}$ is convergent. Assuming the contrary, it has two or more distinct accumulation points. Let u be one of them, let v be the one closest to u (if there are several such v, pick any), and let $r \doteq \| u - v \|$. We first show that $u \in Q$. The sequence $\{\| x^k - q \|\}$ is monotonically decreasing and bounded from below. Since u is an accumulation point of $\{x^k\}$, it follows that $\| x^k - q \| \to \| u - q \|$ as $k \to \infty$, and that, for all $k = 0, 1, 2, \ldots,$

$$\| x^k - q \| \geq \| u - q \| . \tag{3.20}$$

Suppose that $u \notin Q$. Choose $\rho > 0$ such that $\rho < r/2$ and $B = B(u, \rho)$ satisfies $B \cap Q_j = \emptyset$ for every $j \in J(u)$. Let $J = J(B)$, and let γ and α be as in Propositions 3.3.3 and 3.3.4, respectively. Define

$$\epsilon \doteq \rho \frac{\alpha}{\gamma + \alpha},$$

and choose k such that $\| x^k - u \| < \epsilon$. Since v is also an accumulation point and $\rho < r/2$, there exists an $m > k$ such that $x^m \notin B$. Choose the first such m. Then, by Proposition 3.3.4,

$$\| x^m - q \| \leq \| x^k - q \| - \alpha \sum_{t=k}^{m-1} w^t(J) < \| u - q \| + \epsilon - \alpha \sum_{t=k}^{m-1} w^t(J). \tag{3.21}$$

Using (3.20) and (3.21), we get

$$\sum_{t=k}^{m-1} w^t(J) < \frac{\epsilon}{\alpha}. \tag{3.22}$$

On the other hand, by Proposition 3.3.3,

$$\| x^m - u \| \leq \| x^k - u \| + \gamma \sum_{t=k}^{m-1} w^t(J), \tag{3.23}$$

which, together with (3.22), yields

$$\| x^m - u \| < \epsilon + \frac{\gamma\epsilon}{\alpha} = \rho, \tag{3.24}$$

contradicting $x^m \notin B$, and hence showing that $u \in Q$. Using Fejer monotonicity with regard to u, which amounts to the monotonic decrease of $\{\| x^k - u \|\}$, the conclusion that $\{x^k\}$ converges to u follows. Note that $u \in Q$ has only been shown under the (false) assumption of several distinct accumulation points. Hence the limit x^* of any $\{x^k\}$ generated by the **BIP** Algorithm still needs to be characterized, and we show that it belongs to $\hat{Q}$. Assume that $x^* \notin Q_{i_0}$ for some $i_0 \in F(\{w^k\})$, i.e., $\sum_{k=0}^{\infty} w^k(i_0) = +\infty$. Choose a ball B_1 centered at x^* such that $B_1 \cap Q_{i_0} = \emptyset$. Let k be such that $x^m \in B_1$ whenever $m > k$. By Proposition 3.3.4 there exists $\alpha > 0$ for which

$$\| x^m - q \| \leq \| x^k - q \| - \alpha \sum_{t=k}^{m-1} w^t(J(B_1)), \tag{3.25}$$

for every $m > k$. But since $i_0 \in J(B_1)$ and $i_0 \in F(w^k)$, we have

$$\lim_{m \to \infty} \sum_{t=k}^{m-1} w^t(J(B_1)) = +\infty,$$

implying $\| x^m - q \| \to -\infty$, as $m \to \infty$, which is impossible. □

The condition $i \in F(w^k)$ is quite mild. In cases of practical importance, such as the row-action case or cases where fixed weights are attached to the constraint sets, the condition holds and actually yields $\hat{Q} = Q$.

3.4 Bregman's Sequential Algorithm

Bregman constructed in [31] a sequential algorithm which (asymptotically) solves the **CFP** by performing successive generalized projections onto the in-

dividual sets Q_i, starting from an arbitrary point. We present this algorithm now.

Let $f \in \mathcal{B}(\mathrm{S})$ be given as well as a family of closed convex sets $Q_i \subseteq \Re^n, i \in I$.

Algorithm 3.4.1 Successive Generalized Projections.

Initialization: $x^0 \in S$ is arbitrary.

Iterative Step:

$$x^{k+1} = P_{Q_{i(k)}}(x^k), \quad k = 0, 1, 2, \ldots \tag{3.26}$$

where $\{i(k)\}_{k\geq 0}$ is a *control sequence.*

For this algorithm to be well-defined we need the following assumption:
Assumption A2: $f \in \mathcal{B}(S)$ is zone consistent with respect to each Q_i, $i = 1, 2, \cdots, m$.

Theorem 3.4.1 *Let $f \in \mathcal{B}(S)$ be a given Bregman function with zone S. Let a* **CFP** *be given by a finite family of closed convex sets Q_i, $i \in I$, such that $Q \doteq \cap_{i\in I} Q_i$ and $Q \cap \overline{S} \neq \emptyset$. Let $i(k) \doteq k(\mathrm{mod}\ m) + 1$, $k \geq 0$, be a cyclic control sequence. Under these conditions, any sequence $\{x^k\}_{k\geq 0}$ generated by Algorithm 3.4.1, under Assumption A2, converges to a point $x^* \in Q$.*

Proof. The proof consists of three steps: (i) $\{x^k\}_{k\geq 0}$ is bounded, hence contains at least one accumulation point; (ii) every accumulation point of $\{x^k\}_{k\geq 0}$ belongs to Q; (iii) $\{x^k\}_{k\geq 0}$ has a unique limit point.

(i) For any $z \in Q \cap \overline{S}$ we have, according to Theorem 2.5.1 and Algorithm 3.4.1, under Assumption A2,

$$D_f(x^{k+1}, x^k) \leq D_f(z, x^k) - D_f(z, x^{k+1}), \tag{3.27}$$

which yields, using Lemma 2.2.1, the inequality,

$$D_f(z, x^{k+1}) \leq D_f(z, x^k), \quad \forall z \in Q \cap \overline{S}, \quad \forall k \geq 0. \tag{3.28}$$

This property is called D_f*−Fejer monotonicity of the sequence* $\{x^k\}_{k\geq 0}$ *with respect to* $Q \cap \overline{S}$. This implies that $\{x^k\}_{k\geq 0}$ is bounded because

$$x^k \in L_2(z, \alpha), \quad \forall k \geq 0, \tag{3.29}$$

with $\alpha \doteq D_f(z, x^0)$ and assumption (iv) of Definition 2.2.1 applies.

(ii) Let $\{x^k\}_{k\in K}$, with $K \subseteq N_0 \doteq \{0, 1, 2, 3, \ldots\}$ be a subsequence converging to x^*. Let $\{x^k\}_{k\in M}, M \subseteq K$, be a subsequence all whose elements belong to one set, say Q_1. From the sequences $\{x^{k+i-1}\}_{k\in M}$, $i = 2, 3, \ldots, m$, one can extract convergent subsequences, so we assume, without loss of generality, that

$$\lim_{k\to\infty,\, k\in M} x^{k+i-1} = x^{*i}, \quad i = 1, 2, \ldots, m, \tag{3.30}$$

and $x^{*1} = x^*$. Since $\{x^{k+i-1}\}_{k\in M} \subseteq Q_i$, it follows that

$$x^{*i} \in Q_i, \quad \forall i \in I. \tag{3.31}$$

By (3.28) and the nonnegativity of D_f, $\lim_{k\to\infty} D_f(z, x^k)$ exists for any $z \in Q \cap \overline{S}$. In view of (3.27), $\lim_{k\to\infty} D_f(x^{k+1}, x^k) = 0$ thus also

$$\lim_{k\to\infty,\, k\in M} D_f(x^{k+1}, x^k) = 0. \tag{3.32}$$

Assumption (vi) of Definition 2.2.1 can be repeatedly used to show that, from (3.30) and (3.31),

$$x^* = x^{*1} = x^{*2} = \cdots = x^{*m}, \tag{3.33}$$

which, by (3.31), shows that $x^* \in Q$.

(iii) Let

$$\lim_{k\to\infty,\, k\in K_1} x^k = x^* \in Q \cap \overline{S}, \tag{3.34}$$

$$\lim_{k\to\infty,\, k\in K_2} x^k = x^{**} \in Q \cap \overline{S}, \tag{3.35}$$

for some $K_1 \subseteq N_0$ and $K_2 \subseteq N_0$, be two convergent subsequences of $\{x^k\}_{k\in N_0}$ generated by Algorithm 3.4.1. Define

$$h(x^k) \doteq D_f(x^*, x^k) - D_f(x^{**}, x^k), \tag{3.36}$$

and recall an argument from step (ii) of this proof which guarantees that

$$\lim_{k\to\infty} h(x^k) = \overline{h} \tag{3.37}$$

exists. Therefore, repeating this limit once over K_1 and then over K_2, and using assumption (v) of Definition 2.2.1, we obtain

$$\overline{h} = -D_f(x^{**}, x^*) \leq 0, \tag{3.38}$$

$$\overline{h} = D_f(x^{**}, x^*) \geq 0, \tag{3.39}$$

yielding

$$D_f(x^*, x^{**}) = D_f(x^{**}, x^*) = 0, \tag{3.40}$$

which implies $x^* = x^{**}$. □

Chapter 4

Row-Action Algorithms for Linearly Constrained Optimization Problems

4.1 Introduction

The main feature of row-action methods is that they are iterative procedures which, without making any changes to the original constraints matrix A, use the rows of A, one row at a time. Such methods are important and have demonstrated effectiveness for problems with large or huge matrices which do not enjoy any detectable or usable structural pattern, apart from a high degree of sparseness.

In this chapter, we present *row-action methods* for handling huge and sparse systems (i.e., whose number of unknowns and number of equations or inequalities, each fall in the range of hundreds of thousands), linear or

nonlinear, of equalities or inequalities. These methods, scattered in the literature, differ among themselves in various aspects. Some of them are old, some are new, and they were used in different and unrelated fields of application resulting in duplication of efforts and repeated discoveries.

In defining and bringing row-action methods together in a coherent presentation we have two aims in mind. First, to facilitate cross-fertilization between various fields of application where row-action methods are, or might be, used, and second, to pave the way for a better insight into their mathematical nature.

Several row-action methods for linearly constrained optimization can be treated in a unified manner within a framework which employs generalized Bregman distances and generalized projections. We present this uniform framework here but first discuss some general matters related to row-action methods.

4.2 The Problem, Solution Concepts and the Special Environment

4.2.1 The Problem

In many fields of application and in different scientific disciplines, including some of those described in later chapters of these Notes, the modeling of the physical or mathematical problem leads to a system of linear equations.

$$\langle a^i, x \rangle = b_i, \qquad i \in I, \tag{4.1}$$

where $I = \{1, 2, \ldots, m\}$, $a^i \in \Re^n$, $a^i \neq 0$, $x \in \Re^n$, $b_i \in \Re$. We also write the system (4.1) as $Ax = b$ where A denotes the $m \times n$ matrix whose ith

row is $(a^i)^T$ and T stands for transposition.

This system is sometimes underdetermined due to lack of information, often it is greatly overdetermined in which case the chances that it is self-contradictory (inconsistent) are high. Facing reality, we have to be ready to deal with severely ill-conditioned systems. Even if none of these unpleasant situations occur, we might have reason to believe that the exact algebraic solution of the system is less desirable, in terms of the original problem for which the system was set up, then some other, differently defined "solution".

Such a belief may be nourished by evidence about measurements inaccuracy, noise corruption of data, discretization in the model, etc., and therefore, in such cases, as well as in the situations described above, it is useful to turn to a different solution concept rather than aim at the exact algebraic solution of the system.

4.2.2 Approaches and Solution Concepts.

The underlying idea is to try to use the information contained in the system (4.1) in a way which will reflect our limited belief in the equations, and there are indeed several different ways of doing so. Each of the approaches described below leads to a variety of solution concepts and the decision which one to choose in any particular application rests with the user and should usually be made with reference to the specific problem at hand.

(i) **The feasibility approach:** Here one seeks at point x that lies within a specific vicinity of all hyperplanes defined by the equations (4.1). This is done by prescribing some tolerances α_i, for all $i \in I$, and aiming at

any solution $x \in \Re^n$ which lies in the intersection of all hyperslabs

$$b_i - \alpha_i \leq \langle a^i, x \rangle \leq b_i + \alpha_i, \quad i \in I. \tag{4.2}$$

Tolerances α_i have to be set in such a manner that the feasible region is not empty but also not too large. Inequalities of the form (4.2) are called interval inequalities and the feasibility problem will be called an *interval feasibility problem.* If we deal with a solution method that does not take advantage of the fact that the inequalities come in pairs we regard the system (4.2) simply as a system of one sided linear inequalities with twice as many inequalities, obtained by multiplying one of the two sets of inequalities by (-1).

(ii) The optimization approach: A predesignated objective function $f(x)$, $f : \Re^n \to \Re$, according to which a particular element will be sorted out from the feasible region (4.2), as the solution of the original real-world system, must be chosen. If the resulting optimization problem:

$$\text{minimize } f(x), \quad \text{subject to } b_i - \alpha_i \leq \langle a^i, x \rangle \leq b_i + \alpha_i, \quad i \in I, \tag{4.3}$$

does not have a unique solution, a secondary optimization criterion is sometimes used, i.e., another objective function $g(x)$ is optimized over the set of solutions of (4.3).

The problem (4.3) is an *interval programming problem;* and one would prefer a solution method that benefits from the extra slab-structure of the constraints over a one sided linearly constrained optimization technique. In both the feasibility approach and the optimization approach additional information about the real-world problem may take

the form of additional inequalities that will further restrict the feasible set. A common case is that of box constraints

$$w_i \leq x_i \leq v_i, \qquad i = 1, 2, \ldots, n,$$

which may reflect some a-priori information about the desired solution. Although these inequalities can be regarded as interval inequalities (with $x_i = \langle e^i, x \rangle$, where e^i is the ith standard basis vector), a good solution method should take care of them in a simpler way.

(iii) The regularization approach: Here the problem:

$$\text{"solve"} \quad Ax = b \quad \text{subject to} \quad x \in Q, \tag{4.4}$$

where $Ax = b$ is the system (4.1) and Q represents additional constraints, is treated by replacing it with

$$\text{Minimize} \quad [f(x) + rg(Ax - b)] \quad \text{subject to} \quad x \in Q, \tag{4.5}$$

where $f : \Re^n \to \Re$ and $g : \Re^m \to \Re$, are some suitably chosen convex functions and r is a user's specified parameter reflecting the relative importance attached to each summand. Least squares regularization, where $f(x)$ and $g(x)$ are both of the form $\| x \|^2$ is by far the most common approach. Application of row-action methods to (4.5) in this case was suggested by Herman, Hurwitz and Lent [112]. Other choices of f and g are possible, see, e.g., Kovarik [143]. Frieden [95, 96] used an entropy regularization approach where both $f(x)$ and $g(x)$ are of the form $\sum_j x_j \ln x_j$.

4.2.3 The Special Environment.

An indispensable part of the mathematical problem involved in any of the approaches described above is the environment within which the problem is addressed. In some significant application fields the environment is distinguished by a combination of some or all of the following properties.

(i) Dimensionality: The system (4.1) is *huge*, e.g., $n \geq 10^5$ and m even greater, see, e.g., Held et al. [109] and Herman and Lent [114].

(ii) Sparseness: The matrix A of the system (4.1) is sparse. For example, in image reconstruction from projections, see, e.g., Herman and Lent [114], less than 1% of the entries are nonzero.

(iii) Lack of structure: One fails to recognize any structure in the distribution of nonzero entries throughout A. Alternatively, one might sometimes be able to detect a special structure but be unable to take advantage of it.

(iv) Time restriction: It is not only that computation time has a price, but in some applications (image reconstruction from projections for medical diagnosis is an example, again) there might be an inherent time restriction, i.e., a solution which is acceptable in terms of the original real-world problem (say, of radiological usefulness) is demanded within minutes of data collection.

(v) Computation power restriction: Sometimes solution methods are explicitly required to perform efficiently on a machine with low memory or other specifications.

A prime example is the practical problem of computerized tomography, see Chapter 6 below, where the combination of all five properties usually describes the environment for the problem in any of the mentioned approaches. Situations where some of these environmental properties hold, appear in various other fields, some of which are discussed in Chapter 6.

In such an environment the use of general purpose techniques might be impractical and solution techniques have to be carefully chosen so as to be able to cope with the environmental situation in a computationally feasible way.

4.3 Row-Action Methods, Controls and Relaxation Parameters

Definition 4.3.1 (Row-Action Method.) *A row-action method (***RA***-method, for short) is an iterative algorithm which has the following properties:*

1. *no changes are made to the original matrix;*
2. *no operations are performed on the matrix as a whole;*
3. *in a single iterative step access is required to only one row of the matrix;*
4. *in a single iterative step, say when x^{k+1} is calculated, the only iterate needed is the immediate predecessor x^k.*

Several remarks concerning this method are in order.

(a) We consider the concept of **RA**-method as a general framework within which various concrete algorithms may or may not fall. As will be

seen below, the framework is broad enough to include various kinds of algorithms differing in nature and structure, as well as in the problems which they are designed to solve. Moreover, there are algorithms which are not formulated as **RA**-methods at all but lend themselves to a row-action implementation. One such example, which is discussed below, is Hildreth's algorithm, Section 4.6.2. Note that Richardson's method (not considered here, see, e.g., Young [231] also has a row-action implementation.

(b) In what follows we will be particularly interested, for obvious reasons, in **RA**-methods which are further characterized by the fact that the *algorithms present modest arithmetical demands.* While this requirement is not precisely defined and might well be considered as a universal goal, we stress it here to make the point that in the special environment described above (particularly dimensionality), performing operations which are just a little more complicated might quickly reduce the practicality of a method. Some of the methods described in the sequel comply to this demand less than others; e.g., Hildreth's algorithm requires the storage and calculation of a sequence of dual vectors and Bregman's algorithm requires the execution of a generalized projection at each iterative step.

(c) In [45] we used the term "row-generation methods", but row-action methods seems more appropriate and helps to make clear the distinction between the methods and *row-generation capability.* The latter refers to the situation where it is possible to avoid storing the matrix of system (4.1) explicitly, and instead to have the nonzero entries of

the ith row, together with their locations, generated from the experimental data each time, anew. Property (iii) in Definition 4.3.1 makes **RA**-methods favorable in such cases. In the absence of row-generation capability one would, of course, consult any of the efficient methods for storing a sparse matrix (see, e.g., Duff [78]).

The combination of the four restrictions in Definition 4.3.1 together with the additional condition of modest arithmetical demand is enough to make **RA**-methods preferable choices when any of the described approaches is taken within the special environmental of Section 4.2.3. Next, we introduce some notions common to **RA**-methods.

Definition 4.3.2 (Control Sequence.) *A sequence of indices* $\{i(k)\}_{k=0}^{\infty}$ *according to which the rows of the matrix* A *are taken up, is called a control sequence of an* **RA***-method, i.e., in the iterative step* $k \to k+1$ *the* $i(k)$*th row is used.*

The most important controls were already defined in Section 3.1. These are the cyclic, the almost cyclic, the repetitive and the remotest set controls. Here we give some additional controls.

In **RA**-methods, constraint sets Q_i are often associated with the equations of the system (4.1). Usually the sets Q_i are hyperplanes $H_i = \{x \in \Re^n \mid \langle a^i, x\rangle = b_i\}$ or halfspaces $L_i = \{x \in R^n \mid \langle a^i, x\rangle \leq b_i\}$ or hyperslabs, which are intersections of pairs of parallel halfspaces.

Approximately remotest set control. Denote $\max_{i\in I} d(x^k, Q_i) \doteq \theta(x^k)$, and choose the $i(k)$'s to satisfy the condition

$$\lim_{k\to\infty} d(x^k, Q_{i(k)}) = 0 \quad \Rightarrow \quad \lim_{k\to\infty} \theta(x^k) = 0 \tag{4.6}$$

Obviously, every remotest set control is an approximately remotest set control, but it is also true that every cyclic control is an approximately remotest set control (see Gubin et al. [105, Lemma 4]), and these facts make the approximately remotest set control an important theoretical tool.

Most violated constraint control. This control, closely related to (3.2), is obtained by determining which constraint is most violated by the iterate x^k, e.g., if $Q_i = \{x \in \Re^n \mid f_i(x) \leq 0\}$, $i \in I$, are the constraint sets under consideration, then one has to determine at each step

$$f_{i(k)}(x^k) = \max_{i \in I} f_i(x^k), \tag{4.7}$$

and take $i(k)$ as the control index if $f_{i(k)}(x^k) > 0$.

Threshold control. (McCormick [156]). Let $\{\tau_j\}_{j=0}^{\infty}$ be a given sequence of positive real numbers converging to zero, and suppose that x^k, $i(k-1)$ and $j(k-1)$ have been determined. With $j(k) = j(k-1)$, $i(k)$ is chosen as the first member of the sequence

$$i(k-1)+1,\ i(k-1)+2,\ \cdots,\ m-1,\ m,\ 1,\ 2,\ \cdots,\ i(k),$$

that yields

$$|\ \langle a^{i(k)}, x^k \rangle - b_{i(k)} \rangle\ | \geq \tau_{j(k)}\ \|\ a^{i(k)}\ \|, \tag{4.8}$$

and, if this condition cannot be met, the process is repeated with $j(k)$ incremented by one.

Intuition tells us that controls like the remotest set control or the threshold control are more sophisticated than the cyclic control and therefore might

lead to a faster convergence of the **RA**-method to which they are applied. On the other hand, one must bear in mind that in the special environment we are in, any of the controls other than a cyclic or an almost cyclic control may violate the additional requirement of arithmetical simplicity, and thus make the **RA**- method impractical. For this reason we emphasize in the sequel only those two controls, although some **RA**-methods can accommodate other controls too.

4.4 The Method of Bregman

Consider the problem

$$\begin{cases} \min\ f(x), \\ \text{subject to } \langle a^i, x\rangle \le b_i, \qquad i \in I \doteq \{1, 2, \ldots, m\}, \\ x \in \overline{S}. \end{cases} \tag{4.9}$$

Let $H_i \doteq \{x \mid \langle a^i, x\rangle = b_i\}$, $Q_i \doteq \{x \mid \langle a^i, x\rangle \le b_i\}$, denote $Q \doteq \cap_{i=1}^m Q_i$, and assume that $Q \cap \overline{S} \neq \emptyset$. $A = (a_{ij})$ is the $m \times n$ matrix whose ith row is $(a^i)^T$. Assume that all $a^i \neq 0$. Assume that $f \in \mathcal{B}(\mathrm{S})$, see Definition 2.2.1, and that f is strongly zone consistent with respect to every H_i. Define the following sets:

$$Z \doteq \{x \in S \mid \exists\ z \in \Re^m, \text{ such that } \nabla f(x) = -A^T z\}, \tag{4.10}$$

$$Z_0 \doteq \{x \in S \mid \exists\ z \in \Re_+^m, \text{ such that } \nabla f(x) = -A^T z\}, \tag{4.11}$$

which are assumed nonempty in the sequel.

Bregman's iterative method takes at each iteration a hyperplane H_i, projects onto it the current iterate, and decides accordingly what the next

iterate will be. The sequence of indices that governs in this way the application of the method to a problem is the *control sequence* of the method. Bregman deals only with the cyclic case, whereas here the almost cyclic control is established. This extension (almost cyclic control is less restrictive than cyclic) is absolutely necessary for the development of the interval programming method in the next section.

Algorithm 4.4.1 Bregman's Scheme with Almost Cyclic Control.

Initialization: Assume

$$\begin{cases} x^0 \in Z_0 \text{ is arbitrary, and} \\ z^0 \text{ is such that } \nabla f(x^0) = -A^T z^0. \end{cases} \tag{4.12}$$

Iterative Step:

$$\begin{cases} \nabla f(x^{k+1}) &= \nabla f(x^k) + c_k a^{i(k)}, \\ z^{k+1} &= z^k - c_k e^{i(k)}, \\ c_k &\doteq min(z^k_{i(k)}, B_k), \end{cases} \tag{4.13}$$

where B_k is the parameter associated with projecting x^k onto $H_{i(k)}$. We assume throughout that the representation of every hyperplane is fixed during the whole iteration process, so that the values of B_k are well defined.

Control: The sequence $\{i(k)\}_{k=0}^{\infty}$ is almost cyclic on I.

The next lemma gives insight into the behavior of Algorithm 4.4.1 and justifies the geometric interpretation given in Remark 4.4.1.

Lemma 4.4.1 *For $k = 0, 1, 2, \ldots$, the iterates produced by Algorithm 4.4.1 are all $x^{k+1} \in Q_{i(k)}$. Moreover if $x^k \notin intQ_{i(k)}$, then $x^{k+1} \in H_{i(k)}$.*

Proof. Let x and y be any two points in S. Then, the following identity is easily derived:

$$D(x, y) + D(y, x) = \langle \nabla f(x) - \nabla f(y), x - y \rangle. \tag{4.14}$$

Let $\hat{x}^{k+1}$ be the projection of x^k onto $H_{i(k)}$, and assume that (4.13) is solvable for x^{k+1}. Then, replacing x by x^{k+1} and replacing y by $\hat{x}^{k+1}$ in (4.14), we get, by Lemma 2.2.1,

$$(c_k - B_k)\langle a^{i(k)}, x^{k+1} - \hat{x}^{k+1} \rangle \geq 0. \tag{4.15}$$

From (4.13),

$$c_k \leq B_k,$$

and, by definition,

$$\langle a^{i(k)}, \hat{x}^{k+1} \rangle = b_{i(k)};$$

therefore,

$$\langle a^{i(k)}, x^{k+1} \rangle \leq b_{i(k)} \tag{4.16}$$

follows and proves the first part of the lemma.

To prove the second part, we look ahead to (4.17), where it is shown that $z^k \geq 0$, for all k. We are assuming that $x^k \notin intQ_{i(k)}$, so $b_{i(k)} - \langle a^{i(k)}, x^k \rangle \leq 0$, which, from (2.28)–(2.29), implies that $B_k \leq 0$.

But then, from (4.13), $c_k = B_k$, and x^{k+1} is the generalized projection of x^k onto $H_{i(k)}$. □

Remark 4.4.1 For the special case $f(x) = \frac{1}{2} \| x \|^2$ (Example 2.2.1), and with the cyclic control, this is a quadratic programming method, see [121, 73,

150]. In the quadratic programming case, the iterative step (4.13) assumes a very simple form; but, in the general case, the iterative step (4.13), which is a set of n nonlinear equations, has to be solved at each step, and it is the effectiveness of the method chosen to solve this inner-loop subproblem that determines the overall performance of the method.

There are three possible cases that may occur in a typical step when the $i(k)$th constraint is taken up. If $x^k \in H_{i(k)}$, then $x^{k+1} = x^k$; if $x^k \notin Q_{i(k)}$, then x^{k+1} is the projection of x^k onto $H_{i(k)}$ or the projection of x^k onto another hyperplane $\hat{H}_{i(k)}$, say, which is parallel to $H_{i(k)}$ and lies entirely within $Q_{i(k)}$. All of these possibilities are covered by (4.13).

$\triangle$ respectively.

We now present a convergence theorem for Algorithm 4.4.1 as applied to the standard problem (4.9). Bregman's original method of proof, from [31], is followed closely, validating the almost cyclic control.

Theorem 4.4.1 *Assume the following:*

(i) $f \in \mathcal{B}(S)$;

(ii) f *is strongly zone consistent with respect to each* H_i, $i \in I$, *of (4.9); Algorithm 4.4.1.*

(iii) $\{i(k)\}_{k=0}^{\infty}$ *is almost cyclic on* I *with constant* C;

(iv) $Z_0 \neq \emptyset$.

Then, any sequence $\{x^k\}$ *produced by Algorithm 4.4.1 converges to the point* x^* *which is the solution of (4.9).*

Proof. The proof consists of the following steps.

Step 1. $x^k \in Z_0$, for all $k \geq 0$.

Step 2. Define the Lagrangian $L(x, z)$ of (4.9), and show that $\lim L(x^k, z^k)$, as $k \to \infty$, exists.

Step 3. $D_f(x^{k+1}, x^k) \to 0$, as $k \to \infty$.

Step 4. $\{x^k\}$ is bounded.

Step 5. Any accumulation point of $\{x^k\}$ is feasible for (4.9).

Step 6. x^* is the solution of (4.9).

Step 1. We show that

$$x^k \in Z_0, \quad \text{for all } k \geq 0. \tag{4.17}$$

We proceed by induction. Observe that $x^0 \in Z_0$, by (4.12). Assume that $x^l \in Z_0$, for all $l \leq k$; let $i(k)$ be the next index, and abbreviate $i(k) \equiv i$.

Then, from (4.13), together with $A^T e^i = a^i$, we get

$$\nabla f(x^{k+1}) = -A^T z^k + c_k a^i = -A^T(z^k - c_k e^i) = -A^T z^{k+1}, \tag{4.18}$$

showing that $x^{k+1} \in Z$. But $c_k \leq z_i^k$, by (4.13); therefore, $z^k \geq 0$ implies that $z^{k+1} \geq 0$, concluding the proof of (4.17). □

Step 2. Define the Lagrangian of (4.9) as

$$L(x, z) \doteq f(x) + \langle z, Ax - b \rangle. \tag{4.19}$$

First, the values of the Lagrangian at (x^k, z^k) form an increasing sequence, i.e.,

$$d_k \doteq L(x^{k+1}, z^{k+1}) - L(x^k, z^k) \geq 0, \quad \text{for all } k. \tag{4.20}$$

This can be shown as follows. From (4.19), (4.20), (2.1), and from

$$\langle z^k, Ax^k \rangle = \langle A^T z^k, x^k \rangle = -\langle \nabla f(x^k), x^k \rangle, \tag{4.21}$$

which holds for all k because of (4.17), we may write

$$d_k = D_f(x^{k+1}, x^k) - \langle z^{k+1} - z^k, b\rangle + \langle \nabla f(x^k) - \nabla f(x^{k+1}), x^{k+1}\rangle \quad (4.22)$$

which yields, by (4.13),

$$d_k = D_f(x^{k+1}, x^k) + c_k[b_i - \langle a^i, x^{k+1}\rangle]. \quad (4.23)$$

The conclusion that $d_k \geq 0$, for all k, now follows; for, from Lemma 2.2.1, $D_f(x^{k+1}, x^k) \geq 0$, while for the second summand we know from Lemma 4.4.1 that $x^{k+1} \in Q_i$, $i \equiv i(k)$, so that $b_i - \langle a^i, x^{k+1}\rangle \geq 0$.

By (4.17), we know that $z_i^k \geq 0$, so that the only way c_k could be negative is if it were equal to B_k. But, from (2.28)–(2.29), we see that in this case $x^k \notin Q_i$; hence (see Lemma 4.4.1), $x^{k+1} \in H_i$, implying that $b_i - \langle a^i, x^{k+1}\rangle = 0$, thus proving that, in any case, the second summand in (4.23) is also nonnegative.

Going further, we prove existence of the limit

$$\lim_{k\to\infty} L(x^k, z^k) = F, \quad (4.24)$$

say. In view of (4.20), we have only to show that

$$\{L(x^k, z^k)\} \text{ is bounded from above for all } k. \quad (4.25)$$

To see this, take any $z \in Q \cap \overline{S}$, and verify that

$$\langle A^T z^k, z - x^k\rangle = \langle z^k, Az\rangle - \langle z^k, Ax^k\rangle \leq \langle z^k, b - Ax^k\rangle, \quad (4.26)$$

because

$$Az \leq b. \quad (4.27)$$

From (2.1), (4.17), (4.26), we get

$$D_f(z, x^k) \leq f(z) - f(x^k) + \langle z^k, b - Ax^k \rangle = f(z) - L(x^k, z^k), \qquad (4.28)$$

from which we infer that, for any $z \in Q \cap \overline{S}$,

$$L(x^k, z^k) \leq f(x) - D_f(z, x^k) \leq f(z), \qquad (4.29)$$

because of Lemma 2.2.1, thereby proving (4.25).

Step 3. The convergence to zero of the differences of consecutive iterates, often a cornerstone in the proof of convergence of iterative methods, can now be deduced without difficulty. Here the difference is in the sense of the D_f-function associated with $f(x)$ according to (2.1), i.e.,

$$D_f(x^{k+1}, x^k) \to 0, \qquad \text{as } k \to \infty. \qquad (4.30)$$

From (4.23), we take $0 \leq D_f(x^{k+1}, x^k) \leq d_k$ and use (4.24) to see that $d_k \to 0$, as $k \to \infty$, thus validating (4.30). □

Step 4. Now, we show that

$$\text{the sequence } \{x^k\} \text{ is bounded.} \qquad (4.31)$$

From (4.20) and (4.28), we see that $D_f(z, x^k) \leq f(z) - L(x^0, z^0) \doteq \alpha =$ constant; therefore, from the boundedness of the partial level set $L_2(z, \alpha)$ [see Definition 2.2.1(iv)], statement (4.31) follows. □

Step 5. We prove the feasibility of any accumulation point x^* of $\{x^k\}$. Let $x^{k_j} \to x^*$, as $j \to \infty$. By multiple application of Definition 2.2.1(vi), together with (4.30) and (4.31), it follows that

$$x^{k_j+t} \to x^*, \text{as } j \to \infty, \quad \text{for every } t \in \{0, 1, 2, \ldots, C\} \qquad (4.32)$$

where C is the constant of almost cyclicality.

Consider the semi-infinite array which has $C+1$ rows, with $\{x^{k_j+t}\}_{j=1}^{\infty}$ in the tth row. We show that, for each $i \in I$, some row of the array contains an infinite number of elements belonging to Q_i.

For each $i \in I$, at least one element in each column of the array must belong to Q_i, otherwise almost cyclicality is violated. It follows that there must be some row in the array, infinitely many elements of which belong to Q_i, because if all rows had only a finite number of elements belonging to Q_i, there would be only a finite number of elements belonging to Q_i in the entire array. Thus, one can extract from the rows of the array subsequences all of which converge to the same x^*, which belong to each of the Q_i's. Therefore, $x^* \in Q \cap \overline{S}$, where $Q = \bigcap_{i=1}^{m} Q_i$, because Q_i is closed for every $i \in I$ and because of the strong zone consistency assumption. □

Step 6. Here, we show that, if x^* is a cluster point of $\{x^k\}$, then it is a solution of (4.9). We then take advantage of the strict convexity of $f(x)$ to conclude the proof.

Denote by I_1 and I_2 the sets of indices of inactive and active constraints at x^*, respectively,

$$\begin{aligned} I_1 &\doteq \{i \in I \mid \langle a^i, x^* \rangle < b_i\}, \\ I_2 &\doteq \{i \in I \mid \langle a^i, x^* \rangle = b_i\}. \end{aligned} \tag{4.33}$$

Our proof of Step 6 needs the following proposition.

Proposition 4.4.1 *If $x^* \in Q \cap \overline{S}$ is a cluster point of $\{x^k\}$ produced by Algorithm 4.4.1, then there exist a subsequence $\{x^{k_j}\}_{j=0}^{\infty}$ which converges to x^* and has the property that $z_i^{k_j} = 0$, for all $i \in I_1$ and all j.*

The proof of this proposition is constructed by an elaboration of the argument used in Step 5, and is given in full detail in the Appendix of [39].

We now return to Step 6. Using the subsequence of Proposition 4.4.1, we may write

$$\langle z^{k_j}, Ax^{k_j} - b\rangle = \langle A^T z^{k_j}, x^{k_j} - x^*\rangle, \tag{4.34}$$

because $z_i^{k_j} = 0$, for $i \in I_1$, while $\langle a^i, x^*\rangle = b_i$, for $i \in I_2$, and $I = I_1 \cup I_2$.

From (4.17) we have that $A^T z^{k_j} = -\nabla f(x^{k_j})$, so that

$$\langle z^{k_j}, Ax^{k_j} - b\rangle = -\langle \nabla f(x^{k_j}), x^{k_j} - x^*\rangle = D_f(x^*, x^{k_j}) - f(x^*) + f(x^{k_j}), \tag{4.35}$$

which tends to zero as $j \to \infty$, because of Definition 2.2.1(iii) and (iv).

Therefore, using again the continuity of f on $\overline{S}$, we have

$$\lim L(x^{k_j}, z^{k_j}) = \lim[f(x^{k_j}) + \langle z^{k_j}, Ax^{k_j} - b\rangle] = f(x^*), \quad \text{as } j \to \infty. \tag{4.36}$$

From (4.24) and (4.29), it now follows that

$$f(x^*) \leq f(z), \qquad \text{for all } z \in Q \cap \overline{S}, \tag{4.37}$$

which proves that x^* is a solution for (4.9). Because of the strict convexity of f, imposed by Definition 2.2.1(ii), x^* must be unique, which proves that $x^k \to x^*$, as $k \to \infty$, with x^* being the solution of (4.9). □

Remark 4.4.2 Strong zone consistency is required in Theorem 4.4.1 to ensure that the iterates stay within S regardless of the value that c_k of (4.13) takes.

4.5 The Interval-Constrained Problem

Consider the convex programming problem with interval constraints

$$\begin{cases} \min f(y), \\ \text{subject to } \gamma_i \leq \langle \phi^j, y\rangle \leq \delta_j, \quad j \in J \doteq \{1, 2, \ldots, p\}, \\ y \in \overline{S}. \end{cases} \tag{4.38}$$

This problem could easily be transformed to the form of (4.9), to which Bregman's Algorithm 4.4.1 may be applied directly. But doing so would result in a problem with $2p$ constraints, forcing us to deal with that number of dual variables. This doubling of the number of dual variables makes a considerable difference when dealing with a large or huge number of inequalities. For example, in the area of image reconstruction from projections, where an approach leading to a problem of the form (4.38) is taken [114], the number of interval constraints (pairs of inequalities) can be as large as 300,000.

The method for interval convex programming (i) attaches only one dual variable to each interval constraint, and (ii) requires only half as many iterative steps as would be needed if Algorithm 4.4.1 were applied directly to the transformed problem.

This interval convex programming method is obtained by choosing a particular strategy for the application of Algorithm 4.4.1. The almost cyclic control, introduced into Algorithm 4.4.1, is an indispensable tool for validating the process.

Before stating the algorithm, we set up the following definitions and notations,

$$H(\phi^j, \beta_j) \doteq \{y \in \Re^n \mid \langle \phi^j, y \rangle = \beta_j\} \tag{4.39}$$

denotes a hyperplane whose representation is *fixed* and determined by ϕ^j and β_j. The constraints of (4.38) will be described by the half spaces

$$Q_{j^+} \doteq \{y \mid \langle \phi^j, y \rangle \leq \delta_j\}, \tag{4.40}$$

$$Q_{j^-} \doteq \{y \mid \gamma_j \leq \langle \phi^j, y \rangle\}, \tag{4.41}$$

for all $j \in J$, so that $Q_j = Q_{j^+} \cap Q_{j^-}$.

Again denote $Q = \bigcap_{j=1}^{p} Q_j$, and assume that $\phi^j \neq 0$, for all $j \in J$.

Define

$$U \doteq \{y \in S \mid \exists u \in \Re^p, \text{ such that } \nabla f(y) = -\Phi^T u\}, \tag{4.42}$$

where Φ stands for the $p \times n$ matrix whose jth row is $(\phi^j)^T$, and assume $U \neq \emptyset$ throughout.

Algorithm 4.5.1 Method for Interval Convex Programming.

Initialization: Assume that

$$\begin{cases} y^0 \in U \text{ is arbitrary, and} \\ u^0 \text{ is such that } \nabla f(y^0) = -\Phi^T u^0. \end{cases} \tag{4.43}$$

Iterative Step:

$$\begin{cases} \nabla f(y^{k+1}) = \nabla f(y^k) + d_k \phi^{j(k)}, \\ u^{k+1} = u^k - d_k e^{j(k)}, \\ d_k \doteq \text{mid}\{u^k_{j(k)}, \Delta_k, \Gamma_k\}, \end{cases} \tag{4.44}$$

where Δ_k and Γ_k are the parameters associated with the generalized projection of y^k onto $H(\phi^{j(k)}, \delta_{j(k)})$ and $H(\phi^{j(k)}, \gamma_{j(k)})$, respectively. $\text{mid}(a, b, c)$ denotes the median of the three real numbers a, b, c.

Control: The sequence $\{j(k)\}_{k=0}^{\infty}$ is almost cyclic on $J = \{1, 2, \ldots, p\}$.

Remark 4.5.1 For the special case $f(y) = \frac{1}{2} \parallel y \parallel^2$ (see Example 2.2.1 and Remark 4.4.1), Algorithm 4.5.1 coincides with "Algorithm Scheme I" of Herman and Lent [114]. Thus, the algorithm discussed here is a generalization of Herman and Lent's result to linearly constrained problems with an objective function f that is a Bregman function.

The next theorem establishes the convergence of Algorithm 4.5.1.

Theorem 4.5.1 *Let $f \in \mathcal{B}(S)$ be strongly zone consistent with respect to the hyperplanes $H(\phi^j, \delta_j)$ and $H(\phi^j, \gamma_j)$, for all $j \in J$. Assume that $Q \cap \overline{S} \neq \emptyset$ and that $\phi^j \neq 0$, for all $j \in J$. Assume also that $\{j(k)\}_{k=0}^{\infty}$ is almost cyclic on J and that $U \neq \emptyset$. Then, any sequence $\{y^s\}_{s=0}^{\infty}$ generated by Algorithm 4.5.1 converges to a solution of (4.38).*

Sketch of Proof. The complete proof is given in Censor and Lent [47], here we give only its basic outline. Define

$$a^{j^+} \doteq \phi^j, \qquad b_{j^+} \doteq \delta_j, \tag{4.45}$$

$$a^{j^-} \doteq -\phi^j, \qquad b_{j^-} \doteq -\gamma_j, \tag{4.46}$$

for all $j \in J$. Then, the problem

$$\begin{cases} & \min f(y), \\ \text{subject to} & \langle a^{j^+}, y \rangle \leq b_{j^+}, \quad \text{for all } j \in J, \\ & \langle a^{j^-}, y \rangle \leq b_{j^-}, \quad \text{for all } j \in J, \\ & y \in \overline{S}, \end{cases} \tag{4.47}$$

has the same solution as (4.38). Associate dual variables z_j^+ and z_j^- with the constraints of (4.47), and interlace the vectors z^+ and z^- obtained in this way into a dual vector z, having $2p$ components, defined by

$$z^T = (z_1^+, z_1^-, z_2^+, z_2^-, \ldots, z_j^+, z_j^-, \ldots, z_p^+, z_p^-). \tag{4.48}$$

The single dual variable to be used for each interval constraint of (4.38) will be the corresponding component of

$$u \doteq z^+ - z^-. \tag{4.49}$$

The proof consists of applying Algorithm 4.4.1 to the problem (4.47), in such a way that pairs of inequalities originating from one interval constraint in (4.38) are taken up in an almost cyclic order, but the decision as to whether to take first $H(\phi^{j(k)}, \delta_{j(k)})$ and then $H(\phi^{j(k)}, \gamma_{j(k)})$, or vice versa, varies from pair to pair, depending on the sign of the $j(k)$th component of the current vector u^k of dual variables. Of course, such a strategy would not be permissible without the extra feature of almost cyclicality introduced in Theorem 4.4.1. □

j^+ and then to j^-. generalized $z_j^{-k} = 0$. Therefore, □

4.6 Row-Action Algorithms for Norm Minimization

We specialize the methods presented in Sections 4.4 and 4.5 for the Bregman function $f(x) = \frac{1}{2} \parallel x \parallel^2$ (Example 2.2.1). By doing so we obtain the following algorithms:

(i) for linear equality constraints – the algorithm of Kaczmarz [138].

(ii) for linear inequality constraints – the algorithm of Hildreth, see [73, 121, 150] and

(ii) for linear interval constraints – the **ART4** algorithm of Herman and Lent [114].

4.6.1 The Algorithm of Kaczmarz

Algorithm 4.6.1 Kaczmarz's Algorithm.

Initialization: $x^0 \in \Re(A^T)$ – the range of A^T.

Iterative Step:

$$x^{k+1} = x^k + \lambda_k \frac{b_{i(k)} - \langle a^{i(k)}, x^k \rangle}{\| a^{i(k)} \|^2} a^{i(k)} \tag{4.50}$$

Control: The sequence $\{i(k)\}$ is almost cyclic on I.

Relaxation: $\forall k,\ \epsilon_1 \leq \lambda_k \leq 2 - \epsilon_2$, for some $\epsilon_1, \epsilon_2 > 0$.

In the unrelaxed case, i.e., when $\lambda_k = 1$ for all $k \geq 0$, the convergence of this algorithm can be obtained from the study of Bregman's method in Section 4.4. With $f(x) = \frac{1}{2} \| x \|^2$, $S = \overline{S} = \Re^n$, and only equality constraints, the problem (4.9) becomes,

$$\begin{cases} \min \frac{1}{2} \| x \|^2 \\ \text{s.t. } \langle a^i, x \rangle = b_i, \qquad i \in I. \end{cases} \tag{4.51}$$

Kaczmarz's algorithm is then obtained from Algorithm 4.4.1. A look at Lemma 4.4.1 shows that, since only equality constraints are present, $x^{k+1} \in H_{i(k)}$ always holds and $c_k = B_k$, for all $k \geq 0$. Thus, the dual iterates z^k need not be updated at all. For the underrelaxed case, i.e., $\epsilon_1 \leq \lambda_k \leq 1$, for all $k \geq 0$, the convergence can be obtained from the relaxed Bregman method discussed by De Pierro and Iusem [188].

The general case, with the full range of relaxation parameters, can be treated separately, see, e.g., [115].

Geometric interpretation. Given x^k and the hyperplane $H_{i(k)} = \{x \in \Re^n \mid \langle a^{i(k)}, x \rangle = b_{i(k)}\}$, determined by the $i(k)$th equation, x^{k+1} lies on the line, through x^k, perpendicular to $H_{i(k)}$. For unity relaxation, $\lambda_k = 1$ for all $k \geq 0$, x^{k+1} is the orthogonal projection of x^k onto $H_{i(k)}$. The relaxation option actually allows the next iterate x^{k+1} to be inside the line segment connecting x^k and its orthogonal reflection with respect to $H_{i(k)}$.

4.6.2 The Algorithm of Hildreth.

Algorithm 4.6.2 Hildreth's Algorithm.

Initialization: $z^0 \in \Re^m_+$ is arbitrary and $x^0 = -A^T z^0$.

Iterative Step:

$$x^{k+1} = x^k + c_k a^{i(k)} \tag{4.52}$$

$$z^{k+1} = z^k - c_k e^{i(k)} \tag{4.53}$$

with

$$c_k \doteq \min\left(z^k_{i(k)}, \lambda_k \frac{b_{i(k)} - \langle a^{i(k)}, x^k\rangle}{\| a^{i(k)} \|^2}\right) \tag{4.54}$$

Control: The sequence $\{i(k)\}$ is almost cyclic on I.

Relaxation: $\forall k \geq 0,\ \epsilon_1 \leq \lambda_k \leq 2 - \epsilon_2$, for some $\epsilon_1, \epsilon_2 > 0$.

Again, for unity relaxation where $\lambda_k = 1$, for all $k \geq 0$, the convergence follows from Bregman's method in Section 4.4. The underrelaxed case can be obtained from the extension of De Pierro and Iusem [188]. The convergence for general relaxation parameters in the interval $[\epsilon_1, 2 - \epsilon_2]$ has been treated separately by Lent and Censor [150].

Geometric interpretation. Given x^k and the closed halfspace $L_{i(k)} \doteq \{x \in \Re^n \mid \langle a^{i(k)}, x\rangle \leq b_{i(k)}\}$, determined by the $i(k)$th inequality of the problem

$$\begin{cases} \min \frac{1}{2} \| x \|^2 \\ \text{s.t. } \langle a^i, x\rangle \leq b_i, \qquad i \in I, \end{cases} \tag{4.55}$$

then, if $x^k \notin L_{i(k)}$ then x^{k+1} is the (possibly relaxed) orthogonal projection of x^k onto $L_{i(k)}$. If x^k belongs to the bounding hyperplane $H_{i(k)}$ then

$x^{k+1} = x^k$. Finally, if $x^k \in \text{int } L_{i(k)}$, i.e., if $\langle a^{i(k)}, x^k \rangle < b_{i(k)}$, then a move perpendicular to the bounding hyperplane $H_{i(k)}$ is made. In this case, either $c_k = z^k_{i(k)}$ or, otherwise, x^{k+1} is the orthogonal projection of x^k onto $H_{i(k)}$.

A simultaneous version of Hildreth's row-action algorithm was proposed and studied by Iusem and De Pierro [134]. In that algorithm a convex combination of individual Hildreth-steps with respect to all halfspaces, is taken as the next iterate x^{k+1}.

4.6.3 ART4 – An Algorithm for Norm-Minimization Over Linear Intervals

The problem

$$\begin{cases} \min \frac{1}{2} \| x \|^2 \\ \text{s.t. } \gamma_i \leq \langle a^i, x \rangle \leq \delta_i, \qquad i \in I, \end{cases} \tag{4.56}$$

is of the form (4.38) and Algorithm 4.5.1 applies. The desire to solve a problem with such interval constraints comes from the approach described in Section 4.2.2, where an inconsistent system of equality constraints $\langle a^i, x \rangle = b_i$, $i \in I$, is replaced by intervals by defining $\gamma_i \doteq b_i - \alpha_i$ and $\delta_i \doteq b_i + \alpha_i$ for all $i \in I$. A practical difficulty is to properly choose the "tolerances" α_i so that the system of interval constraints is feasible but not too large.

A study of this sort in the field of image reconstruction from projections was reported by Herman and Lent [114].

The **ART4** (ART stands for "Algebraic Reconstruction Technique") algorithm is an extension of Hildreth's Algorithm 4.6.2 designed to efficiently handle the interval constraint. It is actually the idea embodied in **ART4** that motivated Censor and Lent in their development of Algorithm 4.5.1 for interval convex programming. **ART4** is retrieved from Algorithm 4.5.1 by

taking $f(x) = \frac{1}{2} \| x \|^2$.

Algorithm 4.6.3 ART4.

Initialization: $x^0 \in \Re^n$ is arbitrary and $z^0 \in \Re^m$ is such that $x^0 = -A^T z^0$.

Iterative Step:

$$x^{k+1} = x^k + c_k a^{i(k)} \tag{4.57}$$

$$z^{k+1} = z^k - c_k e^{i(k)} \tag{4.58}$$

with

$$c_k \doteq \text{mid} \left(z^k_{i(k)}, \frac{\delta_{i(k)} - \langle a^{i(k)}, x^k \rangle}{\| a^{i(k)} \|^2}, \frac{\gamma_{i(k)} - \langle a^{i(k)}, x^k \rangle}{\| a^{i(k)} \|^2} \right) \tag{4.59}$$

where mid(a,b,c) stands, as mentioned before, for the median of the three real numbers a,b, and c.

Control: $\{i(k)\}$ is almost cyclic on I.

Relaxation: Unity, i.e., $\lambda_k = 1$, for all $k \geq 0$.

In a typical iterative step the i(k)-th hyperslab is employed. Given x^k, the next iterate x^{k+1} will be inside the slab or on one of its bounding hyperplanes, according to the value of c_k. This value of c_k is determined by the "mid" operation and depends on the distances of x^k from the two bounding hyperplanes and on the value of the $i(k)$-th component of the k-th dual vector z^k.

If additional box constraints are present in problem (4.56) then they can be handled in a simple manner as follows. The problem now is

$$\begin{cases} \min \frac{1}{2} \| x \|^2 \\ \text{s.t. } \gamma_i \leq \langle a^i, x \rangle \leq \delta_i, \qquad i \in I, \\ \qquad w_j \leq x_j \leq v_j, \qquad 1 \leq j \leq n. \end{cases} \tag{4.60}$$

and the algorithm, which was suggested by Herman et al. [116], is called **ART2**.

Algorithm 4.6.4 ART2.

Initialization: $\hat{x}^0 \in \Re^n$ is arbitrary and $z^0 \in \Re^m$ is such that $\hat{x}^0 = -A^T z^0$.

Iterative Step:

$$\hat{x}^{k+1} = \hat{x}^k + c_k a^{i(k)} \tag{4.61}$$

$$z^{k+1} = z^k - c_k e^{i(k)} \tag{4.62}$$

with

$$c_k \doteq \text{mid} \left(z^k_{i(k)}, \frac{\delta_{i(k)} - \langle a^{i(k)}, x^k \rangle}{\| a^{i(k)} \|^2}, \frac{\gamma_{i(k)} - \langle a^{i(k)}, x^k \rangle}{\| a^{i(k)} \|^2} \right) \tag{4.63}$$

and x^k denotes the vector whose t-th component is

$$x^k_t \doteq \begin{cases} w_t, & \text{if} \quad \hat{x}^k_t < w_t, \\ \hat{x}^k_t, & \text{if} \quad w_t \leq \hat{x}^k_t \leq v_t, \\ v_t, & \text{if} \quad v_t < \hat{x}^k_t. \end{cases} \tag{4.64}$$

Herman and Lent [114] showed the convergence of **ART2** to a solution of problem (4.60).

4.7 Row-Action Algorithms for Shannon's Entropy Minimization

We now specialize the methods of Sections 4.4 and 4.5 for the Bregman function $f(x) = \sum_{j=1}^{n} x_j \log\ x_j$ (Example 2.2.2). By simply calculating gradients, the first equation of (4.13) takes the form

$$x_j^{k+1} = x_j^k \cdot \exp\ (c_k a_j^{i(k)}), \quad j = 1, 2, \ldots,\ n. \tag{4.65}$$

We look at problems of the form

$$\begin{cases} \min\ \sum_{j=1}^{n} x_j \log x_j \\ \text{s.t.}\ \ x \in Q \cap \overline{S}, \end{cases} \tag{4.66}$$

where $\overline{S} = \Re_+^n$ and Q is one of the following constraints sets:

$$Q_1 \doteq \{x \in \Re^n \mid\ Ax = b\}, \tag{4.67}$$

$$Q_2 \doteq \{x \in \Re^n \mid\ Ax \leq b\}, \tag{4.68}$$

$$Q_3 \doteq \{x \in \Re^n \mid\ c \leq Ax \leq b\}, \tag{4.69}$$

Bregman's algorithm for the solution of (4.66) with $Q = Q_1$ is a row-action iterative procedure which performs successive generalized projections onto the individual hyperplanes of (4.66). Applying Algorithm 4.4.1 in this case yields the following

Algorithm 4.7.1 Bregman's Row-Action Method for Linear Equality constrained Entropy Maximization.

Initialization: $x^0 \in \Re_{++}^n$ is such that for an arbitrary $z^0 \in \Re_+^m$

$$x_j^0 = \exp\ [(-A^T z^0)_j - 1], \ \ j = 1, 2, ..., n. \tag{4.70}$$

Iterative Step: Given x^k choose a control index $i(k)$ and solve the system

$$x_j^{k+1} = x_j^k \cdot \exp\left(c_k a_j^{i(k)}\right), \quad j = 1, 2, \ldots, n, \tag{4.71}$$

$$\langle x^{k+1}, a^{i(k)} \rangle = b_{i(k)}. \tag{4.72}$$

The system (4.71)-(4.72) represents an *entropy projection* of x^k onto the hyperplane $H_{i(k)} \doteq \{x \in \Re^n \mid \langle a^{i(k)}, x \rangle = b_{i(k)}\}$ resulting with the next iterate x^{k+1}. It is a system of $n+1$ equations, of which the first n are not linear, in the $n+1$ unknown x_j^{k+1}, $j = 1, 2, \ldots, n$, and c_k. It must be solved by some iterative method such as the Newton-Raphson method, see, e.g., [184]. This creates a gap between the theoretical algorithm and its practical implementation. An alternative algorithm for the same problem, i.e., (4.66) with $Q = Q_1$, is **MART** (Multiplicative Algebraic Reconstruction Technique) which employs a closed-form formula for the iterative updates.

Algorithm 4.7.2 MART.

Initialization: $u^0 \in \Re^m$ is arbitrary, and $x^0 \in \Re^n$ is defined by

$$1 + \log\ x_j^0 = (-A^T u^0)_j, \quad j = 1, 2, \ldots, n. \tag{4.73}$$

Iterative Step:

$$x_j^{k+1} = x_j^k \left(\frac{b_{i(k)}}{\langle a^{i(k)}, x^k \rangle} \right)^{\lambda_k a_j^{i(k)}}, \quad j = 1, 2, \ldots, n, \tag{4.74}$$

Control: $\{i(k)\}_{k=0}^{\infty}$ is an almost cyclic control sequence on I.

Relaxation: $\{\lambda_k\}_{k=0}^{\infty}$ is a sequence of *relaxation parameters* such that

$$0 < \epsilon \leq \lambda_k \leq 1, \tag{4.75}$$

for all $k \geq 0$, with some fixed $\epsilon > 0$.

To prove convergence of **MART**, the following assumptions need to be made.

Assumption 4.7.1 (i) *Feasibility:* $Q_1 \cap \Re_+^n \neq \emptyset$.

(ii) *Signs:* For all $j = 1, 2, \ldots, n$ and all $i \in I$,

$$a_j^i \geq 0 \qquad \text{and} \qquad b_i > 0, \tag{4.76}$$

and

$$a_j^i \neq 0, \qquad i \in I. \tag{4.77}$$

(iii) *Normalization:* $Ax = b$ is scaled so that for all $j = 1, 2, \ldots, n$, and all $i \in I$, $a_j^i \leq 1$.

Assumption 4.7.1 (ii) is not too restrictive because it can be made to hold in practice. For example, in image reconstruction from projections (which provides some of our own motivation for studying entropy optimization), see, e.g., [40], $a_j^i \geq 0$ holds by the nature of the problem. $b_i > 0$ means that any equation with zero right hand side should be removed from the constraints at the start, a very reasonable thing to do when reconstructing an image from its projections.

Assumption 4.7.1(iii) is also easy to satisfy. We do not know how **MART** would behave if Assumption 4.7.1(i) were violated.

The convergence of a sequence $\{x^k\}$ produced by **MART** to the solution of the linearly constrained entropy optimization problem can be derived indirectly from the general theory of Bregman by studying the relationship between **MART** and Algorithm 4.7.1. This was done by Censor et al. in [49]. However, a direct proof of convergence can be given, see, e.g., [151], to validate the following result.

Theorem 4.7.1 *If Assumptions 4.7.1 hold, then any sequence* $\{x^k\}$ *generated by Algorithm 4.7.2 converges to the unique minimizer of problem (4.66) with* $Q = Q_1$ *as in (4.67).*

Chapter 5

Proximal Minimization Algorithms with D-Functions

5.1 Introduction

The proximal minimization algorithm is designed to solve the optimization problem

$$\begin{cases} \min & f(x) \\ \text{s.t.} & x \in X, \end{cases} \tag{5.1}$$

where $F : \Re^n \to \Re$ is a given convex function and $X \subseteq \Re^n$ is a nonempty closed convex subset of the n-dimensional Euclidean space $\Re^n$. The approach is based on converting (5.1) into a sequence of optimization problems with strictly convex objective functions obtained by adding quadratic terms to $F(x)$.

The method is as follows. There is a (given or constructed) sequence $\{c(t)\}$ of positive numbers for all $t \in N_0$, $N_0 \doteq \{0, 1, 2, 3, ...\}$, with

$$\lim_{t \to \infty} \inf\ c(t) = c > 0. \tag{5.2}$$

Algorithm 5.1.1 The Proximal Minimization Algorithm.

Initialization: $x(0) \in \Re^n$ arbitrary,

Iterative Step:

$$\begin{cases} x(t+1) = \arg\min_{x \in X} \{F(x) + [1/2c(t)] \parallel x - y(t) \parallel^2\}, \\ y(t+1) = x(t+1), \end{cases} \tag{5.3}$$

Equivalently, the algorithm is written as

$$x(t+1) = \arg\min_{x \in X} \{F(x) + [1/2c(t)] \parallel x - x(t) \parallel^2\}. \tag{5.4}$$

The origins of this algorithm go back to Minty [160], Moreau [162], and Rockafellar [197, 198]. In addition to considerable theoretical interest in the family of proximal point algorithms, of which it is a member, this algorithm is also an important computational tool. This is so because the dual problem of a strictly convex optimization problem is differentiable and can be solved by simple iterative procedures like dual coordinate ascent. For several important problem classes, these dual algorithms can be decomposed for parallel computations; the results of such investigation are reported in Nielsen and Zenios [180, 178].

In this chapter, we generalize the proximal minimization algorithm by replacing the quadratic term in (5.3) by D_f-functions for which convergence of the algorithm can be preserved. The material given here is taken from Censor and Zenios [52].

The original proximal minimization algorithm (5.3) is obtained from our scheme by choosing the D_f-function of Example 2.2.1. A different choice leads to a proximal minimization algorithm with entropy additive terms. In the case of linear programming (F and $x \in X$ are all linear), the latter leads to pure entropy optimization problems for which the special-purpose algorithms of Chapter 4 are applicable. Such an approach of replacing a linear programming problem by a sequence of entropy problems was heuristically suggested by Eriksson [89]. He discusses also a specific strategy for choosing the parameters $\{c(t)\}$ and a solution algorithm. However, no overall convergence analysis is given there. The practical question of whether any efficient useful algorithm results from this new look at things has been addressed by Nielsen and Zenios [180, 178, 177], where encouraging computational results are reported.

The fundamental proximal point algorithm for solving the problem $0 \in T(z)$ for an arbitrary maximal monotone operator T and its specialization for $T = \partial F$ (the subdifferential of F) make it clear why quadratic additive terms in (5.3) are mandatory; see, e.g., [197]. Therefore, we do not resort to the operator theory, but rather follow the more direct method of Bertsekas and Tsitsiklis [26]. It is quite conceivable that the idea of incorporating D_f-functions could propagate in other directions within the theory of proximal point and related methods.

The idea of replacing quadratic penalty terms by nonquadratic ones

exists already with respect to other algorithms; see, e.g., [25, Chapter 5]. Teboulle [219] has recently derived what he calls "entropic proximal maps" and used them to construct generalized augmented Lagrangian methods. Although his paper can be considered a close companion to this chapter, his results do not include the **PMD** algorithm (Algorithm 5.1.2) that we describe here. See also Chen and Teboulle [53]. An important work on monotone operators and the proximal point algorithm is Eckstein's thesis [79]. Moreover, in his recent paper [80], Eckstein showed how to construct proximal point algorithms with Bregman functions, thereby further extending the scope of the connection between Bregman functions and proximal minimization presented in this chapter.

Another recent related study is Eggermont's [81]. Nonquadratic additive terms are used there, but with only nonnegativity constraints. Of particular interest is the connection revealed there between multiplicative iterative algorithms and the well-known **EM**- algorithm for maximum likelihood estimation in emission tomography; see Shepp and Vardi [211] and other references in [81]. The algorithms of [81], however, are not special instances of our proximal minimization algorithm with D_f-functions. Finally, we mention the work of Tseng and Bertsekas [222], where they use the entropy proximal term in the proximal minimization algorithm to study the exponential multiplier method.

The *proximal minimization algorithm with D_f-functions,* henceforth abbreviated **PMD**, is as follows. Given are a function f with zone S, and a positive sequence $\{c(t)\}$ for which (5.2) holds.

Algorithm 5.1.2 PMD.

Initialization: $x(0) \in S$ is arbitrary,

Iterative Step:

$$x(t+1) = \arg \min_{x \in X \cap \overline{S}} \{F(x) + [1/c(t)] D_f(x, x(t))\}. \tag{5.5}$$

In order for this algorithm to be well defined, we make the next assumption.

Assumption A3. The **PMD** algorithm (5.5) generates a sequence $\{x(t)\}$ such that $x(t) \in S$, for every t = 0,1,2, ...

This assumption is needed, because D_f is defined on $\overline{S} \times S$. It actually tells us that, given F and X of (5.1), we are free to choose only such f and S that Assumption A3 would hold. If $X \subseteq S$, then Assumption A3 trivially holds, which is true for the quadratic case $f(x) = (1/2) \|x\|^2$, where $S = \Re^n$. We show later that it holds also for the entropy case.

5.2 Convergence Analysis of the PMD Algorithm

The analysis given here follows the one given in [26, Chapter 3.4.3]. Proposition 5.2.1 secures the existence and uniqueness of the minimum of $\{F(x) + (1/c)D_f(x, y)\}$. Propositions 5.2.2, 5.2.3 and 5.2.4 are not directly necessary for the proof of convergence, but they extend to the D_f-function setting some closely related results from [26]. The final convergence result is given in Theorem 5.2.1.

Denote by X^* the solution set of problem (5.1), i.e.,

$$X^* \doteq \{x^* \in X \mid F(x^*) \leq F(x), \ \ \forall x \in X\}. \tag{5.6}$$

Proposition 5.2.1 *Let $f \in \mathcal{B}(S)$. For every $y \in S$ and $c > 0$, the minimum of $\{F(x) + (1/c)D_f(x,y)\}$ over $X \cap \overline{S}$ is attained at a unique point, denoted by $x_f(y,c)$, provided that $F(x)$ is bounded below over X.*

Proof. For all $c > 0$ and $y \in S$, the level sets

$$\{x \in X \cap \overline{S} \mid F(x) + (1/c)D_f(x,y) \leq \alpha\}, \qquad \alpha \in \Re, \tag{5.7}$$

are bounded. This is true because otherwise, for some $c > 0$ and $y \in \Re^n$, there would exist an unbounded sequence $\{x^k\} \subseteq X \cap \overline{S}$ for which

$$D_f(x^k, y) \leq c(\alpha - L), \tag{5.8}$$

where L is the lower bound for $F(x)$ over X. But this would contradict (iv) of Definition 2.2.1; thus, the level sets (5.7) must be bounded.

This allows us to equivalently search for the minimum of

$$F(x) + (1/c)D_f(x,y)$$

over a compact subset of $X \cap \overline{S}$ instead of over $X \cap \overline{S}$. The Weierstrass theorem (e.g., [26, Proposition A.8]), then implies that the above-mentioned minimum is attained. The strict convexity of $D_f(x,y)$ with respect to x for fixed y, (see Lemma 2.2.1), and the strict convexity of the function f, ensure the uniqueness. □

Proposition 5.2.2 *If $D_f(x,y)$ is jointly convex with respect to both x and y, i.e., as a function on $\Re^{2n}$, then the function $\Phi_c : S \to \Re$, defined by*

$$\Phi_c(y) \doteq \min_{x \in X \cap \overline{S}} \{F(x) + (1/c)D_f(x,y)\}, \tag{5.9}$$

is convex over S.

Proof. Let $y^1, y^2 \in S$ and $\alpha \in [0,1]$. Denote $x_f^i = x_f(y^i, c)$, for $i = 1, 2$. Using the convexity of F and joint convexity of $D_f(x, y)$, we have

$$\begin{aligned}
&\alpha\Phi_c(y^1) + (1-\alpha)\Phi_c(y^2) \\
&= \alpha[F(x_f^1) + (1/c)D_f(x_f^1, y^1)] + (1-\alpha)[F(x_f^2) + (1/c)D_f(x_f^2, y^2)] \\
&\geq F(\alpha x_f^1 + (1-\alpha)x_f^2) + (1/c)D_f(\alpha x_f^1 + (1-\alpha)x_f^2, \alpha y^1 + (1-\alpha)y^2) \\
&\geq \min_{x \in X \cap \overline{S}}\{F(x) + (1/c)D_f(x, \alpha y^1 + (1-\alpha)y^2)\} \\
&= \Phi_c(\alpha y^1 + (1-\alpha)y^2).
\end{aligned}$$

□

Proposition 5.2.3 *Let $f \in \mathcal{B}(S)$ together with the assumption of Proposition 5.2.2. Then the function $\Phi_c(y)$ is continuously differentiable on S and its gradient is given by*

$$\nabla\Phi_c(y) = \nabla^2 f(y)^T[[y - x_f(y, c)]/c], \tag{5.10}$$

where T denotes matrix transposition.

Proof. Consider any $y \in S$, $d \in \Re^n$, and $\alpha > 0$ such that $y + \alpha d \in S$. Using the directional derivative $\Phi_c'(y; d)$, we have

$$\begin{aligned}
&F(x_f(y, c)) + (1/c)D_f(x_f(y, c), y + \alpha d) \\
&\geq \Phi_c(y + \alpha d) \geq \Phi_c(y) + \alpha\Phi_c'(y; d) \\
&= F(x_f(y, c)) + (1/c)D_f(x_f(y, c), y) + \alpha\Phi_c'(y; d),
\end{aligned} \tag{5.11}$$

where the second inequality in (5.11) follows from the convexity of Φ_c (Proposition 5.2.2). Therefore, using (2.1), we get from (5.11)

$$\begin{aligned}
&(1/c)[f(y) - f(y + \alpha d) + \langle \nabla f(y) - \nabla f(y + \alpha d), x_f(y, c) - y\rangle \\
&+ \langle \nabla f(y + \alpha d), \alpha d\rangle] \geq \alpha\Phi_c'(y; d).
\end{aligned} \tag{5.12}$$

Since

$$\lim_{\alpha \to 0}[[f(y) - f(y + \alpha d)]/\alpha + (1/\alpha)\langle \nabla f(y + \alpha d), \alpha d\rangle] = 0, \tag{5.13}$$

$$\lim_{\alpha \to 0}[[\nabla f(y) - \nabla f(y + \alpha d)]/\alpha] = -\nabla^2 f(y)d, \tag{5.14}$$

we obtain from (5.12) by dividing by α and letting $\alpha \to 0$,

$$\langle \nabla^2 f(y)d, [y - x_f(y,c)]/c \rangle \geq \Phi_c'(y;d), \quad \forall d \in \Re^n. \tag{5.15}$$

Replacing d by $-d$ in (5.15), we get

$$-\langle \nabla^2 f(y)d, [y - x_f(y,c)]/c \rangle \geq \Phi_c'(y;-d) \geq -\Phi_c'(y;d), \tag{5.16}$$

where the second inequality is a standard relation for directional derivatives of convex functions; see, e.g., [26, p.648].

The relations (5.15) and (5.16) imply that

$$\Phi_c'(y;d) = \langle \nabla^2 f(y)d, [y - x_f(y,c)]/c \rangle, \quad \forall d \in \Re^n, \tag{5.17}$$

or equivalently that Φ_c is differentiable and that its gradient is given by (5.10). Since Φ_c is convex (Proposition 5.2.2), its gradient is continuous; see, e.g., [26, Proposition A.42]. □

The next proposition gives a relation between S^*, the minimum set of $\Phi_c(y)$, the zone S of the Bregman function f, and the solution set X^*. For a function f with zone $S = \Re^n$, we get, as a spacial case, that $X^* = S^*$, which was given in [26, p.234].

Define

$$S^* \doteq \{y^* \in S \mid \Phi_c(y^*) \leq \Phi_c(y), \quad \forall y \in S\}. \tag{5.18}$$

Proposition 5.2.4 *Let $\nabla^2 f(z)$ be nonsingular for all $z \in S^*$. Then,*

$$X^* \cap S = S^*. \tag{5.19}$$

Proof. The function $F(x) + (1/c)D_f(x, y)$ takes the value $F(y)$ for $x = y$ because of Lemma 2.2.1. It follows that

$$\Phi_c(y) \leq F(y), \quad \forall y \in X \cap S. \tag{5.20}$$

If $z^* \in X^*$, then (5.19) holds and we have

$$\begin{aligned} \Phi_c(z^*) &\leq F(z^*) \leq F(x_f(y,c)) \leq F(x_f(y,c)) + (1/c)D_f(x_f(y,c), y) \\ &= \Phi_c(y), \quad \forall y \in S, \end{aligned} \tag{5.21}$$

because always $D_f(x, y) \geq 0$. Thus, z^* minimizes $\Phi_c(y)$ over S, i.e., $z^* \in S^*$. Conversely, if $z^* \in S^*$, then we have, from (5.10),

$$c\nabla\Phi_c(z^*) = \nabla^2 f(z^*)^T[z^* - x_f(z^*, c)] = 0, \tag{5.22}$$

which implies that $z^* = x_f(z^*, c) \in X \cap S$. Using again (5.20), we have

$$F(z^*) = \Phi_c(z^*) \leq \Phi_c(y) \leq F(y), \quad \forall y \in X \cap S, \tag{5.23}$$

and therefore $z^* \in X \cap S$. □

Finally, we present the convergence proof of the **PMD** algorithm.

Theorem 5.2.1 *Let $f \in \mathcal{B}(S)$, let Assumption A3 hold and assume that $X^* \cap \overline{S} \neq \emptyset$. Then any sequence $\{x(t)\}$ generated by a* **PMD** *algorithm, where $c(t) > 0$ and $\liminf_{t\to\infty} c(t) = c > 0$, converges to an element of X^*.*

Proof. The proof consists of three steps. First, we prove that $\{x(t)\}$ is bounded; then, we show that all its accumulation points belong to X^*; and finally, we prove that there is a unique limit point.

Step 1. We have

$$\begin{aligned} F(x(t+1)) &= [1/c(t)]D_f(x(t+1), x(t)) \\ &\leq F(x) + [1/c(t)]D_f(x, x(t)), \quad \forall x \in X \cap \overline{S}, \end{aligned} \tag{5.24}$$

from which follows that, for all $x \in X \cap \overline{S}$ with

$$F(x) \leq F(x(t+1)), \tag{5.25}$$

it is true that

$$D_f(x(t+1), x(t)) \leq D_f(x, x(t)). \tag{5.26}$$

Therefore, $x(t+1)$ is the unique generalized projection of $x(t)$ onto the convex set

$$\Omega \doteq \{x \in X \mid F(x) \leq F(x(t+1))\}. \tag{5.27}$$

Using Theorem 2.5.1 and the fact that $X^* \subseteq \Omega$, we have

$$0 \leq D_f(x(t+1), x(t)) \leq D_f(x^*, x(t)) - D_f(x^*, x(t+1)), \tag{5.28}$$

for every $x^* \in X^* \cap \overline{S}$. Thus,

$$D_f(x^*, x(t+1)) \leq D_f(x^*, x(t)), \quad \forall x^* \in X^* \cap \overline{S}, \quad \forall t. \tag{5.29}$$

This last inequality amounts to saying that $\{x(t)\}$ is D_f-Fèjer-monotone with respect to the set $X^* \cap \overline{S}$, and it implies that $\{x(t)\}$ is bounded because it means that

$$x(t) \in L_2(x^*, \alpha), \quad \forall t, \tag{5.30}$$

with $\alpha \doteq D_f(x^*, x(0))$, and condition (iv) of Definition 2.2.1 applies.

Step 2. Let $\{x(t)\}_{t \in T}$, $T \subseteq N_0$, be a subsequence converging to $x^\infty \in X \cap \overline{S}$. Recall that, by Lemma 2.2.1 and (5.29), the sequence $\{D_f(x^*, x(t))\}_{t \in N_0}$

is nonnegative and nonincreasing; thus, $\lim D_f(x^*, x(t))$ exists for any $x^* \in X^* \cap \overline{S}$. In view of (5.28),

$$\lim D_f(x(t+1), x(t)) = 0, \qquad \text{as } t \to \infty;$$

thus, also

$$\lim_{\substack{t\to\infty \\ t\in T}} D_f(x(t+1), x(t)) = 0, \tag{5.31}$$

which by Assumption A3 implies that $\{x(t+1)\}_{t\in T}$ also converges to x^∞.

Next, observe that (5.24) remains true with $x = x(t)$; thus,

$$F(x(t+1)) = [1/c(t)]D_f(x(t+1), x(t)) \le F(x(t)), \qquad \forall t, \tag{5.32}$$

because $D_f(x(t), x(t)) = 0$. This implies

$$F(x(t)) - F(x(t+1)) \ge [1/c(t)]D_f(x(t+1), x(t)) \ge 0, \quad \forall t. \tag{5.33}$$

Therefore, $\{F(x(t))\}_{t\in N_0}$ is nonincreasing and $\{F(x(t))\}_{t\in T}$ converges to $F(x^\infty)$.

Let $x^* \in X^* \cap \overline{S}$ and $\alpha \in (0,1)$, and set

$$x \doteq \alpha x^* + (1-\alpha)x(t+1)$$

in (5.24). From the convexity of $F(x)$, we get

$$\begin{aligned}
&F(x(t+1)) + [1/c(t)]D_f(x(t+1), x(t)) \\
&\le F(\alpha x^* + (1-\alpha)x(t+1)) \\
&+[1/c(t)]D_f(\alpha x^* + (1-\alpha)x(t+1), x(t)) \\
&\le \alpha F(x^*) + (1-\alpha)F(x(t+1)) \\
&+[1/c(t)]D_f(\alpha x^* + (1-\alpha)x(t+1), x(t)),
\end{aligned} \tag{5.34}$$

which, by (2.1), can be rewritten as

$$\begin{aligned} &\alpha c(t)[F(x(t+1)) - F(x^*)] \\ &\leq f(x(t+1) - \alpha(x^* - x(t+1))) - f(x(t+1)) \\ &-\alpha\langle \nabla f(x(t)), x^* - x(t+1)\rangle. \end{aligned} \tag{5.35}$$

Dividing by α, taking the limit as $\alpha \to 0^+$, and denoting by $f'(\cdot;\cdot)$ the directional derivative, we get

$$c(t)[F(x(t+1)) - F(x^*)]$$

$$\leq f'(x(t+1); x^* - x(t+1)) - \langle \nabla f(x(t)), x^* - x(t+1)\rangle$$

$$= \langle \nabla f(x(t+1)) - \nabla f(x(t)), x^* - x(t+1)\rangle$$

$$= D_f(x^*, x(t)) - D_f(x^*, x(t+1)) - D_f(x(t+1), x(t)). \tag{5.36}$$

The optimality of x^* and the fact that $c(t) \geq c > 0$, for all t, guarantee the nonnegativity of the left-hand side of (5.36) for all t.

From the existence of $\lim_{t\to\infty} D_f(x^*, x(t))$, for any $x^* \in X^* \cap \overline{S}$, we obtain

$$\lim_{\substack{t\to\infty \\ t\in T}} [D_f(x^*, x(t)) - D_f(x^*, x(t+1))] = 0, \tag{5.37}$$

and the remaining term in the right-hand side of (5.36) also tends to zero by (5.31). Thus,

$$0 = \lim_{\substack{t\to\infty \\ t\in T}} [F(x(t+1)) - F(x^*)] = F(x^\infty) - F(x^*), \tag{5.38}$$

and so $x^\infty \in X^*$.

Step 3. Let

$$\lim_{\substack{t\to\infty \\ t\in T_1}} x(t) = x^* \in X^* \cap \overline{S}, \tag{5.39}$$

$$\lim_{\substack{t\to\infty \\ t\in T_2}} x(t) = x^{**} \in X^* \cap \overline{S}, \tag{5.40}$$

for some $T_1 \subseteq N_0$ and $T_2 \subseteq N_0$. Defining

$$H(x(t)) \doteq D_f(x^*, x(t)) - D_f(x^{**}, x(t)), \tag{5.41}$$

it follows that the limit

$$\lim_{t\to\infty} H(x(t)) = H \tag{5.42}$$

exists. Therefore, we have, by Lemma 2.2.1,

$$H = \lim_{\substack{t\to\infty \\ t\in T_1}} H(x(t)) = -D_f(x^{**}, x^*) \leq 0, \tag{5.43}$$

$$H = \lim_{\substack{t\to\infty \\ t\in T_2}} H(x(t)) = D_f(x^*, x^{**}) = 0, \tag{5.44}$$

yielding

$$D_f(x^*, x^{**}) = D_f(x^{**}, x^*) = 0, \tag{5.45}$$

which implies that $x^* = x^{**}$. □

5.3 Special Cases.

Choosing the Bregman function $f(x) = (1/2) \parallel x \parallel^2$ with $\Delta = S = \overline{S} = \Re^n$ gives $D_f(x, y) = (1/2) \parallel x - y \parallel^2$ and immediately returns the **PMD** algorithm to its original form with quadratic additive terms.

For $f(x) = -\text{ent}(x)$, of Example 2.2.2, we have

$$D_f(x, y) = \sum_{j=1}^{n} x_j[\log(x_j/y_j) - 1] + \sum_{j=1}^{n} y_j, \tag{5.46}$$

and the iterative step of the entropy-type **PMD** algorithm is obtained from (5.5).

In the case where problem (5.1) is linear programming, $F(x) = \langle b, x \rangle$ for some given $b = (b_j) \in \Re^n$, and $x \in X$ are linear constraints, (5.5) becomes

$$x(t+1) = \arg \min_{x \in X \cap \Re^n_+} \left\{ \sum_{j=1}^{n} x_j b_j + [1/c(t)] \sum_{j=1}^{n} x_j [\log(x_j/x_j(t)) - 1] + [1/c(t)] \sum_{j=1}^{n} x_j(t) \right\}. \tag{5.47}$$

This is essentially a linearly constrained pure entropy optimization problem obtained by subsuming all linear terms into the entropy functional. For such problems, the iterative algorithms of Chapter 4, which lend themselves efficiently to parallel computation are applicable. The advantages, in practice, of an entropy or otherwise oriented **PMD** algorithm over the original quadratic-additive-term proximal minimization algorithm, depends to a great extent on two factors. One is the specific form of the original problem (5.1); the second is the availability of efficient special-purpose algorithms for performing the step (5.5).

These practical questions have been studied in further experimental research, see, [180, 178, 179].

Chapter 6

Applications

In this chapter we present three practical applications that can be modeled as nonlinear optimization problems. These are (1) the problem of matrix balancing, (2) the problem of image reconstruction from projections, and (3) the problem of planning under uncertainty with stochastic network programs. In all cases the objective function belongs to the class of Bregman functions, and hence the row-action algorithms of Chapter 4 are applicable. A common feature of all applications is that they lead to extremely large problems. The number of variables are of the order of 10^6, and the number of constraints of the order 10^5. The shear size of these problems motivated users to employ row-action, iterative methods, in order to solve them in a practical way.

Along with each nonlinear optimization problem we discuss the practical applications where it arises. We then discuss the mathematical model and, in particular, the real-world considerations that led users to optimize a function that belongs to Bregman's class. Finally we present specific row-action

algorithms that have, occasionally, been tailored to the special structure exhibited by the optimization models.

6.1 Matrix Balancing

A problem that occurs frequently in economics, urban planning, statistics, demography, and stochastic modeling is to adjust the entries of a large matrix to satisfy consistency requirements. It is typically posed as follows:

> Given a rectangular matrix A, determine a matrix X that is *close* to A and satisfies a given set of linear restrictions on its entries.

A well-studied instance of this problem — occurring in transportation planning and input-output analysis — requires that A be adjusted so that the row and column totals equal fixed positive values. A related problem occurring in developmental economics requires that the row and column totals (of a square matrix) be equal to each other, but not necessarily to pre-specified values.

These problems are known as *balancing* the matrix A. The terms *matrix estimation* or *matrix adjustment* have also been used to describe the same problems. Because we discuss several balancing problems with different consistency requirements, the definition of a balanced matrix is problem dependent. That is, a matrix is defined to be *balanced* if it satisfies the given set of linear restrictions of the problem. Applications of matrix balancing can be found in the literature in journals of mathematical programming, statistics, economics, urban planning, numerical analysis, and mathematics. Matrix balancing is an important problem that has attracted attention in many different fields.

The matrix balancing applications that we describe can generally be formulated as one of two problems. These are:

Problem 6.1.1 *Given an $m \times n$ nonnegative matrix $A = (a_{ij})$ and positive vectors $u \in \Re^m$ and $v \in \Re^n$ determine a "nearby" nonnegative matrix $X = (x_{ij})$ (of the same dimensions) such that*

$$\sum_{j=1}^{n} x_{ij} = u_i, \quad \text{for } i = 1, 2, \ldots, m, \tag{6.1}$$

$$\sum_{i=1}^{m} x_{ij} = v_j, \quad \text{for } j = 1, 2, \ldots, n, \tag{6.2}$$

and $x_{ij} > 0$ only if $a_{ij} > 0$.

Problem 6.1.2 *Given an $n \times n$ nonnegative matrix $A = (a_{ij})$, determine a "nearby" non-negative matrix $X = (x_{ij})$ (of the same dimensions) such that*

$$\sum_{j=1}^{n} x_{ij} = \sum_{j=1}^{n} x_{ji}, \quad i = 1, 2, \ldots, n, \tag{6.3}$$

and $x_{ij} > 0$ only if $a_{ij} > 0$.

In general, there are infinitely many matrices satisfying the consistency restrictions (6.1), (6.2) or (6.3). For either problem to be well-posed the notion of a *nearby matrix* has to be defined. Different models and algorithmic approaches follow naturally from different types of such definitions.

6.1.1 Applications of Matrix Balancing

Economics: Social Accounting Matrices (SAMs)

A Social Accounting Matrix, or SAM, is a square matrix A whose entries represent the flow-of-funds between the national income accounts of a country's economy at a fixed point in time. Each index of a row or a column of

A represents an account, or agent, in the economy. Entry a_{ij} is positive if agent j receives funds from agent i. A SAM is a *snapshot* of the critical variables in a general equilibrium model describing the circular flow of financial transactions in an economy. For balancing problems arising from estimating SAMs, the balance conditions are the *a priori* accounting identities that each agent's total expenditures and total receipts must be equal. That is, for each index i of the matrix A, the sum of the entries in row i must equal the sum of the entries in column i. The volume by Pyatt and Round [191] contains a collection of papers that give an introduction to Social Accounting Matrices.

The agents of an economy include institutions, factors of production, households, and the rest-of-the-world (to account for transactions with the economies of other countries). Figure 6.1 shows the major relationships between accounts in a simplified SAM. Briefly, the production activities generate value-added which flows to the factors of production — land, labor, and capital. Factor income is the primary source of income for institutions — households, government and firms — who purchase goods and services supplied by productive activities, thereby completing the cycle. Of course, to be useful for equilibrium modeling, this highly aggregated model must be disaggregated into subaccounts for each sector of the economy. Estimation of disaggregated SAMs with up to 260 agents has been reported in the literature, Baker *et al.* [14], although their solution involved a major computational effort. As recent algorithms for this problem become widely known, it is likely that much larger SAMs will be estimated and their solution will be a routine operation.

SAMs are used as the core database for complex economy-wide general equilibrium models. The entries of the SAM provide a convenient

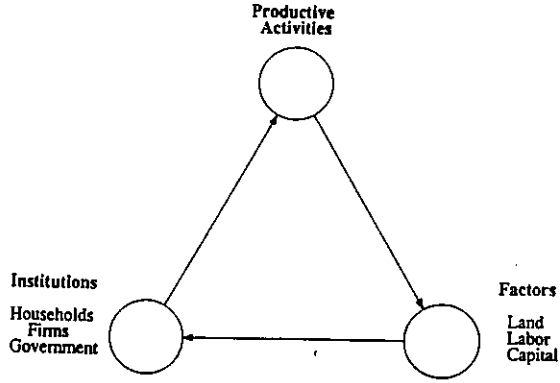


Figure 6.1: A simplified SAM.

database used to estimate parameters in an equilibrium model, see, e.g., Dervis *at al.* [72]. For example, researchers at the World Bank have developed specialized modeling systems based on SAM databases, Kendrick and Drud [141] and Zenios *et al.* [239]. Similar models have been developed by the United Nations Statistical Office, van Tongeren [225], the Cambridge Growth Project, Bacharach [10] and Stone [216], and the Statistical Bureaus of developing and industrialized countries, Pyatt and Round [191].

Inconsistent data is an inherent problem when statistical methods are used to estimate underlying economic models. Morgenstern devoted his 1963 book to the problem of inconsistency in economic measurements [163]. In particular, the direct estimate of a SAM is never balanced. The following quote of Sir Richard Stone (Van der Ploeg [190, page 186] summarizes the sources of inconsistency in SAM modeling.

> " ... it is impossible to establish by direct estimation a system of national accounts free of statistical discrepancies, residual errors, unidentified items, balancing entries and the like since the information available is in some degree incomplete, inconsistent and

> unreliable. Accordingly, the task of measurement is not finished when the initial estimates have been made and remains incomplete until final estimates have been obtained which satisfy the constraints that hold between their true values. "

Therefore, the raw estimates of a SAM must be adjusted so that the consistency requirements are satisfied. The balancing problem of adjusting the initial matrix so that the row and column sums are equal is an example of Problem 6.1.2. This problem motivates much of the work on matrix balancing for economic modeling.

A matrix balancing problem also arises when *partial survey* methods are used to estimate a SAM. Frequently, estimates are available of the total expenditures and receipts for each agent in an economy but current data are not available for the individual transactions between the agents. If a complete (balanced) SAM is available from an earlier period, then the SAM must be updated to reflect the recent index totals. The problem is then to adjust the entries of the old matrix A so that the row and column totals equal the given fixed amounts; this is an example of Problem 6.1.1, see Miller and Blair [158]. A similar balancing problem occurs when the entries of an input-output matrix must be updated to be consistent with exogenous estimates of the total levels of primary inputs and final demands, see, for example, Morrison and Thuman [164], Harrigan and Buchanan [108] or Jensen and McGaurr [136].

Transportation: Estimating Origin-Destination Matrices

Networks are used in urban transportation analysis to model traffic flow over physical transportation systems involving, for example, urban highways,

subways, buses, or airplanes. The transportation system is represented as a network with vertices corresponding to intersections and arcs corresponding to travel links. Each arc has an associated *travel cost function* representing the disutility of travel time as a function of the total flow along that arc. A user entering the transportation network must choose a path from his origin vertex to his destination vertex. The matrix representing the rate of traffic flow between all pairs of vertices is called the *origin-destination matrix.* Each entry of the matrix indicates the fraction of the total network traffic that travels between an origin-destination pair.

The *traffic assignment* problem is to determine the rate of flow along each arc and the travel time between each origin-destination pair given the network representing the system, the travel cost functions, and the origin-destination matrix. For example, under the assumption that users choose the path with the lowest total travel time, the system is in equilibrium if no motorist can lower his or her travel time by altering their route. The resulting steady-state distribution of traffic flows is called a *user equilibrium.* See, for example, Sheffi [210] and Abdfulaal and LeBlanc [2]. We assume that the model represents the network at, for example, a peak travel time and that the number of *trips* is fixed. That is, the total travel demand is represented by a fixed $n \times m$ *origin-destination matrix* A, where a_{ij} is the rate of flow from origin zone i to destination zone j.

The origin-destination matrix, A, is a required input for a wide variety of transportation planning models, LeBlanc and Farhangian [148], Nguyen [174]. Matrix balancing problems arise when origin-destination matrices are estimated from observed traffic flows on selected sets of arcs (Carey *et al.* [37], Jefferson and Scott [135], McNeil [157], Van Zuylen and Willum-

sen [242]). For example, data might be collected measuring the total flow out of each origin vertex and into each destination vertex. The estimated origin-destination matrix should have the property that when it is used as an input into the underlying traffic assignment model, the traffic flows produced as outputs should equal the observed traffic flows. Since there may be many matrices with this property, the estimated matrix is chosen by minimizing some properly defined distance between A and a fixed reference origin-destination matrix A_0 which could be an unreliable estimate based on a previous study.

In practice, the total flow out of each origin and into each destination is frequently used for observed traffic flows. If there is a single route connecting each origin and destination, then the resulting problem of estimating an origin-destination matrix can be formulated as Problem 6.1.1.

A more general framework for estimating origin-destination matrices is produced by assuming that a_{ij}, the aggregate flow from origin i to destination j, is determined by a direct demand function g, $g : \Re^{n^2} \rightarrow \Re$

$$a_{ij} = g(\alpha^{ij}, \beta) + \epsilon_{ij}, \quad \text{for each } (i,j),$$

where α^{ij} is a fixed vector describing the socioeconomic characteristics and travel time for pair (i,j), and ϵ_{ij} is an error term. The vector β is then estimated together with the matrix A, Nguyen [174] and Ben-Akiva [22]. For fixed flows, β is estimated using least-squares, whereas for fixed β, A is estimated using matrix balancing. Therefore, this problem can be decomposed by alternately fixing A and estimating β, and then fixing β and estimating A. This produces an iterative method converging to optimal joint estimates for A and β [175, 22]. The subproblem of estimating A given a fixed vector

β is precisely Problem 6.1.1.

Statistics: Estimating Contingency Tables

Contingency tables are used in the classification of items by several criteria, each of which is partitioned into a finite number of categories. For example, suppose that N individuals in a population are classified according to the criteria marital status and age, which are divided into m and n categories, respectively. The mn distinct classifications are called *cells*, and the resulting $m \times n$ contingency table contains the number of individuals in each cell.

For many applications it is important to estimate the underlying cell probabilities, i.e., the fraction of the population N that is classified in each cell. Determining the cell probabilities directly by sampling the population is generally not possible because such a procedure is often prohibitively expensive. Thus, cell probabilities must be derived from partial observations. A balancing problem arises when prior values for the cell probabilities are revised so that they become consistent with known information. For example, suppose that the marginal distribution of each criterion within the population is known — e.g., the total number of people in each one of the n age categories is known, together with the total number of people in each one of the m categories indicating marital status — and that prior cell values are given. The prior values could be obtained from an out-of-date general census, or from another population with similar ethnic and socioeconomic characteristics, or from a contingency table derived from sampling. The prior cell values must then be adjusted so that they become consistent with the known marginal probabilities. This is an instance of Problem 6.1.1.

Deming and Stephan [71] and Stephan [215] describe applications of ma-

trix balancing procedures for deriving probability estimates from the 1940 census data for the United States. Friedlander [97] discusses the use of those techniques by the British government. Ireland and Kullback [132] discuss algorithms for estimating the underlying cell probabilities of two-way contingency tables based on the principle of minimum discrimination information — entropy minimization. They also describe the extension to four-way tables, that is tables whereby each cell is characterized by four characteristics (e.g., marital status, age, income and education). In the estimation of four-way tables different marginal totals may be available. For example, one-, two- and three-way marginal totals can be given. Darroch and Ratcliff [67] discuss further applications in statistics.

Demography: Modeling Interregional Migration

Estimating interregional migration flows based on partial and outdated information is a recurrent problem in demography. Such problems arise when figures are available for net in- and out-migration from every region and an estimate is needed of the interregional migration patterns. In the United States, for example, flow matrices with detailed migration characteristics become available once every ten years from the general census of the population. In the interim, however, net migration estimates for every region are available as by-products from annual population estimates. (Net migration is the difference between total population change and changes from births and deaths). An updated migration matrix is then needed that reconciles the out-of-date migration patterns with the more recent net figures.

Demographers have postulated several models for estimating interregional migration including gravity models, Markov or fixed transition prob-

ability models, and doubly constrained *minimum-information* models. For a discussion of alternative formulations see Plane [189] or Eriksson [88]. Plane, in particular, develops the minimum-information model and shows that it contains as special cases both the gravity and the fixed rate models.

The doubly constrained minimum-information model gives rise to a matrix balancing problem. It is defined as follows. Let M denote total population in all regions, and for each $i = 1, 2, \ldots, n$, let O_i and I_i be, respectively, the net out-migration and in-migration for region i. Let x_{ij}, $i, j = 1, 2, \ldots, n$, $i \neq j$, be the model estimated probabilities that any individual in the system is a migrant from region i to region j, and let a_{ij} be some prior estimate of the probabilities. The minimum-information model can be written as:

$$\underset{x}{\text{Minimize}} \quad \sum_{i=1}^{n} \sum_{\substack{j=1 \\ j \neq i}}^{n} x_{ij} \left(\ln \left(\frac{x_{ij}}{a_{ij}} \right) - 1 \right) \tag{6.4}$$

$$\text{s.t.} \quad \sum_{\substack{j=1 \\ j \neq i}}^{n} x_{ij} = O_i/M \quad \text{for } i = 1, 2, \ldots, n, \tag{6.5}$$

$$\sum_{\substack{i=1 \\ i \neq j}}^{n} x_{ij} = I_j/M \quad \text{for } j = 1, 2, \ldots, n. \tag{6.6}$$

The constraints are precisely those of Problem 6.1.1. An additional constraint is often imposed to fix the total distance D traveled by all migrants. If d_{ij} is the distance between regions i and j, the additional constraint is

$$\sum_{i} \sum_{j \neq i} x_{ij} d_{ij} = D.$$

Stochastic Modeling: Estimating Transition Probabilities

A problem that appears in the estimation of transition probabilities from macro data is the following:

Given the proportion of observations in n alternative states during T time periods, estimate the probability of transition between any two states at the next time period.

Problems of this sort typically appear in marketing research. Observations are available for the proportion of customers using brand i where $i = 1, 2, \ldots, n$ during the time periods $t = 1, 2, \ldots, T$ when a market survey was conducted. Assuming that transition probabilities are constant over the time interval of the survey, we want to estimate the underlying transition probability matrix. Let x_{it} be the probability of the ith brand during period t, and let a_{ij} be the probability of transition from i to j. Furthermore, let x^t be the (column) vector $x^t = (x_i^t)_{i=1}^n$ and let A be the matrix $\{a_{ij} \mid i, j = 1, 2 \ldots, n\}$. The probability vector at time $t+1$ can be obtained from that at time t by applying the transition probability matrix A as follows:

$$x^{t+1} = A^T x^t, \tag{6.7}$$

where A^T is A transpose. In practice, one is given the observations x^t, $t = 1, 2, \ldots, T$ and would like to compute the transition probability matrix A that satisfies the relation (6.7). There are $n(n-1)$ probabilities a_{ij} to be estimated and a solution exists if the number of observations is sufficiently small. In general $T > n$ and no solution exists. For any estimated matrix A we may expect at most T non-zero discrepancy vectors $x^{t+1} - A^T x^t$. A quadratic programming model for estimating the transition probability matrix A minimizes a semi-definite quadratic form of the discrepancy vectors:

$$\sum_{t=1}^{T} \langle (x^{t+1} - A^T x^t),\ V(x^{t+1} - A^T x^t) \rangle, \tag{6.8}$$

where V is an appropriate symmetric positive semi-definite matrix, such as, for example, an estimate of the inverse of the covariance matrix. In addition, we require that the sum of the entries of every row of the estimated matrix is equal to one (there is always a transition to some state) and that all entries are nonnegative. Thus, the matrix estimation problem that arises in this case can be modeled as the problem of minimizing (6.8) subject to the constraints

$$\sum_{j=1}^{n} a_{ij} = 1, \quad \text{for } i = 1, 2, \ldots, n, \quad \text{and} \tag{6.9}$$

$$a_{ij} \geq 0, \text{ for all } i, j. \tag{6.10}$$

The resulting problem is a reduced version of Problem 6.1.1 without constraints on the column sums. The estimated matrix A is not related to any prior estimate of the transition probabilities. Instead the model is minimizing a quadratic term of discrepancy vectors which are related to the computed matrix A.

A typical example is the following. Consider the market shares of new and used cars among car buyers in the United States over a period of five years. We want to determine the probability of a buyer of a used car switching over to a new car and vice versa. One is therefore estimating the entries of a matrix of transition probabilities, subject to the constraints that all entries are between 0 and 1, and that the sum of the probabilities over all states is equal to 1. Another example of estimating transition probabilities from a set of data on the market shares of three cigarette brands is discussed in Theil and Rey [221] and Theil [220].

6.1.2 Mathematical Models for Matrix Balancing

Many different mathematical formulations have appeared in the literature for solving matrix balancing problems. Some are equivalent to each other, while others are genuinely different, the differences being motivated by the real-world applications. In this section we review the diverse approaches to modeling matrix balancing problems and give some of the most popular mathematical formulations. First we describe the network structure of the two basic matrix balancing Problems 6.1.1 and 6.1.2. This graph structure is valuable for understanding both the mathematical structure of the models and the algorithms.

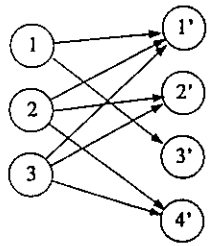
Network Structure of Matrix Balancing Problems

The connection to network models is established by associating with a matrix A a directed graph $G = (V, E)$ representing the sparsity pattern of A. For Problem 6.1.1 the natural graph to associate to A is a bipartite graph with vertex sets corresponding to the rows and columns of A.

Definition 6.1.1 Bipartite Graph of A.
For an $m \times n$ nonnegative matrix $A = (a_{ij})$, define the bipartite graph $G = (V, E)$ where

$$V \doteq \{i \mid i = 1, 2, \ldots, m\} \bigcup \{j \mid j = 1, 2, \ldots, n\}, \quad \textit{and}$$
$$E \doteq \{(i, j) \mid a_{ij} > 0\}.$$

V is the set of **vertices** *(or,* **nodes***) and E is the set of* **edges** *(or,* **arcs***) of the graph.* (See Figure 6.2)



$$A = \begin{bmatrix} 1 & 0 & 4 & 0 \\ 3 & 2 & 0 & 5 \\ 4 & 1 & 0 & 8 \end{bmatrix}$$

Figure 6.2: The transportation graph of A.

This particular graph, in which the vertices are divided into two subsets with all edges leading from nodes of one subset to those of the other, is commonly referred to as *bipartite*. It is possible to define an optimization problem on bipartite graphs, which is known as the *transportation problem* [94]. With each edge (i, j) we associate a variable x_{ij} that denotes flow from vertex i to vertex j and denote the cost of sending x_{ij} units along (i, j) by $f_{ij}(x_{ij})$. We define an optimization problem that minimizes the total cost $\sum_{(i,j)\in E} f_{ij}(x_{ij})$, subject to conditions on the conservation of total flow at each vertex:

$$\sum_{j=1}^{n} x_{ij} = u_i, \quad i = 1, 2, \ldots, m, \tag{6.11}$$

$$\sum_{i=1}^{m} x_{ij} = v_j, \quad j = 1, 2, \ldots, n, \tag{6.12}$$

$$x_{ij} \geq 0, \tag{6.13}$$

$$x_{ij} = 0 \text{ for all } a_{ij} = 0. \tag{6.14}$$

These are precisely the consistency conditions for Problem 6.1.1. The positive elements of A are viewed as flows on the edges of G. That is, the element $a_{ij} > 0$ is the flow on edge (i, j). Hence, the relation between Problem 6.1.1 and transportation problems in optimization is established.

The natural graph for Problem 6.1.2 is a *transshipment graph* with n vertices and an arc for every non-zero entry of A, defined next.

Definition 6.1.2 Transshipment Graph of A. *For an $n \times n$ nonnegative matrix A, define the transshipment graph $G = (V, E)$ where*

$$V \doteq \{1, 2, 3, \ldots, n\}, \quad \textit{and}$$
$$E \doteq \{(i, j) \mid a_{ij} > 0\}.$$

(See Figure 6.3)

Similarly to the transportation optimization problem, defined over a bipartite graph, we can define a transshipment optimization problem defined over a transshipment graph. In this graph model, the entries of A are flows on the edges of G. The conservation of flow conditions of the transshipment problem are precisely the consistency requirements for Problem 6.1.2.

Matrix Construction Formulations

Let $X = (x_{ij})$ be an $m \times n$ matrix and denote

$$E \doteq \{(i, j) \mid i = 1, 2, \ldots, m \text{ and } j = 1, 2, \ldots, n\}.$$

$$A = \begin{bmatrix} 0 & 4 & 3 & 0 \\ 2 & 0 & 1 & 3 \\ 0 & 0 & 0 & 6 \\ 5 & 1 & 0 & 0 \end{bmatrix}$$

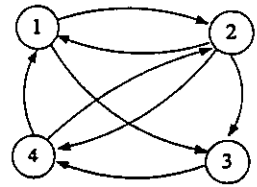


Figure 6.3: The transshipment graph of A.

An *integral of the matrix* X is a sum $\sum_{(i,j)\in T} x_{ij}$ where T is a given subset of E. The matrix balancing problem is to construct a matrix X such that some of its integrals will obey certain conditions and the matrix itself will be related in a specific way to another given matrix $A = (a_{ij})$.

We refer to such problems as *matrix construction* problems. Matrix balancing is a special case of matrix construction. Other, related, problems that can be viewed as special cases of matrix construction appeared in the literature under the terms *matrix scaling* (similarity scaling, equivalence scaling, truncated scaling) [11, 12, 13, 90, 208], *constrained matrix problems* [56], *generalized scaling* [202, 203, 204], and *fair-share matrix allocation* [15, 16].

Various conditions on the integrals of the matrix X have been considered in the aforementioned examples. These include (but are not restricted to): fixing the row sums and column sums to pre-assigned values; constraining

the individual entries of X to lie within specified lower and upper bounds; confining the row sums and column sums to lie within specified lower and upper bounds and fixing the sum of all the entries of X to a pre-assigned value; forcing row sums to be equal to column sums.

The precise relationship between the constructed matrix X and the given matrix A is what differentiates the various instances of matrix construction problems. These relationships are of three general types:

1. *The form relationship:* Here one imposes a certain *form-relation* that dictates the desired form of X. One such relation is to demand the existence of vectors $\lambda \in \Re^m$ and $\mu \in \Re^n$ and a real $\delta \in \Re$ such that for a particular $m \times n$ weight matrix W the form-relation

$$X = A + diag(\lambda)\, W + W\, diag(\mu) + \delta\, diag(\lambda)\, W\, diag(\mu) \qquad (6.15)$$

would hold. Here $diag(\lambda)$ is the diagonal matrix whose diagonal elements are the components of λ. This relation, proposed by Bachem and Korte [12], is the most general, and extends several special relations that were proposed in the past. With the choice $W = A$, $\delta = 1$ and the substitution $r = \lambda + \underline{1}$ and $s = \mu + \underline{1}$ (where $\underline{1}$ is a vector of all ones), relation (6.15) becomes

$$X = diag(r)\, A\, diag(s) \qquad (6.16)$$

which leads to the well-known and extensively studied *equivalence scaling* problem, see, e.g., Schneider [207].

Yet another form-relation between X and A is to demand the existence of a positive vector $z \in \Re^k$ (i.e., $z_j > 0$, $j = 1, 2, \ldots, k$) such that the

lexicographic vectorial reorderings $x \in \Re^{mn}$ and $a \in \Re^{mn}$ of X and A, respectively, are related by

$$x_q = a_q \left(\prod_{p=1}^{k} z_p^{\phi_{pq}} \right), \quad q = 1, 2, 3, \ldots, mn, \tag{6.17}$$

where $\Phi = (\phi_{pq})$ is a $k \times mn$ coefficients matrix of a system of linear equality or linear inequality integral constraints on X. See Darroch and Ratcliff [67] and Rothblum [202, 203]. For special cases of the coefficient matrix Φ this form relationship also leads to the equivalence scaling problem [202].

2. *The axiomatic approach:* This approach, taken recently by Balinski and Demange [15, 16], specifies a list of axioms that the relationship between X and A should satisfy and then establishes a rigid form-relation which is proven to fulfill the given axioms.

3. *Distance optimization:* In this approach the constructed matrix X should be as close as possible to the original matrix A, subject to the integral constraints. The notion "close" is defined by some distance function $f(X; A)$ which measures the "distance" between X and A. The choice $f(X; A) \doteq \| X - A \|_F^2$, where $\| \cdot \|_F$ denotes the Frobenius norm, leads to a linearly constrained quadratic optimization problem, Cottle et al. [56], Zenios et al. [239] and Schneider and Zenios [208]. Another commonly used objective is the negative entropy functional [208, 239]:

$$\sum_{(i,j) \in E} x_{ij} \left[\ln \left(\frac{x_{ij}}{a_{ij}} \right) - 1 \right]. \tag{6.18}$$

It is important to point out that connections exist between some of the seemingly different approaches to matrix construction, in particular between several form- relations and the entropy optimization problem. Entropy optimization formulations have also been justified in the literature as resulting in balanced matrices that are the least-biased, or maximally uncommitted, with respect to missing information from the original matrix A. Some axiomatic approaches also lead naturally to entropy optimization formulations. As a result, matrix balancing problems formulated as entropy optimization models over network flow constraints, are some of the most widely used in practice. We present several such formulations next.

Entropy Optimization Models for Matrix Balancing

We consider the entropy optimization formulations of Problems 6.1.1 and 6.1.2. That is, we consider the estimated, balanced, matrix X to be "nearby" the matrix A if the "entropic distance" between them is minimized, as defined below.

Problem 6.1.3 *Given an $m \times n$ nonnegative matrix $A = (a_{ij})$ and positive vectors $u = (u_i) \in \Re^m$ and $v = (v_j) \in \Re^n$ determine the matrix $X = (x_{ij})$ which solves the optimization problem:*

$$\underset{x}{\text{Minimize}} \quad \sum_{(i,j)\in E} x_{ij}\left[\ln\left(\frac{x_{ij}}{a_{ij}}\right) - 1\right] \tag{6.19}$$

$$\text{s.t.} \quad \sum_{(i,j)\in E} x_{ij} = u_i, \qquad \text{for } i = 1, 2, \ldots, m, \tag{6.20}$$

$$\sum_{(i,j)\in E} x_{ij} = v_j, \qquad \text{for } j = 1, 2, \ldots, n, \tag{6.21}$$

$$x_{ij} \geq 0, \qquad \text{for } (i,j) \in E, \tag{6.22}$$

and so that $a_{ij} = 0 \Rightarrow x_{ij} = 0$.

Problem 6.1.4 *Given an* $n \times n$ *nonnegative matrix* $A = (a_{ij})$, *determine the matrix* $X = (x_{ij})$ *which solves the optimization problem:*

$$\text{Minimize} \quad \sum_{(i,j)\in E} x_{ij}\left[\ln\left(\frac{x_{ij}}{a_{ij}}\right) - 1\right] \tag{6.23}$$

$$\text{s.t.} \quad \sum_{j=1}^{n} x_{ij} = \sum_{j=1}^{n} x_{ji}, \qquad \text{for } i = 1, 2, \ldots, n, \tag{6.24}$$

$$x_{ij} \geq 0, \qquad \text{for } (i,j) \in E, \tag{6.25}$$

and so that $a_{ij} = 0 \Rightarrow x_{ij} = 0$.

We consider now extensions of these problems to situations in which the given row-sums u_i and column-sums v_j are confined to an interval in Problem 6.1.3, or in which the strict agreement between row and column sums, in Problem 6.1.4, is relaxed to having their differences fall within a specified interval. This is known as the *interval-constrained matrix balancing* problem. Such formulations arise in practical applications when the prescribed row and column sums are unreliable or when they cannot be specified more precisely than being confined to certain intervals. Interval-constrained formulations were proposed, independently and in slightly different forms, by Balinksi and Demange [15, 16] and Censor and Zenios [51].

Problem 6.1.5 *Given an* $m \times n$ *nonnegative matrix* $A = (a_{ij})$, *nonnegative vectors* $\underline{u} = (\underline{u}_i)$, $\overline{u} = (\overline{u}_i) \in \Re^m$ *such that* $0 \leq \underline{u} \leq \overline{u}$, *and nonnegative vectors* $\underline{v} = (\underline{v}_j)$, $\overline{v} = (\overline{v}_j) \in \Re^n$ *such that* $0 \leq \underline{v} \leq \overline{v}$, *determine the matrix* $X = (x_{ij})$ *which solves the optimization problem:*

$$\textit{Minimize} \quad \sum_{(i,j)\in E} x_{ij}\left[\ln\left(\frac{x_{ij}}{a_{ij}}\right)-1\right] \tag{6.26}$$

$$s.t. \quad \underline{u}_i \le \sum_{(i,j)\in E} x_{ij} \le \overline{u}_i, \ \textit{for } i=1,2,\ldots,m, \tag{6.27}$$

$$\underline{v}_j \le \sum_{(i,j)\in E} x_{ij} \le \overline{v}_j, \ \textit{for } j=1,2,\ldots,n, \tag{6.28}$$

$$x_{ij} \ge 0, \quad \textit{for } (i,j)\in E, \tag{6.29}$$

and so that $a_{ij}=0 \Rightarrow x_{ij}=0$.

Problem 6.1.6 *Given an* $n\times n$ *nonnegative matrix* $A=(a_{ij})$, *and a positive "tolerance vector"* $\epsilon=(\epsilon_i)\in\Re^n$, $\epsilon>0$, *determine the matrix* $X=(x_{ij})$ *which solves the optimization problem:*

$$\textit{Minimize} \quad \sum_{(i,j)\in E} x_{ij}\left[\ln\left(\frac{x_{ij}}{a_{ij}}\right)-1\right] \tag{6.30}$$

$$s.t. \quad -\epsilon_i \le \sum_{j=1}^{n} x_{ij} - \sum_{j=1}^{n} x_{ji} \le \epsilon_i, \quad \textit{for } i=1,2,\ldots,n, \tag{6.31}$$

$$x_{ij} \ge 0, \quad \textit{for } (i,j)\in E, \tag{6.32}$$

and so that $a_{ij}=0 \Rightarrow x_{ij}=0$.

6.1.3 Iterative Algorithms for Matrix Balancing

The row-action iterative algorithms of Chapter 4 are now applied to the mathematical formulations of the matrix balancing problems. The entropy function is a Bregman function (see Lemma 2.2.3). It is optimized over equality constraints (for Problems 6.1.3 and 6.1.4), and over interval constraints (for Problems 6.1.5 and 6.1.6). We develop here the specific algorithms for the more general, interval-constrained, Problems 6.1.5 and 6.1.6.

Then we obtain as special cases the algorithms for the equality-constrained Problems 6.1.3 and 6.1.4.

The Range-RAS Algorithm (RRAS)

The Range-RAS (**RRAS**) algorithm is designed to solve Problem 6.1.5 with interval constraints on the row and column sums of X. It is an adaptation of the iterative row-action Algorithm 4.5.1 for interval convex programming. It is, therefore, a primal-dual algorithm with the elements of the sequence $\{x^k\}$ as primal variables and dual variables denoted by ρ_i^k, for $i = 1, 2, \ldots, m$, and σ_j^k, for $j = 1, 2, \ldots, n$. While the general algorithm is a dual ascent method which does not perform exact minimization in the dual problem, it turns out that, for the particular case of **RRAS**, the dual ascent is indeed exact. This is the case because the constraints matrix, denoted by Φ below, is a 0-1 matrix, i.e., all its entries are either zero or one.

It is useful to reformulate the constraints of Problem 6.1.5 by lexicographically rearranging the $m \times n$ matrix X into an mn-dimensional vector $x = (x_s), s = 1, 2, \ldots, mn$, where $s \doteq (i-1)m + j$. The constraints then take the form

$$\begin{pmatrix} \underline{u} \\ \underline{v} \end{pmatrix} \leq \Phi x \leq \begin{pmatrix} \overline{u} \\ \overline{v} \end{pmatrix}. \tag{6.33}$$

Let the $m \times mn$ matrix R be given by

$$R \doteq \left[\begin{array}{c|c|c|c} 1\,1\ldots1 & & & \\ \hline & 1\,1\ldots1 & & \\ \hline & & \ldots\ldots & \\ \hline & & & 1\,1\ldots1 \end{array}\right], \tag{6.34}$$

and the $n \times mn$ matrix C be given by

$$C \doteq \left[\begin{array}{ccc|ccc|c|ccc} 1 & & & 1 & & & & 1 & & \\ & 1 & & & 1 & & & & 1 & \\ & & \ddots & & & \ddots & & & & \ddots \\ & & & & & & \cdots & & & \\ & & 1 & & & 1 & & & & 1 \end{array}\right], \tag{6.35}$$

Both matrices are used to define the matrix Φ

$$\Phi \doteq \left[\begin{array}{c} R \\ C \end{array}\right]. \tag{6.36}$$

In Algorithm 4.5.1 we identify Φ as the constraint matrix and take the objective function f to be

$$f(x) = \sum_{s=1}^{mn} x_s \left[\ln\left(\frac{x_s}{a_s}\right) - 1\right]. \tag{6.37}$$

After some algebraic manipulations we obtain the following algorithm.

Algorithm 6.1.1 The Range-RAS (RRAS) Algorithm.

Input: An $m \times n$ nonnegative matrix $A = (a_{ij})$, positive vectors $\underline{u} = (\underline{u}_i), \overline{u} = (\overline{u}_i) \in \Re^m$ and positive vectors $\underline{v} = (\underline{v}_j), \overline{v} = (\overline{v}_j) \in \Re^n$.

Step 0: (Initialization) Set $k = 0$ and $x_{ij}^0 = a_{ij}$ for all $i = 1, 2, \ldots, m$ and $j = 1, 2, \ldots, n$. Set $\rho_i^0 = 1$, for $i = 1, 2, \ldots, m$ and $\sigma_j^0 = 1$, for $j = 1, 2, \ldots, n$.

Step 1: (Row Scaling) For $i = 1, 2, \ldots, m$ define

$$\underline{\rho}_i^k = \frac{\underline{u}_i}{\sum_j x_{ij}^k}, \tag{6.38}$$

$$\overline{\rho}_i^k = \frac{\overline{u}_i}{\sum_j x_{ij}^k}, \tag{6.39}$$

compute

$$\Delta\rho_i^k = mid\left(\rho_i^k, \underline{\rho}_i^k, \overline{\rho}_i^k\right) \tag{6.40}$$

and update:

$$x_{ij}^k \leftarrow x_{ij}^k \Delta\rho_i^k, \quad i = 1, 2, \ldots, n, \; j = 1, 2, \ldots, n, \tag{6.41}$$

$$\rho_i^{k+1} = \frac{\rho_i^k}{\Delta\rho_i^k}, \quad i = 1, 2, \ldots, m. \tag{6.42}$$

Step 2: (Column scaling) For $j = 1, 2, \ldots, n$ define

$$\underline{\sigma}_j^k = \frac{\underline{v}_j}{\sum_i x_{ij}^k}, \tag{6.43}$$

$$\overline{\sigma}_j^k = \frac{\overline{v}_j}{\sum_i x_{ij}^k}, \tag{6.44}$$

compute

$$\Delta\sigma_j^k = mid\left(\sigma_j^k, \underline{\sigma}_j^k, \overline{\sigma}_j^k\right) \tag{6.45}$$

and update

$$x_{ij}^{k+1} = x_{ij}^k \Delta\sigma_j^k, \quad i = 1, 2, \ldots, m, \; j = 1, 2, \ldots, n, \tag{6.46}$$

$$\sigma_j^{k+1} = \frac{\sigma_j^k}{\Delta\sigma_j^k}, \; j = 1, 2, \ldots, n. \tag{6.47}$$

Step 3: Replace $k \leftarrow k + 1$ and return to Step 1.

The RAS Scaling Algorithm

If the input in the **RRAS** algorithm is $\underline{u} = \overline{u}$ and $\underline{v} = \overline{v}$ then for all k, $\underline{\rho}_i^k = \overline{\rho}_i^k$ and $\underline{\sigma}_j^k = \overline{\sigma}_j^k$ will always be equal to $\Delta\rho_i^k$ and $\Delta\sigma_j^k$, respectively. Thus ρ_i^k and σ_j^k — which can be interpreted as dual variables — become irrelevant and the algorithm coincides with the classical **RAS** algorithm [145, 31, 10] for the fixed row-column Problem 6.1.1.

Algorithm 6.1.2 The RAS Algorithm.

Input: An $m \times n$ nonnegative matrix $A = (a_{ij})$, a positive vector $u = (u_i) \in \Re^m$ and a positive vector $v = (v_j) \in \Re^n$.

Step 0: (Initialization) Set $k = 0$ and $x_{ij}^0 = a_{ij}$, for all $i = 1, 2, \ldots, m$, and $j = 1, 2, \ldots, n$.

Step 1: (Row Scaling) For $i = 1, 2, \ldots, m$ define

$$\rho_i^k = \frac{u_i}{\sum\limits_j x_{ij}^k}, \tag{6.48}$$

and update:

$$x_{ij}^k \leftarrow x_{ij}^k \rho_i^k, \quad i = 1, 2, \ldots, m, \quad j = 1, 2, \ldots, n, \tag{6.49}$$

Step 2: (Column scaling) For $j = 1, 2, \ldots, n$ define

$$\sigma_j^k = \frac{v_j}{\sum\limits_i x_{ij}^k} \tag{6.50}$$

and update

$$x_{ij}^{k+1} = x_{ij}^k \sigma_j^k, \quad i = 1, 2, \ldots, m, \quad j = 1, 2, \ldots, n. \tag{6.51}$$

Step 3: Replace $k \leftarrow k+1$ and return to Step 1.

Within the graph model of Problem 6.1.3, the **RAS** algorithm cycles through the vertices on each side of G and multiplies the flows on all edges incident to a vertex by the positive constant necessary to force the sum of the flows equal to the desired total.

The Range-DSS Algorithm (RDSS)

The Range-DSS (**RDSS**) algorithm solves Problem 6.1.6. Here we present the details of its derivation from Algorithm 4.5.1. The interval constraints of Problem 6.1.6 may be rewritten as

$$-\epsilon \leq (R-C)x \leq \epsilon, \tag{6.52}$$

where $R-C$ is the $n \times n^2$ difference matrix of R and C given by (6.34) and (6.35), i.e.,

$R-C \doteq$

$$\left[\begin{array}{ccccc|c|c|ccccc}
0 & 1 & 1 & \dots & 1 & -1 & & & -1 & & & \\
-1 & & & & & 1 & 0 \dots 1 & & & -1 & & \\
 & -1 & & & & & -1 & & & & -1 & \\
 & & \ddots & & & & & & & & & \ddots \\
 & & & \ddots & & & & \dots & & & & \\
 & & & & -1 & & -1 & & 1 & 1 & 1 \dots & 0
\end{array}\right].$$

Let us denote

$$R-C = \Psi = (\psi_{is}), \tag{6.53}$$

and let $\psi^i = (\psi_s^i) = (\psi_{is})$, $s = 1, 2, \ldots, n^2$, be the $i-th$ column of Ψ^T, the transpose of Ψ. Then the iterative step derived from Algorithm 4.5.1 takes the form

$$x_s^{k+1} = x_s^k \exp\left(c_k \psi_s^{i(k)}\right), \; s = 1, 2, \ldots, n^2, \tag{6.54}$$

$$z_t^{k+1} = \begin{cases} z_t^k, & \text{if } t \neq i(k), \\ z_t^k - c_k, & \text{if } t = i(k). \end{cases} \tag{6.55}$$

Here $\{i(k)\}$ is the control sequence according to which the rows of Ψ are chosen. We assume the cyclic control $i(k) = k(mod\, n) + 1$ and henceforth abbreviate $i \equiv i(k)$. $\{z^k\}$ is a sequence of dual vectors.

In this iteration the parameter c_k is

$$c_k = mid\,(z_i^k, \underline{d}^k, \overline{d}^k), \tag{6.56}$$

where $\underline{d}^k$ and $\overline{d}^k$ are determined by solving, for $i = i(k)$, the systems:

$$y_s = x_s^k \exp\left(\underline{d}_k \psi_{is}\right), \; s = 1, 2, \ldots, n^2, \tag{6.57}$$

$$\sum_{s=1}^{n^2} y_s \psi_{is} = -\epsilon_i, \tag{6.58}$$

and

$$y_s = x_s^k \exp\left(\overline{d}_k \psi_{is}\right), \; s = 1, 2, \ldots, n^2, \tag{6.59}$$

$$\sum_{s=1}^{n^2} y_s \psi_{is} = \epsilon_i. \tag{6.60}$$

Substituting (6.57) into (6.58), denoting $\exp \underline{d}_k \doteq \underline{\alpha}_k$ and using the fact that ψ_{is} takes only the values 0, +1, and −1, we get

$$p\underline{\alpha}_k - q\frac{1}{\underline{\alpha}_k} = -\epsilon_i, \tag{6.61}$$

where

$$p \doteq \sum_{\{s|\psi_{is}=1\}} x_s^k\,, \quad q \doteq \sum_{\{s|\psi_{is}=-1\}} x_s^k. \tag{6.62}$$

Similarly, for $\overline{\alpha}_k \doteq \exp \overline{d}_k$, we obtain from (6.59) and (6.60),

$$p\overline{\alpha}_k - q\frac{1}{\overline{\alpha}_k} = \epsilon_i. \tag{6.63}$$

Taking the nonnegative solutions of the quadratic equations (6.61) and (6.63) we obtain the values of $\underline{d}_k$ and $\overline{d}_k$, respectively, which go into the *mid* operator (6.56) to obtain c_k. Using c_k in (6.54)–(6.55) completes the iterative step. The RDSS algorithm is thus formulated as follows.

Algorithm 6.1.3 The Range-DSS (RDSS) Algorithm.

Input: An $n \times n$ nonnegative matrix $A = (a_{ij})$, a positive vector $\epsilon = (\epsilon_i) \in \Re^n$.

Step 0: (Initialization) Set $k = 0$ and $x_{ij}^0 = a_{ij}$ for all $i, j = 1, 2, \ldots, n$. Set $\rho_i^0 = 1$, for $i = 1, 2, \ldots, n$.

Step 1: (Computation of Scaling Parameters) Choose a control index from a cyclic control sequence $\{i(k)\}$ with $i(k) = k(mod\, n)+1$, and calculate the sums

$$p_k \doteq \sum_{j=1}^{n} x_{i(k)j}^k\,, \quad q_k \doteq \sum_{j=1}^{n} x_{ji(k)}^k. \tag{6.64}$$

Compute $\underline{\alpha}_k$ and $\overline{\alpha}_k$ as the nonnegative roots of (6.61) and (6.63), respectively, with $p = p_k$, $q = q_k$, and $i = i(k)$.

Step 2: (Update) Calculate

$$\Delta\rho^k_{i(k)} = mid\left\{\rho^k_{i(k)}, \underline{\alpha}_k, \overline{\alpha}_k\right\} \tag{6.65}$$

and update:

$$x^{k+1}_{ij} = \begin{cases} x^k_{ij}\Delta\rho^k_{i(k)}, & if\ i = i(k),\ j \neq i(k), \\ x^k_{ij}/\Delta\rho^k_{i(k)}, & if\ i \neq i(k),\ j = i(k), \\ x^k_{ij}, & \text{otherwise}, \end{cases} \tag{6.66}$$

$$\rho^{k+1}_i = \begin{cases} \rho^k_i & if\ i \neq i(k), \\ \rho^k_{i(k)}/\Delta\rho^k_{i(k)}, & if\ i = i(k). \end{cases} \tag{6.67}$$

Step 3: Replace $k \leftarrow k+1$ and return to Step 1.

The DSS Scaling Algorithm

In the special case when $\epsilon = 0$ we obtain the **DSS** algorithm that solves Problem 6.1.4. In this case $\underline{\alpha}_k = \overline{\alpha}_k = \alpha_k$ and, therefore, $\underline{d}_k = \overline{d}_k = d_k = c_k$. Since $\alpha_k = \exp c_k$ the iteration (6.54) becomes

$$x^{k+1}_s = x^k_s(\alpha_k)^{\psi_{is}}.$$

Since then also $\alpha_k = \sqrt{\frac{q}{p}}$, the DSS algorithm can be written as follows.

Algorithm 6.1.4 The DSS Algorithm.

Input: An $n \times n$ nonnegative matrix $A = (a_{ij})$.

Step 0: (Initialization) Set $k = 0$ and $x^0_{ij} = a_{ij}$, for all $i, j = 1, 2, \ldots, n$.

Step 1: (Computation of Scaling Parameters) Choose a control index from a cyclic control sequence $\{i(k)\}$ with $i(k) = k(mod\, n)+1$, and calculate the sums

$$p_k \doteq \sum_{j=1}^{n} x^k_{i(k)j}\ ,\quad q_k \doteq \sum_{j=1}^{n} x^k_{ji(k)}. \tag{6.68}$$

Compute

$$\alpha_k = \sqrt{\frac{p_k}{q_k}}. \tag{6.69}$$

Step 2: (Update) Update:

$$x^{k+1}_{ij} = \begin{cases} x^k_{ij}\alpha_k, & if\ i = i(k),\ j \neq i(k), \\ x^k_{ij}/\alpha_k, & if\ i \neq i(k),\ j = i(k), \\ x^k_{ij}, & \text{otherwise}, \end{cases} \tag{6.70}$$

Step 3: Replace $k \leftarrow k + 1$ and return to Step 1.

In the graph model, the **DSS** algorithm scans the vertices of G and selects a vertex that violates the flow conservation conditions. Then the flows on edges directed into and out of that vertex are scaled so that conservation of flow at that vertex is satisfied.

6.2 Image Reconstruction from Projections

Many significant problems in diverse fields of applications in science and technology are inversion problems. An object — described by a vector x — is related to some data b through a relation $\mathcal{O}x = b$, and the recovery of x requires essentially the inversion, in some well-specified sense, of the

operator $\mathcal{O}$. The inversion problems discussed in this section are those of *image reconstruction from projections.*

Image reconstruction problems differ widely depending on the area of application in which they are formulated. Even within one field, such as medical, industrial, etc., very different reconstruction problems are encountered because of the physically different ways the data may be collected. In spite of such differences there is a common mathematical nature to reconstruction problems which is the following. There is an unknown (two- or three-dimensional) distribution of some physical parameter. This parameter could be, for example, the X-ray linear attenuation coefficients of human tissue, or the attenuation coefficients of the material in a nuclear reactor, or the density of electrons in the sun's corona. A finite number of line integrals of this parameter can be estimated from physical measurements, and an estimate of the distribution of the original parameters is desired. In the X-ray Transmission Computerized Tomography (CT) case the total attenuation of the X-ray beam between a source and a detector is approximately the integral of the linear attenuation coefficient along the line between the source and the detector.

Since a line in two- (three-) dimensional space can be parametrized by two (four) independent parameters, the process of associating with a function of two (three) variables the set of all its line integrals is a transformation from a two- (three-) dimensional space into a two- (four-) dimensional space. There is an obvious mathematical generalization, to arbitrary dimensions, of this transformation which has been referred to as the *X-ray transform.* For the two-dimensional case it is the same as the, now classical, transform introduced by Radon.

In this terminology, the aim of image reconstruction from projections can be reformulated as follows:

> Estimate a function from approximate values of its X-ray transform at a finite number of points.

6.2.1 Applications of Image Reconstruction

Problems of image reconstruction appear in numerous and diverse fields of applications. The mode of data-collection, the function to be reconstructed, the dimension of the function and so on may differ widely depending on the application.

Due to the vastness of the field and the desire to keep these Lecture Notes brief, we give here just a short bibliography. The list is selective, rather then exhaustive and is mainly aimed to give the reader some useful directions.

The basic, classical, treatise on image reconstruction is Herman's book [110]. Books with different, more specialized, emphases are written by Natterer [173], by Bates and McDonnel [18], by Deans [68] and by Kak and Slaney [139]. From the many specialized issues of Journals that were devoted to this field over the recent years we mention the following few: a 1983 special issue of the *Proceedings of the IEEE* [111], a 1990 special issue of *Linear Algebra and Its Applications* [44] and the recent special issue of *Physics in Medicine and Biology* [1].

More specialized books in the field are Hurt's [127] book on phase retrieval and zero crossings, the book by Craig and Brown [57] on inverse problems in astronomy, and the books edited by Udupa and Herman [223], by Nolet [181] and by Stark [214]. Recent conference proceedings include [24,

104, 120, 119] and many others. Review and tutorial papers were published in a steady flow, see, e.g., [103, 113, 39, 40, 41, 187, 46] and the recent report by Roerdink [200]. Finally we mention also a recent thesis on computed tomography for nondestructive material evaluation [9].

6.2.2 Mathematical Models for Image Reconstruction

There are two distinctly different approaches to the development of mathematical models for image reconstruction. One methodological path is to formulate a continuous model and use mathematical tools for the analytic inversion of the model operator $\mathcal{O}$. The formulae of the inverse operator are then "discretized" for computer implementation. In contrast to this, the so-called *finite series-expansion* approach to the problem, starts off with a discretized model. This approach leads to a linear or nonlinear system of equations or inequalities, or an optimization problem, in the n-dimensional Euclidean space $\Re^n$. After a solution concept is chosen an iterative algorithm is usually used and when iterations are stopped the current iterate $x^k \in \Re^n$, for some k, is taken as the reconstructed image.

The basic model in the series-expansion approach to the image reconstruction problem of X-ray transmission is formulated in the following way: A Cartesian grid of square picture elements, called pixels, is introduced into the region of interest so that it covers the whole picture that has to be reconstructed. The pixels are numbered in some agreed manner, say from 1 (top left corner pixel) to n (bottom right corner pixel) (see Fig. 6.4). The X-ray attenuation function is assumed to take a constant value x_j throughout the jth pixel, for $j = 1, 2, \ldots, n$. Sources and detectors are assumed to be points and the rays between them are assumed to be lines. Further, assume that

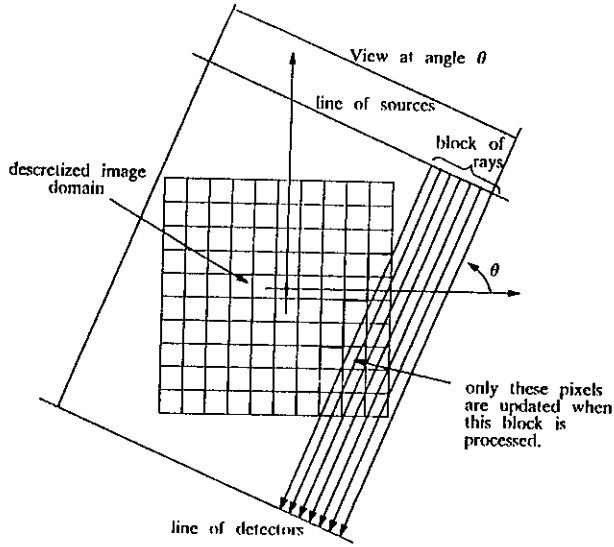


Figure 6.4: Fully discretized model for transmission image reconstruction.

the length of intersection of the ith ray with the jth pixel, denoted by a_{ij}, for all $i = 1, 2, \ldots, m$, $j = 1, 2, \ldots, n$, represents the weight of the contribution of the jth pixel to the total attenuation along the ith ray.

The physical measurement of the total attenuation along the ith ray, denoted by b_i, represents the line integral of the unknown attenuation function along the path of the ray. Therefore, in this discretized model, the line integral turns out to be a finite sum and the whole model is described by a system of linear equations. This is the *feasibility* model for image reconstruction.

Problem 6.2.1 *Given a measurement vector $b \in \Re^m$, and a nonnegative $m \times n$ matrix $A = (a_{ij})$, determine the vector $x \in \Re^n$ which solves the system of equations:*

$$\sum_{j=1}^{n} x_j a_{ij} \simeq b_i, \qquad i = 1, 2, \ldots, m. \tag{6.71}$$

If the system of equations is inconsistent — something which occurs in practice due to measurement errors or missing information — one needs to make additional modeling assumptions. Occasionally, the choice of an algorithm for solving the problem will be determined by the performance of the algorithm when applied to inconsistent systems. For example, some algorithms can be shown to converge to an accumulation point that can be characterized as a "solution" to the feasibility problem.

If the system of equations is underdetermined — a common occurrence in practical applications when only a few observations can be made — one may use an optimization criterion to select the most "suitable" solution. In this respect, the minimization of a negative entropy functional has received widespread attention from users. The image reconstruction problem can then be formulated as the following entropy optimization model.

Problem 6.2.2 *Given a measurement vector $b \in \Re^m$, and a nonnegative $m \times n$ matrix $A = (a_{ij})$, determine the vector $x \in \Re^n$ which solves the optimization problem*

$$\underset{x \in \Re^n}{\textit{Minimize}} \quad \sum_{j=1}^{n} x_j \ln x_j \tag{6.72}$$

$$s.t. \qquad \sum_{j=1}^{n} x_j a_{ij} \simeq b_i, \qquad i = 1, 2, \ldots, m. \tag{6.73}$$

It is worth pointing out the similarity between this entropy optimization model for image reconstruction, and the model for matrix balancing, Problem 6.1.3. In particular, the matrix balancing problem can be viewed as a

problem of image reconstruction in which measurements are taken from only two, orthogonal, projections.

6.2.3 Algorithms for Image Reconstruction

We present now algorithms for both the feasibility and the entropy optimization models for image reconstruction, i.e. Problems 6.2.1 and 6.2.2 respectively. The algorithms are based on the material of Chapters 3 and 4, and they are specialized for the structure of the image reconstruction models.

Algebraic Reconstruction Technique (ART) for Systems of Equations

This section reviews the *variable-block Algebraic Reconstruction Technique* (**ART**) algorithmic scheme for solving systems of linear equations that arise from the fully discretized model of transmission tomography. Mathematically speaking, this scheme is a special instance of the more general *Block-iterative Projections* (**BIP**) method of Section 3.3. However, the variable-block **ART** scheme deserves to be looked at separately because of its importance to image reconstruction from projections. It includes the classical row-action **ART**, the fixed-block **ART** (i.e., Block Kaczmarz algorithms), as well as the Cimmino version of **SIRT** (Simultaneous Iterative Reconstruction Technique), as special cases. It serves as a unifying framework for the description of all these popular iterative reconstruction techniques. The novelty added to all those by the variable-block ART scheme lies in the fact that it allows the processing of blocks (i.e., groups of equations) which need not be fixed in advance, but may rather change dynamically throughout it-

erations. The number of blocks, their sizes and the assignment of equations to blocks may all vary, provided that the weights attached to the equations do not vanish.

The behavior of iterative reconstruction algorithms when the underlying system of equations is inconsistent is an interesting question because noise and other inaccuracies in data would usually make any consistency assumption unrealistic. Results related to the behavior of **ART**, fixed-block **ART** and **SIRT** when applied to an inconsistent system are available in the literature, see, e.g., [43, 82, 107, 217, 218]. This question is still open, in general, for the variable-block **ART**.

Another important issue is the implementation and practical performance of block iterative reconstruction algorithms with fixed or variable blocks. In addition to studying the algorithms in terms of quality of reconstructed images there is a potential of using block-iterative algorithms on parallel machines. We briefly survey some recent results on these matters.

Consider a system of linear equations, obtained from the fully discretized model for transmission tomography image reconstruction, which has the form (6.71). Unless otherwise stated we assumed that the system is consistent, i.e., that, given A and b, the set $\{x \in \Re^n \mid Ax = b\}$ is nonempty. Algorithmic behavior for inconsistent systems is discussed separately below.

Let $w = (w(i)) \in \Re^m$ be a *weight vector* with $w(i) \geq 0$ for all i and $\sum_{i=1}^m w(i) = 1$. Recall (Section 3.3) that a sequence $\{w^k\}_{k=0}^\infty$ of weight vectors is called *fair* if, for every i, there exist infinitely many values of k for which $w^k(i) > 0$. This guarantees that no equation is zero-weighted indefinitely. To prevent equations from fading away by positive but steadily

diminishing weights we require that, for every i, the stronger condition

$$\sum_{k=0}^{\infty} w^k(i) = +\infty, \tag{6.74}$$

holds.

The sequence of weight vectors $\{w^k\}$ actually determines which block is used at each iteration by attaching positive weights to equations that belong to the current block and zero weights to the rest.

Algorithm 6.2.1 Variable-block ART — The General Scheme.

Initialization: $x^0 \in \Re^n$ is arbitrary.

Iterative Step:

$$x^{k+1} = x^k + \lambda_k \sum_{i=1}^{m} w^k(i) \left(\frac{y_i - \langle a^i, x^k \rangle}{\| a^i \|^2} \right) a^i. \tag{6.75}$$

Here $\{\lambda_k\}_{k=0}^{\infty}$ is a sequence of *relaxation parameters* chosen by the user.

As a direct corollary of Theorem 3.3.1 we obtain

Theorem 6.2.1 *If the system of equations in 6.71 is consistent and* $\{w^k\}$ *are weight vectors with property (6.2.3), and if the relaxation parameters* $\{\lambda_k\}$ *are such that* $\tau_1 \leq \lambda_k \leq 2 - \tau_2$, *for all* $k \geq 0$, *with* $\tau_1, \tau_2 > 0$, *then any sequence* $\{x^k\}$ *generated by Algorithm 6.2.1. converges to a solution of Problem 6.2.1.*

The variable-block **ART** algorithm allows processing of the information contained in groups of equations, called *blocks.* The number of blocks, their

sizes (i.e., the number of equations in each block) and their specific structure (i.e., which equations are assigned to each block), may all vary from one iterative step to the next. The following special cases of Algorithm 6.2.1 have been studied separately in the past.

Row-action ART. This classical sequential **ART** is obtained by choosing the weight vectors as $w^k = e^{i(k)}$ where $e^t \in \Re^n$ is the t-th standard basis vector (having one in its t-th coordinate and zeros elsewhere). Each block contains a single equation and the index sequence $\{i(k)\}$ with $i \leq i(k) \leq m$, for all k, is the *control sequence* of the algorithm which determines the index of the single equation upon which the algorithm operates at the k-th iterative step. The iterative steps take obviously the form

$$x_j^{k+1} = x_j^k + \lambda_k \frac{b_{i(k)} - \langle a^{i(k)}, x^k \rangle}{\| a^{i(k)} \|^2} a_j^{i(k)}, \text{ for all } j = 1, 2, \ldots, n. \quad (6.76)$$

Cimmino-type SIRT. In this fully simultaneous reconstruction algorithm all equations are lumped, with fixed weights, into a single block and are acted upon simultaneously in every step. The iteration formula is precisely (6.75) with $w^k = w$, for all $k \geq 0$, and $w(i) \neq 0$, for all $i = 1, 2, \ldots, m$. See Gilbert [101], Lakshminarayanan [146] and van der Sluis and van der Vorst [224].

Fixed Block ART. Here the index set $I = \{1, 2, \ldots, m\}$ is partitioned as $I = I_1 \cup I_2 \cup \cdots \cup I_M$ into M blocks. $\{t(k)\}$ is a control sequence over the set $\{1, 2, \ldots, M\}$ of block indices and the weight vectors are of the form $w^k = \sum_{i \in I_{t(k)}} w^k(i) e^i$. This guarantees that equations outside the $t(k)$-th block are not operated upon in the k-th iterative step. The block-Kaczmarz procedure of [83] is thus obtained.

In addition to these well-known special cases the variable-block **ART** enables the implementation of dynamic **ART** algorithms in which the block formation strategy might vary during the iterative process itself. Such variations in the block formation strategy may be used to accelerate the initial convergence towards an acceptable reconstructed image. It is possible under the regime of the variable-block **ART** to perform *multilevel image reconstruction*. By multilevel approaches we mean methods in which we repeatedly change during the iterative process the system of equations to be solved and/or the way the system is organized into blocks.

Because of their computational appeal (simplicity, row-action nature, etc.) and their efficacious performance in some specific situations, **ART** methods have been applied to inconsistent systems of equations as well. The surprisingly good results prompted studies of the behavior of the algorithms when applied to such systems. For example, row-action Kaczmarz's is known to be "cyclically convergent". This implies the convergence of the subsequences of iterates lying on each of the hyperplanes of the system. This result also holds for the fixed block Kaczmarz method. The fully simultaneous Cimmino method has been shown to converge locally to a weighted least squares solution if the system is inconsistent, see, e.g., Iusem and De Pierro [133]. We are not aware of any study that would in any way unify these results by making a statement about the behavior of the variable block **ART** method for inconsistent systems.

Multiplicative Reconstruction Technique (MART) for Entropy Optimization

A special-purpose block-iterative algorithm for entropy maximization used for image reconstruction is the *block Multiplicative Algebraic Reconstruction Technique* (block-**MART**). This algorithm is a block-variant of the **MART** algorithm of Gordon et al. [102], which in turn can be obtained as a special case of Algorithm 4.7.2. The block-MART algorithm, and its convergence analysis, are given in Censor and Segman [50], see also the recent work of Byrne [35].

The set of row indices of the constraints matrix A is first partitioned into

$$\{1, 2, \ldots, I\} = I_1 \bigcup I_2 \bigcup \ldots \bigcup I_M. \tag{6.77}$$

For each t, $1 \leq t \leq M$, I_t is the index set of a block of constraints. To each block I_t, $t = 1, 2, \ldots, M$, a fixed system of weights is assigned by defining

$$0 < w_i^t < 1, \text{ for all } i \in I_t, \text{ such that } \sum_{i \in I_t} w_i^t = 1. \tag{6.78}$$

If, however, a block contains a single equation then $w_i^t = 1$. The block-iterative algorithm for entropy maximization over linear equality constraints can be stated as follows:

Algorithm 6.2.2 Block-MART.

Initialization: Choose an arbitrary $u^0 \in \Re^m$ and compute $x^0 \in \Re^n$ such that $1 - \log x_j^0 = -\left(A^T u^0\right)_j$, $j = 1, 2, \ldots, n$, where A^T is the transpose of A.

Iterative step:

$$x_j^{k+1} = x_j^k \prod_{i \in I_{t(k)}} \exp\left[w_i^{t(k)} d_i^k a_j^i\right], \quad j = 1, 2, \ldots, n, \tag{6.79}$$

where

$$d_i^k \doteq \log\left(\frac{b_i}{\langle a^i, x^k \rangle}\right), \tag{6.80}$$

The choice of blocks is governed by a cyclic (or almost cyclic) control sequence $\{t(k)\}$:

$$t(k) = k\,(mod M) + 1. \tag{6.81}$$

The iterative step of the algorithm can also be written in the form:

$$x_j^{k+1} = x_j^k \exp\left[\sum_{i \in I_{t(k)}} w_i^{t(k)} d_i^k a_j^i\right]. \tag{6.82}$$

This form is more suitable for computer implementations, both for efficiency and stability.

The block-iteration has the functional form (see Chapter 1):

$$x^{k+1} = B_{t(k)}\left[x^k, \{f_i\}_{i \in I_{t(k)}}\right] \tag{6.83}$$

where $B_{t(k)}$ is an algorithmic operator which uses the current iterate x^k and the information in all rows in the $t(k) - th$ block of constraints. The information in row i is denoted by f_i. In the case $M = I$ the block-iterative scheme is a row-action one and block-**MART** coincides with **MART** [102, 149, 151, 46, 50, 238]. The case M=1 is a fully simultaneous version in which

all constraints are lumped into a single block. This case can be interpreted as a *parallel block-iterative algorithm* which enables several processors to perform in parallel the operations

$$x^{k+1,t} = B_t\left[x^k, \{f_i\}_{i\in I_t}\right], \quad t = 1, 2, \ldots, M, \tag{6.84}$$

and then let

$$x^{k+1} = S\left[\left\{x^{k+1,t}\right\}_{t=1}^{M}\right] \tag{6.85}$$

where the operator S is defined via $\prod_{t=1}^{M}$. While being mathematically equivalent here to the fully simultaneous case it is, of course, different from the implementation point of view and it is suitable for parallel implementation.

6.3 Planning Under Uncertainty: Stochastic Network Optimization

It has long been recognized (Dantzig [63]) that traditional deterministic mathematical programs are not suitable for capturing the truly dynamic behavior of many real-world applications. A primary reason is that such applications involve data uncertainties. Data uncertainties arise in a dynamic setting because information which will be needed in subsequent decision stages is not yet available to the decision maker or modeler. In applications from financial planning such information would be future asset prices and returns. Applications from logistics planning and utility service scheduling exhibit uncertain supplies or demands, as well as uncertain transportation or construction costs. Stochastic programming was first proposed (independently by Dantzig [63] and Beale [20]) as a way to overcome the problem of uncertain problem data.

Although there has been significant progress in the ability of researchers to solve stochastic programs, these programs often turn out to be very complex for practical applications. Their solution on the computer often proves prohibitively expensive. Solving deterministic "mean value" or "worst case" approximations may lead to solutions which are far from optimal, Birge [27]. Reformulating the stochastic program into a large-scale deterministic equivalent, taking uncertainties explicitly into account, is often possible. However, the deterministic program may again be very expensive to solve. This is not only because of sheer size, but also because any special structure exhibited by the stochastic program is largely lost in the deterministic equivalent.

In this section we consider the application of the row-action algorithms of Chapter 4 to the development of an algorithm for strictly convex stochastic network problems. (A survey of models and solution techniques for nonlinear network optimization problems is given by Dembo, Mulvey and Zenios [70].) We employ a decomposition technique for two-stage stochastic network problems by *splitting* (or replicating) first-stage decision variables as suggested in Rockafellar and Wets [199]. This reformulation preserves the network structure. The subsequent application of a row-action algorithm results in further decomposition that is suitable for parallel computing.

A limitation of the algorithm is that it is applicable only to problems with strictly convex objective functions, like the quadratic form we use in this section, or entropy functions of the form $\sum_j x_j \log x_j$. The linear stochastic program cannot be solved directly with this algorithm. Nevertheless, row-action algorithms can be used as building blocks in the proximal minimization algorithms discussed in Chapter 5. Our discussion in the present chapter is restricted to the nonlinear program. The solution of linear stochastic pro-

grams — using the row-action algorithms of this chapter together with the proximal minimization algorithms from Chapter 5 — is discussed by Nielsen and Zenios [178, 179, 180].

There is a vast literature on the theory, algorithms and applications of stochastic programming. We cite here some of the relevant articles that develop solution algorithms for large-scale stochastic optimization models, such as the stochastic network programs we study here. Research over the last twenty years has largely concentrated on (i) devising efficient decomposition methods for solving the deterministic equivalent program (Dantzig, Dempster and Kallio [64] Van Slyke and Wets [212, 213], Ruszczynski , Infanger [130]), (ii) designing or exploiting parallel computing machinery to speed up the solution process (Wets [227], Dantzig [65], Mulvey and Ruszczynski [167], Qi and Zenios [193]), and (iii) on retaining any special structure present in the stochastic program, (Qi [192], Wallace [226], Mulvey and Vladimirou [170, 168], Nielsen and Zenios [177, 179]).

Recently, Rockafellar and Wets [199] introduced the *progressive hedging* algorithm, a decomposition algorithm based on augmented Lagrangean theory. This algorithm retains any structure present in the sub-problems, and is also suitable for implementation on parallel computer architectures. Special structure in the sub-problems is preserved by *splitting* (i.e., replicating) first-stage variables and then using an augmented Lagrangean penalty to ensure that the replicated variables converge to a common value. Progressive hedging was used by Mulvey and Vladimirou [168] for the solution of stochastic programs with network recourse. The split-variable formulation was also used by Lustig et al. [152] in order to exploit the sparsity structure of stochastic programs when using interior point methods. It was also used

by Nielsen and Zenios [177] to exploit the network structure in the design of massively parallel algorithms.

There has also been extensive interest in the use of parallel computing techniques for solving stochastic programs. Wets [227] proposed the use of a large number of very simple processors to solve (multistage) stochastic programs. Ariyawansa and Hudson [7] implement and test on a multiprocessor a version of Van Slyke and Wets' [213] algorithm. Dantzig and Glynn [66] demonstrate that nested decomposition, Monte Carlo importance sampling and parallel programming can be combined to solve a class of multistage stochastic programs.

6.3.1 Problem Formulation

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

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

This framework (and the algorithmic approach we introduce next) can be generalized to more than two stages. We concentrate here exclusively on the two-stage model. Extensions to multi-stage models are given in Nielsen and Zenios [179].

The Two-stage Stochastic Program

To formulate the two-stage stochastic program we need two sets of decision variables:

$x \in \Re^{n_1}$ denotes the vector of *first-stage* decisions. These decisions are made before the future events are observed.

$y \in \Re^{n_2}$ denotes the vector of *second-stage* decisions. These decisions are made after some realization of the uncertain future has been observed. They are unavoidably constrained by the first-stage decisions, and depend on the realization of the uncertain future data.

Let us first formulate the second-stage problem: Once a first-stage decision x has been made, some realization of the uncertain future has been observed. Let bold-face letters $\mathbf{g}$, $\mathbf{r}$, $\mathbf{B}$, $\mathbf{v}$, $\mathbf{C}$ denote the stochastic quantities that describe the uncertain future, and let the corresponding roman-face letters denote a specific realization of these stochastic quantities. The uncertainties of the second-stage problem are represented by a (possibly) stochastic objective function $\mathbf{g} : \Re^{n_2} \mapsto \Re$, the (possibly) stochastic $m_2 \times n_2$ real constraint matrix $\mathbf{B}$, the (possibly) stochastic resource vector, $\mathbf{r} \in \Re^{m_2}$, the vector $\mathbf{v} \in \Re^{n_1}$ of (possibly) stochastic upper bounds on the second-stage variables, and the (possibly) stochastic $m_2 \times n_1$ matrix $\mathbf{C}$ which communicates information about the impact of the first-stage decision to the second-stage problem.

The *second-stage problem* is then to find an y that optimizes the cost of the second-stage decision, for a given value of the first-stage decision x. We denote the value of the second-stage problem by $Q(g, r, B, v, C \mid x)$; its arguments denote its dependence on the realization of the stochastic

quantities, and the fact that it is conditioned on the value of the first-stage variables x. $Q(\cdot)$ is obtained as the optimal value to the following nonlinear program:

$$\underset{y \in \Re^{n_2}}{\text{Minimize}} \quad g(y) \tag{6.86}$$

$$\text{s.t.} \quad By = r - Cx, \tag{6.87}$$

$$0 \leq y \leq v. \tag{6.88}$$

If this second-stage minimization problem is infeasible, we understand its value to be ∞. Let now

$$\mathcal{Q}(x) = \mathcal{E}\{Q(\mathbf{g}, \mathbf{r}, \mathbf{B}, \mathbf{v}, \mathbf{C} \mid x)\} \tag{6.89}$$

denote the *expected value* of the second-stage optimization problem, where expectation is computed over possible realizations of the stochastic quantities $\mathbf{g}$, $\mathbf{r}$, $\mathbf{B}$, $\mathbf{v}$, $\mathbf{C}$. The two-stage stochastic program is then formulated as an optimization problem in the first-stage variables x that optimizes the cost of the first-stage decisions plus the expected cost of the second-stage decisions. It is written as follows:

[SNLP]

$$\underset{x \in \Re^{n_1}}{\text{Minimize}} \quad f(x) + \mathcal{Q}(x) \tag{6.90}$$

$$\text{s.t.} \quad Ax = b, \tag{6.91}$$

$$0 \leq x \leq u^x. \tag{6.92}$$

The first-stage objective function is $f : \Re^{n_1} \mapsto \Re$. The $m_1 \times n_1$ real matrix A, and the vector $b \in \Re^{m_1}$ specify constraints on the first-stage decision, and the vector $u^x \in \Re^{n_1}$ represents upper bounds on the first-stage variables.

We consider here the case where the stochastic quantities $\mathbf{g}$, $\mathbf{r}$, $\mathbf{B}$, $\mathbf{v}$ and $\mathbf{C}$ have a discrete and finite joint distribution, represented by the *scenario set* $\Omega = \{1, 2, \ldots, S\}$. Denote by p_s the probability of realization of scenario s, i.e.,

$$p^s = \text{Prob}\,\{(\mathbf{g}, \mathbf{r}, \mathbf{B}, \mathbf{v}, \mathbf{C}) = (g^s, r^s, B^s, v^s, C^s)\}, \text{ for all } s \in \Omega. \tag{6.93}$$

It is assumed that $p^s > 0$, for all $s \in \Omega$, and that $\sum_{s=1}^{S} p^s = 1$. When this is the case, we can write the expected value of the second-stage optimization problem as:

$$\mathcal{Q}(x) = \sum_{s=1}^{S} p^s Q(g^s, r^s, B^s, v^s, C^s \mid x), \tag{6.94}$$

Under the assumption of a finite, discrete event space, the stochastic nonlinear program [SNLP] can be reformulated to the following *large-scale deterministic equivalent* nonlinear program, see Wets [228]:

[DNLP]

$$\underset{x \in \Re^{n_1}, y^s \in \Re^{n_2}}{\text{Minimize}} \quad f(x) + \sum_{s=1}^{S} p^s g^s(y^s) \tag{6.95}$$

$$\text{s.t.} \qquad Ax = b, \tag{6.96}$$

$$C^s x + B^s y^s = r^s, \qquad \text{for all } s \in \Omega, \tag{6.97}$$

$$0 \leq x \leq u^x, \tag{6.98}$$

$$0 \leq y^s \leq v^s, \qquad \text{for all } s \in \Omega. \tag{6.99}$$

This deterministic equivalent nonlinear program consists of $m_1 + S \cdot m_2$ equality constraints and $n_1 + S \cdot n_2$ variables. The constraint matrix for the deterministic equivalent nonlinear program has the dual, block-angular structure:

$$\begin{pmatrix} A & & & & \\ C^1 & B^1 & & & \\ C^2 & & B^2 & & \\ \vdots & & & \cdots & \\ C^S & & & & B^S \end{pmatrix}.$$

The Case of Network Problems

We now address the case where the constraint set (6.96)–(6.97) takes the form of flow conservation constraints for some network problem. Specifically, we assume that the matrices

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

are both *node-arc incidence* matrices for each scenario $s \in \Omega$. That is, each column of these matrices has two nonzero entries. One entry is $+1$ and the other is a negative real number; we use $-m_{ij}$ to denote this number. Index i specifies the row with the $+1$ entry, and index j specifies the row with the real-valued entry. A network model is obtained by associating each row of the matrix with a node, and by associating an arc between two nodes when the corresponding rows have nonzero entries. The arcs are directed from the row with the $+1$ entry to the row with the real-valued entry.

Even with the assumption that the submatrices represent network flow problems, the full problem [**DNLP**] is not a network problem due to the repeated occurrence of $C^s x$ in (6.97) for all scenarios $s \in \Omega$. The first-stage variables x are, in this context, complicating variables and solution ap-

proaches based on Benders' (or generalized Benders') decomposition suggest themselves [23, 99, 100, 159]. An example of such an algorithm developed specifically for stochastic programs is the L-shaped decomposition method of Van Slyke and Wets [213] and its regularized version, Ruszczynski [205].

Split Variable Formulation

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

[RNLP]

$$\underset{x^s \in \Re^{n_1}, y^s \in \Re^{n_2}}{\text{Minimize}} \quad \sum_{s=1}^{S} p^s (f(x^s) + g^s(y^s)) \tag{6.100}$$

$$\text{s.t.} \qquad Ax^s = b, \qquad \text{for all } s \in \Omega, \tag{6.101}$$

$$C^s x^s + B^s y^s = r^s, \qquad \text{for all } s \in \Omega, \tag{6.102}$$

$$0 \leq x^s \leq u^x, \qquad \text{for all } s \in \Omega, \tag{6.103}$$

$$0 \leq y^s \leq v^s, \qquad \text{for all } s \in \Omega, \tag{6.104}$$

$$x^1 = x^s, \qquad \text{for all } s \in \Omega. \tag{6.105}$$

By this reformulation, we have obtained a *network with side-constraints.* In the absence of the side-constraints (6.105), the problem decomposes completely into S nonlinear network problems. The algorithm we develop exploits this special structure by decomposing the problem into S network problems, iteratively solving these and then enforcing the side-constraints. The side constraints (6.105) are known as *non-anticipativity* constraints.

That is, the first stage decisions x^s can not anticipate the future. Given two scenarios s_1 and s_2 that are undistinguished from each other when the first-stage decisions are made, then we must have $x^{s_1} = x^{s_2}$.

Matrix Structure

Let $M = S \cdot (m_1 + m_2) + (S-1) \cdot n_1$, $N = S \cdot (n_1 + n_2)$ and let I denote the $n_1 \times n_1$ identity matrix. The constraint matrix for [**RNLP**] has dimension $M \times N$. We denote this matrix by Φ, that is,

$$\Phi = \begin{pmatrix} A & & & & & & \\ C^1 & B^1 & & & & & \\ & & A & & & & \\ & & C^2 & B^2 & & & \\ & & & & \ddots & & \\ & & & & & A & \\ & & & & & C^S & B^S \\ I & & -I & & & & \\ \vdots & & & & \ddots & & \\ I & & & & & I & \end{pmatrix}. \tag{6.106}$$

It is evident from the structure of the constraint matrix that the problem decomposes by scenario if the non-anticipativity constraints are ignored. We also denote by $\gamma \in \Re^M$ the right-hand side of [**RNLP**], i.e.,

$$\gamma^T = (b^T \mid (r^1)^T \mid \ldots \mid b^T \mid (r^S)^T \mid 0 \ldots 0).$$

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

$$z^T = ((x^1)^T \mid (y^1)^T \mid \ldots \mid (x^S)^T \mid (y^S)^T),$$

and by $u \in \Re^N$ the upper bounds on z, i.e.,

$$u^T = ((u^x)^T \mid (v^1)^T \mid \ldots \mid (u^x)^T \mid (v^S)^T).$$

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

$$F(z) = F(x^1, y^1, \cdots, x^S, y^S) = \sum_{s=1}^{S} p^s (f(x^s) + g^s(y^s)). \tag{6.107}$$

The replicated nonlinear program (6.100) – (6.105) can be written in compact matrix form as

[**RNLP**]

$$\underset{z \in \Re^N}{\text{Minimize}} \quad F(z) \tag{6.108}$$

$$\text{s.t.} \quad \begin{pmatrix} \gamma \\ 0 \end{pmatrix} \leq \begin{pmatrix} \Phi \\ I_N \end{pmatrix} z \leq \begin{pmatrix} \gamma \\ u \end{pmatrix}, \tag{6.109}$$

where I_N is the $N \times N$ identity matrix. We will be using this compact matrix notation to develop the row-action algorithm. However, the precise iterative steps of the algorithm depend on the network substructures of the matrix Φ. Hence, the algebraic formulation of the network substructures is given next.

Algebraic Representation of Network Problem

We assume for the sake of symmetry that the underlying network structure is the same for all the scenario problems. We denote this structure by the graph $G = (N, A)$, where $N = 1, 2, \ldots, m_1 + m_2$ is the set of nodes, and $A = \{(i,j) \mid i,j \in N\} \subseteq N \times N$ is the set of arcs. Let $\delta_i^+ = \{j \mid (i,j) \in A\}$ be the set of nodes having an arc coming from node i, and $\delta_j^- = \{i \mid (i,j) \in A\}$ be the set of nodes having an arc going to node j. We partition the node set into two disjoint sets, N^1 and N^2. N^1 consists of the m_1 nodes whose

incident arcs are all first-stage so that their flow-conservation constraints do not depend on the realization of the uncertain quantities. $N^2 = N \setminus N^1$ are the m_2 nodes with stochastic right-hand sides or incident second-stage arcs.

We also partition the arc set A into two disjoint sets A^1 and A^2, corresponding to (replicated) first-stage and second-stage decisions, respectively. Denote by x_{ij}^s, $(i,j) \in A^1$, and y_{ij}^s, $(i,j) \in A^2$, the flow on the arc leading from node i to node j under scenario $s \in \Omega$. The upper bound of a (replicated) first-stage arc x_{ij}^s is denoted by u_{ij}^x and the upper bound of a second-stage arc y_{ij}^s is denoted by v_{ij}^s. The multiplier on arc (i,j) is denoted by m_{ij} for $(i,j) \in A^1$ and by m_{ij}^s for $(i,j) \in A^2$. The network optimization model for a single scenario $s \in \Omega$ is given by:

[NLP(s)]

$$\underset{x^s \in \Re^{n_1}, y^s \in \Re^{n_2}}{\text{Minimize}} \quad \sum_{(i,j)\in A^1} p^s f_{ij}(x_{ij}^s) + \sum_{(i,j)\in A^2} p^s g_{ij}^s(y_{ij}^s) \tag{6.110}$$

s.t.

$$\sum_{j\in\delta_i^+} x_{ij}^s - \sum_{k\in\delta_i^-} m_{ki} x_{ki}^s = b_i, \qquad \forall\, i \in N^1, \tag{6.111}$$

$$\sum_{j\in\delta_i^+\cap N^1} x_{ij}^s - \sum_{k\in\delta_i^-\cap N^1} m_{ki}^s x_{ki}^s + \sum_{j\in\delta_i^+\cap N^2} y_{ij}^s - \sum_{k\in\delta_i^-\cap N^2} m_{ki}^s y_{ki}^s = r_i^s, \qquad \forall\, i \in N^2, \tag{6.112}$$

$$0 \le x_{ij}^s \le u_{ij}^x, \qquad \forall (i,j) \in A^1, \tag{6.113}$$

$$0 \le y_{ij}^s \le v_{ij}^s, \qquad \forall (i,j) \in A^1. \tag{6.114}$$

The complete stochastic network problem **[RNLP]** is obtained by replicating the network problem **[NLP(s)]** for each scenario and including the

non-anticipativity side constraints:

$$x_{ij}^1 = x_{ij}^s, \quad \text{for all } s \in \Omega \text{ and } (i,j) \in A^1. \tag{6.115}$$

We have in this section been referring to quantities pertaining to an arc $(i,j) \in A$ under scenario $s \in \Omega$ by using subscripts "(i,j)" and superscript "s". We need to establish the correspondence between the matrix/vector notation established at page 165 and the algebraic notation of this section: If (i_1, j_1) denotes the first arc in A, "z_1" and "$x_{i_1 j_1}^1$" refer to the same quantity (see (6.3.1)). This connection is made formal in an obvious way by defining a lexicographic ordering of the arcs in A.

6.3.2 Applications

We discuss here three real-world applications where data uncertainty is a key factor in formulating a mathematical model. All three applications can be modeled using a stochastic programming formulation, with the network structure described above. While the network structure is not essential for the development of the model, it has certain advantages for the development of specialized solution algorithms. It also makes it easier to understand the model.

Finance: Asset/Liability Management

A generic description of a portfolio manager's problem is the following:

> Construct a portfolio of assets whose performance measures will remain invariant under a wide range of economic scenarios.

For now we leave unspecified the notion of *performance measures* and the precise nature of the *economic scenarios*. The key idea is to decide what goals we want our portfolio to achieve, specify measures that quantify the achievement of the goals, and make sure that the goals will still be met when the economic environment changes.

The precise goal of the portfolio manager depends on the underlying application. We describe here three practical applications where one needs to deal with the inherent uncertainties of the new fixed income securities:

Indexation: Passive portfolio managers would like to build a portfolio that will track a prespecified index. For example, Shearson-Lehman and Salomon Brothers publish a monthly *mortgage* index that reflects the overall state of this segment of fixed-income markets. An investor who wishes to invest in mortgages may be satisfied if his or her portfolio closely tracks the index. The performance measure of such a portfolio is the difference in return between the portfolio and the index. This difference has to be very small, for all changes in the index caused by interest rate movements and by variations in the cashflows generated by the securities in the portfolio.

Liability Funding: Insurance and pension fund companies are typically heavily exposed to complex fixed-income securities, like mortgages. These instruments are considered as an investment for funding a variety of the liabilities held by these institutions. The goal of the portfolio manager is to construct a portfolio of mortgages that will fund the future stream of liabilities. Uncertainty here appears once more in the form of interest rate changes and changes in the timing of payments

from the securities. Furthermore, the timing of the liability stream may also be subject to uncertain variations: For example, the timing of payments to holders of single premium deferred annuities may change as annuitants exercise the option to terminate their investment.

Debt Issuance: Government agencies, like Fannie Mae and Freddie Mac, fund the purchase of mortgage assets by issuing debt. The problem of a portfolio manager is to decide which type of debt — maturity, yield, call-option — to issue in order to fund the purchase of a specific set of assets. Of course, there is no reason to assume that the assets have been pre-specified: The model may choose an appropriate asset mix from a large universe of securities. The timing of the cashflows from both assets and liabilities may be uncertain in this application. The goal of the portfolio manager is to ensure that the payments against the issued debt will be met from the available assets, irrespectively of the timing of cash flows and fluctuations in interest rates.

Problems such as those described above are routinely addressed in practice. For example, Worzel *et al.* [229] describe the development of indexation models for a major insurance company. Zenios and Kang [241] illustrate the use of mathematical models for funding liability streams. Holmer [126] describes the asset/liability management system developed by Fannie Mae. A collection of recent articles on different aspects of these problems is found in Zenios [237].

We classify the asset/liability management models into: 1. Static, 2. Single-period, stochastic, and 3. Multiperiod, dynamic and stochastic. It is important to understand how the models address increasingly more com-

plex aspects of the asset/liability management problem. Only then can the portfolio manager decide which model may be more appropriate for the application at hand. Of course, this decision has to be weighted against the increasing complexity — both conceptual and computational — of the models [123, 236]. The models in the third class are stochastic programming problems.

Static models: Such models build a portfolio that meets the target under the current state of the economy, and hedges against small changes from the current state. For example, a term structure of interest rates is input to the model which matches assets and liabilities under this structure. Conditions are then imposed to guarantee that if the term structure deviates somewhat from the assumed value, the assets and liabilities will move in the same direction and by equal amounts. This is the fundamental principle behind *portfolio immunization.* See, for example, Christensen and Fabozzi [54] for a discussion of the finance-theoretic principles behind immunization, and Dahl *et al.* [61, 62, 60] for operational (optimization) models.

Single-period, stochastic models: A static model does not permit the specification of a stochastic process that describes changes of the economic environment from its current status. However, modern finance abounds with theories that describe interest rates, and other volatile factors, using stochastic processes; see, e.g., Ingersoll [131]. Stochastic differential calculus is often used to price interest-rate contingencies. For complex instruments analysts resort to Monte Carlo simulations, an idea pioneered by Boyle [29] for options pricing. See, for exam-

ple, Hutchinson and Zenios [128] for its application to the pricing of mortgage securities. A stochastic asset/liability model describes the *distribution* of returns of both assets and liabilities in the volatile environment, and ensures that movements of both sides of the balance sheet are highly correlated. This idea is not new: Markowitz pioneered the notion of risk management for equities via the use of correlations in his seminal papers [155, 154]. However, for the fixed income world this approach has only recently received attention. Its validity is advocated in Mulvey and Zenios [172], and an application from the mortgage market is described by Holmer [126].

Multiperiod, dynamic and stochastic models: A stochastic model, as outlined above, is *myopic*. That is, it builds a portfolio that will have a "well behaved" distribution of error (error = asset return - liability return) under the specified stochastic process. For example, the mean value of the error should be positive, and its variance small. However, the single-period model does not account for the fact that the portfolio manager is likely to rebalance the portfolio once some surplus is realized. Furthermore, as the stochastic process evolves across time different portfolios may be more appropriate for capturing the correlations of assets and liabilities. The single-period model may recommend a conservative strategy, while a more aggressive approach would be justified once we explicitly recognize the manager's ability to rebalance the portfolio.

What is needed is a model that explicitly captures both the stochastic nature of the problem, but also the fact that the portfolio is managed

in a dynamic, multi-period context. Stochastic programs with recourse provide the framework for dealing with this broad problem. The significance of this class of models for portfolio optimization was recognized in the early seventies by Bradley and Crane [30]. With the recent advances in high-performance computing this approach has been receiving renewed interest from the academic literature — Mulvey and Vladimirou [169], Zenios [234] and Hiller and Eckstein [122].

The multiperiod and stochastic model captures the dynamics of the following situation:

> The portfolio manager must make investment decisions — i.e., decide on the composition of the portfolio — facing an uncertain future. These are the *first-stage* decisions of the stochastic programming formulation. Once the first-stage decisions are made a realization of the uncertain future is observed, and the manager determines an optimal *second-stage* (or, recourse) decisions. The objective is to maximize the expected utility of terminal wealth.

A precise mathematical formulation of this problem is given in Zenios [234]; see also Mulvey and Vladimirou [171] or Hiller and Eckstein [122] for related formulations. If the uncertain future were known with complete certainty at the time of the first-stage decision, then the portfolio manager would have a much simpler problem to solve.

We now turn to a second application of stochastic network programs from engineering planning.

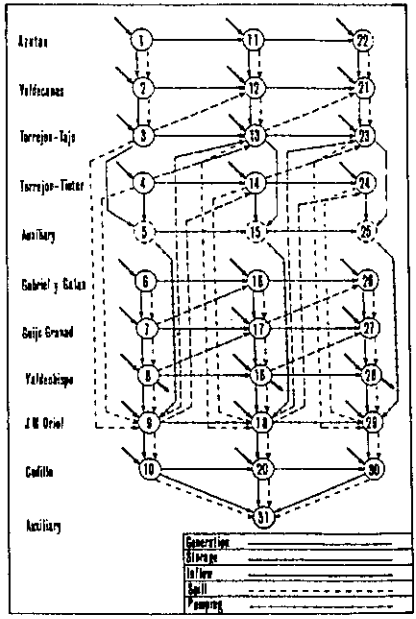


Figure 6.5: The Tajo reservoir system.

Hydroelectric Power Scheduling

Planning the generation of electricity for hydroelectric power systems is a complex process. Decisions made today for the coming hours depend on the current state of the system, electricity demand, water inflows and so on. There are numerous constituents who are affected by the results of decisions on the volume of dam releases. Water release has an impact on irrigation, recreation and flood control, in addition to generating power. Rosenthal [201] modeled the scheduling of reservoir releases as a nonlinear network, and he developed a system for the Tennessee Valley Authority.

Figure 6.5 depicts the reservoir system for the lower Tajo area in Spain, over a three-period model. The system consists of nine dams (names shown)

and an auxiliary reservoir system. Every dam is represent by three nodes — one for each time period of the model. This model was developed by Dembo *et al.* [69] in order to plan the hydroelectric power generation by Hidroeléctrica Espanõla. The necessary data for the model — which is complex to formulate here in its details — consist of: (1) the network topology, specified by the geographical location of the dams and their interconnections, (2) limits on reservoir storage, level of turbine operations, pumping and spillage, (3) hydroelectric production coefficients for each reservoir, obtained from engineering analysis of its storage capacity and the turbine technology. Important input data are also the water level in the reservoirs, and the electricity demand. Both of these quantities are uncertain. Demand for electricity exhibits both a daily and a seasonal variation, that can be estimated at best by a set of scenarios. The same is also true for the water level that depends on rainfall. Different scenarios of rainfall are estimated based on historical observations and on weather forecasting. These uncertainties are fundamental to the operation of the system, and they can not be glossed over. Dembo *et al.* [69] developed, indeed, a stochastic network optimization model for this problem. The objective function is obtained by simulating the marginal benefits of using hydroelectric power over thermal fuel. The detailed formulation developed by Dembo *et al.* [69] fits the structure of a stochastic network program.

Planning Air-Traffic Ground Holding Policies

The US air traffic system is an exceedingly complex web of airports, aircraft, air traffic controllers at all airports, and a centralized flow control facility in Washington, DC. The complexity of the air traffic system in Europe is

intensified by the lack of coordination of 42 different control centers among 22 countries. This situation has been complicated even further with the efforts to integrate the former Eastern European countries into the rest of the European system.

The systems are highly congested. Traffic flow is carefully monitored, and controlled, so that flights proceed without risk to safety. One key control mechanism is *ground holding*, whereby a flight is delayed for departure, if congestion is anticipated at the destination airport. Ground holding is a safe, and relatively inexpensive solution, as opposed to holding aircraft "in flight" before landing clearance is granted. While the air traffic control system does an excellent job in monitoring traffic so that high safety standards are maintained, there is substantial room for improvement. In particular it is considered possible to improve the efficiency — that is, cost effectiveness — of the system without sacrificing anything in safety. It is estimated that the ground delays in the US in 1986 averaged 2000 hours per day. This is equivalent to grounding a total of 250 airplanes, or the equivalent of a carrier the size of Delta Airlines. A study by the West German Institute for Technology estimates the *avoidable* cost of air traffic delays, due to ground holding alone, to be $1.5 billion in 1990. Various aspects of the air traffic system are discussed in Odoni [182], Del Balzo [17], Richetta [195], Zenios [235] and references therein.

The ground holding policy problem attempts to find optimal holding policies, given information about the number of flights scheduled for departure during the planning horizon, and the travel time to the destination airport. Even for the simple case where only a single destination airport is analyzed, its capacity is uncertain due to weather conditions. The problem is compli-

cated further by the presence of multiple airports: ground holding decisions at each one have a cascade effect on all others. A stochastic programming formulation for the single-destination airport problem has been developed by Richetta [195]. Numerical experiments with the model, applied to data obtained from Logan airport in Boston, MA, show that substantial reductions in total delay can be realized when using the stochastic programming, dynamic, models as opposed to the more commonly used static models.

6.3.3 An Iterative Algorithm for Stochastic Network Optimization

We apply now the row-action Algorithm 4.4.1 to the replicated nonlinear programming formulation of the stochastic program. We first present a detailed description of the algorithm for the equality and bound constraints on z, (6.109). We then go further to specialize the algorithm for the network flow constraints and the non-anticipativity constraints, thus giving the details of an implementable procedure. After initialization, the algorithm proceeds by projecting upon one constraint of (6.109) at a time, updating the dual price of the constraint and the primal variables occurring in the constraint in order to maintain dual feasibility and complementary slackness. We express the order in which constraints are considered using a *control sequence*, $\ell(\nu)$, such that constraint $\ell(\nu)$ is considered at iteration ν. For clarity of notation we will abbreviate $\ell(\nu)$ by ℓ. Let Φ^ℓ denote the ℓ^{th} row of Φ and let γ_ℓ denote the ℓ^{th} element of γ. If $\ell \in \{1, ..., M\}$ the constraint considered is $H(\Phi^l, \gamma_l)$. If $\ell \in \{M+1, ..., M+N\}$ we consider the simple bounds on the $(\ell - M)^{\text{th}}$ variable.

Let $\pi \in \Re^{M+N}$ be a vector of dual prices associated with the rows of

the constraint set (6.109), and let $e^\ell \in \Re^{M+N}$ denote the ℓ^{th} unit vector. Assuming that $F(z)$ is a Bregman's function and has the strong zone consistency property with respect to the hyperplanes $H(\Phi^\ell, \gamma_\ell)$, we can re-state the general Algorithm 4.4.1 for the formulation of the replicated stochastic program:

Algorithm 6.3.1 General Row-Action Algorithm for problem RNLP.

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

$$\nabla F(z^0) = -\begin{pmatrix} \Phi \\ I_N \end{pmatrix}^T \pi^0. \tag{6.116}$$

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

$$\nabla F(z^{\nu+\frac{1}{2}}) = \nabla F(z^\nu) + \beta^\nu \Phi^\ell, \tag{6.117}$$

$$z^{\nu+\frac{1}{2}} \in H(\Phi^\ell, \gamma_\ell). \tag{6.118}$$

Update the dual price:

$$\pi^{\nu+\frac{1}{2}} = \pi^\nu - \beta^\nu e^\ell. \tag{6.119}$$

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

If $z_{\ell-M}^{\nu+\frac{1}{2}} \leq 0$, let β^ν and $z^{\nu+1}$ be the solution of:

$$\nabla F(z^{\nu+1}) = \nabla F(z^{\nu+\frac{1}{2}}) + \beta^\nu, \tag{6.120}$$

$$z_{\ell-M}^{\nu+1} = 0. \tag{6.121}$$

If $z_{\ell-M}^{\nu+\frac{1}{2}} \geq u_{\ell-M}$, let β^ν and $z^{\nu+1}$ be the solution of:

$$\nabla F(z^{\nu+1}) = \nabla F(z^{\nu+\frac{1}{2}}) + \beta^\nu, \tag{6.122}$$

$$z_{\ell-M}^{\nu+1} = u_{\ell-M}. \tag{6.123}$$

If $0 < z_{\ell-M}^{\nu+\frac{1}{2}} < u_{\ell-M}$, let β^ν and $z^{\nu+1}$ be the solution of:

$$\nabla F(z^{\nu+1}) = \nabla F(z^{\nu+\frac{1}{2}}) + \beta^\nu, \tag{6.124}$$

$$\beta^\nu = \pi_\ell^{\nu+\frac{1}{2}}. \tag{6.125}$$

Update the dual price:

$$\pi^{\nu+1} = \pi^{\nu+\frac{1}{2}} - \beta^\nu e^\ell, \tag{6.126}$$

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

Specialization to Quadratic Stochastic Networks

We specialize now the general algorithm to the case of quadratic network flow problems with non-anticipativity constraints. Since we work with the replicated problem it is convenient to use "x_{ij}^s" for both first- and second-stage variables. We thus assume that F takes the form

$$F(x) = \sum_{(i,j)\in A, s\in\Omega} \left(\frac{1}{2} w_{ij}^s (x_{ij}^s)^2 + q_{ij}^s x_{ij}^s \right). \tag{6.127}$$

for $w_{ij}^s > 0$. The coefficients w_{ij}^s and q_{ij}^s incorporate the scenario probabilities, p^s. Let $M_1 = S \cdot (m_1 + m_2)$. Then rows $1, ..., M_1$ of the constraint matrix Φ correspond to network flow conservation constraints, and rows

$M_1 + 1, ..., M$ correspond to the non-anticipativity constraints that take the simple form

$$x_{ij}^1 - x_{ij}^s = 0,$$

for some $(i,j) \in A^1$ and $s \in \Omega$. For the dual price π_ℓ, $\ell \in \{1, ..., M_1\}$, associated with the flow conservation constraint for node $i \in N$ under scenario $s \in \Omega$ we use the notation π_i^s. For the dual price π_ℓ, $\ell \in \{M+1, ..., M+N\}$, associated with the simple bound constraints for x_{ij}^s (i.e., the reduced cost of x_{ij}^s), we use the notation π_{ij}^s. We now proceed to develop the specific projection formulae for use in Steps 1 and 2 of Algorithm 6.3.1. The complete algorithm is given as Algorithm 6.3.2.

Projection on Flow Conservation Constraints

First we derive the projection on the flow conservation constraints of a generalized network, (6.111)–(6.112). We consider in this section the flows on the incoming arcs, x_{ki}^s for $k \in \delta_i^-$, and the flows on the outgoing arcs, x_{ij}^s for $j \in \delta_i^+$ for a node $i \in N^1$ under some scenario $s \in \Omega$. The derivation of the projection upon flow conservation constraints for nodes $i \in N^2$ is similar.

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

$$\nabla F(x^s) = \nabla F(y^s) + \beta_i^s \Phi^i, \tag{6.128}$$

$$x^s \in H(\Phi^i, \gamma_i). \tag{6.129}$$

Of course, if $y^s \in H(\Phi^i, \gamma_i)$, then $\beta_i^s = 0$ and $x^s = y^s$. If the current iterate

does not satisfy flow conservation, we define the *node surplus* σ_i^s as

$$\sigma_i^s = b_i - (\sum_{j \in \delta_i^+} y_{ij}^s - \sum_{k \in \delta_i^-} m_{ki} y_{ki}^s). \tag{6.130}$$

Applying the iterative step (6.117) to the functional form of the objective function (6.127) and using the structure of the rows of the constraint matrix Φ that correspond to network flow constraints we get:

$$x_{ij}^s = y_{ij}^s + \beta_i^s \cdot \frac{1}{w_{ij}^s} \text{ for } j \in \delta_i^+, \tag{6.131}$$

$$x_{ki}^s = y_{ki}^s - \beta_i^s \cdot \frac{m_{ki}}{w_{ki}^s} \text{ for } k \in \delta_i^-. \tag{6.132}$$

Substituting these expressions for x_{ij}^s and x_{ki}^s into (6.111) we get:

$$\sum_{j \in \delta_i^+} (y_{ij}^s + \beta_i^s \cdot \frac{1}{w_{ij}^s}) - \sum_{k \in \delta_i^-} m_{ki}(y_{ki}^s - \beta_i^s \cdot \frac{m_{ki}}{w_{ki}^s}) = b_i. \tag{6.133}$$

From this and (6.130) we get

$$\beta_i^s = \frac{\sigma_i^s}{\sum_{j \in \delta_i^+} \frac{1}{w_{ij}^s} + \sum_{k \in \delta_i^-} \frac{m_{ki}^2}{w_{ki}^s}}. \tag{6.134}$$

Using this result in (6.131) and (6.132) gives us the desired formulae for updating all primal variables incident to node i. The dual variable for this node (π_i^s) is updated by adding β_i^s to its current value, $\pi_i^s \leftarrow \pi_i^s + \beta_i^s$.

Projection on Simple Bound Constraints

We now develop the specific projections on the simple bounds, (6.113)–(6.114) corresponding to Step 2 of the general row-action algorithm. Denote by y_{ij}^s the current value of the variable. If $y_{ij}^s < 0$, we get from (6.120) that

$$0 = x_{ij}^s = y_{ij}^s + \frac{\beta}{w_{ij}^s} \tag{6.135}$$

The primal variable is thus set to 0, and the Bregman parameter is

$$\beta = -w_{ij}^s y_{ij}^s. \tag{6.136}$$

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

If $y_{ij}^s > u_{ij}^x$ we similarly set the primal variable x_{ij}^s to the upper bound, u_{ij}^x, and from (6.122) compute the Bregman parameter to be

$$\beta = (u_{ij}^x - y_{ij}^s) w_{ij}^s \tag{6.137}$$

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

Finally, if $0 \leq y_{ij}^s \leq u_{ij}^x$ we get from (6.124)

$$x_{ij}^s = y_{ij}^s + \frac{\pi_{ij}^s}{w_{ij}^s} \tag{6.138}$$

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

Projections on Non-Anticipativity Constraints

We now develop the iterative step (6.117) for the equality, non-anticipativity constraints. A non-anticipativity constraint (6.115) takes the form

$$x_{ij}^1 - x_{ij}^s = 0 \tag{6.139}$$

for some $(i,j) \in A^1$ and some $s \in \Omega$. In order to map these constraints onto the matrix Φ (6.106), let $\mu(s)$ be a row index such that $\Phi^{\mu(s)}$ for $s = 2, 3, \ldots, S$ is the row of Φ that corresponds to the constraint $x_{ij}^1 - x_{ij}^s = 0$.

If y is the current iterate, the Bregman projection upon this constraint solves

$$\nabla F(x) = \nabla F(y) + \beta \Phi^{\mu(s)}, \tag{6.140}$$

$$x_{ij}^1 = x_{ij}^s, \tag{6.141}$$

where x is the projected point. This system can be written as

$$x_{ij}^1 = x_{ij}^s = y_{ij}^1 + \frac{\beta}{w_{ij}^1} = y_{ij}^s - \frac{\beta}{w_{ij}^s}. \tag{6.142}$$

Solving this, we get

$$x_{ij}^1 = x_{ij}^s = \frac{w_{ij}^1 y_{ij}^1 + w_{ij}^s y_{ij}^s}{w_{ij}^1 + w_{ij}^s}, \tag{6.143}$$

i.e., the point (y_{ij}^1, y_{ij}^s) is projected upon the point with coordinates equal to the weighted average of y_{ij}^1 and y_{ij}^s, where w_{ij}^1 and w_{ij}^s are the weights.

Closed Form Solution for Non-Anticipativity Constraints

In this section we obtain a closed form solution for the projection on all $S-1$ non-anticipativity constraints relating to a first-stage variable x_{ij}, i.e., the constraints $x_{ij}^1 = x_{ij}^s$ for all $s \in \Omega$, and some $(i,j) \in A^1$. This result has important implications for the massively parallel implementation of the algorithm. We consider the effect of repeated projections on this subset of the constraints (6.115). The almost cyclic control framework of the row-action algorithm allows repeated projections upon only these constraints until convergence (within some tolerance) of the variables x_{ij}^s to a common value, $\hat{x}_{ij}$. We show that $\hat{x}_{ij}$ can be obtained analytically rather than using the iterative scheme.

We focus on the non-anticipativity constraints for the replications of a single first-stage variable x_{ij}, which take the form

$$\begin{aligned} x_{ij}^1 - x_{ij}^2 &= 0, \\ x_{ij}^1 - x_{ij}^3 &= 0, \\ &\cdots \end{aligned} \tag{6.144}$$

$$x_{ij}^1 - x_{ij}^S = 0.$$

Let $\nabla F_{ij} : \Re^S \mapsto \Re^S$ denote the subvector of the gradient ∇F corresponding to the S replications of the first-stage variable, $x_{ij}^1, ..., x_{ij}^S$, and, similarly, let Φ_{ij} denote the submatrix of Φ consisting of the columns corresponding to $x_{ij}^1, ..., x_{ij}^S$.

By repeated projection upon these non-anticipativity constraints, such that the ν^{th} projection is upon the constraint $x_{ij}^1 - x_{ij}^{\ell(\nu)} = 0$ we obtain a sequence of points $x^\nu \in \Re^S$ satisfying

$$\nabla F_{ij}(x^\nu) = \nabla F_{ij}(y) + \sum_{k=1}^{\nu} \lambda^k \Phi_{ij}^{\mu(\ell(\nu))}, \tag{6.145}$$

where λ^k is the Bregman parameter corresponding to the k^{th} projection, y is the starting point, and $\mu(\ell)$ is the row index that corresponds to the non-anticipativity constraints; see the discussion on page 182. The limiting point $x^* \in \Re^S$ satisfies

$$\nabla F_{ij}(x^*) = \nabla F_{ij}(y) + \sum_{k=1}^{\infty} \lambda^k \Phi_{ij}^{\mu(\ell(\nu))} \tag{6.146}$$

and must, by (6.144), have all components identical, i.e., $x^* = (\hat{x}_{ij}, ..., \hat{x}_{ij})^T$ for some $\hat{x}_{ij} \in \Re$.

Let

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

for $k = 2, ..., S$. Using the fact that $F(y)$ is the quadratic function (6.127), rewrite (6.146) as the square system in S variables, $\hat{x}_{ij}, \Lambda^2, ..., \Lambda^S$:

$$\hat{x}_{ij} = y_{ij}^1 + \frac{1}{w_{ij}^1} \sum_{s=2}^{S} \Lambda^s,$$

$$\begin{aligned} \hat{x}_{ij} &= y_{ij}^2 - \frac{1}{w_{ij}^2}\Lambda^2, \\ &\vdots \\ \hat{x}_{ij} &= y_{ij}^S - \frac{1}{w_{ij}^S}\Lambda^S. \end{aligned} \tag{6.147}$$

In matrix form, this is

$$Ht = y, \tag{6.148}$$

where

$$H = \begin{pmatrix} 1 & \frac{-1}{w_{ij}^1} & \frac{-1}{w_{ij}^1} & \cdots & \frac{-1}{w_{ij}^1} \\ 1 & \frac{1}{w_{ij}^2} & & & \\ 1 & & \frac{1}{w_{ij}^3} & & \\ \cdots & & & \ddots & \\ 1 & & & & \frac{1}{w_{ij}^S} \end{pmatrix}, \tag{6.149}$$

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

$$h = \frac{1}{\det(H)} \cdot \prod_{s=1}^{S} \frac{1}{w_{ij}^s} \cdot (w_{ij}^1, w_{ij}^2, ..., w_{ij}^S)^T,$$

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

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

Note that $\det(H) > 0$, so system (6.148) has a unique solution. Solving for $\hat{x}_{ij}$ we get

$$\hat{x}_{ij} = h^T y = \frac{\sum_{s=1}^{S} w_{ij}^s y_{ij}^s}{\sum_{s=1}^{S} w_{ij}^s}. \tag{6.150}$$

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

$$\hat{x}_{ij} = \sum_{s=1}^{S} p^s y_{ij}^s. \tag{6.151}$$

The Row-Action Algorithm for Quadratic Stochastic Networks

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

Algorithm 6.3.2 Row-Action Algorithm for Quadratic Stochastic Networks

Step 0: (Initialization) $\nu \leftarrow 0$. Get π^0 and z^0 such that $\nabla F(z^0) = -\begin{pmatrix} \Phi \\ I_N \end{pmatrix}^T \pi^0$. For example, $\pi^0 = 0$ and

$$(x_{ij}^s)^0 = -\frac{q_{ij}^s}{w_{ij}^s}, \text{ for all } (i,j) \in A^1, s \in \Omega, \tag{6.152}$$

$$(y_{ij}^s)^0 = -\frac{q_{ij}^s}{w_{ij}^s}, \text{ for all } (i,j) \in A^2, s \in \Omega, \tag{6.153}$$

Step 1: (Solve scenario subproblems). For all $s \in \Omega$:

Step 1.1: (Solve for flow conservation constraints) Let

$$(\beta_i^s)^{\nu+\frac{1}{2}} = \frac{\sigma_i^s}{\sum_{j\in\delta_i^+} \frac{1}{w_{ij}^s} + \sum_{k\in\delta_i^-} \frac{m_{ki}^2}{w_{ki}^s}} \text{ for all } i \in N^1, \tag{6.154}$$

$$(\beta_i^s)^{\nu+\frac{1}{2}} = \frac{\sigma_i^s}{\sum_{j\in\delta_i^+} \frac{1}{w_{ij}^s} + \sum_{k\in\delta_i^-} \frac{(m_{ki}^s)^2}{w_{ki}^s}} \text{ for all } i \in N^2 \tag{6.155}$$

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

$$(x_{ij}^s)^{\nu+\frac{1}{2}} = (x_{ij}^s)^{\nu} + (\beta_i^s)^{\nu+\frac{1}{2}} \cdot \frac{1}{w_{ij}^s} \text{ for all } j \in \delta_i^+, \quad (6.156)$$

$$(x_{ki}^s)^{\nu+\frac{1}{2}} = (x_{ki}^s)^{\nu} - (\beta_i^s)^{\nu+\frac{1}{2}} \cdot \frac{m_{ki}}{w_{ki}^s} \text{ for all } k \in \delta_i^-, \quad (6.157)$$

$$(\pi_i^s)^{\nu+\frac{1}{2}} = (\pi_i^s)^{\nu} - (\beta_i^s)^{\nu+\frac{1}{2}}. \quad (6.158)$$

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

$$(y_{ij}^s)^{\nu+\frac{1}{2}} = (y_{ij}^s)^{\nu} + (\beta_i^s)^{\nu+\frac{1}{2}} \cdot \frac{1}{w_{ij}^s} \text{ for all } j \in \delta_i^+, \quad (6.159)$$

$$(y_{ki}^s)^{\nu+\frac{1}{2}} = (y_{ki}^s)^{\nu} - (\beta_i^s)^{\nu+\frac{1}{2}} \cdot \frac{m_{ki}^s}{w_{ki}^s} \text{ for all } k \in \delta_i^-, \quad (6.160)$$

$$(\pi_i^s)^{\nu+\frac{1}{2}} = (\pi_i^s)^{\nu} - (\beta_i^s)^{\nu+\frac{1}{2}}. \quad (6.161)$$

Step 1.2: (Solve for the simple bounds).

For all first-stage arcs $(i,j) \in A^1$:

$$(x_{ij}^s)^{\nu+1} = \begin{cases} u_{ij}^x & \text{if } (x_{ij}^s)^{\nu+\frac{1}{2}} \geq u_{ij}^x, \\ 0 & \text{if } (x_{ij}^s)^{\nu+\frac{1}{2}} \leq 0, \\ (x_{ij}^s)^{\nu+\frac{1}{2}} + \frac{(\pi_{ij}^s)^{\nu+\frac{1}{2}}}{w_{ij}^s} & \text{if } 0 < (x_{ij}^s)^{\nu+\frac{1}{2}} < u_{ij}^x. \end{cases} \quad (6.162)$$

and

$$(\pi_{ij}^s)^{\nu+1} = \begin{cases} (\pi_{ij}^s)^{\nu+\frac{1}{2}} - w_{ij}^s(u_{ij}^x - (x_{ij}^s)^{\nu+\frac{1}{2}}) & \text{if } (x_{ij}^s)^{\nu+\frac{1}{2}} \geq u_{ij}^x, \\ (\pi_{ij}^s)^{\nu+\frac{1}{2}} + w_{ij}^s(x_{ij}^s)^{\nu+\frac{1}{2}} & \text{if } (x_{ij}^s)^{\nu+\frac{1}{2}} \leq 0, \\ 0 & \text{if } 0 < (x_{ij}^s)^{\nu+\frac{1}{2}} < u_{ij}^x. \end{cases} \quad (6.163)$$

For all second-stage arcs $(i,j) \in A^2$:

$$(y_{ij}^s)^{\nu+1} = \begin{cases} v_{ij}^s & \text{if } (y_{ij}^s)^{\nu+\frac{1}{2}} \geq v_{ij}^s, \\ 0 & \text{if } (y_{ij}^s)^{\nu+\frac{1}{2}} \leq 0, \\ (y_{ij}^s)^{\nu+\frac{1}{2}} + \frac{(\pi_{ij}^s)^{\nu+\frac{1}{2}}}{w_{ij}^s} & \text{if } 0 < (y_{ij}^s)^{\nu+\frac{1}{2}} < v_{ij}^s. \end{cases} \tag{6.164}$$

and

$$(\pi_{ij}^s)^{\nu+1} = \begin{cases} (\pi_{ij}^s)^{\nu+\frac{1}{2}} - w_{ij}^s(v_{ij}^s - (y_{ij}^s)^{\nu+\frac{1}{2}}) & \text{if } (y_{ij}^s)^{\nu+\frac{1}{2}} \geq v_{ij}^s, \\ (\pi_{ij}^s)^{\nu+\frac{1}{2}} + w_{ij}^s(y_{ij}^s)^{\nu+\frac{1}{2}} & \text{if } (y_{ij}^s)^{\nu+\frac{1}{2}} \leq 0, \\ 0 & \text{if } 0 < (y_{ij}^s)^{\nu+\frac{1}{2}} < v_{ij}^s. \end{cases} \tag{6.165}$$

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

For all first-stage arcs $(i,j) \in A^1$:

$$\hat{x}_{ij} = \sum_{s=1}^{S} p^s (x_{ij}^s)^{\nu+1}, \tag{6.166}$$

$$(x_{ij}^s)^{\nu+1} \leftarrow \hat{x}_{ij} \text{ for all } s \in \Omega. \tag{6.167}$$

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

Chapter 7

Decompositions for Parallel Computing

In this chapter we address the issue of implementation of row-action algorithms on parallel computer architectures. The nature of these algorithms — often in conjunction with the structure of the application — makes them suitable for decomposition into independent tasks. The decomposition is sometimes facilitated by the structure of the mathematical algorithm. For example, simultaneous algorithms (as characterized in Chapter 1) decompose naturally for parallel computations. Sometimes the decomposition is facilitated by the structure of the problem. In image reconstruction, for example, we can envision the partitioning of a discretized image into domains that are reconstructed independently from each other.

The discussion in this chapter concentrates first on machine-independent issues. The potential parallelism of an algorithm or an application is analyzed without special attention to the architecture of any particular system.

But we then proceed to discuss details of mapping the algorithm/problem onto specific architectures. Computational results are presented that illustrate the effectiveness of alternative implementations.

There are two reasons why row-action algorithms have become very attractive methods for parallel optimization. First, due to their row-action nature — that is, working with only one row of the constraints matrix at a time — it has been possible to devise specific implementations that decompose naturally for parallel processing. Second, the large scale of the applications where the row-action algorithms have traditionally been applied prompted users to look into the technological development with parallel computers in order to advance their modeling capabilities.

We review first some general principles of parallel computations. For each of the applications presented in Chapter 6 we select a suitable algorithm, and discuss its potential for parallel implementation. Finally, we present a summary of empirical results to illustrate the performance of each algorithm when implemented on a suitable parallel architecture.

7.1 Introduction to Parallel Computing

Parallelism in computer systems is not a recent concept. Indeed, ENIAC — the first electronic digital computer built at the University of Pennsylvania in Philadelphia between 1943 and 1946 — was designed with multiple functional units for adders, multipliers and so on. The primary motivation behind this design was to deliver the computing power required by the applications but not yet feasible with the electronic technology of that time. The shift from vacuum tubes to transistors, integrated circuits, and very

large scale integrated circuits (VLSI) rendered parallel designs obsolete and uniprocessor systems reigned throughout the seventies.

The first milestone in the evolution of parallel computers was the Illiac IV project at the University of Illinois in the 1970's. A brief historical note on this project can be found in Desrochers [74]. The array architecture of the Illiac prompted studies on the design of suitable algorithms for scientific computing. Interestingly enough, a study of this sort was carried out for linear programming [186] — one of the first studies in parallel optimization. The Illiac never went past the stage of the research project however and only one machine was ever built.

The second milestone was the introduction of CRAY 1S in 1976. The term *supercomputer* was coined at that time to indicate the fastest available computer. The CRAY 1S is not manufactured any more and is not the fastest machine available. It still serves, however, as the yardstick against which supercomputers are measured. The vector architecture of the CRAY introduced the notion of *vectorization* of scientific computing. Designing or restructuring of numerical algorithms to exploit the innovative computer architecture — in this case vector registers and functional units — became once more a critical issue. Vectorization of an application could range from simple modifications of the software implementation with the use of computational kernels, streamlined for the machine architecture, to more substantive changes in data structure and the design of algorithms that are rich in vector operations.

Since the mid-seventies supercomputers and parallel computers have been evolving rapidly in the level of performance they can deliver, the size of memory available, and the increasing number of parallel processors that can

be applied to a single job. At the same time such equipment are becoming widely accessible to the scientific and academic community in North America, Europe and the Far East; see for example Duff [77]. Supercomputers are usually multimillion dollar systems based on state-of-the-art electronic circuitry and cooling systems, installed at a few sites worldwide and usually accessed through high speed networks. Only very few systems qualify for this title. On the other hand, parallel computers are usually designed with off-the-shelf processing units communicating either through shared memory or via some message passing network. Their price is usually well below $1 million. Several manufacturers are marketing parallel systems which are more accessible to researchers and practitioners. Dongarra and Duff [75] provide a comprehensive list of high performance computers that is periodically updated as new products enter — and leave — the market. Textbook introductions to parallel processing systems are given in Desrochers [74] Hockney and Jesshope [125] and a collection of papers is available in the tutorials by Hwang [129] or Kowalik [144].

The proliferation of parallel hardware is having a significant impact on several areas of scientific computing. Researchers can now simulate complex phenomena and analyze alternative hypotheses without resorting to expensive and time consuming experimentation. Furthermore, computational experiments are carried out without exhausting one's computer budget and patience. Vector and parallel computing is having a significant impact in fields like weather forecasting, aircraft and automobile design, image processing, astrophysics, quantum mechanics and molecular chemistry, oil reservoir simulation, seismology, computer animation, manufacturing, electronic circuit design and so on. Buzbee and Sharp [34] offer a perspective on the state

of supercomputers and their usage. Studies on the strategic importance of supercomputing, its potential for further development and its influence on technology and international cooperation (or competition) are provided by Karin and Smith [140] or Lazou [147].

7.1.1 Models of Computation: Flynn's Taxonomy

There is a lot of flexibility in how a parallel computer architecture can be realized. Indeed, this flexibility has hampered acceptance of parallel computers by a broad spectrum of industrial and business users, since no *standard* parallel architecture seems possible. Before we proceed with a classification of computer architectures it is important to understand the distinction between a *supercomputer* and a *parallel* computer.

Supercomputer is defined as the fastest machine at any point in time. Some computers, however, may be very efficient for some tasks (e.g., array processing), while they lack substantial capability for others (e.g., list processing). Hence, this definition is rather vague.

Parallel computer refers to a class of computer architectures, with multiple processing units. Detailed classifications are given below. A parallel computer does not automatically qualify as a supercomputer. However, it is widely accepted that improved supercomputing performance can only be achieved using parallel architectures.

A broad classification of parallel architectures was offered in Flynn [93]. While several extensions have been added, see, for example, Hockney and Jeeshope [125], Flynn's taxonomy is still fundamental in understanding par-

allel architectures. He identified four classes, depending on the interaction of the instructions of a program with the data of the problem:

SISD (*Single Instruction stream Single Data stream.*) Systems in this class execute a single instruction on a single piece of data before moving on to the next instruction and the next piece of data. Traditional uniprocessor, scalar, computers fall under this category.

SIMD (*Single Instruction stream Multiple Data stream.*) A single instruction can be executed simultaneously on multiple data. This of course implies that the operations of an algorithm are identical over a set of data, and that data can be arranged for concurrent execution. This is one type of parallel computing and some of the successful instances of SIMD machines are the Connection Machine CM-2, the Active Memory Technology DAP and Masspar.

MISD (*Multiple Instruction stream Single Data stream.*) Multiple instructions can be executed concurrently on a single piece of data. This form of parallelism has not received, to our knowledge, extensive attention from researchers. It appears in Flynn's taxonomy for the sake of completeness.

MIMD (*Multiple Instruction stream Multiple Data stream.*) Multiple instructions can be executed concurrently on multiple pieces of data. The majority of parallel computer systems fall into this category. Multiple instructions indicate the presence of independent code modules that may be executing independently from each other. Each module may be operating either on a subset of the data of the problem, have

copies of all the problem data, or access all the data of the problem together with the other modules in a way that avoids read/write conflicts. The Intel hypercubes, the CRAY X-MP and Y-MP, the Alliant, the Convex and the Connection Machine CM–5 are MIMD systems.

A mode of computing that deserves special classification is that of *vector computers*. While vector computers are SIMD machines they constitute a class of their own. This is due to the frequent appearance of vector capabilities in many parallel systems and most supercomputers. Also the development of algorithms or software for a vector computer — like, for example, the CRAY — poses different problems than the design of algorithms for a system with multiple processors that operate synchronously on multiple data — like, for example, the Connection Machine. The processing elements of a vector computer are equipped with functional units that can operate efficiently on long vectors. This is usually achieved by segmenting the functional unit so that arrays of data can be processed in a pipeline fashion. The pipeline units are loaded either directly through memory or via vector registers. Furthermore, multiple functional units may be available both for scalar and vector operations. These functional units may operate concurrently or *chained*, with the results of one unit being fed directly into another without need for memory access. Using these machines efficiently is a problem of structuring the underlying algorithm with (long) homogeneous vectors and arranging the operations to maximize chaining or overlap of the multiple units.

Another major classification of parallel machines is made based on their memory organization: *Shared memory* machines are those where multiple processors can access directly all the memory of the system. Communica-

tion among processors takes place by writing and reading data to and from this common memory. The algorithm designer has to ensure that no read or write conflicts arise (i.e., no two processors access the same memory location simultaneously) otherwise the integrity of the data can not be guaranteed. *Distributed memory* machines are those where each processor has direct access only to some local memory. Communication among processors takes place by exchanging messages through some communication network. Accessing the local memory is a very efficient operation, while communication among processors is much more expensive. Distributed memory systems are equipped with sophisticated routing algorithms that direct messages from the sending to the receiving processors. Nevertheless, it is the algorithm designer who has to specify which processors communicate at different phases of the execution of an algorithm.

7.1.2 Some Unifying Concepts of Parallel Computers

Flynn's taxonomy created the impression that a uniform model of parallel computer architectures were not possible. However, some recent concepts allow end-users to a get a uniform view of a parallel computer system, independently of the underlying architecture. We discuss these concepts here. Going a step further, we have also seen the emergence of models for parallel programming that are independent of the underlying hardware: *control-level* and *data-level* parallelism. These concepts are also discussed in this section.

Single Program Multiple Data, (or, SPMD). The distinction between MIMD and SIMD gives way to the unified notion of SPMD. Parallelism can be viewed as the operation of a single program on multiple sets

of problem data. Each data set is operated upon by its very own processing elements.

Using a linear programming solver to solve multiple related instances of a linear program — as, for example, in the analysis of multiple scenarios in a portfolio management model — is an SPMD application. Whether each solver executes exactly the same pivot steps or not is irrelevant to the programmer. Of course an SIMD computer would impose this restriction, and could be inefficient if each solver should execute different pivot steps. If most of the solvers execute identical steps, efficiency will be substantially improved. An MIMD computer might be more efficient. But the main point is that the parallel program looks, at least to the user, the same for both computers.

SPMD has been motivated by the computing paradigm of SIMD architectures, as introduced by Hillis [124] for the Connection Machines CM–1 and CM–2. However, it is likely that MIMD architectures will be more effective environments for the execution of SPMD applications. The advantage of SPMD is that the user need not worry about the details of the computer architecture.

Distributed Virtual Memory, (or, DVM). The distinction between distributed memory and shared memory architectures becomes less significant with the introduction of the notion of distributed virtual memory. This idea was originated by software systems that facilitate the development of parallel applications on distributed, and possibly heterogeneous, computing environments. Examples of such systems are LINDA from Scientific Computing Associates [38], Parasoft EXPRESS [185] or

PVM [21].

Within a DVM environment a distributed architecture can be viewed as a system with (shared) virtual memory. The algorithm designer, of course, has to deal with the problem of data integrity to avoid incorrect results. However, the fact that memory is implemented in a distributed environment is of no direct consequence to him. It is the developer of the distributed computing environment who has to keep track of global objects. Where is a global object (like a common variable) stored? One approach would be to distribute all global objects to local memories. Another approach would be to distribute the address of all global objects to local memories, and let each processor fetch the data only if and when it is been used by the application. Of course, these designs tradeoff space with time efficiency. Once more, however, the analyst need not be concerned about the shared/distributed memory distinction. Indeed, systems like LINDA, EXPRESS or PVM have made it possible to port applications between shared memory machines, distributed memory machines and heterogeneous networks of workstations with little effect on performance and no reprogramming effort. See Fig. 7.1 for the performance of a simulation model for pricing financial instruments across a variety of computer platforms and parallel architectures, Cagan et al. [36].

Heterogeneous Parallel Processing. In the early days of parallel processing — and in the quest for a "winner" parallel computer — the debate on the right architecture was quite heated and often dogmatic. With the emergence of unifying computing paradigms — like SPMD

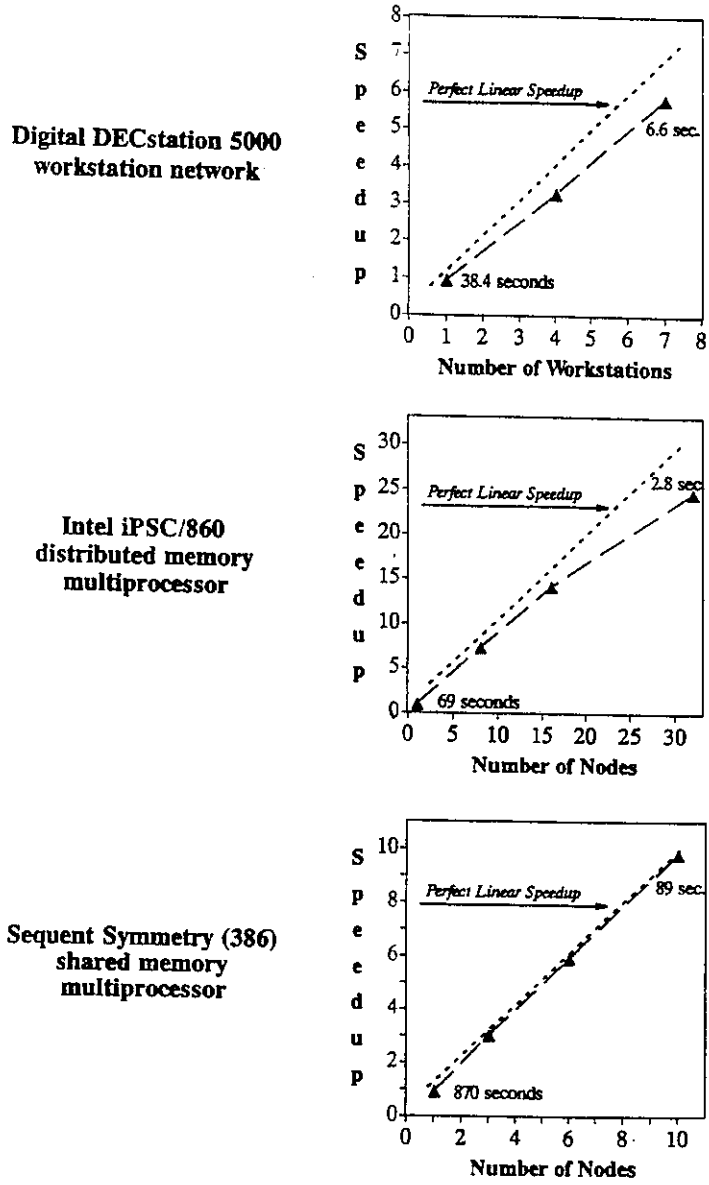


Figure 7.1: Distributed computing performance of a simulation model across different computer platforms and parallel architectures.

and the distributed virtual memory — the notion of heterogeneous parallel processing is finding widespread acceptance. By this term we mean a parallel computing environment consisting of multiple machines — some of which could be parallel processors themselves — linked together through some communication network or a fast switching device.

For example, networks of workstations can be linked together in a most cost-effective form of parallel processing. Fiber optics provide data-transfer rates (100 Mbits/sec) that were until recently found only on tightly coupled systems. Software systems like LINDA or EXPRESS allow users to decouple their problems in distributed virtual memory environment. Multiple workstations will then grab pieces of work from this environment, complete them at their own pace (depending on workload, and the performance of each workstation) and return the results. Of course there is no reason to assume that all the servers on the network are identical workstations, or even just workstations. Some of the servers could be more advanced parallel architectures. For example, an array processor attached to the network could be used to execute any linear algebra calculations, while workstations could be executing the less homogeneous operations. The results illustrated in Fig. 7.1(a) were obtained on a heterogeneous network of DECstations.

Models of Parallel Programming

We now turn to the two distinct types of parallel programming: *control parallelism* and *data parallelism.* These modes of parallel programming should not be confused with the variety of parallel computer architectures charac-

terized by Flynn [93] in his taxonomy of computer architectures. In this section we discuss models for writing parallel programs. These models are independent of the hardware on which the programs are written.

Control-Level Parallelism

In *control parallelism* a program is decomposed into instructions that are independent of each other. These instructions can be executed concurrently. A well-known type of control parallelism is *pipelining*. Consider, for example, the simple operation

$$Y \leftarrow \alpha X + Y$$

where X, Y are n-dimensional vectors and α is a scalar. (This is known as a SAXPY.) On a computer with two functional units (an adder and a multiplier) it is possible to pipeline the αX multiplication with the $+Y$ operation. As soon as the first product result becomes available both the adder and the multiplier will be kept busy. Assuming that the adder and multiplier operate on scalars, and that addition and multiplication require the same number of machine cycles, then pipelining the two operations will complete the SAXPY twice as fast than a machine with a single functional unit. The maximal improvement in performance that can be gained from the control parallel execution of a SAXPY is a factor of two.

The performance of control parallelism is bounded by the number of independent instructions. Even a small fraction of the code that has to be executed serially will give rise to a bottleneck. We know, from Amdahl's law [6], that if a fraction s of the execution time of an algorithm is serial, while the remaining $1 - s$ is parallelized on P processors, then the speedup

of the application is bounded from above by $1/s$; see equation (7.1).

Data-Level Parallelism

In *data parallelism* the same instruction (or, similar instructions) are performed on many data elements simultaneously by many processors. For example, the SAXPY operation on n-dimensional vectors can be completed in just two steps using n processors : one step will execute the product αX for all components of X simultaneously and the next step will execute $+Y$ on all components of Y simultaneously. The improvement in performance is, in this case, proportional to the size of the problem, i.e., n.

The performance of data parallelism is bounded by the number of data elements that can be handled concurrently, while the performance of control parallelism is bounded by the fraction of the instructions that have to be executed serially. The argument in favor of data parallelism is the following: As problems get bigger the number of data that need to be operated upon increases, while the complexity of the algorithm/program remains unchanged. Hence, more and more processors can be utilized efficiently for large scale problems in the case of data parallelism.

Early applications of data parallel programming were implemented on SIMD architectures, like the Connection Machine CM–2, Active Memory Technology DAP and Masspar. Hence, identical instructions had to be executed on all selected vector memory addresses. With the introduction of single-program multiple-data systems, like the Connection Machine CM–5, it is possible for the vector processor to decode its own instructions. This allows for somewhat inhomogeneous operations to be carried out at different addresses of the vector memory. For example, a *linear-programming-solve*

can be an instruction sent by the controller to the parallel vector processor. Multiple linear programs could be stored in vector memory and the vector processor could execute the same instruction on selected memory locations. The fact that *linear-programming-solve* will consist of several instructions executing a simplex method or an interior-point algorithm is transparent to the user. So is the fact that, most likely, a different sequence of simplex or interior-point steps will be executed for each vector. When the notion of *instruction* in its classical meaning (i.e., add, multiply, fetch) is replaced by a sequence of such simple instructions (i.e., by a program), executed in lock-step, we refer to the underlying parallel machine as *Single Program Multiple Data* (SPMD). This designation distinguishes the parallel computer realization from the simpler — but more restrictive — SIMD architecture, or the more general — but more complex — MIMD architecture.

7.1.3 Parallel Prefix Operators for Data-Parallel Computing

The fact that identical (or very similar) operations need to be executed in the data parallel model might imply that only highly-regular numerical algorithms are suitable for this form of parallelism. That this need not be the case can be shown using an abstract model of a data parallel computer: the *Vector-Random Access Machine* (V-RAM) of Blelloch [28].

The V-RAM is a standard RAM with the addition of vector memory and a parallel vector processor, Fig. 7.2. The vector memory is a sequence of locations containing linearly ordered collections of scalar values, known as *simple vectors*. It is an important feature of this model that the vector lengths need not be identical. This is one of the major distinguishing features of data parallel machines from vector computers. The vector processor exe-

cutes primitive instructions (in parallel) on sets of simple vectors and scalars from the vector and scalar memories. For example, the SAXPY operation corresponds to the multiplication of a vector X in the vector memory by an element α from scalar memory, and the addition of the result to a second vector Y in vector memory.

Blelloch introduced two classes of vector instructions: *scans*, or parallel prefix operations, and *segmented scans*. He showed that a V-RAM endowed with these two instructions could implement a wide variety of applications, while maintaining a level of parallelism proportional to the amount of data in the problem. (In particular, the level of parallelism is proportional to the length of the parallel vector processor of a specific realization of a data parallel machine.) An equally important outcome of the V-RAM model is that it permits the complexity analysis of data parallel algorithms, but this issue is not addressed here.

Parallel Prefix Operations

The data-parallel implementations of some of the algorithms discussed later make use of the parallel prefix operators. The two basic prefix operators are the *scan* and *spread*.

The $\otimes$-scan primitive, for an associative binary operator $\otimes$ takes a sequence $\{x_0, x_1, \ldots, x_n\}$ and produces another sequence $\{y_0, y_1, \ldots, y_n\}$ such that $y_i = x_0 \otimes x_1 \otimes \ldots \otimes x_i$. These sequences are referred to as *parallel variables*, as their elements reside on different processing elements of the parallel computer. The $\otimes$-spread primitive, for an associative binary operator $\otimes$, takes a sequence $\{x_0, x_1, \ldots, x_n\}$ and produces another sequence $\{y_0, y_1, \ldots, y_n\}$ such that $y_i = x_0 \otimes x_1 \otimes \ldots \otimes x_n$. In the implementations

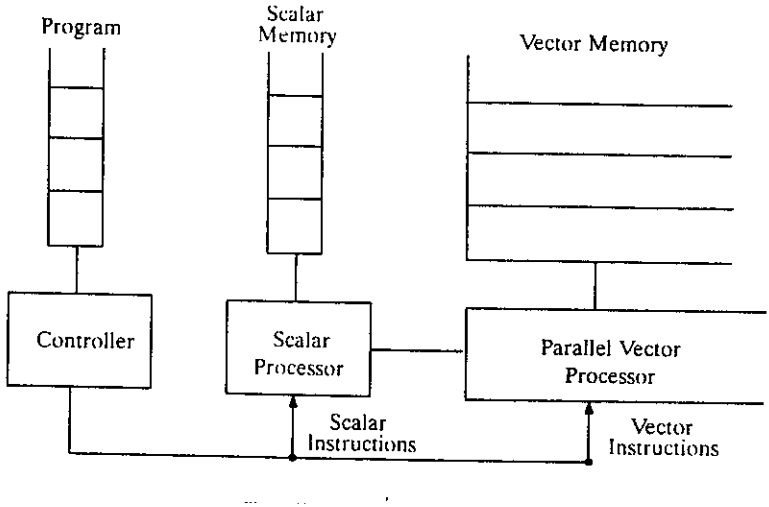


Figure 7.2: The Vector – Random Access Machine (V-RAM) model of Blelloch for data parallel computers.

of these primitives on parallel machines (like, for example, the Connection Machine CM–2 or MassPar) the communication network facilitates the exchange of data between processing elements during the calculation of the $\otimes$ operator. However, the parallel prefix operators hide from the user the details of the hardware and the communication network.

Another variation of the scan primitives allows their operation within *segments* of a parallel variable. These primitives are denoted as *segmented-$\otimes$-scan*. They take as arguments a parallel variable and a set of segment bits which specify a partitioning of the elements of the variable into contiguous segments. Segment bits have a 1 at the starting location of a new segment

Processing Element	= [	0	1	2	3	4	5	6	7	8	9]
X	= [	5	1	3	4	3	9	2	6	1	0]
Segment Bits (Sb)	= [	1	0	1	0	0	0	1	0	0	1]
$Y = add - scan(X, Sb)$	= [	0	5	6	3	7	10	19	2	8	9]
$copy - scan(Y, Sb)$	= [	0	0	0	6	6	6	6	19	19	19]
$reverse - copy - scan(Y, Sb)$	= [	6	6	19	19	19	19	9	9	9	9]

Figure 7.3: The Segmented *add-scan and copy-scan* primitives.

and a 0 elsewhere. A *segmented-⊗-scan* operation re-starts at the beginning of every segment. Fig.7.3 illustrates the use of *segmented-add-scan* and *segmented-copy-scan* on a small example. When processors are configured as a two-dimensional grid, scans within rows or columns are special cases of segmented scans called *grid-scans.*

7.1.4 Measures of Performance

How does one evaluate the performance of an algorithm executing on a parallel computer? There is no simple answer to this question. Instead, several measures of performance are usually employed. Depending on the objective of the experimental study — that is, of the algorithm implementation — different measures may be more appropriate [76, 137].

Definition 7.1.1 (Speedup.) *Speedup (or "speedup factor") is the ratio of solution time of the algorithm executing on a single processor, to the solution time of the same algorithm when executing on multiple processors. It is, of*

course, understood that the sequential algorithm is executed on one of the processors of the parallel system.

Linear speedup is observed when a parallel algorithm on p processors runs p times faster than on a single processor. *Sublinear* speedup is achieved when the improvement in performance is less than p due to the presence of sequential segments of code, processor synchronization, overhead in spawning independent tasks and so on. *Superlinear* speedup is unusual. It indicates that the parallel algorithm takes a different — and more efficient — solution path than the sequential algorithm. It is often possible in such situations to improve the performance of the sequential algorithm based on insight gained from the parallel algorithms.

Amdahl's law [6] specifies that the speedup when moving from a uniprocessor machine to one with P processors is given by

$$\text{Speedup} = \frac{1}{s + p/P}, \tag{7.1}$$

where s is the fraction of serial execution of the application, and p is the fraction of execution time that can be performed in parallel ($s + p = 1$). Even with an infinite number of processors, speedup can not exceed $1/s$. As scientists observed that 10% of a typical application could not be parallelized (input/output, initializations, serial bottlenecks of the algorithm) they concluded that a speedup of 10 was the best one could expect.

This limited view of parallelism has today few followers. Parallel computers are not just used to solve 10 times faster an existing application. Instead, they were used to solve in about the same time 1000-fold bigger instances of the same problem. When an application is scaled in size, to fit in

the larger number of processors, the serial part usually remains unchanged. What scales up is the parallel part. Hence, linear speedups can be expected. Consider a problem with a serial execution part s and a parallelizable part p. When a P processor system becomes available, the application would be scaled. If the parallel part would scale linearly then the execution time for the larger application would be $s + p \cdot P$. A modified Amdahl's law, due to E. Barsis from Sandia, is given in Gustafson [106]:

$$\text{Scaled speedup} = s + p \cdot P.$$

Definition 7.1.2 (Efficiency.) *Efficiency is the ratio of speedup to the number of processors.*

Efficiency provides a convenient way to measure the performance of an algorithm independently from the level of parallelism of the computer architecture. Linear speedup corresponds to 100% (or 1.00) efficiency. Factors less than 1.00 indicate sublinear speedup while superlinear speedup is indicated by factors greater than 1.00.

Definition 7.1.3 (MFLOPS.) *The acronym stands for* Million Floating-point Operations per Second. *Observed MFLOPS rates are considered the yardstick for the evaluation of algorithms on supercomputers.*

All measures have merit and provide insightful information. At the same time they provide only a partial measure of performance. High speedup for example with an inherently slow algorithm is of little use when an alternative algorithm exhibits much faster convergence when executing on a uniprocessor. Similarly, high MFLOPS rate with an algorithm that is performing

redundant calculations is not as meaningful as reduced MFLOPS rate by an algorithm that requires fewer operations.

7.2 Parallel Computing for Matrix Balancing

We now discuss parallel computing techniques for the implementation of algorithms for the matrix balancing problem. We focus in particular on Problem 6.1.3, and we look at alternative ways to parallelize the **RAS** Algorithm 6.1.2. The application of **RAS** to this problem is well suited for both control-level and data-level parallel implementations. Computational results with the alternative implementations are reported.

The control-parallel implementation of **RAS** was developed in Zenios and Iu [240] and the data-parallel implementation in Zenios [233]. The Range-RAS **RRAS** algorithm has also been implemented using data-level parallelism, and the implementation as well as computational results are reported in Censor and Zenios [51].

7.2.1 Control-Parallel Computing with RAS

The **RAS** algorithm 6.1.2 iterates by scaling the rows of the matrix (Step 1) before it proceeds with the scaling of the columns of the matrix (Step 2). It is easy to observe that the row-updates are independent from each other. Each row can be updated given the current value of the row entries, and the target row sum. Hence, multiple rows can be scaled simultaneously and in parallel. Once all row-scaling operations are completed the algorithm may proceed with the simultaneous and parallel scaling of the columns.

How, precisely, is the control of the algorithm arranged for parallel com-

putations leaves some room for experimentation. At the one extreme the algorithm may consider each row-scaling operations as a task that can be scheduled for execution on the next available processor. If there are more rows than there are processors then each processor will process the next available task. The advantage of this approach to *task scheduling* is that *load balancing* is achieved. That is, processors will terminate in approximately the same time: slower processors will process fewer rows. If there are some dense rows in the matrix then those processors assigned to them will process fewer rows. The disadvantage of this approach is that processors must inquire about the availability of tasks and initiate a task. On some computers the overhead from task initiation could be substantial.

At the other extreme of task scheduling one could consider grouping rows together into as many groups as there are processors, and assign one group to each processor. This approach requires only one task initialization, and hence reduces the overhead of the implementation. The size of each group — i.e., number of rows — has to be adjusted for the speed of the processors and the number of non-zero entries in each row.

An alternative approach to the parallel solution of Problem 6.1.3 is to work with a simultaneous version of **RAS**. Such a version will work concurrently on both rows and columns of the matrix. It is stated as follows:

Algorithm 7.2.1 Simultaneous RAS Algorithm.

Input: An $m \times n$ nonnegative matrix $A = (a_{ij})$, a positive vector $u = (u_i) \in \Re^m$ and a positive vector $v = (v_j) \in \Re^n$.

Step 0: (Initialization) Set $k = 0$ and $X^0 = A$. Choose $\lambda_\rho, \lambda_\sigma \in (0, 1)$.

Step 1: (Compute Row Scaling Factors) For $i = 1, 2, \ldots, m$ define

$$\rho_i^k = \left(\frac{u_i}{\sum_j x_{ij}^k} \right)^{\lambda_\rho}, \tag{7.2}$$

Step 2: (Compute Column Scaling Factors) For $j = 1, 2, \ldots, n$ define

$$\sigma_j^k = \left(\frac{v_j}{\sum_i x_{ij}^k} \right)^{\lambda_\sigma} \tag{7.3}$$

Step 3: (Update the matrix) For $i = 1, 2, \ldots, m$ and $j = 1, 2, \ldots, n$ define

$$x_{ij}^{k+1} = \rho_i^k x_{ij}^k \sigma_j^k. \tag{7.4}$$

Step 4: Replace $k \leftarrow k + 1$ and return to Step 1.

The simultaneous **RAS** algorithm can be implemented in parallel in a way similar to **RAS**. That is, multiple processors could compute the scaling factors for multiple rows. Then the scaling factors for multiple columns will be computed in parallel. And finally the updating of the matrix entries can be performed in parallel for all entries. We note however that the simultaneous **RAS** can better utilize a larger number of processors, since both rows and columns can be iterated upon in parallel.

Computational Results

We report now computational results with the parallel implementation of both **RAS** and simultaneous **RAS** on the Alliant FX/8. The Alliant FX/8 is a shared-memory vector multiprocessor, with 8 processors. Each processor has vector functional units. The implementations take advantage of the vector architecture.

Problem	No. of rows × columns	No. of non-zero entries
USE537	504 × 473	57247
MAKE537	523 × 533	9586
MRIO	(approx.) 6000 × 6000	(approx.) 370000

Table 7.1: Test problem characteristics.

Test problems were derived from regional input/output accounts. The first source is the National table for the U.S. for 1977 [32]. This set of data consists of the "Make" and "Use" matrices, with a classification scheme of 537 sectors. The second source is the multiregional input/output accounts of the U.S. for 1977 [166]. This set of accounts consists of input/output tables for the 50 States and the District of Columbia, using an industrial classification scheme of approximately 120 sectors. For our test problem we assembled a table of Make submatrices along the diagonal, and inter-regional trade flows off the diagonal that provide coupling between the regions. The resultant table has approximately 6000 (= 50 × 120) accounts and over 370000 nonzero transactions, both inter- and intra-regional.

To create balancing problems based on the available data we computed the row and column sums of each test problem, and then added noise to the entries of the matrix. Thus we obtained tables whose entries do not add up to the computed control totals. The characteristics of the test problems are summarized in Table 7.1.

To provide a benchmark against which to evaluate the performance of the parallel algorithms we first run **RAS** on a VAX 8700 large mainframe

Problem	Error	VAX 8700		FPS M64/60	
		Iterations	CPU Time	Iterations	CPU Time
USE537	10^{-6}	31	0:00:10	25	0:00:1
	10^{-8}	46	0:00:15	41	0:00:2
	10^{-12}	78	0:00:26	73	0:00:4
MAKE537	10^{-4}	443	0:00:30	713	0:00:11
	10^{-6}	66827	1:13:06	22280	0:05:54
	10^{-8}	—	> 20:00:00	273701	1:12:28

Table 7.2: Benchmark solution times with **RAS** in hrs:min:sec.

and a Floating Point Systems M64/60 attached array processor. The results are summarized in Table 7.2.

Parallel Computing on the Alliant FX/8

The results on the Alliant FX/8 with the parallel implementation of **RAS** are summarized in Table 7.3. The algorithm was implemented on a single processor in a way that takes advantage of the vector capabilities of the Alliant.

The algorithm was then parallelized by allowing multiple row-scaling operations to be performed concurrently, before the algorithm would proceed with the concurrent scaling of multiple columns. Significant speedup is achieved with parallel computations using all 8 processors of the system.

Problem:	MAKE537	MAKE537	MAKE537	USE537
Error:	10^{-4}	10^{-6}	10^{-8}	10^{-12}
Iterations:	802	80057	342081	70
Scalar:	00:01:19.9	02:10:20.8	09:20:51.0	00:00:31.0
Vectorized:	00:00:51.3	01:23:13.4	05:55:15.9	00:00:18.5
2-CPUs:	00:00:25.7	00:41:37.6	02:59:33.9	00:00:09.5
3-CPUs:	00:00:17.5	00:28:13.0	02:00:46.9	00:00:06.4
4-CPUs:	00:00:13.5	00:21:39.1	01:31:54.9	00:00:04.9
5-CPUs:	00:00:11.0	00:17:32.9	01:14:57.7	00:00:04.1
6-CPUs:	00:00:09.5	00:14:56.1	01:03:37.7	00:00:03.5
7-CPUs:	00:00:08.3	00:13:04.0	00:55:39.6	00:00:03.1
8-CPUs:	00:00:07.5	00:11:45.0	00:49:45.9	00:00:02.7

Table 7.3: Solution time on the Alliant FX/8 in hrs:min:sec.

The average speedup factor — average for the solution of MAKE537 to three different levels of accuracy, and for the solution of USE537 — is shown in Fig. 7.4 as a function of the number of processors. The speedup is almost linear. The small inefficiency (around 10%) is due to the overhead in initializing the tasks and the convergence test which is a sequential operation.

Using the simultaneous **RAS** with weights $\lambda_\rho = \lambda_\sigma = 0.8$ in the multiprocessing environment and comparing it with the uniprocessor implementation of **RAS** we obtained the speedup curve of Fig. 7.5. Although it appears that the speedup is superlinear, a more careful examination of the results indicated that the simultaneous **RAS** is more efficient than **RAS** even on a single processor. Were we to develop the speedup curve based

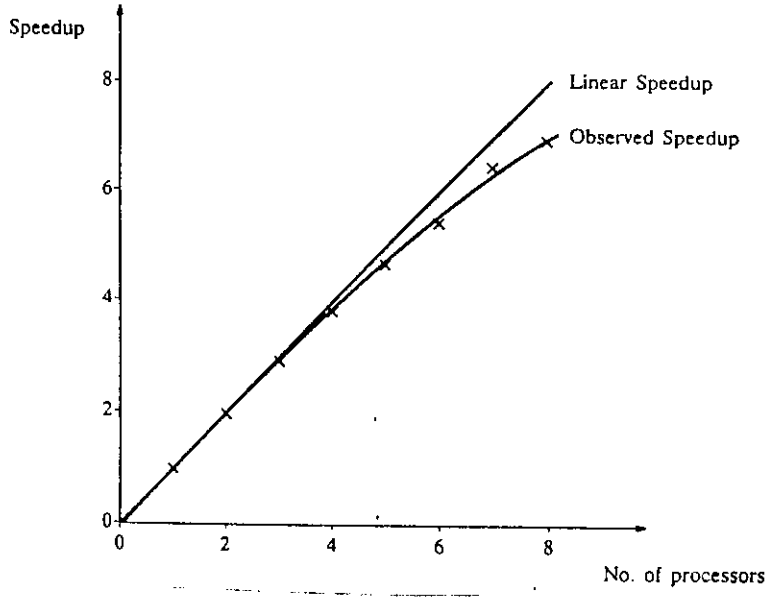


Figure 7.4: Speedup factors *vs* no. of processors on the Alliant FX/8 with the parallel implementation of **RAS**.

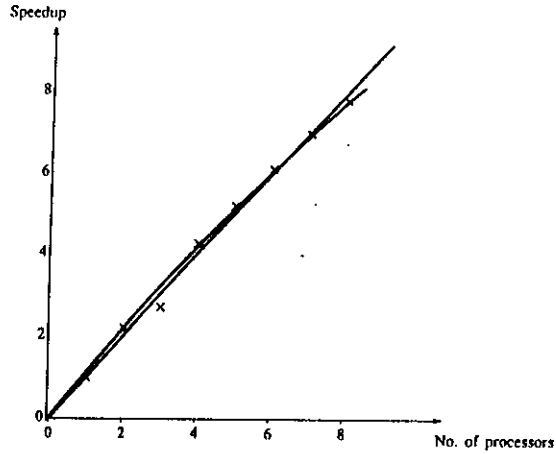


Figure 7.5: Speedup factors with simultaneous **RAS** on the Alliant FX/8.

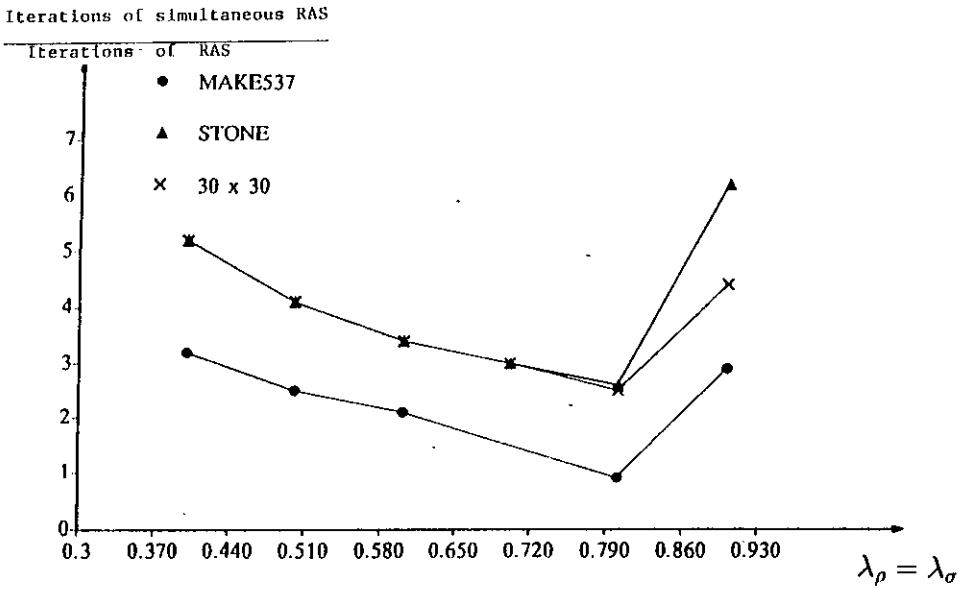


Figure 7.6: The performance of simultaneous **RAS** compared to the performance of **RAS** for varying values of the parameters $\lambda_\rho = \lambda_\sigma$.

on a comparison of the parallel simultaneous **RAS** with the uni-processor implementation of the simultaneous **RAS** we would obtain a curve identical to that of Fig. 7.4. Nevertheless, we present the results of Fig. 7.5 since it is the experimentation with parallel computing that revealed the potential of the simultaneous **RAS** as a more efficient sequential algorithm.

Fig. 7.6 illustrates the relative performance of simultaneous **RAS** with respect to **RAS** with varying parameters λ_ρ, λ_σ for three test problems: STONE, MAKE537 and a randomly generated 30×30 matrix. These results were obtained on the VAX 8700. We observe that a fair amount of fine tuning is required in order to find the set of parameters that make the simultaneous **RAS** outperform **RAS**.

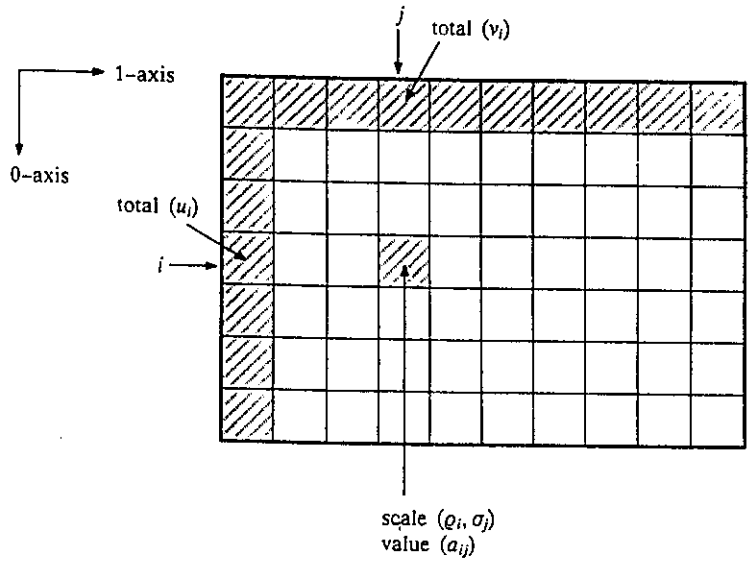


Figure 7.7: Data-parallel representation of a dense matrix.

7.2.2 Data-Parallel Computing with RAS

The **RAS** algorithm for matrix balancing problems has a natural map to data-parallel computations. In particular, each element of the matrix can be assigned to a distinct processing element. Processing elements that store the data of a row will coordinate to compute a row-sum, and then proceed with the scaling of the row-entries. Multiple rows can be updated concurrently, and the implementation will proceed similarly to update column sums. It is convenient to think of the processing elements as being arranged into a two-dimensional grid, with each element connected to its North, East, West, and South neighbors. Fig. 7.7 illustrates the mapping of data onto a two dimensional grid of processing elements.

Data-Parallel Implementation for Dense Problems

The data-parallel implementation of **RAS** for dense problems uses the mapping of data to processors as illustrated in Fig. 7.7. Every processor stores the matrix entry a_{ij} and the scaling factor. In addition, both the row total u_i and the column total v_j are stored at the same processor. Hence the (i,j)-th processor stores a_{ij}, u_i, v_j and the scaling factor.

With this mapping of data to processors the **RAS** algorithm is implemented as follows: First, use an add-spread parallel prefix operation along the vertical $(0-th)$ axis of the processors to spread the partial sum of the matrix entries in each column to some temporary memory location of each processor in the column. Each processor then proceeds to divide the column total v_j by the partial sum, therefore computing its own value of the scaling factor. All processors in the same row will compute identical scaling factors at this point, but they do so locally without any need for communications. The implementation uses redundant computations in order to eliminate communication overhead. Finally, each processor uses its local copy of the scaling factor to multiply its entry of the matrix. The same operations are then repeated along the horizontal $(1-st)$ axis of the processors to carry out the row scaling.

Data-Parallel Implementation for Sparse Problems

To represent sparse problems we will make use of the segmented parallel prefix operations. The non-zero entries of the matrix are arranged along multiple processors in a one-dimensional array. The non-zero entries are arranged both row-wise and column-wise. Segment bits are used to denote

contiguous segments that contain the data of complete rows and columns. Every non-zero entry of the matrix is, therefore, stored twice: once in a row-wise format that allows row scaling and once in a column-wise format that allows column scaling. This scheme has some redundancy, but allows the efficient use of segmented-scan operations. This scheme for representing sparse matrices was proposed by Blelloch [28]. Zenios and Lasken [232] proposed, independently, a similar scheme for representing sparse network problems. The two schemes are equivalent, if one associates with a sparse matrix an incidence graph. That is, with each row or column of a matrix associate a node and with a non-zero entry introduce an arc between the corresponding row-node and column-node. See, for example, definitions 6.1.1 and 6.1.2. Blelloch's scheme for representation of a sparse matrix is identical to the Zenios-Lasken scheme for the representation of a network. A comparison of alternative data- structures for the representation of network problems is given in Nielsen and Zenios [176].

Fig. 7.8 illustrates the sparse representation of a small matrix. The various data stored at each processor are described below:

1. A field *s-row* of segment bits. It is used to partition the processors into contiguous segments such that processors in the same segment correspond to entries of the same row.

2. A field *a-row* that holds the value of the entry a_{ij}. The matrix entries are allocated row-wise: entries of the same row belong to the same segment.

3. A field *r-total* holds the row total values u_i. All processors in the same row segment hold identical values.

	1	2	3	4	5	
1		a_{12}	a_{13}	a_{14}	a_{15}	u_1
2	a_{21}					u_2
3	a_{31}					u_3
4		a_{42}	a_{43}			u_4
5		a_{52}	a_{53}	a_{54}		u_5
	v_1	v_2	v_3	v_4	v_5	

NEWS address

of VP	1	2	3	4	5	6	7	8	9	10	11	12
S-ROW	1	0	0	0	1	1	1	0	1	0	0	1
A-ROW	a_{12}	a_{13}	a_{14}	a_{15}	a_{21}	a_{31}	a_{42}	a_{43}	a_{52}	a_{53}	a_{54}	
R-TOTAL	u_1	u_1	u_1	u_1	u_2	u_3	u_4	u_4	u_5	u_5	u_5	
P-ROW	3	6	9	11	1	2	4	7	5	8	10	
P-COL	5	6	1	7	9	2	8	10	3	11	4	
C-TOTAL	v_1	v_1	v_2	v_2	v_2	v_3	v_3	v_3	v_4	v_4	v_5	
A-COL	a_{21}	a_{31}	a_{12}	a_{42}	a_{52}	a_{13}	a_{43}	a_{53}	a_{14}	a_{54}	a_{15}	
S-COL	1	0	1	0	0	1	0	0	1	0	1	1

Figure 7.8: Data-parallel representation of a sparse matrix.

4. A second field of segment bits *s-col* partitions the processors into contiguous segments, such that processors in the same segment correspond to entries of the same column.

5. A field *a-col* holds the value of the entry a_{ij} in column-wise order. This field provides redundant information, since the non-zero entries have already been stored row-wise in field $a - row$.

6. A field *c-total* holds the column total values v_j. All processors that correspond to the same column (i.e., are in the same column segment) hold identical values.

7. A field *scale* holds the scaling factors for either row or column.

8. Two fields *p-row* and *p-col* hold pointers to map the non-zero entries from the row-wise field *a-row* to the column-wise field *a-col* and vice versa.

With this representation of sparse problems one iteration of **RAS** is executed as follows: A segmented-add-scan operation on field *a-row* (that is, an add-scan over the row segments) computes the sum of the entries in each row. This partial sum is then used to divide the row total field *r-total* thus computing the scaling factors. Note that — by the definition of the segmented-scan operator — only the last processor in each segment has the sum of all the entries in the row and hence only the scaling factor of the last processor is the correct one. A reverse segmented-copy-scan is used to copy the correct scaling factor to all processors in the same segment. Finally, each processor proceeds to multiply its copy of the matrix value with its local copy of the scaling factor. Before the algorithm can proceed with the

Test problems	Error	32K CM–2		Alliant FX/8	CRAY X-MP
		(sparse)	(dense)		
USE537	10^{-6}	0.45	0.18	1.04	0.30
MAKE537	10^{-4}	11.41	6.13	7.50	6.90

Table 7.4: Comparisons of execution times (seconds) of the data-parallel and control-parallel implementations of **RAS**.

column scaling the non-zero entries, just scaled following a row operation, have to be copied from the row-wise to the column-wise representation. This is a rearrangement of the contents of the memory field *a-row* according to the sparsity structure of the matrix. The addresses stored in the pointer field *p-row* are used to copy the non-zero entries of the matrix from the row field *a-row* to the column field *a-col*. The algorithm then proceeds with a column scaling, which is similar to the row scaling described above.

Computational Results on the Connection Machine CM–2

We summarize now computational results obtained using **RAS** on the Connection Machine CM–2. (For a brief description of the CM–2 see Appendix A.) The test problems are identical to those used for the control-parallel implementation of section 7.2.1. Table 7.4 summarizes the results, and compares the performance of the data-parallel with the control-parallel implementation, and against the benchmark implementation of the algorithm on a CRAY X-MP vector supercomputer.

7.3 Parallel Computing for Image Reconstruction

We discuss now the parallel implementation of algorithms for image reconstruction. We consider, in particular, the implementation of the **block-MART** Algorithm of Censor and Segman [50] applied to the entropy optimization model for image reconstruction, i.e. Problem 6.2.2. We propose three parallel variants of the algorithm. All three are logical special cases of the general **block-MART**. Therefore no separate mathematical analysis is needed. However, the three schemes demonstrate different speedup factors. This example serves as an indication of the flexibility of block-iterative algorithms for parallel computations. The implementation was carried out on a CRAY X-MP/48 which is a shared-memory vector multiprocessor. Special attention was placed on implementing the algorithm in a way that take advantage of the vector architecture.

The **block-MART** algorithm was implemented as the reconstruction algorithm within the SNARK77 package [161]. SNARK77 is a programming environment designed to facilitate the implementation and testing of reconstruction algorithms. In particular, based on user specifications of an image and the geometry of data collection, SNARK77 will create a discretized image and simulate the data measurements. This information is then passed on to the reconstruction algorithm that proceeds to reconstruct the image specified by the user. In this section we discuss alternative implementation that take advantage of the multiprocessor and vector capabilities of the CRAY. This material is based on the work of Zenios and Censor [238].

7.3.1 Vector Computing with Block-MART

The iterative step of the **block-MART** Algorithm 6.2.2 can be written in the form:

$$x_j^{k+1} = x_j^k \exp\left[\sum_{i \in I_{t(k)}} w_i^{t(k)} d_i^k a_{ij}\right]. \tag{7.5}$$

Instead of computing the term under the summation sign for each pixel j and then updating the corresponding x_j, we proceed as follows. For every pixel j we accumulate the terms

$$w_i^{t(k)} d_i^k a_{ij} \tag{7.6}$$

in a *correction* array. Once all the terms are accumulated we update the jth elements of x^k. This computational scheme can be computed in two intermediate steps — accumulating the correction terms and updating the vector x^k — that are rich in vector operations.

We formalize these two steps, omitting temporarily the iteration index k for the sake of simplicity of explanation. Given an iteration index k a block index $t = t(k)$ is chosen according to a cyclic rule. A list of all pixels intersected by rays that belong to the block I_t is given by $J_t(s)$, $s = 1, 2, \ldots, S_t$, where S_t is the number of elements in the set $\{j \in J \mid a_{ij} > 0\ ,\ i \in I_t\}$. $C_t(s)$ is a temporary correction array, associated with the tth block, for $s = 1, 2, \ldots, S_t$. If the total number of pixels intersected by the ith ray is L_i then $LIST_i(.)$ is an array of the indices of these pixels. $LIST_i(.)$ is itself indexed by $l = 1, 2, \ldots L_i$. Together with $LIST_i(.)$ we get from SNARK77 an array $LENGTH_i(.)$ which contains, for every $l = 1, 2, \ldots, L_i$, the length of intersection of the pixel $LIST_i(l)$ with the ith ray. For a given block, the correction array $C_t(.)$ is computed as follows:

Initialize:

$$C_t(s) \leftarrow 0\,, \quad s = 1, 2, \ldots, S_t. \tag{7.7}$$

Compute, for all $i \in I_t$:

$$C_t\left(LIST_i(l)\right) \leftarrow C_t\left(LIST_i(l)\right) + w_i^t d_i * LENGTH_i(l). \tag{7.8}$$

Updating the image vector x is computed by the iterative step, for all $s = 1, 2, \ldots, S_t$:

$$x_r \leftarrow \exp\left[\log x_r + C_t(s)\right], \tag{7.9}$$

where $r \in J_t(s)$. The computations in equations (7.7)–(7.9) are implemented using coded BLAS (i.e., Basic Linear Algebra Subroutines) kernels from the CRAY library, [58]. The kernels use the vector capabilities of each processor of the CRAY. Results with the scalar and vectorized codes are summarized in Table 7.5. All runs with **block-MART** were carried out with 45 blocks per view. We observe significant improvement in performance due to vectorization: the improvements range from a factor of 5.8 with row-action MART on the smaller problems to over 10 with **block-MART** on the larger problems. **block-MART** achieves slightly better performance than **MART** due to the longer vectors that appear in the **block-MART** calculations.

7.3.2 Parallel Scheme I: Parallelism Within a Block

The first parallel scheme deals with local changes in the algorithm and is therefore the easiest to implement. The operations performed on equations of a single block are structured for possible concurrent execution. The following four steps are executed during one iteration over the equations of a block:

Phantom	MART	Block-MART
HEAD64	452.86	77.53
SHEPP64	471.24	81.19
SHEPP512	3543.10	108.81
SHEPP1024	N/A	312.71

Table 7.5: Solution times for test phantoms on the CRAY X-MP/48 (CPU seconds.)

Step 1: Choose the rays that form the block using SNARK77 utilities. Given a user specified size of the block (i.e., the number of rays per block) SNARK77 will determine the rays that form this particular block. Recall that each ray gives rise to an equation which is a constraint in the optimization problem.

Step 2: For each ray i in the block determine the simulated measurement b_i, the list of pixels j intersected by it — $LIST_i(.)$ — and the lengths of the intersections a_{ij}. This information is computed by SNARK77 (using the utilities WRAY and PSEUDO) based on the geometry of data collection and the discretization of the phantom.

Step 3: Compute the correction terms $C_t(.)$ as discussed in Section 7.3.1.

Step 4: Update all pixel intensities x_j as discussed in Section 7.3.1.

These four steps are executed for each block and are repeated until some iteration number limit is reached or some convergence criterion is satisfied. Blocks are processed in a cyclical manner. It is possible to utilize multiple processors during the execution of some of the steps.

Step 1 is executed in a GUARDed mode, thus avoiding two or more processors from assigning the same ray to a block more than once. This step is not particularly time consuming. SNARK77 just goes through the list of rays and picks a user specified number of them according to a simple rule. For example, choose the first ray that has not been used already.

Step 2 is computationally intensive. SNARK77 uses the geometry of the phantom and the orientation of the rays to determine the list of pixels intersected by each ray and the lengths of intersections. The exact specifications of the phantom — known to SNARK77 but, of course, unknown to the reconstruction algorithm — are used to compute the simulated measurements. These operations can be executed concurrently for all rays in the block. The relevant SNARK77 utility routines (PSEUDO and WRAY) are declared *microtasked* and a copy of each is executed in parallel for multiple rays in a block. Computations are executed in parallel for up to four rays (i.e., the number of processors on the CRAY we use) at a time.

In Step 3 the correction term is computed for each ray. Again multiple rays can be processed in parallel. Finally, updating of the pixel intensities x_j using equation (7.9) is executed in a guarded mode. This avoids the updating of a single pixel by multiple processors when such a pixel is intersected by several rays that are operated upon in parallel. However, the use of the protected mode inhibits the vectorization of Step 3.

The performance of this parallel scheme was evaluated reconstructing the phantoms SHEPP512 and SHEPP1024 with the vectorized **block-MART** code using block sizes of 45 blocks/view. Results are summarized in Table 7.6. While this scheme demonstrates almost linear speedup factors the results of Table 7.6 are misleading: the microtasked code is significantly

No. of CPUs	Speedups	
	SHEPP512	SHEPP1024
2	1.97	1.83
3	2.97	2.95
4	3.72	3.97

Table 7.6: Speedup factors with parallel scheme I and no vectorization.

slower than the non-microtasked code when executed on a single processor. This is due to the way the GUARD statement is used during execution of Step 4. Updating the pixel intensities x_j as per equation (7.9) involves the computation of the expression on the right hand side and storage of the results in vector x. While the computations can be executed in parallel, updating the vector must be guarded from concurrent execution. Placing the updating step in a GUARDed segment of code introduces minimal wait states between the processors. This is the reason why Scheme I achieves significant speedup factors. However, the presence of the GUARD statement inside the iterative loop inhibits vectorization and the microtasked code runs much slower on a single processor as shown in Table 7.7. On machines with few processors multitasking should be used only if it will not interfere with the execution of vector calculations. In the modification of the parallel scheme I, that we introduce next, we implement multitasking in a way that preserves vectorization. Speedup factors are then reduced somewhat. The almost linear speedup factors reported in Table 7.6 are indicative of the performance that can be expected from the parallel algorithm on a scalar mutliprocessor.

Problem	Scalar	Vectorized	Parallel	
			1-CPU	4-CPUs
SHEPP512	1140.49 sec	108.81 sec	459.89 sec	123.49 sec
SHEPP1024	N/A	312.71 sec	945.94 sec	263.00 sec

Table 7.7: Trading off vectorization for parallelism.

No. of CPUs	Speedups	
	SHEPP512	SHEPP1024
2	1.82	1.80
3	2.49	1.97
4	2.97	3.15

Table 7.8: Speedup Factors with Vectorized Parallel Scheme I.

Placing the whole iterative step inside a GUARDed segment of code increases the wait states between the processors but maintains vectorization. Speedup factors with this modification (which we call vectorized parallel scheme I) are summarized in Table 7.8. The overhead due to the use of microtasked directives is around 10% (i.e., the microtasked code will run only 10 % slower on a single processor than the non-microtasked code).

7.3.3 Parallel Scheme II: Parallelism with Independent Blocks

The block nature of the algorithm and the geometry of image reconstruction problems provides a mechanism for introducing parallelism with large granularity tasks. It is possible to instruct SNARK77 to generate independent blocks. (Two blocks are termed *independent* if their respective rays do not intersect any common pixels.) With independent blocks it is known a priori

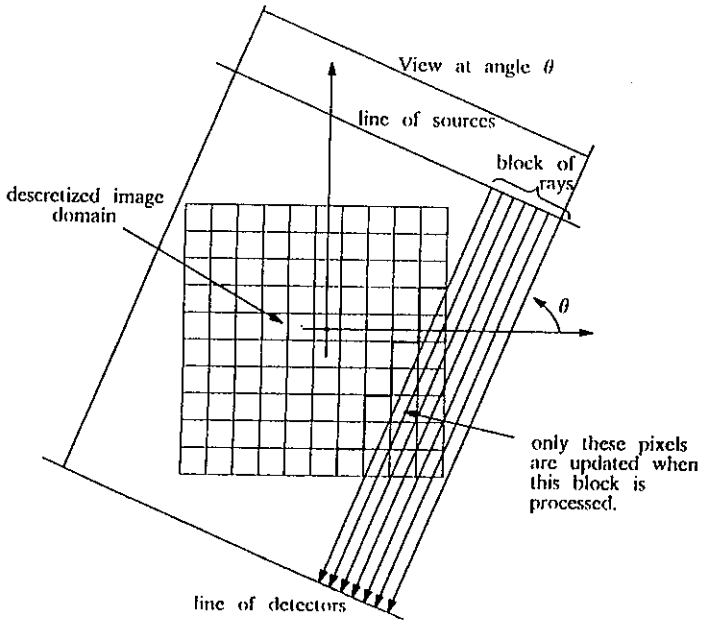


Figure 7.9: Parallel computing with independent blocks.

that no write conflicts will arise between the tasks for a given view, and there is no need to use GUARDed code segments.

Independent blocks are formed by grouping together rays that belong to the same view, as illustrated in Fig. 7.9. With this parallel scheme, synchronization is needed whenever the algorithm starts operating on blocks in a different view. It is possible that some processors may still be operating on blocks in a different view, in which case write conflicts may arise in updating the intensities of pixels shared between tasks. A barrier mechanism is set up to avoid this problem by imbedding the iteration over all blocks in a view, within an outer iteration over all views. The inner iteration is then parallelized while the outer is executed sequentially. Hence, all processors

No. of CPUs	Speedup	
	SHEPP512	SHEPP1024
2	1.82	1.79
3	2.32	1.93
4	3.01	3.11

Table 7.9: Speedup factors with parallel scheme II.

must terminate execution of the inner iteration before operations on a new view may begin.

The performance of this parallel scheme was evaluated by reconstructing the phantoms SHEPP512 and SHEPP1024 with the vectorized **block-MART** code using 45 blocks/view. Results are summarized in Table 7.9. The observed speedup factors are almost identical to those of the vectorized scheme I above. This is due to the efficiency of the implementation of parallel computations with the microtasking directives on the CRAY. Even tasks with small granularity (such as those of scheme I) start with very little overhead. On systems with large overhead for task initiation Parallel Scheme II is preferable.

7.3.4 Parallel Scheme III: Parallelism Between Views

A third parallel scheme is illustrated in Fig. 7.10 where all rays in a single view are grouped together as a block and are operated upon in parallel. The algorithm is a simultaneous block-iterative algorithm as characterized in Chapter 1. For all M different views in the parallel geometry of data collection, let I_t contain the indices of all rays in the tth view. For each

t, $1 \leq t \leq M$, define an *intermediate image* $x^{k+1,t}$ by using, for all j, $j = 1, 2, \ldots J$, the formula

$$x_j^{k+1,t} = (x_j^k)^{1/M} \prod_{i \in I_t} \exp\left(w_i^t d_i^k a_{ij}\right). \tag{7.10}$$

With this as a particular representation of equation (1.10) take the operator S of equation (1.11) to be the following:

$$x_j^{k+1} = \prod_{t=1}^{M} x_j^{k+1,t}, \quad j = 1, 2, \ldots J. \tag{7.11}$$

This simultaneous block-iterative version of MART is mathematically equivalent to a fully simultaneous algorithm with

$$x_j^{k+1} = x_j^k \prod_{i=1}^{I} \exp\left(w_i d_i^k a_{ij}\right), \tag{7.12}$$

for all $j = 1, 2, \ldots J$, where all rays are lumped into a single block. However, from the computational point of view it offers yet another, completely different, parallel scheme for implementing **block-MART** for image reconstruction. Since all pixel intensities must be updated by the rays of each view (and hence in parallel by multiple processors) temporary storage must be used for the correction factors computed by each view. The algorithmic operator then aggregates the intermediate correction factors and updates the vector x. At present we have no computational experiences with this parallel scheme, which we introduce for the sake of completeness. It has, however, the highest potential for performance improvement on a multiprocessor due to the large granularity of the tasks. Furthermore, the synchronization step where the intermediate correction terms are updated can be partitioned for

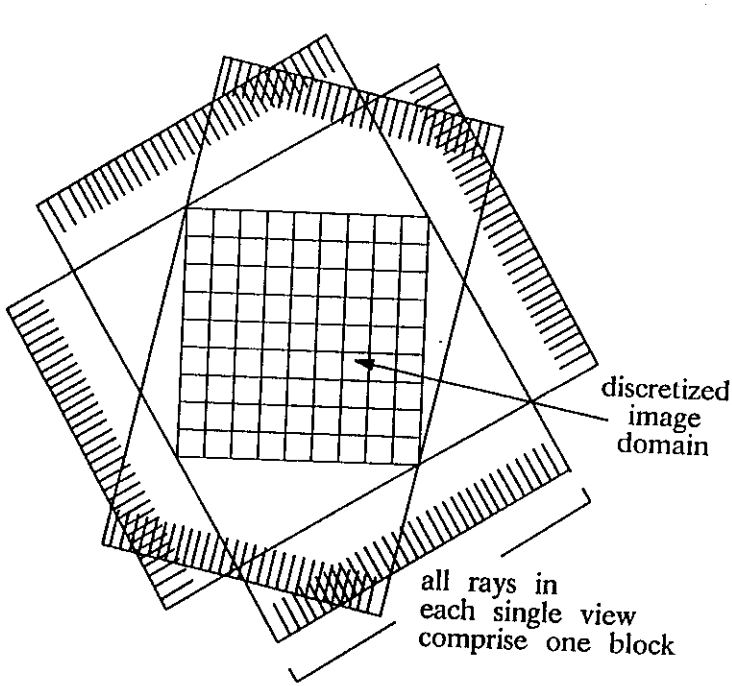


Figure 7.10: Parallel computing between views.

parallel execution. This scheme is more suitable for message passing architectures or with shared memory multiprocessors with higher overhead for task initialization.

7.3.5 Parallel Computations with MART and Block-MART

By varying the number of equations in a block we may execute a range of algorithms from row-action **MART** to a fully simultaneous **block-MART**. All previous experiments were carried out with **block-MART** using block sizes of 45 blocks/view. One of the advantages of **block-MART** is that the structure of the algorithm provides a natural mechanism for the introduction of parallelism using different task sizes. Hence, we repeated the

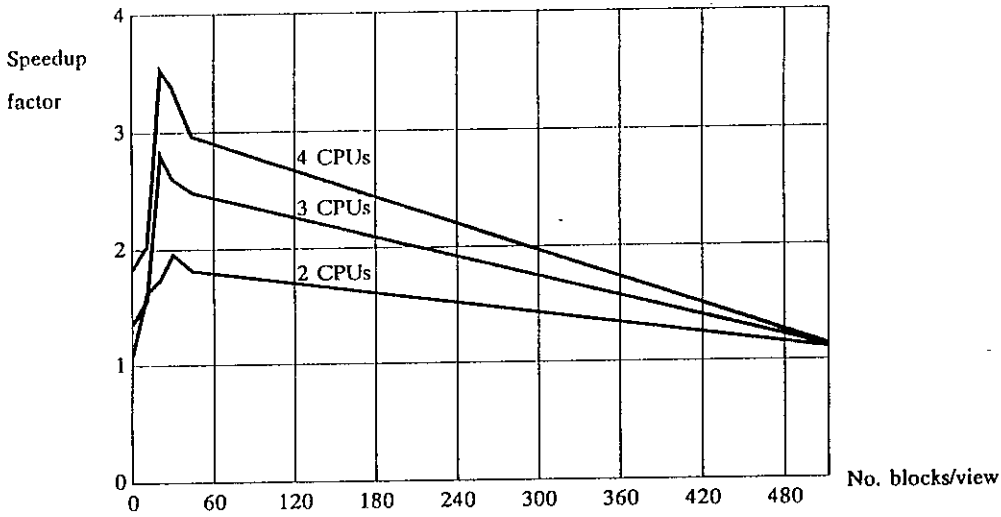


Figure 7.11: Speedup factors with varying block sizes.

experiments with parallel scheme II on problem SHEPP512 and varying the block sizes. Results are summarized in Fig. 7.11. Intermediate block sizes (30 blocks/view for this particular problem) appear to be optimal. At this point blocks are large enough to provide large tasks. At the same time there are enough blocks (and therefore tasks) available to achieve load balancing between the processors and to avoid excessive synchronization delays.

7.4 Parallel Computing for Stochastic Network Optimization

Stochastic programming problems remain one of the major challenging problems in computational mathematical programming. The size of these prob-

lems grows with the number of scenarios that are included in the formulation. Furthermore, any special structure that might have been present in a deterministic formulation of the problem is substantially complicated with the stochastic programming formulation.

In this section we describe the data-level parallel implementation of the row-action algorithm for stochastic network problems presented in section 6.3. The implementation is carried out on the Connection Machine CM–2, and is used to solve some of the largest stochastic network problems that have been reported in the literature. The specialization of the row-action algorithms for the massively parallel solution of stochastic networks is reported in Nielsen and Zenios [177].

In order to implement a sparse, stochastic network solver we use an extension of the data-structures used to represent the graph arising from the sparse matrix balancing problem, see section 6.1.2. Each network subproblem is stored in a one-dimensional grid of processors of dimension $(m_1 + m_2) + 2(n_1 + n_2)$. With each arc (i, j) we associate two processors: one at the tail node i and one at the head node j. An additional processor is associated with each node. Processors that correspond to the same node — i.e., are assigned to arcs incident to the same node — are grouped together into a contiguous segment. This is the data-structures introduced in Zenios and Lasken [232]. In this way the segmented scan operations (Section 7.1.3) can be employed for the implementation of the algorithm.

In order to solve the stochastic network problems we use a two- dimensional grid of processors, of dimensions $S \times (m_1 + m_2) + 2(n_1 + n_2)$. Each row of this grid is used to represent a single network problem as outlined above. Since the network topology is identical for all scenarios, the mapping

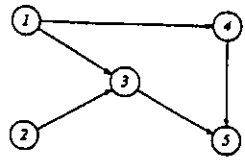
of arcs into processors and the partitioning of processors into segments, will be identical for each row of the grid. Hence, the control of the algorithm is identical for each row of the grid (i.e., for each network problem). Row s will store the data of the network problem for the s-th scenario. This configuration is illustrated in Fig. 7.12. The algorithm iterates along the row-axis until some convergence criteria is satisfied for all the rows. Once the single scenario networks are solved by iterations along the row- axis, the algorithm executes the projection on the non-anticipativity constraints using scan operations along the column-axis.

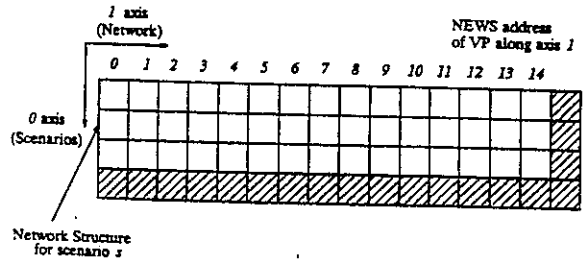
The projection on the non-anticipativity constraints, i.e., (6.166)–(6.167), is executed in parallel for all first-stage (replicated) variables. Each first-stage variable (with replications) occupies two columns of the two-dimensional grid representing the stochastic program (Fig. 7.12). Each processor holds the scenario probability p^s and the component x_{ij}^s of the current iterate, and computes their product. The products are then added and distributed back to each processor (using the *spread-with-add* parallel prefix) as the projected point, $\hat{x}_{ij}$.

7.4.1 Experiments and Numerical Results

In this section we report on the performance of the algorithm on a number of test problems from financial applications. We also investigate the effect of stochasticity in the network parameters on the performance of the algorithm. The row-action algorithm for solving stochastic network problems was implemented on the Connection Machine CM-2.

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





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

NEWS address of VP along axis 1	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
Node	1	1	1	2	2	3	3	3	3	4	4	4	5	5	5
Segment bits	1	0	0	1	0	1	0	0	0	1	0	0	1	0	0
Supply/Demand	$a_1(1)$	$a_2(1)$	$a_3(1)$	$a_1(2)$	$a_2(2)$	$a_3(3)$	$a_1(3)$	$a_2(3)$	$a_3(3)$	$a_1(4)$	$a_2(4)$	$a_3(4)$	$a_1(5)$	$a_2(5)$	$a_3(5)$
Capacity	∞	$U(1,3)U(1,4)$	∞	$U(2,3)$	∞	$U(1,3)U(2,3)U(3,5)$	∞	$U(1,4)U(4,5)$	∞	$U(3,5)U(4,5)$					
Send address in NEWS coordinates along axis 1	0	6	10	3	7	5	1	4	13	9	2	14	12	8	11
Stage bit	0	1	0	0	1	0	1	1	0	0	0	0	0	0	0

Figure 7.12: Data-parallel representation of a stochastic network problem.

Problem	Scenarios	Nodes	Arcs	Det. Eqv. Rows × Columns
Deter 0	18	121	334	1923 × 5468
Deter 1	52	121	335	5527 × 15737
Deter 2	80	91	249	6095 × 17313
Deter 3	72	121	335	7647 × 21777
Deter 4	70	61	163	3235 × 9133
Deter 5	48	121	335	5103 × 14529
Deter 6	40	121	335	4255 × 12113
Deter 7	60	121	335	6375 × 18153
Deter 8	36	121	335	3831 × 10905

Table 7.10: Characteristics of the test problems Deter 0 – Deter 8.

minor.

Test Problems

We use as test problems a set of asset allocation models from Mulvey and Vladimirou [170, 171]. Due to the requirement of our implementation that the objective must be quadratic, we minimize $\sum_{(i,j)\in\mathcal{A},s\in S}(x_{ij}^s)^2$. Table 7.10 lists the characteristics of the test problems and the size of the deterministic equivalent nonlinear program.

Results on the Connection Machine CM–2

The problems were solved on a CM–2 with 8K processors (1K = 1024). The algorithm was terminated when both the absolute node surplus/deficit was

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

Table 7.11: Solution times (seconds) of test problems on the CM–2 with 8K processing elements.

below a small tolerance $\epsilon > 0$ for each node, and each non-anticipativity constraint was violated by less than ϵ, for $\epsilon = 10^{-3}$. Results are shown in Table 7.11. We imposed a limit of 100 minor iterations between the execution of a projection on the non-anticipativity constraints (constituting a major iteration). Convergence of the network subproblems (to within the tolerance) was checked every 25 iterations.

The results show that all of these problems were solved in less than half a minute, and required only up to 11 major iterations. These problems could be solved in under 5 seconds each on a CM–2 with 64K processing elements. It is also interesting to observe that the solution time does not depend on

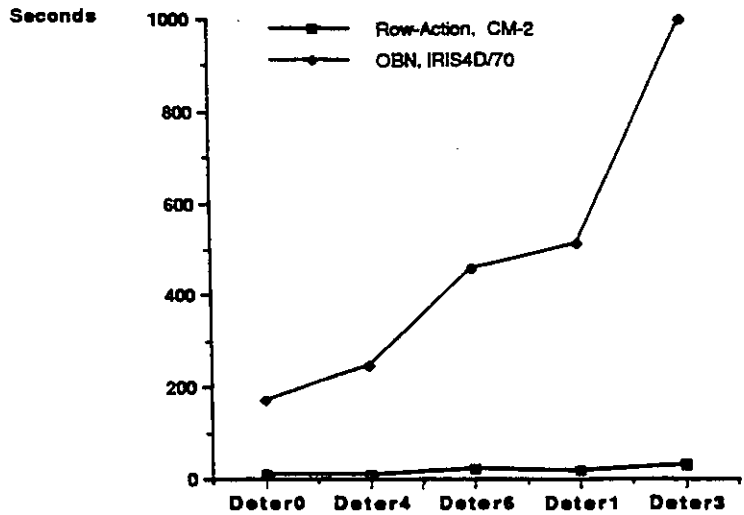


Figure 7.13: Comparing the row-action algorithm on the 8K CM–2 with the general purpose solver OBN on an IRIS4D/70.

the number of scenarios, or the size of the equivalent nonlinear program. For example, Deter4 was solved in much less time than Deter6 although it has twice as many scenarios.

To provide some basis for evaluation of the results of Table 7.11 we summarize in Fig. 7.13 the results obtained using the row-action algorithm with the results obtained by Lustig et al. [152] using the interior point code OBN. Lustig et al. used OBN within a successive-quadratic programming (SQP) framework in order to solve a nonlinear program. The time reported in Fig. 7.13 is the average time for the solution of several quadratic programs within SQP, as obtained from their report. The row-action algorithm

Scenarios	Det. Eqv.	8K PEs		32K PEs	
	Rows × Columns	$\epsilon = 10^{-3}$	$\epsilon = 10^{-4}$	$\epsilon = 10^{-3}$	$\epsilon = 10^{-4}$
128	13583 × 38689	30.1	46.2	10.4	16.2
512	54287 × 154657	108.2	155.4	30.7	46.3
1024	108559 × 309281	210.8	326.5	57.3	86.3
2048	217103 × 618529	407.5	623.1	113.1	163.6
8196	868367 × 2474017	N/A	N/A	N/A	11 min.

Table 7.12: Solving large-scale problems (solution times in seconds)

substantially outperforms OBN. It should be noted, however, that OBN is a general purpose optimizer while the row-action algorithm is specialized for the network structured problems. It is interesting to observe that the solution time for the row-action algorithm increases only marginally with problem size, while substantial increase is observed for OBN. The interior point algorithms cannot be used to solve directly the much larger problems we solve next since that would involve factorization of 100K × 100K matrices. The conclusion drawn from the results of Fig. 7.13 is that the special-purpose algorithms scale nicely with problem size and are very competitive with general purpose optimizers.

Solving Large-Scale Problems

To test the algorithm on large-scale problems, we modified the largest problem, Deter3, by replicating the scenarios until there were 128, 512,1024, 2048 and 8196 scenarios, respectively.

The problems were run on an 8K and a 32K Connection Machine CM–2

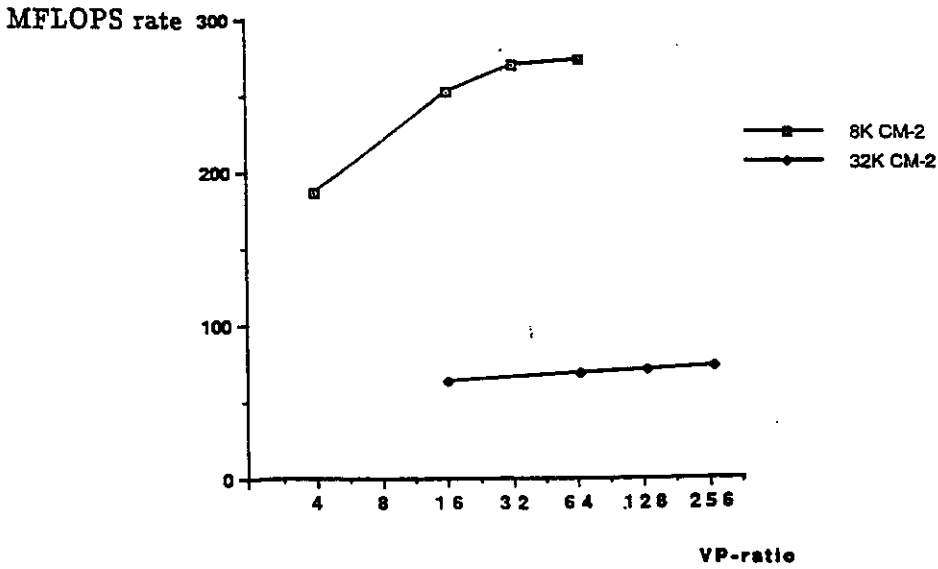


Figure 7.14: CM–2 MFLOPS rates for an 8K and a 32K CM–2.

with 32Kbytes of random-access memory per processing element. Results are shown in Table 7.12, with final tolerances of $\epsilon = 10^{-3}$ and $\epsilon = 10^{-4}$.

This experiment demonstrates the suitability of the algorithm for solving large-scale stochastic problems. The largest problem, 8196 scenarios, having a deterministic nonlinear equivalent of 868,367 constraints and 2,474,017 variables, was solved to the tightest tolerance in about 11 minutes on the 32K CM–2. We also observe that the algorithm scales very effectively for larger problems on bigger machine sizes. For example, 512 scenarios are solved in 108.2 sec. on the 8K CM–2. Using a system with 32K processing elements we solve a problem with four times as many scenarios in almost the same time, 113.1 sec.

Based on the solution of the large-scale problems (Table 7.12) we can calculate the computational rate of the algorithm. The MFLOP rates for the large scale runs are shown in Fig. 7.14. For example , the 2048 scenario problem was solved in 113.1 seconds on a 32K CM–2 and the number of floating point operations per second for this run is 276 MFLOPS. A fully configured CM–2 with 64K processing elements, solving a problem with 4096 scenarios would achieve twice this computing rate, 552 MFLOPS. On a 64K CM–2 equipped with maximum memory the solution of a 64K scenario problem — the largest problem which would fit on the machine — would take about 30 minutes for $\epsilon = 10^{-3}$ and about 44 minutes for $\epsilon = 10^{-4}$. Although such systems are available we do not have access to them at this time.

Appendix A

The Connection Machine System

In this Appendix we describe briefly the characteristics of the Connection Machine CM–2 that has been used for all the data-parallel implementations in these lecture notes. Other existing systems, like MassPar, DAP and the Connection Machine CM–5, have characteristics similar to the CM–2. The implementations discussed in the lecture notes are applicable to those systems as well. The details of the hardware of the CM–2 discussed here are not crucial in the understanding of the data-parallel implementations. Nevertheless, it is instructive to understand the hardware configuration of one system where data-parallel algorithms can be implemented.

The Connection Machine is a fine grain SIMD — Single Instruction stream, Multiple Data stream — system. Its basic hardware component is an integrated circuit with sixteen processing elements (PEs) and a *router* that handles general communication. A fully configured CM has 4,096 chips for a

total of 65,536 PEs. The 4,096 chips are interconnected as a 12-dimensional hypercube. Each processor is equipped with local memory of 8Kbytes, and for each cluster of 32 PEs a floating point accelerator handles floating point arithmetic.

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

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

To achieve high performance with data level parallelism one needs a large number of processors that could operate on multiple copies of the data concurrently. While the full configuration of the CM has 65,536 PEs this number is not large enough for several applications. For example, in balancing matrices of dimension 1000×1000 we need 10^6 processors. The CM provides the mechanism of *virtual processors* (VPs) that allows one PE to operate in a serial fashion on multiple copies of data. VPs are specified by slicing the local memory of each PE into equal segments and allowing the physical processor to loop over all slices. The number of segments is called the *VP ratio* (i.e., ratio of virtual to physical PEs). Looping by the PE over

all the memory slices is executed, in the worst case, in linear time. The set of virtual processors associated with each element of a data set is called a *VP set.* VP sets are under the control of the software and are mapped onto the underlying CM hardware in a way that is transparent to the user.

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

The *send* address indicates the location of the PE (hypercube address) that supports a specific VP and the relative address of the VP in the VP set that is currently active. NEWS address is an n-tuple of coordinates which specifies the relative position of a VP in an n-dimensional Cartesian-grid geometry. A *geometry* (defined by the software) is an abstract description of such an n-dimensional grid. Once a geometry is associated with the currently active VP set a relative addressing mechanism is established among the processors in the VP set. Each processor has a relative position in the n-dimensional geometry and NEWS allows the communication across the North, East, West and South neighbors of each processor, and enables the execution of operations along the axes of the geometry. Such operations are efficient since the n-dimensional geometry can be mapped onto the underlying hypercube in such a way that adjacent VPs are mapped onto vertices of the hypercube connected with a direct link. This mapping of an n-dimensional mesh on a hypercube is achieved through a Gray coding, (see, e.g., Bertsekas and Tsitsiklis [26]).

Before using a programming language to execute the instructions of a program, the user has to specify the VP set, create a geometry, and asso-

ciate the VP set with the geometry. Thus a communications mechanism is established (along both *send* and NEWS addresses).. For high-level languages, like CM Fortran or C*, these specifications are achieved automatically by the compiler when the data-structures are created. When using a low level language, like the *parallel instruction set* (Paris) the VP set specification and the geometry configuration must be specified explicitly. One the geometry is specified Paris instructions — parallel primitives — can then be invoked to execute operations along some axis of the geometry (using NEWS addresses), operate on an individual processor using *send* addresses, or to translate NEWS to *send* addresses for general interprocessor communication or communication with the front end. Parallel primitives for the CM–2, which also appear in other systems, and which have been used in our implementations, were reviewed in section 7.1.3.

Bibliography

[1] *Special Issue on Fully Three-Dimensional Image Reconstruction*, volume 37 of *Physics in Medicine and Biology*. 1992.

[2] M. Abdfulaal and L.J. LeBlanc. Methods for combining modal split and equilibrium assignment models. *Transportation Science*, 13:292–314, 1979.

[3] A. Agmon. The relaxation method for linear inequalities. *Canadian Journal of Mathematics*, 6:382–392, 1954.

[4] R. Aharoni, A. Berman, and Y. Censor. An interior points algorithm for the convex feasibility problem. *Advances in Applied Mathematics*, 4:479–489, 1983.

[5] R. Aharoni and Y. Censor. Block-iterative projection methods for parallel computation of solutions to convex feasibility problems. *Linear Algebra and Its Applications*, 120:165–175, 1989.

[6] G. Amdahl. The validity of single processor approach to achieving large scale computing capabilities. In *AFIPS Proceedings*, volume 30, pages 483–485, 1967.

[7] K.A. Ariyawansa and D.D. Hudson. Performance of a benchmark parallel implementation of the Van Slyke and Wets algorithm for two-stage stochastic programs on the Sequent/Balance. *Concurrency: Practice and Experience*, 3:109–128, 1991.

[8] A. Auslender. *Optimization: Méthodes Numériques.* Masson, Paris, 1976.

[9] S.G. Azevedo. *Model-based Computed Tomography for Nondestructive Evaluation.* PhD thesis, Lawrence Livermore National Laboratory, University of California, Livermore, CA 94551, 1991.

[10] M. Bacharach. *Bi-proportional Matrices and Input-Output Change.* Cambridge University Press, U.K., 1970.

[11] A. Bachem and B. Korte. On the RAS-Algorithm. *Computing*, 23:189–198, 1979.

[12] A. Bachem and B. Korte. Estimating matrices. *Metrika*, 28:273–286, 1981.

[13] A. Bachem and B. Korte. Primal and dual methods for updating input-output matrices. In K. Brockhoff and W. Krelle, editors, *Unternehmensplanung*, pages 117–127, Berlin, W. Germany, 1981. Springer-Verlag.

[14] T. Baker, F. Van der Ploeg, and M. Weale. A balanced system of National accounts for the United Kingdom. *The Review of Income and Wealth*, 30:461–485, 1979.

[15] M.L. Balinski and G. Demange. Algorithms for proportional matrices in reals and integers. *Mathematical Programming*, 45:193–210, 1989.

[16] M.L. Balinski and G. Demange. An axiomatic approach to proportionality between matrices. *Mathematics of Operations Research*, 14:700–719, 1989.

[17] J.M. Del Balzo. The US national airspace system for the year 2010. *Journal of ATC*, July-September 1989.

[18] R.H.T. Bates and M.J. McDonnell. *Image Restoration and Reconstruction.* Clarendon Press, Oxford, 1986.

[19] M.S. Bazaraa and C.M. Shetty. *Nonlinear Programming: Theory and Algorithms.* John Wiley and Sons, New York, 1979.

[20] E.M.L. Beale. On minimizing a convex function subject to linear inequalities. *Journal of the Royal Statistical Society*, 17:173–184, 1955.

[21] A. Beguelin, J. Dongarra, A. Geist, R. Manchek, and V. Sunderam. A user's guide to PVM: Parallel Virtual Machine. Technical Report ORNL/TM-11826, Oak Ridge National Laboratory, 1991.

[22] M. Ben-Akiva. Methods to combine different data sources and estimate origin-destination matrices. Working paper, Department of Civil Engineering, Massachusetts Institute of Technology, Cambridge, Mass., 1987.

[23] J.F. Benders. Partitioning procedures for solving mixed variables programming problems. *Numerische Mathematik*, 4:238–252, 1962.

[24] M. Bertero and E.R. Pike, editors. *Inverse Problems in Scattering and Imaging.* Adam Hilger, 1992.

[25] D. P. Bertsekas. *Constrained Optimization and Lagrange Multipliers Method.* Academic Press, N. York, 1982.

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

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

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

[29] P. Boyle. Options: A Monte Carlo approach. *Journal of Financial Economics*, 4:323–338, May 1977.

[30] S.P. Bradley and D.B. Crane. A dynamic model for bond portfolio management. *Management Science*, 19:139–151, Oct. 1972.

[31] L.M. Bregman. The relaxation method for finding the common point of convex sets and its application to the solution of problems in convex programming. *USSR Computational Mathematics and Mathematical Physics*, 7:200–217, 1967.

[32] Bureau of Economic Analysis, U.S. Department of Commerce. *Data tape from Interindustry Economic Division*, 1977.

[33] D. Butnariu and Y. Censor. On the behavior of a block-iterative projection method for solving convex feasibility problems. *International Journal of Computer Mathematics*, 34:79–94, 1990.

[34] B.L. Buzbee and D.H. Sharp. Perspectives on supercomputing. *Science*, 227:591–597, 1985.

[35] C.L. Byrne. Iterative image reconstruction algorithms based on cross-entropy minimization. *IEEE Transactions on Image Processing*, to appear.

[36] L.D. Cagan, N.S. Carriero, and S.A. Zenios. Pricing mortgage-backed securities with network Linda. *Financial Analysts Journal*, 1993. (to appear).

[37] M.C. Carey, C. Hendrickson, and K. Siddharathan. A method for direct estimation of origin/destination trip matrices. *Transportation Science*, 15:32–49, 1981.

[38] N. Carriero and D. Gelernter. How to write parallel programs: A guide to the perplexed. Technical report, Scientific Computing Associates, New Heaven, CT, 1989.

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

[40] Y. Censor. Finite series-expansion reconstruction methods. *Proceedings of IEEE*, 71:409–419, 1983.

[41] Y. Censor. Parallel application of block-iterative methods in medical imaging and radiation therapy. *Mathematical Programming*, 42:307–325, 1988.

[42] Y. Censor, M.D. Altschuler, and W.D. Powlis. On the use of Cimmino's simultaneous projections method for computing a solution of the inverse problem in radiation therapy treatment planning. *Inverse Problems*, 4:607–623, 1988.

[43] Y. Censor, P.P.B. Eggermont, and D. Gordon. Strong underrelaxation in Kaczmarz's method for inconsistent systems. *Numerische Mathematik*, 41:83–92, 1983.

[44] Y. Censor, T. Elfving, and G.T. Herman, editors. *Linear Algebra in Image Reconstruction from Projections*, volume 130 of *Linear Algebra and Its Applications*. 1990.

[45] Y. Censor and G.T. Herman. Row-generation methods for feasibility and optimization problems involving sparse matrices and their applications. In I.S. Duff and G.W. Stewart, editors, *Sparse Matrix Proceedings-1978*, pages 197–219. SIAM, Philadelphia, PA., 1979.

[46] Y. Censor and G.T. Herman. On some optimization techniques in image reconstruction from projections. *Applied Numerical Mathematics*, 3:365–391, 1987.

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

[48] Y. Censor and A. Lent. Cyclic subgradient projections. *Mathematical Programming*, 24:233–235, 1982.

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

[50] Y. Censor and J. Segman. On block-iterative entropy maximization. *Journal of Information and Optimization Sciences*, 8:275–291, 1987.

[51] Y. Censor and S.A. Zenios. Interval-constrained matrix balancing. *Linear Algebra and Its Applications*, 150:393–421, 1991.

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

[53] G. Chen and M. Teboulle. Convergence analysis of a proximal-like minimization algorithm using Bregman functions. *SIAM Journal on Optimization*, 3, 1992. (to appear).

[54] P.E. Christensen and F.J. Fabozzi. Bond immunization: An asset liability optimization strategy. In F.J. Fabozzi and I.M. Pollack, editors, *The Handbook of Fixed Income Securities*. Dow Jones Irwin, 1987.

[55] G. Cimmino. Calcolo approssimato per le soluzioni dei sistemi di equazioni lineari. *La Ricerca Scientifica XVI, Series II, Anno IX,*, 1:326–333, 1938.

[56] R. W. Cottle, S. G. Duvall, and K. Zikan. A Lagrangean relaxation algorithm for the constrained matrix problem. *Naval Research Logistics Quarterly*, 33:55–76, 1986.

[57] I.J.D. Craig and J.C. Brown. *Inverse Problems in Astronomy: a guide to inversion strategies for remotely sensed data.* Adam Hilger Ltd, Bristol and Boston, 1986.

[58] CRAY Research Inc., Mendota Heights. *Multitasking User Guide, Technical Note SN–0222*, March 1986.

[59] I. Csiszár. Why least squares and maximum entropy? An axiomatic approach to inference for linear inverse problems. *The Annals of Statistics*, 19:2032–2066, 1991.

[60] H. Dahl, A. Meeraus, and S.A. Zenios. Some financial optimization models: III. an algebraic modeling system library. Report 89–12–03, Decision Sciences Department, The Wharton School, University of Pennsylvania, Philadelphia, 1989.

[61] H. Dahl, A. Meeraus, and S.A. Zenios. Some financial optimization models: I. risk management. In S.A. Zenios, editor, *Financial Optimization*, pages 3–36. Cambridge University Press, 1992.

[62] H. Dahl, A. Meeraus, and S.A. Zenios. Some financial optimization models: II. financial engineering. In S.A. Zenios, editor, *Financial Optimization*, pages 37–71. Cambridge University Press, 1992.

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

[64] G. B. Dantzig, M. A. H. Dempster, and M. J. Kallio, editors. *Large-Scale Linear Programming (Volume 1).* IIASA Collaborative Proceedings Series. International Institute of Applied Systems Analysis, Laxenburg, Austria, 1981. CP-81-51.

[65] G.B. Dantzig. Planning under uncertainty using parallel computing. *Annals of Operations Research*, 14:1–16, 1988.

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

[67] J.N. Darroch and D. Ratcliff. Generalized iterative scaling for log-linear models. *The Annals of Mathematical Statistics*, 43:1470 – 1480, 1982.

[68] S.R. Deans. *The Radon Transform and Some of Its Applications.* John Wiley & Sons, New York, 1983.

[69] R.S. Dembo, A. Chiarri, J.Gomez Martin, and L. Paradinas. Managing Hidroeléctrica Española's hydroelectric power system. *Interfaces*, 20:115–135, Jan.–Feb. 1990.

[70] R.S. Dembo, J.M. Mulvey, and S.A. Zenios. Large scale nonlinear network models and their application. *Operations Research*, 37:353–372, 1989.

[71] W.E. Deming and F.F. Stephan. On a least squares adjustment of sampled frequency table when the expected marginal totals are known. *The Annals of Mathematical Statistics*, 11:427–444, 1940.

[72] K. Dervis, J. De Melo, and S. Robinson. *General Equilibrium Models for Development Policy.* Cambridge University Press, Cambridge, 1982.

[73] D.A. D'Esopo. A convex programming procedure. *Naval Research Logistics Quarterly*, 6:33–42, 1959.

[74] G. R. Desrochers. *Principles of Parallel and Multi-Processing.* McGraw Hill, New York, NY, 1987.

[75] J. J. Dongarra and I. S. Duff. Advanced architecture computers. Technical memorandum 57, Mathematics and Computer Science Division, Argonne National Laboratory, Chicago, Illinois, 1985. (revised 1987).

[76] K.W. Dritz and J.M. Boyle. Beyond speedup: Performance analysis of parallel programs. Report ANL–87–7, Argonne National Laboratory, Chicago, Illinois, Feb. 1987.

[77] I. S. Duff. The use of supercomputers in Europe. Report css –161, United Kingdom Atomic Energy Authority, Harwell Laboratory, Computer Science Division, Oxfordshire, England, Sept. 1984.

[78] I.S. Duff. A survey of sparse matrix research. *Proceedings of The IEEE*, 65:500–535, 1977.

[79] J. Eckstein. *Splitting Methods for Monotone Operators with Applications to Parallel Optimization.* PhD thesis, Department of Civil Engineering, MIT, Cambridge, MA, 1989.

[80] J. Eckstein. Nonlinear proximal point algorithms using Bregman functions, with applications to convex programming. *Mathematics of Operations Research*, 18(1):202–226, 1992.

[81] P.P.B. Eggermont. Multiplicative iterative algorithms for convex programming. *Linear Algebra and Its Applications*, 130:25–42, 1990.

[82] P.P.B. Eggermont, G.T. Herman, and A. Lent. Iterative algorithms for large partitioned linear systems, with applications to image reconstruction. *Linear Algebra and Its Applications*, 40:37–67, 1981.

[83] P.P.B. Eggermont, G.T. Herman, and A. Lent. Iterative algorithms for large partitioned linear systems, with applications to image reconstruction. *Linear Algebra and Its Applications*, 40:37–67, 1981.

[84] I.I. Eremin. The relaxation method of solving systems of inequalities with convex functions on the left hand side. *Soviet Math. Doklady*, 6:219–222, 1965.

[85] I.I. Eremin. Certain iteration methods in convex programming. *Ekon. Math. Method.*, 2:870–886, (in Russian), 1966.

[86] I.I. Eremin. On systems of inequalities with convex functions in the left sides. *Amer. Math. Soc. Translations*, 88:67–83, 1970.

[87] I.I. Eremin and V.D. Mazurov. Iteration method for solving problems of convex programming. *Soviet Phys. Doklady*, 11:757–759, 1967.

[88] J. Eriksson. A note on solution of large sparse maximum entropy problems with linear equality constraints. *Mathematical Programming*, 18:146–154, 1980.

[89] J.R. Eriksson. An iterative primal-dual algorithm for linear programming. Report lith-mat-r-1985-10, Department of Mathematics, Linkoping University, Linkoping, Sweden, 1985.

[90] S. Erlander, K.O. Jőrnsten, and J.T. Lundgren. On the estimation of trip matrices in the case of missing and uncertain data. *Transportation Research*, 19B:123–141, 1985.

[91] K. Fan, I. Glicksberg, and A.J. Hoffman. Systems of inequalities involving convex functions. *Proceedings of the American Math. Soc.*, 8:617–622, 1957.

[92] S.D. Flåm and D. Zowe. Relaxed outer projections, weighted averages and convex feasibility. *BIT*, 30:289–300, 1990.

[93] M.J. Flynn. Some computer organizations and their effectiveness. *IEEE Transactions on Computers*, C–21:948–960, 1972.

[94] L.R. Ford, Jr. and D.R. Fulkerson. *Flows on Networks*. Princeton University Press, Princeton, New Jersey, 1962.

[95] B.R. Frieden. Restoring with maximum likelihood and maximum entropy. *Journal of the Optical Society of America*, 62:511–518, 1972.

[96] B.R. Frieden. Image enhancement and restoration. In T.S. Huang, editor, *Picture Processing and Digital Filtering*, volume Chapter 5. Springer-Verlag, New York, 1975.

[97] D. Friedlander. A technique for estimating a contingency table given the marginal totals and some supplementary data. *Journal of the Royal Statistical Society*, 124:412–420, 1961.

[98] N. Gastinel. *Linear Numerical Analysis.* Hermann, Paris, 1970.

[99] A. M. Geoffrion. Elements of large-scale mathematical programming. *Management Science*, 16:652–691, 1970.

[100] A.M. Geoffrion. Generalized Benders decomposition. *Journal of Optimization Theory and Applications*, 10:237–260, 1972.

[101] P.F.C. Gilbert. Iterative methods for the three-dimensional reconstruction of an object from projections. *Journal of Theoretical Biology*, 36:105–117, 1972.

[102] R. Gordon, R. Bender, and G.T. Herman. Algebraic reconstruction techniques (art) for three-dimensional electron microscopy and x-ray photography. *Journal of Theoretical Biology*, 29:471–481, 1970.

[103] R. Gordon and G.T. Herman. Three-dimensional reconstruction from projections: A review of algorithms. In G.F. Bourne and J.F. Danielli, editors, *International Review of Cytology*, volume 38, pages 111–151. Academic Press, N.Y., 1974.

[104] E. Grinberg and E.T. Quinto, editors. *Integral Geometry and Tomography*, volume 113, American Mathematical Society, Providence, RI, 1990.

[105] L.G. Gubin, B.T. Polyak, and E.V. Raik. The method of projections for finding the common point of convex sets. *USSR Computational Mathematics and Mathematical Physics*, 7:1–24, 1967. (in Russian).

[106] J. L. Gustafson. Reevaluating Amdahl's law. *Communications of the ACM*, 31:532–533, 1988.

[107] M. Hanke and W. Niethammer. On the acceleration of Kaczmarz's method for inconsistent linear systems. *Linear Algebra and Its Applications*, 130:83–98, 1990.

[108] F. Harrigan and I. Buchanan. A quadratic programming approach to input-output estimation and simulation. *Journal of Regional Science*, 24:339-358, 1984.

[109] M. Held, P. Wolfe, and H.P. Crowder. Validation of subgradient optimization. *Mathematical Programming*, 6:62–68, 1974.

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

[111] G.T. Herman, editor. *Special Issue on Computerized Tomography*, volume 71 of *Proceedings of The IEEE.* The Institute of Electrical and Electronics Engineers, 1983.

[112] G.T. Herman, H. Hurwitz, and A. Lent. A storage-efficient algorithm for finding the regularized solution of a large, inconsistent system of equations. *J. Inst. Maths Applics*, 25:361–366, 1980.

[113] G.T. Herman and A. Lent. Iterative reconstruction algorithms. *Computers in Biology and Medicine*, 6:273–294, 1976.

[114] G.T. Herman and A. Lent. A family of iterative quadratic optimization algorithms for pairs of inequalities, with application in diagnostic radiology. *Mathematical Programming Study*, 9:15–29, 1978.

[115] G.T. Herman, A. Lent, and P.H. Lutz. Relaxation methods for image reconstruction. *Communications of the ACM*, 21:152–158, 1978.

[116] G.T. Herman, A. Lent, and S.W. Rowland. ART: mathematics and applications. *Journal of Theoretical Biology*, 42:1–32, 1973.

[117] G.T. Herman and H. Levkowitz. *Initial Performance of Block-Iterative Reconstruction Algorithms.* Springer-Verlag, 1988.

[118] G.T. Herman, H. Levkowitz, H.K. Tuy, and S. McCormick. Multilevel image reconstruction. In A. Rosenfeld, editor, *Multiresolution Image Processing and Analysis*, pages 121–135. Springer-Verlag, New York, 1984.

[119] G.T. Herman, A.K. Louis, and F. Natterer, editors. *Mathematical Methods in Tomography*, volume 1497. Springer-Verlag, 1990.

[120] G.T. Herman and F. Natterer, editors. *Mathematical Aspects of Computerized Tomography*, volume 8. Springer-Verlag, 1981.

[121] C. Hildreth. A quadratic programming procedure. *Naval Research Logistics Quarterly*, 4:79–85, Erratum, ibid., p.361, 1975.

[122] R.S. Hiller and J. Eckstein. Stochastic dedication: Designing fixed income portfolios using massively parallel Benders decomposition. *Management Science*, 1993. (to appear).

[123] R.S. Hiller and C. Schaack. A classification of structured bond portfolio modeling techniques. *Journal of Portfolio Management*, pages 37–48, Fall 1990.

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

[125] R. Hockney and C. Jeeshope. *Parallel Computers.* Adam Hilger Ltd., Bristol, England, 1981.

[126] M.R. Holmer. The asset/liability management strategy at Fannie Mae. *Interfaces*, 1993. (to appear).

[127] N.E. Hurt. *Phase Retrieval and Zero Crossings.* Kluwer Academic Publishers, Dordrecht, The Netherlands, 1989.

[128] J.M. Hutchinson and S.A. Zenios. Financial simulations on a massively parallel Connection Machine. *International Journal of Supercomputer Applications*, 5:27–45, 1991.

[129] K. Hwang. *Supercomputers: Design and Applications.* IEEE Computer Society, Arlington, Virginia, 1984.

[130] G. Infanger. Monte Carlo (importance) sampling with a Benders decomposition algorithm for stochastic linear programs. Report sol–89-13r, Department of Operations Research, Stanford University, Palo Alto, CA, 1990.

[131] Jr. Ingersoll, J.E. *Theory of Financial Decision Making.* Studies in Financial Economics. Rowman & Littlefield, 1987.

[132] C.T. Ireland and S. Kullback. Contingency tables with given marginals. *Biometrika*, 55:179–188, 1968.

[133] A.N. Iusem and A.R. De Pierro. Convergence results for an accelerated nonlinear Cimmino algorithm. *Numerische Mathematik*, 49:367–378, 1986.

[134] A.N. Iusem and A.R. De Pierro. A simultaneous iterative method for computing projections on polyhedra. *SIAM Journal on Control and Optimization*, 25:231–243, 1987.

[135] T.R. Jefferson and C.H. Scott. The analysis of entropy models with equality and inequality constraints. *Transportation Research*, 13B:123–132, 1979.

[136] R.C. Jensen and D. McGaurr. Reconciliation techniques in input-output analysis: some comparisons and implications. *Urban Studies*, 14:327–337, 1977.

[137] H.F. Jordan. Interpreting parallel processor performance measurements. *SIAM Journal on Scientific and Statistical Computing*, 8:5–11, March 1985.

[138] S. Kaczmarz. Angenaherte Auflősung von Systemen Linearer Gleichungen. *Bull. Acad. Polon. Sci. Lett.*, A35:355–357, 1937.

[139] A.C. Kak and M. Slaney. *Principles of Computerized Tomographic Imaging*. IEEE Press, The Institute of Electrical and Electronic Engineers, Inc., New York, 1988.

[140] S. Karin and N.P. Smith. *The Supercomputer Era*. Harcourt Brace Jovanovich, Boston, Massachusetts, 1987.

[141] D. Kendrick and A. Drud. SIMS: The SAM integrated modeling system. Technical report, The World Bank, 1985. (in preparation).

[142] D. Kinderlehrer and G. Stampacchio. *An Introduction to Variational Inequalities and their Applications.* Academic Press, New York, 1980.

[143] Z.V. Kovarik. Minimal compatible solutions of linear equations. *Linear Algebra and and Its Applications*, 17:95–106, 1977.

[144] J. S. Kowalik. *High Speed Computations*, volume 7 of *NATO ASI Series F in Computer and System Sciences.* Springer-Verlag, Berlin, 1984.

[145] J. Kruithof. Telefoonverkeersrekening. *De Ingenieur*, 3:15–25, 1937.

[146] A.V. Lakshminarayanan and A. Lent. Methods of least squares and SIRT in reconstruction. *Journal of Theoretical Biology*, 76:267–295, 1980.

[147] C. Lazou. *Supercomputers and Their Use.* Oxford Science Publications, Oxford, England, 1985.

[148] L.J. LeBlanc and K. Farhangian. Selection of a trip table which reproduces observed link flows. *Transportation Research*, 16B:83–88, 1982.

[149] A. Lent. A convergent algorithm for maximum entropy image restoration, with a medical x-ray application. In R. Shaw, editor, *Image Analysis and Evaluation*, pages 249–257. Society of Photographic Scientists and Engineers, Washington, D.C., 1977.

[150] A. Lent and Y. Censor. Extensions of Hildreth's row-action method for quadratic programming. *SIAM Journal on Control and Optimization*, 18:444–454, 1980.

[151] A. Lent and Y. Censor. The primal-dual algorithm as a constraint-set manipulation device. *Mathematical Programming*, 50:343–357, 1991.

[152] I.J. Lustig, J.M. Mulvey, and T.J. Carpenter. Formulating two-stage stochastic programs for interior point methods. *Operations Research*, 39:757–770, 1991.

[153] J. Mandel. Convergence of the cyclical relaxation method for linear equalities. *Mathematical Programming*, 30:218–228, 1984.

[154] H. Markowitz. Portfolio selection. *Journal of Finance*, 7:77–91, 1952.

[155] H. Markowitz. *Mean-Variance Analysis in Portfolio Choice and Capital Markets.* Basil Blackwell, Oxford, 1987.

[156] S.F. McCormick. The methods of Kaczmarz and row orthogonalization for solving linear equations and least squares problems in Hilbert space. *Indiana Univ. Math. J.*, 26:1137–1150, 1977.

[157] S. McNeil. *Quadratic Matrix Estimation Methods.* PhD thesis, Department of Civil Engineering, Carnegie-Mellon University, Pittsburgh, 1983.

[158] R.E. Miller and P.D. Blair. *Input-Output Analysis. Foundations and Extensions.* Prentice Hall, N. Jersey, 1985.

[159] M. Minoux. *Mathematical Programming: Theory and Algorithms.* John Wiley and Sons, New York, N.Y., 1986.

[160] G.J. Minty. Monotone (nonlinear) operators in Hilbert space. *Duke Mathematical Journal*, 29:341–346, 1962.

[161] MIPG, Department of Radiology, Hospital of the University of Pennsylvania, Philadelphia, PA 19104. *SNARK77 User's Manual, Version 2*, 1978.

[162] J.J. Moreau. Proximité et dualité dans un espace Hilbertien. *Bull. Soc. Math. France*, 93:273–299, 1965.

[163] O. Morgenstern. *On the accuracy of economic observations*. Princeton University Press, Princeton, New Jersey, 1963.

[164] W.I. Morrison and R.G. Thumann. A Lagrangian multiplier approach to the solution of a special constrained matrix problem. *Journal of Regional Science*, 20:279–292, 1980.

[165] T.S. Motzkin and I.J. Schoenberg. The relaxation method for linear qualities. *Canadian Journal of Mathematics*, 6:393–404, 1967.

[166] The MRPIS Project, The Social Welfare Research Institute, Boston College. *The Multiregional Input-Output Accounts for 1977*, Feb. 1988.

[167] J.M. Mulvey and A. Ruszczynski. A diagonal quadratic approximation method for large scale linear programs. *Operations Research Letters*, 12:205–215, 1992.

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

[169] J.M. Mulvey and H. Vladimirou. Stochastic network optimization models for investment planning. *Annals of Operations Research*, 20:187–217, 1989.

[170] J.M. Mulvey and H. Vladimirou. Solving multistage stochastic networks: An application of scenario aggregation. *Networks*, 21:619–643, 1991.

[171] J.M. Mulvey and H. Vladimirou. Stochastic network programming for financial planning problems. *Management Science*, 38:1643–1664, 1992.

[172] J.M. Mulvey and S.A. Zenios. Capturing the correlations of fixed-income instruments. *Management Science*. (to appear).

[173] F. Natterer. *The Mathematics of Computerized Tomography*. B.G. Teubner, 1986.

[174] S. Nguyen. An algorithm for the traffic assignment problem. *Transportation Science*, 8:203–216, 1974.

[175] S. Nguyen. Estimation of origin-destination matrices from observed flows. In M. Florian, editor, *Transportation Planning Models*. Elsevier Science Publishers, Amsterdam, 1984.

[176] S. Nielsen and S.A. Zenios. Data structures for network algorithms on massively parallel architectures. *Parallel Computing*, 18:1033–1052, 1992.

[177] S. Nielsen and S.A. Zenios. Massively parallel algorithms for nonlinear stochastic network problems. *Operations Research*, 41, 1993. (to appear).

[178] S.S. Nielsen and S.A. Zenios. Proximal minimizations with D-functions and the massively parallel solution of linear stochastic network programs. Report 92–01–05, Decision Sciences Department, The Wharton School, University of Pennsylvania, Philadelphia, PA, 1992.

[179] S.S. Nielsen and S.A. Zenios. Solving multistage stochastic network programs. Report 92–08–04, Decision Sciences Department, The Wharton School, University of Pennsylvania, Philadelphia, PA19104, 1992.

[180] S.S. Nielsen and S.A. Zenios. Proximal minimizations with D-functions and the massively parallel solution of linear network programs. *Computational Optimization and Applications*, 1(4):375–398, 1993.

[181] G. Nolet, editor. *Seismic Tomography: With Applications in Global Seismology and Exploration Geophysics.* D. Reidel Publishing Company, Dordrecht, Holland, 1987.

[182] A.R. Odoni. The flow management problem in air-traffic control. In A.R. Odoni, L. Bianco, and G. Szegő, editors, *Flow Control of Congested Networks*, pages 269–288, New York, 1987. Springer-Verlag.

[183] W. Oettli. Symmetric duality, and a convergent subgradient method for discrete linear, constrained approximation problems with arbitrary norms appearing in the objective function and in the constraints. *Journal of Approximation Theory*, 14:43–50, 1975.

[184] J.M. Ortega and W.C. Rheinboldt. *Iterative Solution of Nonlinear Equations in Several Variables.* Academic Press, New York, 1970.

[185] Parasoft Corporation, Pasadena, CA. *Express FORTRAN: User's Guide*, 1990.

[186] C.E. Pfefferkorn and J.A. Tomlin. Design of a linear programming system for the Illiac IV. Technical report, Department of Operations Research, Stanford University, Stanford, California, April 1976.

[187] A.R. De Pierro. Multiplicative iterative methods in computed tomography. In G.T. Herman, A.K. Louis, and F. Natterer, editors, *Mathematical Methods in Tomography, Lecture Notes Mathematics*, volume 1497, pages 167–186, Berlin, Germany, 1991. Springer-Verlag.

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

[189] D.A. Plane. An information theoretic approach to the estimation of migration flows. *Journal of Regional Science*, 22:441–456, 1982.

[190] F. Van Der Ploeg. Reliability and the adjustment of sequences of large economic accounting matrices. *Journal of the Royal Statistical Society*, 145:169–194, 1982.

[191] G. Pyatt and J. I. Round, editors. *Social Accounting Matrices: A Basis for Planning.* The World Bank, Washington, D.C., 1985.

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

[193] R-J. Qi and S.A. Zenios. On the scalability of data-parallel decomposition algorithms for stochastic programs. Working paper, Decision Sciences Department, The Wharton School, University of Pennsylvania, Philadelphia, PA 19104, 1993.

[194] E. Raik. Fejer type methods in Hilbert space. *Esti. NSV Tead. Akad. Toimetised Fuus.-Mat.*, 16:286–293 (in Russian), 1967.

[195] O. Richetta. *Ground holding strategies for air traffic control under uncertainty.* PhD thesis, Department of Electrical Engineering and Computer Science, Massachusetts Institute of Technology, Cambridge, MA, 1991.

[196] R. T. Rockafellar. *Convex Analysis.* Princeton University Press, Princeton, New Jersey, 1970.

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

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

[199] R.T. Rockafellar and R.J.-B. Wets. Scenarios and policy aggregation in optimization under uncertainty. *Mathematics of Operations Research*, 16:119–147, 1991.

[200] J.B.T.M. Roerdink. Computerized tomography and its applications: A guided tour. Technical report, Department of Computer Science, University of Groningen, 1992.

[201] R. E. Rosenthal. A nonlinear network flow algorithm for maximization of benefits in a hydroelectric power system. *Operations Research*, 29:763–786, 1981.

[202] U.G. Rothblum. Generalized scaling satisfying linear equations. *Linear Algebra and Its Applications*, 114/115:765–784, 1989.

[203] U.G. Rothblum. Linear inequality scaling problems. *SIAM Journal on Optimization*, 2:635–648, 1992.

[204] U.G. Rothblum and S.A. Zenios. Scaling of matrices satisfying line-product constraints and generalizations. *Linear Algebra and its Applications*, 175:159–175, 1992.

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

[206] M. H. Schneider. Matrix scaling, entropy minimization and conjugate duality. I. Existence conditions. *Linear Algebra and Its Applications*, 114/115:785–813, 1989.

[207] M. H. Schneider. Matrix scaling, entropy minimization and conjugate duality. II. The dual problem. *Mathematical Programming*, 48:103–124, 1990.

[208] M. H. Schneider and S. A. Zenios. A comparative study of algorithms for matrix balancing. *Operations Research*, 38:439–455, 1990.

[209] M.I. Sezan and H. Stark. Applications of convex projection theory to image recovery in tomography and related areas. In H. Stark, editor,

Image Recovery: Theory and Application, pages 415–462. Academic Press, New York, 1987.

[210] Y. Sheffi. *Urban Transportation Networks.* Prentice Hall, Englewood Cliffs, N.J., 1985.

[211] L.A. Shepp and Y. Vardi. Maximum likelihood reconstruction in emission tomography. *IEEE Transactions on Medical Imaging*, MI-1:113–122, 1982.

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

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

[214] H. Stark, editor. *Image Recovery: Theory and Applications.* Academic Press, Inc., 1987.

[215] F.F. Stephan. An iterative method of adjusting sample frequency tables when expected marginal totals are known. *The Annals of Mathematical Statistics*, 13:166–178, 1942.

[216] R. Stone. The development of economic data systems. In G. Pyatt et al., editors, *Social Accounting for Development Planning with.* Cambridge University Press, U.K., 1976.

[217] K. Tanabe. Projection method for solving a singular system of linear equations and its applications. *Numerische Mathematik*, 17:203–214, 1971.

[218] K. Tanabe. Characterization of linear stationary iterative processes for solving a singular system of linear equations. *Numerische Mathematik*, 22:349–359, 1974.

[219] M. Teboulle. Entropic proximal mappings with applications to nonlinear programming. *Mathematics of Operations Research*, 17:670–690, 1992.

[220] H. Theil. *Economics and Information Theory.* North-Holland, Amsterdam, 1967.

[221] H. Theil and G. Rey. A quadratic programming approach to the estimation of transition probabilities. *Management Science*, 12:714–721, 1966.

[222] P. Tseng and D.P. Bertsekas. On the convergence of the exponential multiplier method for convex programming. Technical report no. lids-p-1995, Laboratory for Information and Decision Systems, MIT, 1992.

[223] J.K. Udupa and G.T. Herman, editors. *3D Imaging in Medicine.* CRC Press, 1991.

[224] A. van der Sluis and H.A. van der Vorst. SIRT- and CG-type methods for the iterative solution of sparse linear least-squares problems. *Linear Algebra and Its Applications*, 130:257–302, 1990.

[225] J.W. van Tongeren. Development of an algorithm for the compilation of National accounts and related statistics. *Review of Income and Wealth*, 32:25–68, 1986.

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

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

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

[229] K.J. Worzel, C.V. Zenios, and S.A. Zenios. Integrated simulation and optimization models for tracking fixed-income indices. *Operations Research.* (to appear).

[230] D.C. Youla. Mathematical theory of image restoration by the method of convex projections. In H. Stark, editor, *Image Recovery: Theory and Application*, pages 29–77. Academic Press, New York, 1987.

[231] D.M. Young. *Iterative Solution of Large Linear Systems.* Academic Press, New York, 1971.

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

[233] S.A. Zenios. Matrix balancing on a massively parallel Connection Machine. *ORSA Journal on Computing*, 2:112–125, 1990.

[234] S.A. Zenios. Massively parallel computations for financial modeling under uncertainty. In J. Mesirov, editor, *Very Large Scale Computing in the 21-st Century*, pages 273–294. SIAM, Philadelphia, PA, 1991.

[235] S.A. Zenios. Network based models for air-traffic control. *European Journal of Operational Research*, 49:166–178, 1991.

[236] S.A. Zenios. Asset/liability management under uncertainty: The case of mortgage-backed securities. Report 92–08–05, Decision Sciences Department, The Wharton School, University of Pennsylvania, Philadelphia, PA19104, 1992.

[237] S.A. Zenios, editor. *Financial Optimization.* Cambridge University Press, 1992.

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

[239] S.A. Zenios, A. Drud, and J.M. Mulvey. Balancing large social accounting matrices with nonlinear network programming. *Networks*, 17:569–585, 1989.

[240] S.A. Zenios and S-L. Iu. Vector and parallel computing for matrix balancing. *Annals of Operations Research*, 22:161–180, 1990.

[241] S.A. Zenios and P. Kang. Mean-absolute deviation portfolio optimization for mortgage backed securities. *Annals of Operations Research.* (to appear).

[242] H.J. Van Zuylen and L.G. Willumsen. The most likely trip matrix estimated from traffic counts. *Transportation Research*, 14B:281–293, 1980.

Impresso na Gráfica do
impa
pelo Sistema Xerox/1065