DOE GRANT NO. DE-FG06-87ER25045

**PARALLEL ALGORITHMS FOR STOCHASTIC LINEAR PROGRAMS—
A SUMMARY OF RESEARCH PERFORMED**

K.A. Ariyawansa
Department of Pure and Applied Mathematics
Washington State University
Pullman
Washington 99164-2930

DOE/ER/25045--T3

DE91 001306

DISCLAIMER

This report was prepared as an account of work sponsored by an agency of the United States Government. Neither the United States Government nor any agency thereof, nor any of their employees, makes any warranty, express or implied, or assumes any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States Government or any agency thereof. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States Government or any agency thereof.

DOE/ER/25045--T3

27

1. Introduction

This report is a summary of activities performed in the research project "Parallel Algorithms for Stochastic Linear Programs" during the period September 1, 1987 to August 31, 1989. The research was sponsored by DOE Grant No. DE-FG06-87ER25045.

The research described in this report has been carried out by the principal investigator and his graduate research assistants Ms. Kelly S. Lessor and Mr. David D. Hudson. Research performed has been benefitted by collaborations and communications with Professor R.J. B. Wets (Department of Mathematics, University of California at Davis, Davis, CA 95616) and Dr. D.C. Sorensen (Mathematics and Computer Science Division, Argonne National Laboratory, 9700 South Cass Avenue, Argonne, IL 60439).

In the sequel we shall freely refer to the appropriate sections of the proposal submitted to DOE that led to the award of the above grant. In particular, we shall have occasion to refer to the objectives of the project mentioned in Section 5 of that research proposal. When in this report we need to refer to the research proposal we shall do so by indicating within square brackets the letter P and the item number of the relevant section, equation, or the figure of the proposal.

In this report our use of mathematical symbols is minimal. We shall use technical terms without providing definitions, unless they would not be discernible to a person familiar with optimization algorithms in general or they are not defined in more precise terms either in the proposal [P] or in the appendices of this report. Thus this report serves as an extended abstract of the technical details provided in the appendices.

Two technical reports, a precise statement of an algorithm, and the MS project report of Ms. Lessor are provided as appendices to this report.

The rest of this report is structured as follows. In Section 2 we provide a brief description of the research carried out. We do so by using the objectives of the project [P, Section 5] as a basis. Section 3 is a brief description of the activities of the two graduate research assistants supported by this grant. In Section 4 we indicate the meetings attended and presentations made by the principal investigator and his graduate students.

2. A summary of research performed

We describe the research carried out with reference to the objectives of the project [P, Section 5]. The following four major accomplishments have been made. The four technical appendices provide the technical details pertinent to these four accomplishments.

(2.1) Accomplishment 1:

This is related to [P, Section 5, Objectives 1, 5]. Given a stochastic linear program

in the form of [P, Equation 5], [P, Section 5, Objective 1] was concerned with the development of an algorithm to generate a collection $\mathcal{D}$ of nonsingular matrices in $\mathbb{R}^{m_2 \times m_2}$ to be used in [P, Equation 17]. Of course, as explained in [P, Sections 3,4], $\psi_{\mathcal{D}}$ of [P, Equation 17] provides an upper bound on the function ψ of [P, Equation 8]. As illustrated in [P, Figure 1], such a collection $\mathcal{D}$ would therefore define an upper bound $Q_{\mathcal{D}}$ on the expected recourse function Q of the stochastic linear program [P, Equation 5]. The underlying idea is to approximate the values and subgradients of Q (which are difficult to compute and are needed in an algorithm for the stochastic linear program [P, Equation 5] for *many argument values*) with the values and subgradients of $Q_{\mathcal{D}}$ (which are relatively easy to compute [P, Section 3]).

Such an algorithm to construct a collection $\mathcal{D}$ has been developed. It is described and derived in Appendix 1 and is stated formally as Algorithm 3.1 therein. We now state the following desirable features of Algorithm 3.1 of Appendix 1.

- (i) The functions ψ and $\psi_{\mathcal{D}}$ agree along a set of directions that are specified as input to Algorithm 3.1 of Appendix 1.
- (ii) As explained in Sections 3.1 of Appendix 1 the data defining $\psi_{\mathcal{D}}$ when $\mathcal{D}$ is produced by Algorithm 3.1 has special structure that may be exploited for the purposes of computation of values and subgradients of $\psi_{\mathcal{D}}$.
- (iii) As indicated in Section 4 of Appendix 1 the properties mentioned in (ii) may be utilized to design schemes to compute values and subgradients of $Q_{\mathcal{D}}$ suitable for parallel vector processors.

In [P, Section 5, Objective 5], we proposed to develop a lower bound on the expected recourse function Q of [P, Equation 7]. Such a lower bound, $Q_{\mathcal{V}}$, indexed by a collection $\mathcal{V}$ of vectors in $\mathbb{R}^{m_2}$ is also described in Appendix 1. Indeed, Algorithm 3.1 of Appendix 1 generates the collection $\mathcal{V}$ while it generates the collection $\mathcal{D}$ that defines the upper bound $Q_{\mathcal{D}}$.

(2.2) Accomplishment 2:

- (i) This is really not related to an explicitly stated objective in [P, Section 5]. The upper bound $Q_{\mathcal{D}}$ mentioned in (2.1) is induced by a pointwise infimum of a finite collection of convex functions [Appendix 1, Equation 11]. It is a well known fact that a function defined as a pointwise infimum of a set of convex functions is not necessarily convex. This implies that unless special care is taken in designing $\mathcal{D}$, $Q_{\mathcal{D}}$ may not be convex. Of course Q which $Q_{\mathcal{D}}$ is supposed to approximate is convex and hence the mathematical program in [P, Equation 5] is convex. If Q in [P, Equation 5] is replaced by the nonconvex $Q_{\mathcal{D}}$ then the resulting mathematical program that is supposed to approximate [P, Section 5] would be nonconvex. Since nonconvex mathematical programs are difficult to solve than convex ones, in trying

to approximate Q by a function $Q_{\mathcal{D}}$ that is suitable for faster parallel processing, we have destroyed a desirable mathematical property of Q . We therefore spent a considerable amount of time on the difficult question of deciding on modifications necessary to [Appendix 1, Algorithm 3.1], so that $Q_{\mathcal{D}}$ it produces is convex and possesses the three desirable properties we indicated in (2.1). The modified form of [Appendix 1, Algorithm 3.1] we have developed is stated in Appendix 2. It should be pointed out that this task took more time than we anticipated due to the technical difficulty of the underlying mathematical question: How should one select a finite collection of convex functions of the type in [Appendix 1, Equation 11], so that the function defined as their pointwise infimum is convex?

(ii) We should point out that this problem of nonconvexity of $Q_{\mathcal{D}}$ defined by $\mathcal{D}$ produced by [Appendix 1, Algorithm 3.1] arises only if $m_2 > 2$. For $m_2 = 2$, [Appendix 1, Algorithm 3.1] and the algorithm in Appendix 2 are identical. Indeed, it can be shown that when $m_2 = 2$, the input to [Appendix 1, Algorithm 3.1] can be selected so that $Q_{\mathcal{D}}$ is *exact*.

More details on (i) and (ii) may be found in [1] (which is being revised).

(2.3) Accomplishment 3:

This is related to [P, Section 5, Objective 2]. We proposed to implement and computationally test the algorithm developed as part of [P, Section 5, Objective 1]. The algorithm mentioned in (2.1) has been implemented. Details of this implementation may be found in Appendix 3. Appendix 3 also describes results of a preliminary numerical evaluation of $\mathcal{D}$ produced by this implementation of Algorithm 3.1 of Appendix 1 on the Sequent/Balance 8000 of the Department of Computer Science, Washington State University. Since as stated in (2.2) (ii), the input to Algorithm 3.1 of Appendix 1 can be chosen so that $Q_{\mathcal{D}}$ is exact for $m_2 = 2$ in problems [P, Equation 5], the evaluation was limited to random instances of [P, Section 5] generated by the test problem generator of Kall and Keller [2] with $m_2 = 2$. Apart from this restriction the evaluation was quite extensive. We refer the reader to Sections 3 and 4 of Appendix 2 for details and indicate the major observations here.

(i) The exact computation, namely the computation of values and subgradients of Q , as illustrated in Figure 1 of Appendix 1 parallelizes almost ideally except when K the number of realizations associated with the (discrete or discretized) probability distribution F of [P, Equation 5] is 'small'. (Of course meaningful problems have 'large' K .)

(ii) The upper bound computation, namely the computation of values and subgradients of $Q_{\mathcal{D}}$, as illustrated in Figure 2 of Appendix 1 together with schemes that exploit properties of $\mathcal{D}$ mentioned in (2.1)(ii), does not parallelize as well as the exact

computation.

(iii) The upper bound computation is significantly faster than the exact computation, when the comparison is done with the same number of processors on the Sequent. We refer the reader to Tables 4.5 through 4.8 of Appendix 2 for the very encouraging speed-up values observed.

We believe that the parallelization of the upper bound computation (and hence its speed relative to the exact computation) will be improved when the evaluation is performed on parallel, vector processors. (The Sequent is a parallel, scalar processor.) Research on schemes like Algorithm 4.1 of Appendix 1 for performing the upper bound computation on parallel, vector processors is in progress.

(2.4) Accomplishment 4:

This is related to [P, Section 5, Objective 3]. We proposed to develop a parallel version of the *exact* L-shaped algorithm. Such a version has been developed and is stated as Algorithm 3.1 in Appendix 4. We have also developed an implementation of this algorithm for the Sequent/Balance. Section 4 of Appendix 4 contains results of a carefully designed experiment on the Sequent/Balance using this implementation. These performance results, to the best of our knowledge, are the first available for the L-shaped algorithm on a parallel processor. Two general observations can be made regarding these performance results: (a) Our implementation of the exact L-shaped algorithm parallelizes well. See the speed-up values indicated in Section 4 of Appendix 4. (b) Even with the use of parallel processing, *exact* solution of stochastic programs take prohibitively large amounts of computation when the number of realizations of the underlying random variables are large. See the parallel cpu time values indicated in Section 4 of Appendix 4.

3. Graduate Students Supported

Ms. Kelly S. Lessor, MS student in the Department of Computer Science, Washington State University, assisted the principal investigator during the first year of this grant. She completed her MS degree in August 1988 and is currently employed by Advanced Nuclear Fuels Inc. of Richland, Washington. The work she performed for this project formed the basis of her MS project report which forms Appendix 2 of this report. Mr. David D. Hudson, MS student in the Department of Pure and Applied Mathematics, assisted the principal investigator during the second year of this grant. The work that he performed with the principal investigator is reported in Appendix 4.

4. Travel

The following conferences were attended using funds from this research grant.

- (4.1) TIMS/ORSA Conference, Washington, D.C., April 25-27, 1988.

Title of Presentation: On collinear scaling algorithms for sparse unconstrained minimization (joint presentation with graduate student D.T.M. Lau).

- (4.2) 13th International Symposium on Mathematical Programming, Tokyo, Japan, August 29-September 2, 1988.

Title of presentation: On implementing a parallel scheme to compute approximate values and subgradients of the expected recourse function in stochastic programs with complete recourse.

- (4.3) Third SIAM Conference on Optimization, Boston, Massachusetts, April 3-5, 1989.

Title of Presentation: Local convergence of collinear scaling algorithms that extend quasi-Newton methods with updates from the Broyden family (joint presentation with graduate student D.T.M. Lau).

- (4.4) SIAM Annual Meeting, San Diego, California, July 17-21, 1989.

Title of Presentation: A collinear scaling algorithm for sparse unconstrained minimization with sparse Cholesky factor approximants (joint presentation with graduate student D.T.M. Lau).

In addition, the principal investigator was invited to make a presentation titled "On the use of parallel processing in the solution of stochastic linear programs with complete recourse", at the Department of Mathematics of University of Washington, Seattle, Washington. This presentation was made on January 25, 1988. Funds from this research grant were not used for this travel. The principal investigator will also make a presentation titled "Parallel processing in the solution of two-stage stochastic linear programs with complete recourse" at the Fourth SIAM Conference on Parallel Processing for Scientific Computing, to be held in Chicago, Illinois, during December 11-13, 1989.

5. References

- [1] Ariyawansa, K.A., D.C. Sorensen and R.J-B. Wets (1987), "On the convexity of an upper bounding approximant to the optimum value of linear programs with varying right hand side". (Being revised.)
- [2] Kall, P and E. Keller (1985), "GENSLP: A program for generating input for stochastic linear programs with complete fixed recourse", Manuscript, Institut für Operations Research der Universität Zürich, CH-8006.

Appendix 1

This appendix is part of the final report of DOE Grant No. DE-FG06-87ER25045. The technical report which forms this appendix is under revision.

Parallel schemes to approximate values and subgradients of the recourse function in certain stochastic programs

K. A. Ariyawansa¹, D. C. Sorensen² and R. J. -B. Wets³

Abstract: We consider the class of stochastic linear programs with fixed, relatively complete recourse. The major difficulty in solving such programs is due to the prohibitively large amount of computation needed to obtain values and subgradients of the expected recourse function. In this paper we discuss schemes to approximate values and subgradients of the expected recourse function. These schemes are well suited for parallel processing.

1. Introduction

Consider the class of stochastic programs

find $x \in \mathbb{R}^{n_1}$ such that

$z(x) = c^T x + Q(x)$ is minimized, and

$Ax = b, x \geq 0$

where

$Q(x) = E[Q(x, \xi)],$

$Q(x, \xi) = \inf_{y \in \mathbb{R}^{n_2}} \{q^T y : My = h - Tx, y \geq 0\},$

$A \in \mathbb{R}^{m_1 \times n_1}$, $b \in \mathbb{R}^{m_1}$, $c \in \mathbb{R}^{n_1}$, $q \in \mathbb{R}^{n_2}$, $M \in \mathbb{R}^{m_2 \times n_2}$ are deterministic and given and $h \in \mathbb{R}^{m_2}$, $T \in \mathbb{R}^{m_2 \times n_1}$ are random with a given cumulative probability distribution F of $\xi =$

$[h_1, h_2, \dots, h_{m_2}, T_{11}, T_{12}, \dots, T_{1n_1}, T_{21}, T_{22}, \dots, T_{2n_1}, \dots, T_{m_2 1}, T_{m_2 2}, \dots, T_{m_2 n_1}]^T$
on $\Xi \subset \mathbb{R}^N$, with $N = m_2 + m_2 n_1$.

(1)

We refer the reader to the survey papers by Dempster (1980), Wets (1982) and by Ermoliev and Wets (1984) (and references contained therein) for details on theory, algorithms and applications pertinent to (1).

In this paper we make the following assumptions on problem (1).

(A1) The set $X = \{x : x \in \mathbb{R}^{n_1}, Ax = b, x \geq 0\}$ is nonempty.

(A2) For all $x \in X$, $Q(x) < \infty$.

¹ Department of Pure and Applied Mathematics, Washington State University, Pullman, WA 99164-2930, U.S.A.

² Mathematics and Computer Science Division, Argonne National Laboratory, 9700 South Cass Avenue, Argonne, IL 60439, U.S.A.

³ Department of Mathematics, University of California at Davis, Davis, CA 95616, U.S.A.

(A3) The rank of M is m_2 and the set $\{v : v \in \mathbb{R}^{m_2}, M^T v \leq q\}$ is nonempty.

(A4) The random variables h and T have finite first moments with respect to F .

The function Q in (1) is referred to as the *recourse function* and when assumption (A2) is satisfied we say that problem (1) has *relatively complete recourse*. If $Q(x) < \infty$ for all $x \in \mathbb{R}^{n_1}$ (which implies relatively complete recourse) (1) is said to have *complete recourse* while if $M = [I, -I]$ (which implies complete recourse) (1) is said to have *simple recourse*. It is the case of (1) with relatively complete recourse that we consider in this paper.

Assumption (A3) simply states that the dual of the linear program that defines the recourse function is feasible.

When assumptions (A3) and (A4) are satisfied it can be shown that we do not lose generality by assuming $q \geq 0$ in (1). We demonstrate this fact in the appendix of the paper. Consequently, in the sequel we shall also assume that

(A5) $q \geq 0$.

Problem (1) is a convex program. However, solving (1) using an algorithm for general convex programs is difficult because obtaining values and subgradients of Q is difficult unless Q , Ξ and F have special structure that can be exploited. The difficulty in computing values and subgradients of Q are two fold. First, to compute Q and ∂Q at a given x , difficult multiple quadrature on $\mathbb{R}^N$ is needed. Secondly, a single evaluation of Q needs the solution of a linear program (lp)—a multiple quadrature routine to compute $Q(x)$ may need evaluation of $Q(x, \xi)$ for several $\xi \in \Xi$.

Multiple quadrature is usually avoided by using a discretization $\Xi^K = \{\xi^1, \xi^2, \dots, \xi^K\}$ of Ξ and a corresponding discretization $P^K = \{p^1, p^2, \dots, p^K\}$ of F where $\Pr(\xi = \xi^k) = p^k$, $k = 1, 2, \dots, K$. We shall let the realizations of h and T contained in ξ^k be h^k and T^k respectively for $k = 1, 2, \dots, K$. In view of assumptions (A2) and (A5), we note that h^k and T^k must be selected so that for every $x \in X$, $(h^k - T^k x) \in \text{pos } M = \{w : w \in \mathbb{R}^{m_2}, My = w, y \geq 0\}$, $k = 1, 2, \dots, K$. In the rest of the paper we shall assume that Ξ^K and P^K specifying such a discrete probability distribution are available to us.

We now have that for $x \in X$

$$Q(x) = \sum_{k=1}^K p^k Q(x, \xi^k) \quad (2.1)$$

$$\partial Q(x) = - \sum_{k=1}^K p^k (T^k)^T v^*(x, \xi^k) \quad (2.2)$$

where for $k = 1, 2, \dots, K$

$$Q(x, \xi^k) = \min_{y \in \mathbb{R}^{n_2}} \{q^T y : My = h^k - T^k x, y \geq 0\} \quad \text{and} \quad (2.3)$$

FIGURE 1. EXACT COMPUTATION OF $Q(x)$ AND $\partial Q(x)$

$$v^*(x, \xi^k) = \operatorname{argmax}_{v \in \mathbb{R}^{m_2}} \{[h^k - T^k x]^T v : M^T v \leq q\}. \quad (2.4)$$

In (2.2) and in the rest of the paper for $f : \mathbb{R}^n \rightarrow \mathbb{R} \cup \{\infty\}$ the symbol $\partial f(x)$ denotes the set of subgradients of f at x or a member of that set depending on the context.

Using (2.1) and (2.2) to compute $Q(x)$ and $\partial Q(x)$ is still difficult because we usually desire to have 'large' K ($K \approx 10^4$ – 10^6) and an optimization algorithm usually requires $Q(x)$ and $\partial Q(x)$ at several x values.

Suppose now that we have a parallel computer with n_p processors. Then it is clear that we can solve the K lp's on the right hand side of (2.3) (to obtain the minima $Q(x, \xi^k)$ and the dual maximizers $v^*(x, \xi^k)$, $k = 1, 2, \dots, K$) by solving n_p of them at a time in parallel. We illustrate the computation of $Q(x)$ and $\partial Q(x)$ schematically in figure 1. While this is certainly a possibility it would be desirable if the work that we perform in parallel could be reduced from that of solving linear programs.

In a recent paper, Wets (1985) has proposed a certain approximation Q_D to Q such that

$$Q(x) \leq Q_D(x) \quad \text{for } x \in X. \quad (3)$$

The reader is also referred to Birge and Wets (1986a,b,c) for other related material. The approximation Q_D is based on a collection $\mathcal{D}$ of nonsingular matrices in $\mathbb{R}^{m_2 \times m_2}$. The computation of $Q_D(x)$ and $\partial Q_D(x)$ are well suited for parallel processing. The amount of work associated with computing $Q_D(x)$ and $\partial Q_D(x)$ depends on the structure of $\mathcal{D}$, but the most expensive computation is the solution of a system of equations of the form $D(t^k(x)) = h^k - T^k x$ for $t^k(x)$, $k = 1, 2, \dots, K$ for each $D \in \mathcal{D}$. If $\nu = \operatorname{card}(\mathcal{D})$ then we need to solve $K\nu$ systems of equations for each x and note that these $K\nu$ systems could be solved in parallel n_p at a time. Since solving a system of linear equations generally requires less computation than solving a linear program, we see that computing $Q_D(x)$ and $\partial Q_D(x)$ in a parallel processing scheme may be cheaper than computing $Q(x)$ and $\partial Q(x)$ in parallel as indicated in the previous paragraph.

Wets (1985) only provides guidelines on the design of the collection $\mathcal{D}$. In this paper we provide a specific scheme for developing a collection $\mathcal{D}$. In particular, we show that for $\mathcal{D}$ developed this way the number of systems of equations that need to be solved in parallel for each x is considerably less than $K\nu$.

Since Q_D only provides an upper bound for Q on X we also indicate a way to compute a lower bound Q_γ for Q on X so that

$$Q_\gamma(x) \leq Q(x) \leq Q_D(x) \quad \text{for } x \in X. \quad (4)$$

The lower bound Q_V depends on a collection V of vectors in $\mathbb{R}^{m_2}$ and we indicate a way to design V while we design D . Moreover, computing $Q_V(x)$ and $\partial Q_V(x)$ are also well suited for parallel processing.

The reason for developing bounds Q_V and Q_D of course is to use them in place of Q in (1) for the purpose of solving the resulting 'easier' programs. If z_V^* and z_D^* are the minima that result when Q_V and Q_D are substituted for Q respectively, then $z_V^* \leq z^* \leq z_D^*$, where z^* is the minimum of (1). In this paper however, we do not consider details of algorithms for solving (1) or its approximants obtained by using Q_V and Q_D in place of Q . We shall be concerned only with approximants of the form (4) for Q and the design of collections V and D so that approximate values of $Q(x)$ and $\partial Q(x)$ may be computed cheaply using Q_V and Q_D in parallel processing schemes.

The rest of the paper is structured as follows. The bounds Q_V and Q_D for Q on X are really induced by two approximants $\psi_V : \mathbb{R}^{m_2} \rightarrow \mathbb{R} \cup \{\infty\}$ and $\psi_D : \mathbb{R}^{m_2} \rightarrow \mathbb{R} \cup \{\infty\}$ respectively, for $\psi : \mathbb{R}^{m_2} \rightarrow \mathbb{R} \cup \{\infty\}$ defined by $\psi(w) = Q(x, \xi)$, $w = h - Tx$, so that $\psi_V(w) \leq \psi(w) \leq \psi_D(w)$. In section 2 we describe how the approximants ψ_V and ψ_D are constructed. The description in section 2 assumes that we have at hand appropriate collections D and V relative to which ψ_D and ψ_V are defined. We also indicate how ψ_D and ψ_V lead to schemes for computing $Q_D(x)$, $\partial Q_D(x)$, $Q_V(x)$ and $\partial Q_V(x)$ that are suitable for parallel processing. Section 3 of the paper is devoted to a detailed discussion of a specific way we propose to develop the collection D and the way we design the collection V while we design D . We end section 3 with an algorithm that sets up the data structures pertinent to D and V which are efficient in terms of storage while leading to streamlined procedures for computing $Q_D(x)$, $\partial Q_D(x)$, $Q_V(x)$ and $\partial Q_V(x)$. Section 4 of the paper is devoted to a brief discussion on the suitability of the schemes proposed for implementation on shared memory multiprocessors. We end the paper in section 5 where we comment on the use of the schemes proposed in conjunction with optimization algorithms for (1) (like the L-shaped algorithm (Wets 1982)).

We shall adopt the following notational conventions throughout the paper.

$$\overline{\mathbb{R}} := \mathbb{R} \cup \{+\infty\}$$

$$\overline{\overline{\mathbb{R}}} := \mathbb{R} \cup \{+\infty\} \cup \{-\infty\}$$

$$\overline{\mathbb{R}}^n := \overline{\mathbb{R}} \times \overline{\mathbb{R}} \times \cdots \times \overline{\mathbb{R}} \text{ (n times)}$$

$$\overline{\overline{\mathbb{R}}}^n := \overline{\overline{\mathbb{R}}} \times \overline{\overline{\mathbb{R}}} \times \cdots \times \overline{\overline{\mathbb{R}}} \text{ (n times)}$$

$$\mathbb{N} := \text{the set of nonnegative integers}$$

$$\mathbb{N}^n := \text{the set of nonnegative integer n-tuples}$$

As in this section, throughout the paper we shall treat vectors as column vectors

unless specified otherwise. For $x \in \mathbb{R}^n$ the symbol x_i denotes the i th component of x , $i = 1, 2, \dots, n$. For $A \in \mathbb{R}^{m \times n}$ the symbol A_i denotes the i th row of A , $i = 1, 2, \dots, m$ while the symbol A_j denotes the j th column of A , $j = 1, 2, \dots, n$. We shall say that $x^k \in \mathbb{R}^n$ and $A^k \in \mathbb{R}^{m \times n}$ when we need to refer to the k -th member in a sequence of vectors in $\mathbb{R}^n$ and a sequence of matrices in $\mathbb{R}^{m \times n}$ respectively. Components of x^k will be denoted by x_i^k , $i = 1, 2, \dots, n$ while columns, rows and elements of A^k will be denoted by A_j^k , A_i^k and A_{ij}^k respectively for $i = 1, 2, \dots, m$, $j = 1, 2, \dots, n$. The statement $e_j \in \mathbb{R}^n$ means that e_j is the j -th unit vector in $\mathbb{R}^n$ for some $j \in \{1, 2, \dots, n\}$.

For $\lambda \in \mathbb{R}$, $\text{sgn}(\lambda) := \begin{cases} 1 & \text{if } \lambda \geq 0 \\ -1 & \text{if } \lambda < 0. \end{cases}$

For $x \in \mathbb{R}^n$, $\text{sgn}(x) \in \mathbb{R}^n$ is defined by $e_i^T \text{sgn}(x) := \text{sgn}(e_i^T x)$, $i = 1, 2, \dots, n$.

For $x \in \mathbb{R}^n$, $\text{diag}(x)$ shall denote the diagonal matrix in $\mathbb{R}^{n \times n}$ whose ii -th diagonal element is x_i , $i = 1, 2, \dots, n$.

Given $x \in \mathbb{R}^n$, $x_+, x_z, x_- \in \mathbb{R}^n$ are defined respectively by

$$\begin{aligned} (x_+)_i &:= \begin{cases} x_i & \text{if } x_i \geq 0 \\ 0 & \text{otherwise} \end{cases} \\ (x_z)_i &:= \begin{cases} 1 & \text{if } x_i = 0 \\ 0 & \text{otherwise} \end{cases} \\ \text{and } (x_-)_i &:= \begin{cases} x_i & \text{if } x_i < 0 \\ 0 & \text{otherwise} \end{cases} \end{aligned}$$

for $i = 1, 2, \dots, n$.

Given $A \in \mathbb{R}^{m \times n}$, $A_+, A_z, A_- \in \mathbb{R}^{m \times n}$ are defined respectively by

$$\begin{aligned} (A_+)_j &:= (A_j)_+ \\ (A_z)_j &:= (A_j)_z \\ \text{and } (A_-)_j &:= (A_j)_- \end{aligned}$$

for $j = 1, 2, \dots, n$.

2. Approximations for the recourse function

The assumptions that we have made in section 1 imply that during the course of an algorithm for (1) we would need (or need to approximate) the value and subgradients (w.r.t. x) of $Q(x, \xi)$ for $x \in X$ with $(h - Tx) \in \text{pos } M$. We therefore define $\psi : \mathbb{R}^{m_2} \rightarrow \overline{\mathbb{R}}$ by

$$\psi(w) = \inf_{y \in \mathbb{R}^{n_2}} \{q^T y : My = w, y \geq 0\} = \inf \left\{ \alpha : \begin{pmatrix} \alpha \\ w \end{pmatrix} \in \text{pos} \begin{pmatrix} q^T \\ M \end{pmatrix} \right\} \quad (5)$$

and seek ways to approximate ψ and $\partial\psi$ on $\text{pos } M \subseteq \mathbb{R}^{m_2}$. In (5) we shall let $\psi(w) = \infty$ for $w \notin \text{pos } M$. We can now state the following properties of ψ that are useful in the sequel. (See for example Wets (1966).)

- (1) $0 \leq \psi(w) < \infty$ for $w \in \text{pos } M$ and $\psi(0) = 0$.
- (2) ψ is convex on $\text{pos } M$.
- (3) ψ is positively homogeneous on $\text{pos } M$. i.e. $\psi(\lambda w) = \lambda\psi(w)$ for all $\lambda > 0$ for each $w \in \text{pos } M$.
- (4) The epigraph $\text{epi } \psi = \{[\alpha, w^T]^T : \alpha \geq \psi(w), w \in \text{pos } M\}$ of ψ is a convex polyhedral cone.

The approximants $\psi_D : \mathbb{R}^{m_2} \rightarrow \overline{\mathbb{R}}$ and $\psi_V : \mathbb{R}^{m_2} \rightarrow \overline{\mathbb{R}}$ that we define below so that $\psi_V(w) \leq \psi(w) \leq \psi_D(w)$ for $w \in \mathbb{R}^{m_2}$, are based on these properties of ψ .

2.1 The upper bound ψ_D

The bound we discuss below was initially proposed by Wets (1985). It was further discussed in Birge and Wets (1986a,b). The scheme of Wets (1985) is to approximate ψ by attempting to construct $\text{epi } \psi$ as follows. Suppose that we have $\alpha^j \in \overline{\mathbb{R}}$ and $w^j \in \mathbb{R}^{m_2}$ with $\alpha^j \geq \psi(w^j)$, $j = 1, 2, \dots, J$. Let $a = [\alpha^1, \alpha^2, \dots, \alpha^J]^T$, $W = [w^1, w^2, \dots, w^J]$ and $U = \begin{pmatrix} a^T \\ W \end{pmatrix}$. Then

$$\psi(w) \leq \inf \left\{ \alpha : \begin{pmatrix} \alpha \\ w \end{pmatrix} \in \text{pos } U \right\}. \quad (6)$$

In (6) we augment the usual rules of handling ∞ by the convention $\infty \times 0 = 0$.

Note that if $w^j \in \text{pos } M$ for $j = 1, 2, \dots, J$ then (6) simply follows from the fact that $\begin{pmatrix} \alpha^j \\ w^j \end{pmatrix} \in \text{epi } \psi$, $j = 1, 2, \dots, J$ which implies that $\text{pos } U \subseteq \text{epi } \psi$. If on the other hand $w^j \notin \text{pos } M$ for some j then we must necessarily have $\alpha^j = \infty$ and it is possible for the right hand side (rhs) of (6) to be ∞ . If the rhs of (6) is ∞ then (6) would be trivially satisfied. It is easy to see that if $w^j \notin \text{pos } M$ for some j and if (with the convention $\infty \times 0 = 0$) the rhs of (6) is finite for some w then the rhs of (6) would be equal to $\inf \left\{ \alpha : \begin{pmatrix} \alpha \\ w \end{pmatrix} \in \text{pos} \left(\begin{pmatrix} \alpha^j \\ w^j \end{pmatrix}, \alpha^j < \infty, j = 1, 2, \dots, J \right) \right\}$. (6) then follows as in the case of $\alpha^j < \infty$ for $j = 1, 2, \dots, J$.

The rhs of (6) provides a majorization of ψ . However, for general U , computing values and subgradients of the majorization in (6) is no easier than computing values and subgradients of ψ using (5). Wets (1985) proposes a way to select U so that the values and subgradients of the majorization could be computed easily. We now briefly discuss that majorization.

Let $D = [D_{.1}, D_{.2}, \dots, D_{.m_2}] \in \mathbb{R}^{m_2 \times m_2}$ be nonsingular. Then $\text{pos}[D, -D] = \mathbb{R}^{m_2}$. Now define $r, s \in \overline{\mathbb{R}}^{m_2}$ by

$$r = \sum_{j=1}^{m_2} \psi(D_{.j}) e_j, \quad s = \sum_{j=1}^{m_2} \psi(-D_{.j}) e_j. \quad (7.1)$$

If we now define $\psi_D : \mathbb{R}^{m_2} \rightarrow \overline{\mathbb{R}}$ by

$$\psi_D(w) = \inf \left\{ \alpha : \begin{pmatrix} \alpha \\ w \end{pmatrix} \in \text{pos } U \right\}, \quad U = \begin{pmatrix} r^T & s^T \\ D & -D \end{pmatrix} \quad (7.2)$$

then as we indicate below in lemma 2.1, $\psi_D(w)$ and $\partial\psi_D(w)$ can be computed very easily.

Lemma 2.1: Suppose that $\psi_D : \mathbb{R}^{m_2} \rightarrow \overline{\mathbb{R}}$ is defined by (7.1) and (7.2). Let $w \in \mathbb{R}^{m_2}$ be given. Let $t \in \mathbb{R}^{m_2}$ be the solution to the system $Dt = w$ and define the diagonal matrices $Y, Z \in \mathbb{R}^{m_2 \times m_2}$ by

$$Y = \text{diag}(\text{sgn}(t)), \quad Z = \text{diag}(t_z). \quad (8)$$

Then if assumption (A5) is satisfied the value $\psi_D(w)$ and the subgradient $\partial\psi_D(w)$ of ψ_D at w are given respectively by

$$\psi_D(w) = r^T t_+ - s^T t_- \quad \text{and} \quad (9.1)$$

$$\partial\psi_D(w) = \{u : D^T u = v, v = (Y_+ - Z)r + Y_- s + Zh, h \in [-s, r]\}. \quad (9.2)$$

Proof: Let $B = DY$. We now show that B is an optimal basis for the lp on the rhs of (7.2). Since $B^{-1}w \geq 0$, it suffices to show that

$$c_B^T B^{-1}[D, -D] - [r^T, s^T] \leq 0 \quad (10.1)$$

where $c_B^T = r^T Y_+ - s^T Y_-$. A direct computation noting that $Y = Y^{-1}$, $Y_+ Y = Y Y_+ = Y_+$ and $Y_- Y = Y Y_- = -Y_-$ yields

$$c_B^T B^{-1}[D, -D] - [r^T, s^T] = [(r^T(Y_+ - I) + s^T Y_-), (-r^T Y_+ - s^T(Y_- + I))]. \quad (10.2)$$

Now note that $Y_+ - I \leq 0$, $Y_- \leq 0$, $Y_+ \geq 0$ and $Y_- + I \geq 0$ and that by assumption (A5) $r \geq 0$ and $s \geq 0$ to obtain (10.1). Therefore, B is an optimal basis for the lp on the rhs of (7.2) and we have

$$\begin{aligned} \psi_D(w) &= c_B^T B^{-1} w \\ &= [r^T Y_+ - s^T Y_-] Y^{-1} D^{-1} w \\ &= r^T Y_+ t + s^T Y_- t \\ &= r^T t_+ - s^T t_-. \end{aligned}$$

In order to show that (9.2) is true just note that

$$\begin{aligned}\partial\psi_D(w) &= \partial_w(r^T t_+ - s^T t_-) \\ &= D^{-T} \partial_t(r^T t_+ - s^T t_-) \\ &= D^{-T} \partial_t[(r^T Y_+(t) - s^T Y_-(t))t].\end{aligned}\tag{10.3}$$

(9.2) follows from (10.3) and the fact $-s \leq 0 \leq r$. ■

Note that by lemma 2.1 and (7.1)

$$\psi_D(D_{.j}) = r_j = \psi(D_{.j})$$

and that

$$\psi_D(-D_{.j}) = s_j = \psi(-D_{.j})$$

for $j = 1, 2, \dots, m_2$. Consequently, by the discussion at the beginning of this section we have

$$\psi(w) \leq \psi_D(w), \quad \text{for } w \in \mathbb{R}^{m_2}.$$

We also note in passing that computing $\psi_D(w)$ and $\partial\psi_D(w)$ using lemma 2.1 essentially amounts to solving the system $Dt = w$ for t and the system $D^T u = v$ for u .

Now suppose that instead of just one matrix D we have a collection of nonsingular matrices $\mathcal{D} = \{D^i : D^i \in \mathbb{R}^{m_2 \times m_2}, D^i \text{ nonsingular}, i = 1, 2, \dots, \nu\}$. Relative to the collection $\mathcal{D}$ we define the function $\psi_{\mathcal{D}} : \mathbb{R}^{m_2} \rightarrow \overline{\mathbb{R}}$ by

$$\psi_{\mathcal{D}}(w) = \inf_i \{\psi_{D^i}(w), i = 1, 2, \dots, \nu\} = \psi_{D^{i^*}}(w)\tag{11}$$

where $i^* = i^*(w)$ is an index i where the infimum is achieved. We then have that

$$\psi(w) \leq \psi_{\mathcal{D}}(w) \quad \text{for } w \in \mathbb{R}^{m_2}$$

and

$$\psi(\pm\lambda D_{.j}^i) = \psi_{\mathcal{D}}(\pm\lambda D_{.j}^i) \quad \forall \lambda > 0; j = 1, 2, \dots, m_2; i = 1, 2, \dots, \nu.$$

We may therefore approximate ψ and $\partial\psi$ at $w \in \text{pos } M$ as follows.

$$\psi(w) \approx \psi_{\mathcal{D}}(w)\tag{12.1}$$

$$\partial\psi(w) \approx \partial\psi_{\mathcal{D}}(w) = \partial\psi_{D^{i^*}}(w)\tag{12.2}$$

We observe that $\psi_{\mathcal{D}}$ possesses properties (1) and (3) (but not necessarily properties (2) and (4)) that we stated for ψ above. (12.2) however is meaningful only if $\psi_{\mathcal{D}}$ is convex. We

FIGURE 2. COMPUTATION OF $Q_D(x)$ AND $\partial Q_D(x)$.

shall consider the issue of convexity further in section 3. By (2.1) and (2.2), (11) induces the following majorization $\mathcal{Q}_D$ for $\mathcal{Q}$.

$$\mathcal{Q}(x) \leq \mathcal{Q}_D(x) = \sum_{k=1}^K p^k \psi_D(h^k - T^k x) \quad \text{for } x \in X \quad (13.1)$$

with

$$\partial \mathcal{Q}_D(x) = - \sum_{k=1}^K p^k (T^k)^T \partial \psi_D(h^k - T^k x) \quad \text{for } x \in X. \quad (13.2)$$

Suppose now that we have the data D^j , r^j and s^j (where r^j and s^j are obtained by substituting $D = D^j$ in (7.1)) $j = 1, 2, \dots, \nu$ stored. We can then illustrate the computation of $\mathcal{Q}_D(x)$ and $\partial \mathcal{Q}_D(x)$ as in figure 2. We invite the reader to compare figures 1 and 2. In particular, compare the computations in the k -th row of figures 1 and 2. Note that the computations involved in obtaining $\psi_D(h^k - T^k x)$ and $\partial \psi_D(h^k - T^k x)$ in figure 2 are simpler in general than the solution of the linear program involved in obtaining $\psi(h^k - T^k x)$ and $\partial \psi(h^k - T^k x)$. Moreover, the computations to obtain $\psi_D(h^k - T^k x)$ and $\partial \psi_D(h^k - T^k x)$ for each k are well suited for parallel processing. The scheme laid out in figure 2, therefore, exhibits parallelism across rows (corresponding to realizations h^k and T^k) and across columns (corresponding to members D^j in the collection $\mathcal{D}$). We shall have more to say on the organization of these computations in section 4 of the paper.

Of course the simplicity of computing $\mathcal{Q}_D(x)$ and $\partial \mathcal{Q}_D(x)$ is obtained at the expense of having to store D^j , r^j and s^j for $j = 1, 2, \dots, \nu$. As we shall see in section 3, the collection $\mathcal{D}$ that we propose would have a special structure that would eliminate the need to store all of the data D^j , r^j and s^j , $j = 1, 2, \dots, \nu$. Moreover, this specially structured collection $\mathcal{D}$ would further simplify the computation of $\mathcal{Q}_D(x)$ and $\partial \mathcal{Q}_D(x)$.

2.2 The lower bound ψ_ν

It is possible to select $\mathcal{D}$ so that $\psi(w) = \psi_D(w)$ for all $w \in \text{pos } M$, which will ensure that $\mathcal{Q}(x) = \mathcal{Q}_D(x)$ for all $x \in X$. One such collection $\mathcal{D}$ is described in proposition 2.4 of Birge and Wets (1986b). Let us define $\varphi : \mathbb{R}^{m_2} \rightarrow \overline{\mathbb{R}}$ by

$$\varphi(w) = \sup_{v \in \mathbb{R}^{m_2}} \{w^T v : M^T v \leq q\} \quad (14)$$

so that the lp on the rhs of (14) is the dual of the lp on the rhs of (5).

Let $B \in \mathbb{R}^{m_2 \times m_2}$ be a nonsingular submatrix of M . Then B^T is referred to as a feasible basis for (the lp in) (14) if there exists a v such that $B^T v = q_B$ and $M^T v \leq q$. Here $q_B \in \mathbb{R}^{m_2}$ is the vector that consists of the components of q corresponding to the columns of B .

Proposition 2.4 of Birge and Wets (1986b) states that if

$$\mathcal{D} = \{B : B \in \mathbb{R}^{m_2 \times m_2}, B \text{ is a submatrix of } M, B^T \text{ is a feasible basis for (14)}\} \quad (15)$$

then $\psi(w) = \psi_{\mathcal{D}}(w)$ for all $w \in \text{pos } M$. The cardinality of the collection $\mathcal{D}$ in (15) can be very large. If we use a subset of $\mathcal{D}$ of (15) then $\psi_{\mathcal{D}}$ would only provide an upper bound for ψ . Moreover, $\psi_{\mathcal{D}}$ that results from such a subset $\mathcal{D}$ need not be convex.

In section 3 we shall provide a specific scheme for designing a collection $\mathcal{D}$ which would under certain conditions (see Ariyawansa, Sorensen and Wets (1987)) can be made to yield an exact approximant $\psi_{\mathcal{D}}$. The cardinality of this collection, in general, is much small in comparison to that of the collection in (15). It also ensures the desirable property of convexity of $\psi_{\mathcal{D}}$ even when $\mathcal{D}$ is not developed to the extent so that $\psi_{\mathcal{D}}$ is exact.

In solving stochastic programs of the form (1) due to various approximations that have been made (discretization, estimates of probability distributions etc.) usually in practice one desires approximate estimates of optimizers and optima that may be obtained cheaply. It is therefore possible that a collection $\mathcal{D}$ that does not provide an exact $\psi_{\mathcal{D}}$ (even when we have schemes like the one we propose in section 3 where it is practical to develop $\mathcal{D}$ to ensure exactness of $\psi_{\mathcal{D}}$) is used to approximate ψ . In such situations it is desirable to be able to assess how accurate $\psi_{\mathcal{D}}$ is.

With the remarks in the previous paragraph in mind we now indicate a way to construct a lower bound $\psi_{\mathcal{V}}$ for ψ .

Assumption (A3) implies that for $w \in \mathbb{R}^{m_2}$, either $-\infty < \varphi(w) < +\infty$ with a $v^*(w) \in \mathbb{R}^{m_2}$ so that $M^T v^*(w) \leq q$ and $\varphi(w) = w^T v^*(w)$ (the lp on the rhs of (14) is feasible and bounded), or $\varphi(w) = \infty$ with $v_p^*(w), v_d^*(w) \in \mathbb{R}^{m_2}$, so that $M^T(v_p^*(w) + \lambda v_d^*(w)) \leq q$ for all $\lambda \geq 0$, and $w^T v_d^*(w) > 0$ (the lp on the rhs of (14) is feasible and unbounded). The reader will note that when we evaluate $\psi(w)$ of (5) (by solving the lp on the rhs of (5)) the dual optimizer $v^*(w)$ if $w \in \text{pos } M$ or $v_p^*(w)$ and $v_d^*(w)$ if $w \notin \text{pos } M$ usually are available to us.

Let us now define $\pi : \mathbb{R}^{m_2} \rightarrow \overline{\mathbb{R}}^{m_2}$ by

$$\pi(w) = \begin{cases} v^*(w) & \text{if } w \in \text{pos } M \\ \infty \cdot v_d^*(w) & \text{if } w \notin \text{pos } M. \end{cases} \quad (16)$$

In (16) $v^*(w)$ and $v_d^*(w)$ are as described in the preceding paragraph. The representation $\infty \cdot v_d^*(w)$ in (16) is a convenient symbol. As we shall see the only operation that we need to perform with $\pi(w)$ is the formation of scalar products with $u \in \mathbb{R}^{m_2}$. Given $u \in \mathbb{R}^{m_2}$, we assign values to the scalar product $[\infty \cdot v_d^*(w)]^T u = \infty \cdot [v_d^*(w)]^T u$ as follows. If $[v_d^*(w)]^T u > 0$ we assign the value ∞ to the scalar product; otherwise we assign the

FIGURE 3. COMPUTATION OF $Q_v(x)$ AND $\frac{dQ_v(x)}{dv}$

value $-\infty$. In actual computational algorithms we can simply store $v_d^*(w)$ together with an indicator to specify that the actual value is $\infty \cdot v_d^*(w)$ and that scalar products are to be formed according to the conventions described above.

We are now ready to describe the collection $\mathcal{V}$. Let $w^j, j = 1, 2, \dots, \eta$ be a sequence of w values at which we evaluate ψ . We shall assume that $w^j \in \text{pos } M$ for some $j \in \{1, 2, \dots, \eta\}$. Let $v^j = \pi(w^j), j = 1, 2, \dots, \eta$ where π is defined as in (16). The collection we propose is

$$\mathcal{V} = \{v^1, v^2, \dots, v^\eta\}. \quad (17)$$

Relative to $\mathcal{V}$ we define $\psi_{\mathcal{V}} : \mathbb{R}^{m_2} \rightarrow \overline{\mathbb{R}}$ by

$$\psi_{\mathcal{V}}(w) = \sup_j \{\psi_{v^j}(w), j = 1, 2, \dots, \eta\} = \psi_{v^{j^*}}(w) \quad (18.1)$$

where

$$\psi_{v^j}(w) = (v^j)^T w, j = 1, 2, \dots, \eta. \quad (18.2)$$

In (17.1) $j^* = j^*(w)$ is an index j where the supremum is attained. Note that because $w^j \in \text{pos } M$ for some j , by (16) (and the conventions we adopt regarding π of (16)), $\psi_{\mathcal{V}}(w) > -\infty$ for all $w \in \mathbb{R}^{m_2}$.

By (17), (16) and by the duality theorem in linear programming it should now be clear that

$$\psi_{\mathcal{V}}(v^j) = \psi(v^j), j = 1, 2, \dots, \eta \quad (19.1)$$

$$\psi_{\mathcal{V}}(w) = \psi_{v^{j^*}}(w) \leq \psi(w) \quad \text{for } w \in \mathbb{R}^{m_2}. \quad (19.2)$$

We also have

$$\partial \psi_{\mathcal{V}}(w) = \partial \psi_{v^{j^*}}(w) = v^{j^*} \quad \text{for } w \in \mathbb{R}^{m_2}. \quad (19.3)$$

By (2.1) and (2.2), (18.1) induces the following lower bound $\mathcal{Q}_{\mathcal{V}}$ for $\mathcal{Q}$.

$$\mathcal{Q}_{\mathcal{V}}(x) = \sum_{k=1}^K p^k \psi_{\mathcal{V}}(h^k - T^k x) \leq \mathcal{Q}(x) \quad \text{for } x \in X \quad (20.1)$$

with

$$\partial \mathcal{Q}_{\mathcal{V}}(x) = - \sum_{k=1}^K p^k (T^k)^T \partial \psi_{\mathcal{V}}(h^k - T^k x) \quad \text{for } x \in X. \quad (20.2)$$

Suppose now that we have the data $v^j, j = 1, 2, \dots, \eta$ stored. We illustrate the computation of $\mathcal{Q}_{\mathcal{V}}(x)$ and $\partial \mathcal{Q}_{\mathcal{V}}(x)$ in figure 3. Figure 3 may be compared with figures 1

and 2 and the comments we made at the end of section 2.1 on the possibility of computing $Q_D(x)$ and $\partial Q_D(x)$ using a doubly parallel scheme apply for the computations in figure 3. We shall have more to say on the organization of these computations in section 4 of the paper.

3. The design of the collections $\mathcal{D}$ and $\mathcal{V}$

In this section we describe the way we propose to design the collections $\mathcal{D}$ and $\mathcal{V}$. (In the case of $\mathcal{D}$ the word collection is used not just for referring to members of $\mathcal{D}$. It is used to refer to members of $\mathcal{D}$ and other information necessary to compute ψ_D and $\partial\psi_D$.) At the end of the section we shall give a single algorithm that would set up the data necessary to compute $Q_D(x)$, $\partial Q_D(x)$, $Q_V(x)$ and $\partial Q_V(x)$ for $x \in X$. For ease of exposition however, we describe the design of $\mathcal{D}$ and $\mathcal{V}$ separately.

3.1 The design of the collection $\mathcal{D}$

The scheme that we propose to develop $\mathcal{D}$ is motivated by the following properties that we desire in ψ_D . Since ψ_D is to approximate ψ we would desire ψ_D to be exact along many directions. Moreover, we want $\partial\psi_D(x)$ to approximate $\partial\psi(x)$ and consequently we would desire ψ_D to be convex since ψ is convex. Also as mentioned in the introduction the purpose of designing $\mathcal{D}$ is to use Q_D in place of Q in (1) and to solve the resulting mathematical program. It would be easier, in general, to design algorithms to solve convex programs than to solve nonconvex programs. Since convexity of ψ_D would ensure that the mathematical program that results when Q is replaced by Q_D is convex, we desire to design $\mathcal{D}$ so that ψ_D is convex.

The collection that we describe below would ensure that ψ and ψ_D agree along a given set of directions. This fact would become clear during our description of the design of $\mathcal{D}$. The collection $\mathcal{D}$ would also ensure that ψ_D is convex. The proof of this fact is somewhat lengthy and we indicate that in a separate paper. (See Ariyawansa, Sorensen and Wets (1987).)

Suppose then that we are given a set of vectors u^j ($u^j \neq 0$), $j = 1, 2, \dots, J$ along which we need ψ_D to be exact. In section 3.3 we indicate a possible choice for such a set of vectors. We propose to build the collection $\mathcal{D}$ by an updating scheme as follows. We begin with a trivial collection $\mathcal{D}_0 = \{D_0^1\}$, $D_0^1 \in \mathbb{R}^{m_2 \times m_2}$, D_0^1 nonsingular. Here the subscript 0 indicates that this is the initial collection (i.e. we have done zero updates) and the superscript 1 indicates that D_0^1 is the first (and of course the only) member in the

initial collection $\mathcal{D}_0$. A possible choice is $D_0^1 = B$, where B is a member of the collection (15). Another choice is $D_0^1 = I \in \mathbb{R}^{m_2 \times m_2}$. Let

$$r_0^1 = \sum_{j=1}^{m_2} \psi((D_0^1)_{\cdot j}) e_j, \quad s_0^1 = \sum_{j=1}^{m_2} \psi(-(D_0^1)_{\cdot j}) e_j, \quad (21.1)$$

$$U_0^1 = \begin{pmatrix} (r_0^1)^T & (s_0^1)^T \\ D_0^1 & -D_0^1 \end{pmatrix} \quad (21.2)$$

and

$$\psi_{\mathcal{D}_0}(w) = \psi_{D_0^1}(w) = \inf \left\{ \alpha : \begin{pmatrix} \alpha \\ w \end{pmatrix} \in \text{pos } U_0^1 \right\}. \quad (21.3)$$

Note that

$$\psi_{\mathcal{D}_0}(\pm \lambda (D_0^1)_{\cdot j}) = \psi(\pm \lambda (D_0^1)_{\cdot j}) \quad \forall \lambda > 0, j = 1, 2, \dots, m_2,$$

and that

$$\psi(w) \leq \psi_{\mathcal{D}_0}(w), \quad w \in \mathbb{R}^{m_2}.$$

Now if we have $\psi(u^j) = \psi_{\mathcal{D}_0}(u^j)$ for $j = 1, 2, \dots, J$ then the initial collection does provide an approximant that is exact along u^j , $j = 1, 2, \dots, J$ and we terminate. Otherwise let

$$j^1 = \min \{ j : \psi(u^j) < \psi_{\mathcal{D}_0}(u^j), j = 1, 2, \dots, J \}. \quad (22)$$

We now wish to update the collection $\mathcal{D}_0$ to $\mathcal{D}_1$ so that $\psi_{\mathcal{D}_1}$ is exact along u^{j^1} while retaining exactness in all directions along which $\psi_{\mathcal{D}_0}$ is exact. Note that this can be achieved if we simply let $\mathcal{D}_1$ consist of D_0^1 and a nonsingular matrix obtained by replacing a column of D_0^1 by u^{j^1} . Indeed, because of the way we define $\psi_{\mathcal{D}_1}$, such a collection would make $\psi_{\mathcal{D}_1}$ exact along $-u^{j^1}$ as well. Such a $\psi_{\mathcal{D}_1}$ need not be convex. If however, we let $\mathcal{D}_1$ consist of D_0^1 and *all* nonsingular matrices obtained by replacing a single column of D_0^1 by u^{j^1} , then $\psi_{\mathcal{D}_1}$ would be convex as shown in Ariyawansa, Sorensen and Wets (1987).

It would now appear that the natural way to represent $\mathcal{D}_1$ would be

$$\mathcal{D}_1 = \{D_1^1, D_1^2, \dots, D_1^{\nu_1}\} \quad (23)$$

where $D_1^1 = D_0^1$, and D_1^j , $j = 2, 3, \dots, \nu_1$ are the nonsingular matrices obtained by replacing a single column of D_0^1 by u^{j^1} . However, when we complete our description of the collection $\mathcal{D}_l$ (i.e. after l updates of $\mathcal{D}_0$), $l \geq 1$ it will become clear that the following is a more natural way to describe $\mathcal{D}_l$. In fact it turns out that the notation that follows from this way of viewing the collection $\mathcal{D}_l$ is helpful in streamlining the computation of $\psi_{\mathcal{D}_l}$ and $\partial \psi_{\mathcal{D}_l}$, $l \geq 1$.

We make the following remarks for $l \geq 1$. The collection $\mathcal{D}_l$ consists of l *groups* of matrices. (The word group is used in a nonrigorous literal sense to indicate that the

matrices in the group are *related*.) Each group has a *base matrix* and the other *nonbase* members in the group are obtained by simply replacing an appropriate column of the base matrix by a vector that is *given* for the group. When D_l is updated to D_{l+1} the following occur. A new group $(l+1)$ is formed. Certain quantities associated with *one* of the groups $1, 2, \dots, l$ are changed.

With the structure described in the previous paragraph in mind we adopt the following notational conventions for describing D_l and quantities necessary to compute ψ_{D_l} and $\partial\psi_{D_l}$ for $l \geq 1$. In symbols we use to describe quantities that refer to D_l we avoid *explicitly* indicating that we have performed l updates of D_0 . Since the number of updates we perform to obtain D_l is the same as the number of groups we have in the collection, information on the number of updates would be readily available. This convention would considerably simplify our notation. If a quantity is same for all the members in a group numbered g we describe that quantity by a symbol with a *single* superscript g . If a quantity is different for different members in a group g then we denote the quantity associated with member number m by a symbol with *two* superscripts — gm (in that order). The base member in a group will always be given the member number $m = 0$.

D_1 has just *one* group of matrices in the sense described above. Certain parts of the notation we now use to describe D_1 may therefore appear to be redundant when one considers D_1 in isolation. We request the reader to be patient and mention that the notation that we use to describe D_1 is consistent with the notational requirements of D_l for general l ($l \geq 1$).

We now describe the collection D_1 in precise terms. Let $w^1 = u^{j^1}$. In (22) we compute ψ_{D_0} using (11) and lemma 2.1 so that

$$\psi_{D_0}(w^1) = \psi_{D_0^1}(w^1) = (r_0^1)^T t_+^1 - (s_0^1)^T t_-^1 \quad \text{where } D_0^1 t^1 = w^1.$$

Let N^1 be the number of nonzero components of t^1 . Let $(P^1)^T \in \mathbb{R}^{m_2 \times m_2}$ be any permutation matrix that permutes the components of t^1 so that the first N^1 components are nonzero and the remaining $m_2 - N^1$ components are zero. A particular P^1 may be described as follows.

$$P^1 = [e_{k_1}, e_{k_2}, \dots, e_{k_{m_2}}] \quad (24.1)$$

where $k \in \mathbb{N}^{m_2}$ is defined by

$$k_j = \begin{cases} \text{index of } j\text{-th nonzero component of } t^1, \text{ for } j = 1, 2, \dots, N^1 \\ \text{index of } (j - N^1)\text{-th zero component of } t^1, \text{ for } j = N^1 + 1, N^1 + 2, \dots, m_2. \end{cases} \quad (24.2)$$

We now let $t^1 := (P^1)^T t^1$. The collection $\mathcal{D}_1$ is written as

$$\begin{aligned} \mathcal{D}_1 &= \{D^{10}, D^{11}, \dots, D^{1\mu^1}\} \quad \text{where} \\ \mu^1 &= N^1 \\ D^{10} &= D_0^1 P^1 \\ D^{1m} &= D^{10} + (w^1 - D_0^1 e_m) e_m^T, \quad m = 1, 2, \dots, \mu^1. \end{aligned} \tag{25}$$

We note that we are abusing the notation somewhat by using the same symbol to denote *different representations* for the *same* collection in (23) and (25). We believe however that this would not lead to confusion.

We now define the following in preparation to write down an expression for $\psi_{\mathcal{D}_1}(w)$.

$$\begin{aligned} \rho^1 &= \psi(w^1), \quad \sigma^1 = \psi(-w^1) \\ r^{10} &= (P^1)^T r_0^1, \quad s^{10} = (P^1)^T s_0^1 \\ r^{1m} &= r^{10} + (\rho^1 - e_m^T r^{10}) e_m, \quad m = 1, 2, \dots, \mu^1 \\ s^{1m} &= s^{10} + (\sigma^1 - e_m^T s^{10}) e_m, \quad m = 1, 2, \dots, \mu^1 \\ U^{1m} &= \begin{pmatrix} (r^{1m})^T & (s^{1m})^T \\ D^{1m} & -D^{1m} \end{pmatrix}, \quad m = 0, 1, \dots, \mu^1 \end{aligned} \tag{26}$$

$\psi_{\mathcal{D}_1}(w)$ can then be expressed as

$$\psi_{\mathcal{D}_1}(w) = \inf_m \{ \psi_{D^{1m}}(w), m = 0, 1, \dots, \mu^1 \} \tag{27.1}$$

where

$$\psi_{D^{1m}}(w) = \inf \left\{ \alpha : \begin{pmatrix} \alpha \\ w \end{pmatrix} \in \text{pos } U^{1m} \right\}, \quad m = 0, 1, \dots, \mu^1. \tag{27.2}$$

We note that $\psi_{\mathcal{D}_1}$ is exact along any direction along which $\psi_{\mathcal{D}_0}$ was exact,

$$\psi_{\mathcal{D}_1}(\pm \lambda w^1) = \psi(\pm \lambda w^1), \quad \forall \lambda > 0$$

and that

$$\psi(w) \leq \psi_{\mathcal{D}_1}(w), \quad w \in \mathbb{R}^{m_2}.$$

Note that by (25), (26), (27) and lemma 2.1, we can compute $\psi_{\mathcal{D}_1}(w)$ and $\partial \psi_{\mathcal{D}_1}(w)$ for $w \in \mathbb{R}^{m_2}$ if we just have D^{10} , μ^1 , w^1 , r^{10} , s^{10} , ρ^1 and σ^1 . We shall later see that when we need to compute $\psi_{\mathcal{D}_1}(w)$ and $\partial \psi_{\mathcal{D}_1}(w)$ for many w we can speed up the computations by having $t^1 (= (D^{10})^{-1} w^1)$ at hand. We therefore treat D^{10} , μ^1 , t^1 , r^{10} , s^{10} , ρ^1 , and σ^1 as the data we store to define $\mathcal{D}_1$ and to compute $\psi_{\mathcal{D}_1}(w)$ and $\partial \psi_{\mathcal{D}_1}(w)$.

If we now have $\psi(u^j) = \psi_{\mathcal{D}_1}(u^j)$ for $j = j^1 + 1, j^1 + 2, \dots, J$ then the collection $\mathcal{D}_1$ provides an approximant that is exact along u^j , $j = 1, 2, \dots, J$, and we terminate.

Otherwise let $j^2 = \min\{j : \psi(u^j) < \psi_{D_1}(u^j), j = j^1 + 1, j^1 + 2, \dots, J\}$, and $w^2 = u^{j^2}$. We now wish to update the collection D_1 to D_2 so that ψ_{D_2} is exact along w^2 while being exact along all directions along which ψ_{D_1} is exact. The specific way we update D_1 to D_2 is as follows. Let

$$\psi_{D_1}(w^2) = \inf_m \{\psi_{D^{1m}}(w^2), m = 0, 1, \dots, \mu^1\} = \psi_{D^{1m^1}}(w^2) \quad (28)$$

where m^1 is an index m where the infimum is achieved. $\psi_{D^{1m}}, m = 0, 1, \dots, \mu^1$ in (28) is of course computed using (27) and lemma 2.1, so that

$$\psi_{D_1}(w^2) = \psi_{D^{1m^1}}(w^2) = (r^{1m^1})^T t_+^2 - (s^{1m^1})^T t_-^2 \quad (29)$$

where $D^{1m^1} t^2 = w^2$.

Let N^2 be the number of nonzero components of t^2 , and define the permutation matrix P^2 by (24.1) and (24.2) using t^2 , N^2 and P^2 in place of t^1 , N^1 and P^1 respectively.

We now create a new group—group 2—as follows. The base member and quantities needed to compute $\psi_{D^{2m}}(w)$ and $\partial\psi_{D^{2m}}(w)$, $m = 1, 2, \dots, N^2$ are

$$\begin{aligned} D^{20} &:= D^{1m^1} P^2 \\ \mu^2 &:= N^2 \\ r^{20} &:= (P^2)^T r^{1m^1} \\ s^{20} &:= (P^2)^T s^{1m^1} \\ \rho^2 &:= \psi(w^2) \\ \sigma^2 &:= \psi(-w^2) \\ t^2 &:= (P^2)^T t^2. \end{aligned} \quad (30)$$

If m^1 , the index that yielded the infimum implied by (29) (via (27)) is zero, then we do not change any quantity associated with group 1 we have already created. If however, $m^1 \neq 0$ then we update certain quantities associated with group 1 as follows. We will elaborate on this further after we define D_2 and ψ_{D_2} .

$$\begin{aligned} P &:= I - (e_{\mu^1} - e_{m^1})(e_{\mu^1} - e_{m^1})^T, \quad m^1 \neq 0 \\ D^{10} &:= D^{10} P \\ r^{10} &:= P^T r^{10} \\ s^{10} &:= P^T s^{10} \\ t^1 &:= P^T t^1 \\ \mu^1 &:= \mu^1 - 1 \end{aligned} \quad (31)$$

The collection $\mathcal{D}_2$ we propose is

$$\mathcal{D}_2 = \{D^{gm}; m = 0, 1, \dots, \mu^g; g = 1, 2\} \quad (32.1)$$

where

$$D^{gm} = D^{g0} + [w^g - D^{g0}e_m]e_m^T; \quad m = 1, 2, \dots, \mu^g; \text{ if } \mu^g > 0 \quad (32.2)$$

and $w^g = D^{g0}t^g$ for $g = 1, 2$. $\psi_{\mathcal{D}_2}$ can now be written as

$$\psi_{\mathcal{D}_2}(w) = \inf_{m,g} \{\psi_{D^{gm}}(w); m = 0, 1, \dots, \mu^g; g = 1, 2\} \quad (33.1)$$

where

$$\begin{aligned} \psi_{D^{gm}}(w) &= \inf \left\{ \alpha : \begin{pmatrix} \alpha \\ w \end{pmatrix} \in \text{pos } U^{gm} \right\}; \quad m = 0, 1, \dots, \mu^g; g = 1, 2 \\ U^{gm} &= \begin{pmatrix} (r^{gm})^T & (s^{gm})^T \\ D^{gm} & -D^{gm} \end{pmatrix}; \quad m = 0, 1, \dots, \mu^g; g = 1, 2 \\ r^{gm} &= r^{g0} + [\rho^g - e_m^T r^{g0}]e_m; \quad m = 1, 2, \dots, \mu^g; \text{ if } \mu^g > 0; g = 1, 2 \\ s^{gm} &= s^{g0} + [\sigma^g - e_m^T s^{g0}]e_m; \quad m = 1, 2, \dots, \mu^g; \text{ if } \mu^g > 0; g = 1, 2. \end{aligned} \quad (33.2)$$

At this point it is appropriate to elaborate further on the way we updated $\mathcal{D}_1$ to $\mathcal{D}_2$. We decided to update $\mathcal{D}_1$ to $\mathcal{D}_2$ because $\psi(w^2) < \psi_{\mathcal{D}_1}(w^2)$ and consequently to ensure $\psi_{\mathcal{D}_2}$ is exact along w^2 while being exact along all directions along which $\psi_{\mathcal{D}_1}$ was exact. Note that we can achieve that if we let $\mathcal{D}_2$ be $\mathcal{D}_1$ and any matrix D with $\psi(w^2) = \psi_D(w^2)$. Such a $\mathcal{D}_2$ however need not yield $\psi_{\mathcal{D}_2}$ that is convex. $\mathcal{D}_2$ we define via (30), (31) and (32) adds more than one such matrix D to $\mathcal{D}_1$. As shown in Ariyawansa, Sorensen and Wets (1987) this $\mathcal{D}_2$ ensures that $\psi_{\mathcal{D}_2}$ of (33) is convex in addition to being exact along w^2 and along all directions along which $\psi_{\mathcal{D}_1}$ was exact.

Note that updating $\mathcal{D}_1$ to $\mathcal{D}_2$ consisted of the following. We first find the index m^1 that yields the infimum in $\psi_{\mathcal{D}_1}(w^2)$. We then wish to add all nonsingular matrices that may be created by replacing a column of D^{1m^1} by w^2 to $\mathcal{D}_1$ to obtain $\mathcal{D}_2$. The permutation matrix P^2 is used simply to make sure that these μ^2 matrices may be obtained by replacing the first μ^2 columns of the permuted matrix. Note that we can now remove the matrix D^{1m^1} from group 1 since the permuted matrix is in the collection as the base matrix of group 2. However, since we define a group via a base matrix and other associated quantities rather than *explicitly* defining all the matrices in a group, we cannot remove D^{1m^1} from group 1 if $m^1 = 0$. We therefore change group 1 only if $m^1 \neq 0$.

We stress that the collection $\mathcal{D}_2$ and $\psi_{\mathcal{D}_2}$ are completely defined by the data D^{g0} , μ^g , t^g , r^{g0} , s^{g0} , ρ^g and σ^g , $g = 1, 2$. As we shall see later having t^g readily available would make the computation of $\psi_{\mathcal{D}_2}(w)$ and $\partial\psi_{\mathcal{D}_2}(w)$ for many w faster. For this reason we prefer to store t^g rather than w^g , $g = 1, 2$.

We shall now describe the general step of updating D_l to D_{l+1} ($l \geq 1$). The collection D_l and ψ_{D_l} would be described by D^{g0} , μ^g , t^g , r^{g0} , s^{g0} , ρ^g and σ^g , $g = 1, 2, \dots, l$, and suppose therefore that we have these data already set up. Then

$$D_l = \{D^{gm}; m = 0, 1, \dots, \mu^g; g = 1, 2, \dots, l\} \quad (34.1)$$

where

$$D^{gm} = D^{g0} + [w^g - D^{g0}e_m]e_m^T; \quad m = 1, 2, \dots, \mu^g; \text{ if } \mu^g > 0 \quad (34.2)$$

and $w^g = D^{g0}t^g$ for $g = 1, 2, \dots, l$. ψ_{D_l} is defined by

$$\psi_{D_l}(w) = \inf_{m,g} \{\psi_{D^{gm}}(w); m = 0, 1, \dots, \mu^g; g = 1, 2, \dots, l\} \quad (35.1)$$

where

$$\begin{aligned} \psi_{D^{gm}}(w) &= \inf \left\{ \alpha : \begin{pmatrix} \alpha \\ w \end{pmatrix} \in \text{pos } U^{gm} \right\}; \quad m = 0, 1, \dots, \mu^g; g = 1, 2, \dots, l \\ U^{gm} &= \begin{pmatrix} (r^{gm})^T & (s^{gm})^T \\ D^{gm} & -D^{gm} \end{pmatrix}; \quad m = 0, 1, \dots, \mu^g; g = 1, 2, \dots, l \\ r^{gm} &= r^{g0} + [\rho^g - e_m^T r^{g0}]e_m; \quad m = 1, 2, \dots, \mu^g; \text{ if } \mu^g > 0; g = 1, 2, \dots, l \\ s^{gm} &= s^{g0} + [\sigma^g - e_m^T s^{g0}]e_m; \quad m = 1, 2, \dots, \mu^g; \text{ if } \mu^g > 0; g = 1, 2, \dots, l. \end{aligned} \quad (35.2)$$

Suppose without loss of generality that $\psi(u^j) = \psi_{D_l}(u^j)$ for $j = 1, 2, \dots, j^{l+1} - 1$ and $\psi(u^{j^{l+1}}) < \psi_{D_l}(u^{j^{l+1}})$ for some $j^{l+1} \leq J$. Let $w^{l+1} = u^{j^{l+1}}$. Of course $\psi_{D_l}(w^{l+1})$ would be computed using (35.1), (35.2) and lemma 2.1 so that

$$\psi_{D_l}(w^{l+1}) = \psi_{D^{g^l m^l}}(w^{l+1}) = (r^{g^l m^l})^T t_+^{l+1} - (s^{g^l m^l})^T t_-^{l+1} \quad (36)$$

where $D^{g^l m^l} t^{l+1} = w^{l+1}$. In (36) m^l and g^l are indices m and g respectively in (35.1) that yield the infimum when $w = w^{l+1}$.

Let N^{l+1} be the number of nonzero components of t^{l+1} and define the permutation matrix P^{l+1} by (24.1) and (24.2) using t^{l+1} , N^{l+1} and P^{l+1} in place of t^1 , N^1 and P^1 respectively.

A new group—group $l+1$ —is now created as follows.

$$\begin{aligned} D^{(l+1)0} &:= D^{g^l m^l} P^{l+1} \\ \mu^{l+1} &:= N^{l+1} \\ r^{(l+1)0} &:= (P^{l+1})^T r^{g^l m^l} \\ s^{(l+1)0} &:= (P^{l+1})^T s^{g^l m^l} \\ \rho^{l+1} &:= \psi(w^{l+1}) \\ \sigma^{l+1} &:= \psi(-w^{l+1}) \\ t^{l+1} &:= (P^{l+1})^T t^{l+1}. \end{aligned} \quad (37)$$

DATA ACTUALLY STORED TO SPECIFY $\mathcal{D}_l$.

FIGURE 4. THE COLLECTION $\mathcal{D}_l (l \geq 1)$

We update the quantities associated with groups $g = 1, 2, \dots, l$ as follows.

$$\begin{aligned} D^{g^0}, \mu^g, t^g, r^{g^0}, s^{g^0}, \rho^g \text{ and } \sigma^g & \text{ are not changed} \\ & \text{for } g = 1, 2, \dots, l; \text{ if } m^l = 0 \\ & \text{for } g = 1, 2, \dots, l; g \neq g^l; \text{ if } m^l > 0 \end{aligned} \quad (38.1)$$

If $m^l > 0$ quantities associated with group g^l are changed as follows.

$$\begin{aligned} P &:= I - (e_{\mu^{g^l}} - e_{m^l})(e_{\mu^{g^l}} - e_{m^l})^T, \quad m^l > 0 \\ D^{g^l 0} &:= D^{g^l 0} P \\ r^{g^l 0} &:= P^T r^{g^l 0} \\ s^{g^l 0} &:= P^T s^{g^l 0} \\ t^{g^l} &:= P^T t^{g^l} \\ \mu^{g^l} &:= \mu^{g^l} - 1 \end{aligned} \quad (38.2)$$

$\mathcal{D}_{l+1}$ and $\psi_{\mathcal{D}_{l+1}}$ are completely determined by $D^{g^0}, \mu^g, t^g, r^{g^0}, s^{g^0}, \rho^g$ and σ^g , $g = 1, 2, \dots, l+1$ via (34) and (35) with l replaced by $l+1$.

We shall continue updating the collection in the manner described above until we have $\mathcal{D}_L$ with $\psi_{\mathcal{D}_L}(u^j) = \psi(u^j)$ for $j = 1, 2, \dots, J$ for some L . It should be clear that

$$\psi_{\mathcal{D}_L}(\pm \lambda (D_0^1)_{\cdot j}) = \psi(\pm \lambda (D_0^1)_{\cdot j}), \quad \forall \lambda > 0, j = 1, 2, \dots, m_2,$$

$$\psi_{\mathcal{D}_L}(\pm \lambda u^j) = \psi(\pm \lambda u^j), \quad \forall \lambda > 0, j = 1, 2, \dots, J$$

and that

$$\psi(w) \leq \psi_{\mathcal{D}_L}(w), \quad \text{for } w \in \mathbb{R}^{m_2}.$$

Ariyawansa, Sorensen and Wets (1987) show that $\psi_{\mathcal{D}_l}$, $l = 0, 1, \dots, L$ are also convex.

Our description of the way we design the collection $\mathcal{D}$ is now complete. For $l \geq 1$, the collection $\mathcal{D}_l$ may be illustrated as in figure 4. Figure 4 illustrates that the collection $\mathcal{D}_l$ ($l \geq 1$) consists of l groups of members, and that the collection $\mathcal{D}_l$ and the function $\psi_{\mathcal{D}_l}$ are completely specified by storing the data $D^{g^0}, \mu^g, t^g, r^{g^0}, s^{g^0}, \rho^g$ and σ^g for groups $g = 1, 2, \dots, l$.

We now state the following proposition on the computation of $\psi_{\mathcal{D}_l}(w)$ and $\partial \psi_{\mathcal{D}_l}(w)$ for $\mathcal{D}_l$ ($l \geq 1$) developed as described above.

Proposition 3.1: Let $D^{g^0} \in \mathbb{R}^{m_2 \times m_2}$ (D^{g^0} nonsingular), $t^g \in \mathbb{R}^{m_2}$ and $\mu^g \in \mathbb{N}$ ($0 \leq \mu^g \leq m_2$), $g = 1, 2, \dots, l$ ($l \geq 1$) be given. Suppose that if $\mu^g > 0$ then $t_i^g \neq 0$ for

$i = 1, 2, \dots, \mu^g$, $g = 1, 2, \dots, l$. Let $w^g = D^{g0}t^g$ and define $r^{g0} \in \overline{\mathbb{R}}^{m^2}$, $s^{g0} \in \overline{\mathbb{R}}^{m^2}$, $\rho^g \in \overline{\mathbb{R}}$ and $\sigma^g \in \overline{\mathbb{R}}$ respectively by $r^{g0} = \sum_{j=1}^{m^2} \psi(D^{g0}e_j)e_j$, $s^{g0} = \sum_{j=1}^{m^2} \psi(-D^{g0}e_j)e_j$, $\rho^g = \psi(w^g)$ and $\sigma^g = \psi(-w^g)$ for $g = 1, 2, \dots, l$ where $\psi: \mathbb{R}^{m^2} \rightarrow \overline{\mathbb{R}}$ is as defined in (5). Suppose that the collection of matrices D_l and the function $\psi_{D_l}: \mathbb{R}^{m^2} \rightarrow \overline{\mathbb{R}}$ are defined by (34) and (35) respectively. Let assumption (A5) be satisfied. Then for $w \in \mathbb{R}^{m^2}$,

$$\psi_{D_l}(w) = \inf_{m,g} \{ \psi_{D^{gm}}(w) = (r^{gm})^T t_+^{gm} - (s^{gm})^T t_-^{gm} : m = 0, 1, \dots, \mu^g; g = 1, 2, \dots, l \} \quad (39.1)$$

where if $\mu^g > 0$, for $m = 1, 2, \dots, \mu^g$, $r^{gm} = r^{g0} + [\rho^g - e_m^T r^{g0}]e_m$, $s^{gm} = s^{g0} + [\sigma^g - e_m^T s^{g0}]e_m$ and

$$t^{gm} = t^{g0} - (e_m^T t^{g0} / e_m^T t^{g*})(t^{g*} - e_m) \quad (39.2)$$

for $g = 1, 2, \dots, l$. In (39.2) t^{g0} is the solution to the system $D^{g0}t^{g0} = w$, $g = 1, 2, \dots, l$.

Suppose now that $m = m^*$ and $g = g^*$ ($0 \leq m^* \leq \mu^{g^*}$, $1 \leq g^* \leq l$) are indices that yield the infimum in (39.1). Then

$$\partial \psi_{D_l}(w) = \begin{cases} \left\{ \begin{array}{l} u : u = u^{g^*0}, (D^{g^*0})^T u^{g^*0} = v, \\ v = (Y_+ - Z)r^{g^*0} + Y_- s^{g^*0} + Zh, \\ h \in [-s^{g^*0}, r^{g^*0}] \end{array} \right\} & \text{if } m^* = 0 \\ \left\{ \begin{array}{l} u : u = u^{g^*0} - [((w^{g^*})^T u^{g^*0} - e_{m^*}^T v) / e_{m^*}^T t^{g^*}] e^{g^*m^*}, \\ (D^{g^*0})^T u^{g^*0} = v, (D^{g^*0})^T e^{g^*m^*} = e_{m^*}, \\ v = (Y_+ - Z)r^{g^*m^*} + Y_- s^{g^*m^*} + Zh, \\ h \in [-s^{g^*m^*}, r^{g^*m^*}] \end{array} \right\} & \text{if } m^* \neq 0 \end{cases} \quad (40)$$

where $Y = \text{diag}(\text{sgn}(t^{g^*m^*}))$, $Z = \text{diag}(t_z^{g^*m^*})$ and if $\mu^{g^*} > 0$, $t^{g^*m^*} = t^{g^*0} - (e_{m^*}^T t^{g^*0} / e_{m^*}^T t^{g^*})(t^{g^*} - e_{m^*})$, $D^{g^*0}t^{g^*0} = w$, $r^{g^*m^*} = r^{g^*0} + [\rho^{g^*} - e_{m^*}^T r^{g^*0}]e_{m^*}$, and $s^{g^*m^*} = s^{g^*0} + [\sigma^{g^*} - e_{m^*}^T s^{g^*0}]e_{m^*}$.

Proof: We first show the validity of (39.1) and (39.2). We now compute $\psi_{D^{gm}}(w)$ in (35.1) $m = 0, 1, \dots, \mu^g$; $g = 1, 2, \dots, l$ using lemma 2.1. Note that we have

$$\psi_{D^{g0}}(w) = (r^{g0})^T t_+^{g0} - (s^{g0})^T t_-^{g0} \quad g = 1, 2, \dots, l \quad (41.1)$$

directly from lemma 2.1. Now if $\mu^g > 0$, for $m = 1, 2, \dots, \mu^g$ again by lemma 2.1 we have

$$\psi_{D^{gm}}(w) = (r^{gm})^T t_+^{gm} - (s^{gm})^T t_-^{gm} \quad (41.2)$$

where

$$[D^{g0} + (w^g - D^{g0}e_m)e_m^T]t^{gm} = w \quad (41.3)$$

for $g = 1, 2, \dots, l$. By a straightforward application of the Sherman-Morrison-Woodbury formula (see for instance Ortega and Rheinboldt (1970), page 50) to (41.3) we get (39.2). By (35.1), (35.2), (41.1) and (41.2), (39.1) follows.

We now show the validity of (40). By (34), (35), (12.2) and lemma 2.1 we have

$$\partial\psi_{D_l}(w) = \{(D^{g^*m^*})^{-T}v : v = (Y_+ - Z)r^{g^*m^*} + Y_-s^{g^*m^*} + Zh, h \in [-s^{g^*m^*}, r^{g^*m^*}]\} \quad (42.1)$$

where

$$D^{g^*m^*} = D^{g^*0} + (w^{g^*} - D^{g^*0}e_{m^*})e_{m^*}^T, \quad \text{if } m^* > 0 \quad (42.2)$$

and $t^{g^*m^*}$, Y , Z , $r^{g^*m^*}$ and $s^{g^*m^*}$ are as indicated in the statement of the proposition. If $m^* = 0$ (40) follows directly from (42.1). If $m^* > 0$ let $(D^{g^*m^*})^T u = v$ and use Sherman-Morrison-Woodbury formula to get

$$u = u^{g^*0} - [(w^{g^*})^T u^{g^*0} - e_{m^*}^T v] / e_{m^*}^T t^{g^*} e^{g^*m^*} \quad (42.3)$$

where $(D^{g^*0})^T u^{g^*0} = v$ and $(D^{g^*0})^T e^{g^*m^*} = e_{m^*}$.

(40) now follows from (42.1), (42.2) and (42.3). ■

The reader will now note that when the collection $\mathcal{D}_l$ ($l \geq 1$) is developed starting from nonsingular D_0^1 and u^j , $j = 1, 2, \dots, J$ as described above, all the hypotheses in proposition 3.1 are satisfied. Proposition 3.1 can therefore be used to compute $\psi_{D_l}(w)$ and $\partial\psi_{D_l}(w)$ for $l \geq 1$.

Note that proposition 3.1 indicates that in order to compute $\psi_{D_l}(w)$ and $\partial\psi_{D_l}(w)$ ($l \geq 1$) we need to solve the following systems of equations. We need to solve the l systems, $D^{g^0}t^{g^0} = w$ for t^{g^0} , $g = 1, 2, \dots, l$ first. We can then compute $\psi_{D_l}(w)$ and obtain m^* and g^* using (39.1) and (39.2). It is now necessary to solve the system $(D^{g^*0})^T u^{g^*0} = v$ for u^{g^*0} and if $m^* \neq 0$ the system $(D^{g^*0})^T e^{g^*m^*} = e_{m^*}$ for $e^{g^*m^*}$ to compute $\partial\psi_{D_l}(w)$ using (40). Therefore, the computation of $\psi_{D_l}(w)$ and $\partial\psi_{D_l}(w)$ at a given w (and given $h \in [-s^{g^*m^*}, r^{g^*m^*}]$) would need the solution of $l + 1$ systems if $m^* = 0$ and $l + 2$ systems if $m^* \neq 0$.

We mention that $\mathcal{D}_l$ ($l \geq 1$) has $l + \sum_{g=1}^l \mu^g$ members and if these members were arbitrary, (11), (12) and lemma 2.1 indicate that we would have to solve $1 + l + \sum_{g=1}^l \mu^g$ systems to obtain $\psi_{D_l}(w)$ and $\partial\psi_{D_l}(w)$ at a given w . The structure in $\mathcal{D}_l$ we described in this section thus leads to a considerable saving in the computations necessary to obtain $\psi_{D_l}(w)$ and $\partial\psi_{D_l}(w)$ ($l \geq 1$).

3.2 The design of the collection $\mathcal{V}$

In section 3.1 we discussed in detail the development of the collection $\mathcal{D}_l$ and the corresponding function ψ_{D_l} which majorizes ψ , $l = 0, 1, \dots$. Corresponding to each $\mathcal{D}_l$ we

now define a collection $\mathcal{V}_l$ of vectors in $\overline{\mathbb{R}}^{m_2}$ and a function $\psi_{\mathcal{V}_l}$ relative to $\mathcal{V}_l$ that bound ψ from below. Once the collection $\mathcal{V}_l$ is defined $\psi_{\mathcal{V}_l}$ would be defined relative to $\mathcal{V}_l$ as in section 2.2.

The representation we use for $\mathcal{V}_l$ is

$$\mathcal{V}_l = \{v^j, j = 1, 2, \dots, \eta_l\}. \quad (43)$$

The initial collection $\mathcal{V}_0$ is related to $\mathcal{D}_0$ of section 3.1 and is defined as

$$\mathcal{V}_0 = \{v^j, j = 1, 2, \dots, \eta_0\} \quad (44.1)$$

where

$$\eta_0 = 2m_2 \quad \text{and} \quad (44.2)$$

$$v^{2j-1} = \pi((D_0^1)_{\cdot j}), \quad v^{2j} = \pi((-D_0^1)_{\cdot j}), \quad j = 1, 2, \dots, m_2. \quad (44.3)$$

In (44.3), $\pi : \mathbb{R}^{m_2} \rightarrow \overline{\mathbb{R}}^{m_2}$ is as defined in (16).

For $l = 1, 2, \dots, L$ (where L is the number of updates we performed on $\mathcal{D}_0$ in section 3.1) we define $\mathcal{V}_l$ as follows.

$$\mathcal{V}_l = \{v^j, j = 1, 2, \dots, \eta_l\} \quad (45.1)$$

where

$$\eta_l = 2(m_2 + l) \quad \text{and} \quad (45.2)$$

$$v^{2(m_2+g)-1} = \pi(w^g), \quad v^{2(m_2+g)} = \pi(-w^g), \quad g = 1, 2, \dots, l. \quad (45.3)$$

We recall that while building the collection $\mathcal{D}_0$ we evaluate $\psi(w)$ of (5) at $w = \pm(D_0^1)_{\cdot j}$, $j = 1, 2, \dots, m_2$. The vectors v^j , $j = 1, 2, \dots, 2m_2$ in the collection $\mathcal{V}_0$ are simply related to the lp in (14) at $w = \pm(D_0^1)_{\cdot j}$, $j = 1, 2, \dots, m_2$ via the function π of (16). When we update $\mathcal{D}_{l-1}$ to $\mathcal{D}_l$ for $l = 1, 2, \dots, L$ we evaluate $\psi(w)$ for $w = \pm w^l$ related to the new group l we create. The vectors $v^{2(m_2+l)-1}$ and $v^{2(m_2+l)}$ we add to $\mathcal{V}_{l-1}$ to get $\mathcal{V}_l$ are simply related to the lp in (14) at $w = \pm w^l$ via π of (16).

The function $\psi_{\mathcal{V}_l} : \mathbb{R}^{m_2} \rightarrow \overline{\mathbb{R}}$ for $l = 0, 1, \dots, L$ is now defined as in section 2.2 so that

$$\psi_{\mathcal{V}_l}(w) = \sup_j \{\psi_{v^j}(w), j = 1, 2, \dots, \eta_l\} \quad (46.1)$$

where

$$\psi_{v^j}(w) = (v^j)^T w, \quad j = 1, 2, \dots, \eta_l. \quad (46.2)$$

We shall now assume that at least one of the vectors $\pm(D_0^1)_{\cdot j}$, $j = 1, 2, \dots, m_2$ or one of the vectors $\pm w^j$, $j = 1, 2, \dots, J$ is in the set $\text{pos } M$. (We shall see that this indeed would

be the case for the choice for the vectors u^j , $j = 1, 2, \dots, J$ we indicate in section 3.3.) It then follows that $\psi_{\mathcal{V}_L}(w) > -\infty$ for all $w \in \mathbb{R}^{m_2}$. Therefore, if j^* is the index that achieves the supremum in (46.1) (when $l = L$),

$$\partial\psi_{\mathcal{V}_L}(w) = v^{j^*}. \quad (46.3)$$

It should now be clear that

$$\psi_{\mathcal{V}_l}(w) = \psi(w) = \psi_{\mathcal{D}_l}(w)$$

for

$$w = \pm\lambda(D_0^1)_{.j}, \quad \forall \lambda > 0, j = 1, 2, \dots, m_2$$

and for

$$w = \pm\lambda w^g, \quad \forall \lambda > 0, g = 1, 2, \dots, l$$

and that

$$\psi_{\mathcal{V}_l}(w) \leq \psi(w) \leq \psi_{\mathcal{D}_l}(w) \quad \text{for } w \in \mathbb{R}^{m_2}.$$

Our description of the way we develop the collection $\mathcal{V}$ is now complete.

3.3 An algorithm that designs the collections $\mathcal{D}$ and $\mathcal{V}$

In this section we give a specific algorithm that develops the collections $\mathcal{D}$ and $\mathcal{V}$ as described in sections 3.1 and 3.2 respectively. Several comments are in order before we do that.

- (1) We shall assume that as input to the algorithm we have $D_0^1 \in \mathbb{R}^{m_2 \times m_2}$, D_0^1 non-singular and a collection of vectors u^j , $j = 1, 2, \dots, J$. The algorithm will perform the necessary number of updates L , to obtain $\mathcal{D}_L$ and $\mathcal{V}_L$ so that $\psi_{\mathcal{V}_L}$, ψ and $\psi_{\mathcal{D}_L}$ agree along $\pm u^j$, $j = 1, 2, \dots, L$ (and of course along $\pm(D_0^1)_{.j}$, $j = 1, 2, \dots, m_2$). $\psi_{\mathcal{D}_L}$ and $\psi_{\mathcal{V}_L}$ will bound ψ from above and below respectively.
- (2) In our discussion of the collection $\mathcal{D}_l$ in section 3.1 we indicated that the data we need to define $\mathcal{D}_l$ and $\psi_{\mathcal{D}_l}$ for $l \geq 1$ are D^{g^0} , μ^g , t^g , r^{g^0} , s^{g^0} , ρ^g and σ^g , $g = 1, 2, \dots, l$. Recall that by our discussion in section 3.1, when we compute $\psi_{\mathcal{D}_l}(w)$ and $\partial\psi_{\mathcal{D}_l}(w)$, ($l \geq 1$) we need to appeal to lemma 3.1. We would therefore have to solve the systems $D^{g^0}t^{g^0} = w$ (for t^{g^0}), $g = 1, 2, \dots, l$, and the two systems $(D^{g^*0})^T u^{g^*0} = v$ (for u^{g^*0}), and $(D^{g^*0})^T e^{g^*m^*} = e_{m^*}$ (for $e^{g^*m^*}$) for appropriate indices m^* and g^* ($0 \leq m^* \leq \mu^{g^*}$, $1 \leq g^* \leq l$). In order to facilitate the solution of such systems in a stable manner the algorithm will maintain the QR factors Q^{g^0} and R^{g^0} of D^{g^0} .

(see Stewart (1973) for example) for $g = 1, 2, \dots, l$. The data that the algorithm maintains to define D_l and ψ_{D_l} therefore are $Q^{g^0}, R^{g^0}, \mu^g, t^g, r^{g^0}, s^{g^0}, \rho^g$ and σ^g , $g = 1, 2, \dots, l$ ($l \geq 1$).

- (3) In the algorithm we do not give procedures for computing $\psi_{D_l}(w)$. We shall simply say that these computations may be performed using lemma 2.1 (for $l = 0$) or proposition 3.1 (for $l \geq 1$). We will consider the issue of streamlining the computation of $\psi_{D_l}(w)$ (and $\partial\psi_{D_l}(w)$, $\psi_{v_l}(w)$ and $\partial\psi_{v_l}(w)$) in section 4.

Algorithm 3.1: (Design of the collections $\mathcal{V}$ and $\mathcal{D}$)

Input:

$D_0^1 \in \mathbb{R}^{m_2 \times m_2}$, D_0^1 nonsingular;
 $u^k \in \mathbb{R}^{m_2}$, $k = 1, 2, \dots, J$

Step 0: (Initialization)

$l := 0$;
 for $j := 1$ to m_2 do
 $(\tau_0^1)_j := \psi((D_0^1)_j)$; $v^{2j-1} := \pi((D_0^1)_j)$;
 $(s_0^1)_j := \psi(-(D_0^1)_j)$; $v^{2j} := \pi(-(D_0^1)_j)$;
 end do;
 form the QR factorization $Q_0^1 R_0^1 = D_0^1$;

Main Step:

for $k := 1$ to J do
 $w^{l+1} := u^k$;
 if $\psi(w^{l+1}) < \psi_{D_l}(w^{l+1})$ then
 /* If $l = 0$, then $\psi_{D_l}(w^{l+1})$ is computed using (21.3) and lemma 2.1. Otherwise, $\psi_{D_l}(w)$ is computed using (34), (35) and proposition 3.1. In this latter case m^* , g^* ($0 \leq m^* \leq \mu^{g^*}$, $1 \leq g^* \leq l$) would become available. */

 if $l \geq 1$ then
 $g^l := g^*$; $m^l := m^*$;
 end if;
 if $l = 0$ then
 call step 1;
 $l := l + 1$;

```

    else
      call step 2;
       $l := l + 1$ ;
    end if;
  end if;
end do;
 $L := l$ ;
call output;
end main step.

```

Step 1: (Form collections $\mathcal{V}_1$ and $\mathcal{D}_1$)

```

  solve  $Q_0^1 R_0^1 t^1 = w^1$  for  $t^1$ ;
   $\mu^1 :=$  number of nonzero components of  $t^1$ ;
  form the QR factorization  $Q^{10} R^{10} = Q_0^1 R_0^1 P^1$ ;
  /*  $P^1$  is permutation matrix defined in (24). */

```

```

 $t^1 := (P^1)^T t^1$ ;
 $r^{10} := (P^1)^T r_0^1$ ;  $s^{10} := (P^1)^T s_0^1$ ;
 $\rho^1 := \psi(w^1)$ ;  $v^{2m_2+1} := \pi(w^1)$ ;
 $\sigma^1 := \psi(-w^1)$ ;  $v^{2m_2+2} := \pi(-w^1)$ ;
end step 1.

```

Step 2: (Form the collections $\mathcal{V}_{l+1}$ and $\mathcal{D}_{l+1}$ for $l = 1, 2, \dots$)

```

  if  $m^l \neq 0$  then

```

```

    form the QR factorization  $Q^{g^l m^l} R^{g^l m^l} = Q^{g^l 0} R^{g^l 0} +$ 
       $+ (w^{g^l} - Q^{g^l 0} R^{g^l 0} e_{m^l}^T) e_{m^l}^T$ ;

```

/* This rank 1 modification of a QR factorization may be performed using routines of Buckley (1981). */

```

  end if;
  solve  $Q^{g^l m^l} R^{g^l m^l} t^{l+1} = w^{l+1}$  for  $t^{l+1}$ ;
   $\mu^{l+1} :=$  number of nonzero components of  $t^{l+1}$ ;
  form the QR factorization  $Q^{(l+1)0} R^{(l+1)0} = Q^{g^l m^l} R^{g^l m^l} P^{l+1}$ ;
  /*  $P^{l+1}$  is permutation matrix defined in section 3.1. */

```

```

 $t^{l+1} := (P^{l+1})^T t^{l+1}$ ;

```

```

 $r^{g'l} := r^{g'0} + [\rho^{g'} - e_{m'}^T r^{g'0}] e_{m'}$ ;
 $s^{g'l} := s^{g'0} + [\sigma^{g'} - e_{m'}^T s^{g'0}] e_{m'}$ ;
 $r^{(l+1)0} := (P^{l+1})^T r^{g'l}$ ;  $s^{(l+1)0} := (P^{l+1})^T s^{g'l}$ ;
 $\rho^{l+1} := \psi(w^{l+1})$ ;  $v^{2(m_2+l)+1} := \pi(w^{l+1})$ ;
 $\sigma^{l+1} := \psi(-w^{l+1})$ ;  $v^{2(m_2+l+1)} := \pi(-w^{l+1})$ ;
if  $m^l \neq 0$  then
   $P := I - (e_{\mu^{g'}} - e_{m'}) (e_{\mu^{g'}} - e_{m'})^T$ ;
  form the QR factorization  $Q^{g'0} R^{g'0} = Q^{g'0} R^{g'0} P$ ;
   $t^{g'} := P^T t^{g'}$ ;
   $r^{g'0} := P^T r^{g'0}$ ;  $s^{g'0} := P^T s^{g'0}$ ;
   $\mu^{g'} := \mu^{g'} - 1$ ;
end if;
end step 2.

```

Output:

```

output
 $L$ ; and
if  $L = 0$  then
   $Q_0^1, R_0^1, r_0^1, s_0^1$ ;
/* This is data defining  $\mathcal{D}_L, L = 0.$  */

  and  $v^j, j = 1, 2, \dots, 2m_2$ ;
/* This is data defining  $\mathcal{V}_L, L = 0.$  */

else
  for  $g := 1$  to  $L$ 
     $Q^{g0}, R^{g0}, \mu^g, t^g, r^{g0}, s^{g0}, \rho^g$  and  $\sigma^g$ ;
/* This is data defining  $\mathcal{D}_L, L \geq 1.$  */

    and for  $j := 1$  to  $2(m_2 + L)$ 
       $v^j$ ;
/* This is data defining  $\mathcal{V}_L, L \geq 1.$  */

  end if;
end output.

```

The input to algorithm 3.1 consists of $D_0^1 \in \mathbb{R}^{m_2 \times m_2}$, D_0^1 nonsingular and a collection of vectors $u^j \in \mathbb{R}^{m_2}$, $j = 1, 2, \dots, J$. We indicated earlier that a member of the collection (15) or the identity matrix in $\mathbb{R}^{m_2 \times m_2}$ may be used as D_0^1 . A possible choice for the collection of vectors u^j , $j = 1, 2, \dots, J$ is

$$u^j = M_{\cdot j}, \quad j = 1, 2, \dots, n_2. \quad (47)$$

The justification for the choice (47) is that using the results given in Ariyawansa, Sorensen and Wets (1987) it can be shown that ψ_{D_L} is exact for D_L produced by algorithm 3.1 with the choice (47) for u^j , $j = 1, 2, \dots, J (= n_2)$ and any nonsingular $D_0^1 \in \mathbb{R}^{m_2 \times m_2}$ as input. We refer the reader to Ariyawansa, Sorensen and Wets (1987) for details. Note also that since $M_{\cdot j} \in \text{pos } M$ for $j = 1, 2, \dots, n_2$, the function ψ_{V_L} we defined in section 3.2 will never take the value $-\infty$ if V_L is developed by algorithm 3.1 with the choice (47) for u^j , $j = 1, 2, \dots, n_2$.

4. Some comments on the implementation of schemes for computing $Q_D(x)$, $\partial Q_D(x)$, $Q_V(x)$ and $\partial Q_V(x)$

We shall now assume that we have applied algorithm 3.1 to the recourse problem in (1) starting with some $D_0^1 \in \mathbb{R}^{m_2 \times m_2}$, D_0^1 nonsingular, and $u^j \in \mathbb{R}^{m_2}$, $j = 1, 2, \dots, J$. Let us also assume that algorithm 3.1 has performed a number of updates $L \geq 1$. Although it is possible for algorithm 3.1 to terminate with $L = 0$ (as for example when (1) has simple recourse so that $M = [I, -I]$, if we started with $D_0^1 = I$) it is the case with $L \geq 1$ that is interesting. The output of algorithm 3.1 then is

$$Q^{g0} \in \mathbb{R}^{m_2 \times m_2}, Q^{g0} \text{ orthogonal}, R^{g0} \in \mathbb{R}^{m_2 \times m_2}, R^{g0} \text{ upper triangular},$$

$$\mu^g \in \mathbb{N}, t^g \in \mathbb{R}^{m_2}, r^{g0} \in \overline{\mathbb{R}}^{m_2}, s^{g0} \in \overline{\mathbb{R}}^{m_2}, \rho^g \in \overline{\mathbb{R}} \text{ and } \sigma^g \in \overline{\mathbb{R}}$$

for $g = 1, 2, \dots, L$ characterizing D

and

$$v^j \in \overline{\mathbb{R}}^{m_2} \text{ for } j = 1, 2, \dots, 2(m_2 + 1) \text{ characterizing } V.$$

(48)

In (48) we are denoting the collections simply by D and V dropping the subscript L .

We may now identify the collection D in (48) (or more precisely the collection D that data in (48) specifies as indicated in figure 4 for $l = L$) as the collection of matrices in figure 2 and the collection V in (48) as the collection of vectors in figure 3.

At this point we wish to state further reasons to justify the claim that the schemes for computing $Q_D(x)$, $\partial Q_D(x)$, $Q_V(x)$ and $\partial Q_V(x)$ laid out in figures 2 and 3 are suit-

able for processing on shared memory multiprocessors. Issues that need to be considered when implementing algorithms (in the context of numerical linear algebra) so that peak performance is achieved across a wide range of architectures are discussed in Dongarra and Sorensen (1986). Our aim here is to indicate one specific way for implementing one of the above four computations so that we may expect good performance on shared memory parallel vector processors like the CRAY 2, CRAY X-MP-2/4 and the Alliant FX/8. We shall choose the computation of $Q_D(x)$ for this purpose.

Dongarra and Sorensen (1986) stress the importance of avoiding unnecessary data transfer from and to the memory as a key factor in obtaining peak performance. They also demonstrate how computations organized and expressed in terms of matrix and vector operations help in this respect. In fact, it is the ability to cast the operations that are laid out in figures 2 and 3 (supplemented with the structure of D that algorithm 3.1 produces) in terms of matrix-vector operations, that makes the schemes suitable for implementation for peak performance.

In view of the fact that we have large K we shall process blocks of b_s realizations. For convenience let us assume that b_s is such that $n_b b_s = K$ where n_b is the number of such blocks of realizations. Figure 5 is a modification of figure 2 to illustrate processing of blocks of realizations and the structure of D that algorithm 3.1 has produced.

A possible way to implement the computation of $Q_D(x)$ is indicated below in algorithm 4.1.

Algorithm 4.1: (Compute $Q_D(x)$)

Input:

$K, b_s, n_b \in \mathbb{N}$ ($K = n_b b_s$); $L \in \mathbb{N}$ ($L \geq 1$);
 $h^k \in \mathbb{R}^{m_2}$, $T^k \in \mathbb{R}^{m_2 \times n_1}$, $p^k \in [0, 1]$, $k = 1, 2, \dots, K$, ($\sum_{k=1}^K p^k = 1$);
 $Q^{g0}, R^{g0} \in \mathbb{R}^{m_2 \times m_2}$; $r^{g0}, s^{g0} \in \mathbb{R}^{m_2}$; $\mu^g \in \mathbb{N}$; $t^g \in \mathbb{R}^{m_2}$; $\rho^g, \sigma^g \in \mathbb{R}$;
 $g = 1, 2, \dots, L$; and
 $x \in \mathbb{R}^{n_1}$.

Step 0: (Initialization)

$Q_D := 0$;

Main Step:


```

for  $i := 1$  to  $n_b$  do
   $\mathcal{R} := \begin{pmatrix} [h^{(i-1)b_s+1}]^T - x^T [T^{(i-1)b_s+1}]^T \\ [h^{(i-1)b_s+2}]^T - x^T [T^{(i-1)b_s+2}]^T \\ \vdots \\ [h^{ib_s}]^T - x^T [T^{ib_s}]^T \end{pmatrix};$ 
   $\mathcal{P} := \begin{pmatrix} p^{(i-1)b_s+1} \\ p^{(i-1)b_s+2} \\ \vdots \\ p^{ib_s} \end{pmatrix};$ 
   $\Psi := \begin{pmatrix} \infty \\ \infty \\ \vdots \\ \infty \end{pmatrix};$ 
  for  $g := 1$  to  $L$  do
     $T^0 := [Q^{g0} R^{g0}]^{-1} \mathcal{R}^T;$ 
    /* This is solution of  $b_s$  systems corresponding to base matrix of group  $g$  and the block
    of realizations. */

     $\Psi_{trial}^T := (r^{g0})^T T_+^0 - (s^{g0})^T T_-^0;$ 
     $\Psi := \inf(\Psi, \Psi_{trial});$ 
    /* Component-wise infimum */

    if  $\mu^g > 0$  then
      for  $m := 1$  to  $\mu^g$  do
        for  $l := 1$  to  $b_s$  do
           $T_{el} := T^0 e_l - (e_m^T T^0 e_l / e_m^T t^g) (t^g - e_m);$ 
        end do;
        /* This is using proposition 3.1 to update base system solutions to obtain solutions
        corresponding to nonbase members of group. */

         $\Psi_{trial}^T := (r^{g0} + (\rho^g - e_m^T r^{g0}) e_m)^T T_+ - (s^{g0} + (\sigma^g - e_m^T s^{g0}) e_m)^T T_-;$ 
         $\Psi := \inf(\Psi, \Psi_{trial});$ 
        /* Component-wise infimum */

      end do;
    end if;
  end do;
   $\mathcal{Q}_D := \mathcal{Q}_D + \mathcal{P}^T \Psi;$ 
end do;

```

```

call output;
end main step.

```

Output:

```

output  $Q_D$ ;
end output.

```

In terms of figure 5 we may visualize the operations of algorithm 4.1 as follows. A block of realizations consisting of b_s realizations is processed by computing columns of $\psi_{D^{sm}}$ values from left to right of figure 5 by solving base systems and using proposition 3.1. The result is a column of ψ_D values in Ψ . These values are then weighted with appropriate probabilities, summed and the result is added to the location for Q_D .

Schemes like the one in algorithm 4.1 may be given for the computation of $\partial Q_D(x)$, $Q_V(x)$ and $\partial Q_V(x)$. Such schemes are currently being implemented and full details together with numerical results will be reported in a companion paper.

5. Concluding Remarks

Our concern in this paper has been to obtain two approximants Q_V and Q_D which bound Q from below and above respectively with the property that their values and subgradients could be computed in parallel processing schemes. Of course, the motivation to consider such approximants come from the desire to replace Q in (1) with these approximants and to solve the resulting 'easier' mathematical programs.

We do not intend to discuss such optimization algorithms here. However, given a discretized version of (1) as outlined in section 1 the general scheme we have in mind is as follows. We first use algorithm 3.1 to design collections D_L and V_L . At this point we shall have the necessary *fixed* input (the input parameters that do not depend on x) to schemes like algorithm 4.1 that compute $Q_D(x)$, $\partial Q_D(x)$, $Q_V(x)$ and $\partial Q_V(x)$. Such schemes implemented on a parallel vector processor can then take x as input and produce (approximations to) values and subgradients at x of the recourse function in (1) that are needed by an optimization algorithm during its optimization of (1).

Acknowledgements

The first author wishes to thank the Mathematics and Computer Science Division of

Argonne National Laboratory, Argonne, Illinois 60439, U.S.A. for the hospitality extended to him during his visits in summer 86 and in November 86.

Appendix

We demonstrate that under assumptions (A3) and (A4) we stated in section 1, we do not lose generality in assuming $q \geq 0$ in (1). In fact we establish the following lemma.

Lemma A1: *Suppose that assumptions (A3) and (A4) hold. Then there exist a $c' \in \mathbb{R}^{n_1}$ and a $q' \in \mathbb{R}^{n_2}$, $q' \geq 0$ such that (1) and the following problem are equivalent.*

find $x \in \mathbb{R}^{n_1}$ such that

$z(x) = (c')^T x + Q'(x)$ is minimized, and

$Ax = b$, $x \geq 0$

where

$Q'(x) = E[Q'(x, \xi)],$

$Q'(x, \xi) = \inf_{y \in \mathbb{R}^{n_2}} \{(q')^T y : My = h - Tx, y \geq 0\},$

$A \in \mathbb{R}^{m_1 \times n_1}$, $b \in \mathbb{R}^{m_1}$, and $M \in \mathbb{R}^{m_2 \times n_2}$ are deterministic and are as in (1), and

$h \in \mathbb{R}^{m_2}$, $T \in \mathbb{R}^{m_2 \times n_1}$ are random with the same cumulative probability distribution

F of ξ on Ξ in (1).

(A1)

Proof: By assumption (A3) there exists some $v_0 \in \mathbb{R}^{m_2}$ such that

$$M^T v_0 \leq q. \quad (A2)$$

Note that

$$\begin{aligned} Q(x, \xi) &= \inf_{y \in \mathbb{R}^{m_2}} \{q^T y : My = h - Tx, y \geq 0\} \\ &= \inf_{y \in \mathbb{R}^{m_2}} \{q^T y - v_0^T (My - (h - Tx)) : My = h - Tx, y \geq 0\} \\ &= \inf_{y \in \mathbb{R}^{m_2}} \{(q - M^T v_0)^T y + v_0^T (h - Tx) : My = h - Tx, y \geq 0\} \\ &= v_0^T (h - Tx) + \inf_{y \in \mathbb{R}^{m_2}} \{(q')^T y : My = h - Tx, y \geq 0\} \end{aligned}$$

where $q' = q - M^T v_0$ and $q' \geq 0$ by (A2).

Now let $Q'(x, \xi) = \inf_{y \in \mathbb{R}^{m_2}} \{(q')^T y : My = h - Tx, y \geq 0\}$. We have

$$\begin{aligned} & \inf \{c^T x + \int_{\xi \in \Xi} Q(x, \xi) dF(\xi) : Ax = b, x \geq 0\} \\ &= \inf \{c^T x + \int_{\xi \in \Xi} [v_0^T (h(\xi) - T(\xi)x) + Q'(x, \xi)] dF(\xi) : Ax = b, x \geq 0\} \\ &= v_0^T E(h) + \inf \{(c')^T x + \int_{\xi \in \Xi} Q'(x, \xi) dF(\xi) : Ax = b, x \geq 0\} \end{aligned}$$

where $c' = c - (E(T))^T v_0$ and we have used assumption (A4) to assert the existence of $E(h)$ and $E(T)$.

It now follows that (1) and (A1) are equivalent (except for the constant $v_0^T E(h)$). ■

References

1. Ariyawansa, K. A., D. C. Sorensen and R. J.-B. Wets (1987), "On the convexity of an upper bounding approximant to the recourse function in a class of stochastic programs", in preparation.
2. Birge, J. R. and R. J.-B. Wets (1986a), "Designing approximation schemes for stochastic optimization problems, in particular for stochastic programs with recourse", *Mathematical Programming Study*, 27, pp. 54-102.
3. Birge, J. R. and R. J.-B. Wets (1986b), "On-line solution of linear programs using sublinear functions", Technical Report 86-25, Department of Industrial and Operations Engineering, The University of Michigan, Ann Arbor, Michigan.
4. Birge, J. R. and R. J.-B. Wets (1986c), "Sublinear upper bounds for stochastic programs with recourse", to appear in *Mathematical Programming*.
5. Buckley, A. (1981), "QRUP: A set of Fortran routines for updating QR factorizations", *ACM Transactions of Mathematical Software*, 7, pp. 548-549.
6. Dempster, M. A. H. (1980), *Stochastic Programming*, Academic Press, New York.
7. Dongarra, J. J. and D. C. Sorensen (1986) "Linear algebra on high-performance computers", *Parallel Computing* 85, eds., M. Feilmeier, G. Joubert and U. Schendel, pp. 3-32.
8. Ermoliev, Y. and R. J.-B. Wets (1984) "Numerical techniques for stochastic optimization problems", Technical Report PP-84-04, International Institute for Applied Systems Analysis, A-2361 Laxenburg, Austria.
9. Ortega, J. M. and W. C. Rheinboldt (1970), *Iterative Solution of Nonlinear Equations in Several Variables*, Academic Press, New York.

10. Wets, R. J. -B. (1966) "Programming under uncertainty: The equivalent convex program", *J. SIAM Appl. Math.*, Vol. 14, No. 1. pp. 89-105.
11. Wets, R. J.-B. (1982), "Stochastic programming: Solution techniques and approximation schemes", in *Mathematical Programming—The State of the Art*, eds., A. Bachem, M. Grötschel and B. Korte, pp. 566-603.
12. Wets, R. J.-B. (1985), "On parallel processors design for stochastic programs", Working Paper WP 85-67, International Institute for Applied Systems Analysis, A-2361 Laxenburg, Austria.
13. Stewart, G. W. (1973), *Introduction to Matrix Computations*, Academic Press, New York.

Appendix 2

This appendix is part of the final report of DOE Grant No. DE-FG06-87ER25045.

A Variant of Algorithm 3.1 of Ariyawansa, Sorensen and Wets (1987)

The purpose of this appendix is to present a variant of Algorithm 3.1 of ASW (1987). In order to keep the presentation brief we do so under the following conditions. We give only the steps that design the collection $\mathcal{D}_L$. We assume that $\text{pos } M = \mathbb{R}^{m_2}$, $q \geq 0$ and that a certain lp has a unique optimal basis. (This latter assumption is specified in the comments of the statement below.) Finally, we avoid the use of matrix factorizations.

Algorithm A2.1: (Design of the collection $\mathcal{D}$)

Input:

nonsingular $D_0 \in \mathbb{R}^{m_2 \times m_2}$;
nonzero $u^k \in \mathbb{R}^{m_2}$, $k = 1, 2, \dots, J$.

Step 0: (Initialization)

$l := 0$;
for $j := 1$ **to** m_2 **do**
 $r_0^T e_j := \psi(D_0 e_j)$; $s_0^T e_j := \psi(-D_0 e_j)$;
end do;
 $a_0 := [r_0^T, s_0^T]^T$;
 $V_0 := [D_0, -D_0]$;
end initialization.

Main Step:

for $k := 1$ **to** J **do**
 $w^{l+1} := u^k$;
 $\rho^{l+1} := \psi(w^{l+1})$; $\hat{\rho}^{l+1} := \text{co } \psi_{\mathcal{D}_l}(w^{l+1})$;
 /* $\text{co } \psi_{\mathcal{D}_l}(w) := \inf \{ \alpha : \begin{bmatrix} \alpha \\ w \end{bmatrix} \in \text{pos} \begin{bmatrix} a_l^T \\ V_l \end{bmatrix} \}$ */
 if $\rho^{l+1} < \hat{\rho}^{l+1}$ **then**
 if $l = 0$ **call** *first update* ($[\cdot, \dots, \cdot]$);
 if $l \geq 1$ **call** *update* ($[\cdot, \dots, \cdot]$);
 end if;
end do;
 $L := l$;
call *output*;
end main step.

First Update: (Form collection $\mathcal{D}_1$)

solve $D_0 t^1 = w^1$ **for** t^1 ;
 $\mu^1 := \text{number of nonzero components of } t^1$;
 $D^{10} := D_0 P^1$;
/* P^1 is any permutation matrix that permutes the components of t^1 so that its first μ^1 components are nonzero. */

$t^1 := (P^1)^T t^1;$
 $r^{10} := (P^1)^T r_0; s^{10} := (P^1)^T s_0;$
 $\sigma^1 := \psi(-w^1);$
 $a_1 := [(r^{10})^T, \rho^1, (s^{10})^T, \sigma^1];$
 $V_1 := [D^{10}, w^1, -D^{10}, -w^1];$
 $l := 1;$
 end first update.

Update: (Form the collections $\mathcal{D}_{l+1}$ for $l = 1, 2, \dots$)

$B^* :=$ the optimal basis of $\text{co } \psi_{\mathcal{D}_l}$ at $w^{l+1};$

/* We assume that $\text{co } \psi_{\mathcal{D}_l}$ has a unique optimal basis at w^{l+1} . */

for $i := 1$ to m_2 do

$j^i := \underset{j=1,2,\dots,2(m_2+l); j \notin \mathcal{J}_{B^*}}{\text{argmin}} \{ [(a_l)_B^T (B^*)^{-1} (V_l)_{.j} - (a_l)_j] / [-e_i^T (B^*)^{-1} (V_l)_{.j}] :$

$e_i^T (B^*)^{-1} (V_l)_{.j} < 0 \};$

end do;

/* $\mathcal{J}_{B^*}$ is the set of column indices of V_l corresponding to B^* */

$\bar{\rho}_{l+1} := \hat{\rho}^{l+1} - \min_{1 \leq i \leq m_2} e_i^T (B^*)^{-1} w^{l+1} [(a_l)_B^T (B^*)^{-1} (V_l)_{.j^i} - (a_l)_{j^i}] / [e_i^T (B^*)^{-1} (V_l)_{.j^i}];$

if $\rho^{l+1} < \bar{\rho}_{l+1}$ then $\rho^{l+1} := \bar{\rho}_{l+1};$

$\sigma^{l+1} := \psi(-w^{l+1}); \hat{\sigma}^{l+1} := \text{co } \psi_{\mathcal{D}_l}(-w^{l+1});$

if $\sigma^{l+1} < \hat{\sigma}^{l+1}$ then

if $-(a_l)_{(-B^*)}^T (B^*)^{-1} (V_l)_{.j} - (a_l)_j < 0 \ \forall j = 1, 2, \dots, 2(m_2 + l); j \notin \mathcal{J}_{(-B^*)}$ then

for $i = 1$ to m_2 do

$j^i := \underset{j=1,2,\dots,2(m_2+l); j \notin \mathcal{J}_{(-B^*)}}{\text{argmin}} \{ -[(a_l)_{(-B^*)}^T (B^*)^{-1} (V_l)_{.j} + (a_l)_j] / [e_i^T (B^*)^{-1} (V_l)_{.j}] :$

$e_i^T (B^*)^{-1} (V_l)_{.j} > 0 \};$

end do;

$\bar{\sigma}_{l+1} := \hat{\sigma}^{l+1} + \min_{1 \leq i \leq m_2} e_i^T (B^*)^{-1} w^{l+1} [(a_l)_{(-B^*)}^T (B^*)^{-1} (V_l)_{.j^i} +$
 $+(a_l)_{j^i}] / [e_i^T (B^*)^{-1} (V_l)_{.j^i}];$

if $\sigma^{l+1} < \bar{\sigma}_{l+1}$ then $\sigma^{l+1} := \bar{\sigma}_{l+1};$

else $\sigma^{l+1} := \hat{\sigma}^{l+1};$

end if;

end if;

for $m \in \{0, 1, \dots, \mu^g\}, g \in \{1, 2, \dots, l\}$ let $D^{gm} := D^{g0} + [w^g - D^{g0} e_m] e_m^T$ and let

$m^* \in \{0, 1, \dots, \mu^g\}, g^* \in \{1, 2, \dots, l\}$ be such that

$[\pm D_{.1}^{g^* m^*}, \pm D_{.2}^{g^* m^*}, \dots, \pm D_{.m_2}^{g^* m^*}] = B^*;$

solve $D^{g^* m^*} t^{l+1} = w^{l+1}$ for $t^{l+1};$

$\mu^{l+1} :=$ number of nonzero components of $t^{l+1};$

$D^{(l+1)0} := D^{g^* m^*} P^{l+1};$

*/** P^{l+1} is any permutation matrix that permutes the components of t^{l+1} so that its first μ^{l+1} components are nonzero. **/*

$t^{l+1} := (P^{l+1})^T t^{l+1};$

if $m^* = 0$ **then**

$r^{(l+1)0} := (P^{l+1})^T r^{g^*0}; s^{(l+1)0} := (P^{l+1})^T s^{g^*0};$

else

$r^{g^*m^*} := r^{g^*0} + [\rho^{g^*} - e_{m^*}^T r^{g^*0}] e_{m^*}; s^{g^*m^*} := s^{g^*0} + [\sigma^{g^*} - e_{m^*}^T s^{g^*0}] e_{m^*};$

$r^{(l+1)0} := (P^{l+1})^T r^{g^*m^*}; s^{(l+1)0} := (P^{l+1})^T s^{g^*m^*};$

$P := I - (e_{\mu^{g^*}} - e_{m^*})(e_{\mu^{g^*}} - e_{m^*})^T;$

$t^{g^*} := P^T t^{g^*};$

$r^{g^*0} := P^T r^{g^*0}; s^{g^*0} := P^T s^{g^*0};$

$\mu^{g^*} := \mu^{g^*} - 1;$

end if;

$l := l + 1;$

end update.

Output:

output

L ; and

if $L = 0$ **then**

$D_0, r_0, s_0;$

/ This is data defining $\mathcal{D}_L, L = 0.$ **/**

else

for $g := 1$ **to** L

$D^{g0}, \mu^g, t^g, r^{g0}, s^{g0}, \rho^g$ and $\sigma^g;$

/ This is data defining $\mathcal{D}_L, L \geq 1.$ **/**

end if;

end output.

Appendix 3

This appendix is part of the final report of DOE Grant No. DE-FG06-87ER25045.

**A preliminary numerical evaluation of a parallel algorithm
for approximating the values and subgradients of the recourse function
in a stochastic program with complete recourse**

Kelly S. Lessor

A project submitted in partial fulfillment of
the requirements for the degree of
Master of Science

Washington State University
Department of Computer Science

August 26, 1988

1. Introduction

Consider the class of stochastic programs:

Find $x \in \mathbb{R}^{n_1}$ such that

$z(x) = c^T x + Q(x)$ is minimized, and

$Ax = b, x \geq 0$

where

$Q(x) := E[Q(x, h, T)] = \sum_{k=1}^K p^k Q(x, h^k, T^k);$

$Q(x, h, T) := \inf_{y \in \mathbb{R}^{n_2}} \{q^T y : My = h - Tx, y \geq 0\};$ and

$A \in \mathbb{R}^{m_1 \times n_1}, b \in \mathbb{R}^{m_1}, c \in \mathbb{R}^{n_1}, q \in \mathbb{R}^{n_2}, M \in \mathbb{R}^{m_2 \times n_2}$ are deterministic and given and $(h, T) \in \mathbb{R}^{m_2} \times \mathbb{R}^{m_2 \times n_1}$ is a random variable with a given discrete distribution $\{(p^k, (h^k, T^k)), k = 1, 2, \dots, K\}$ where $p^k := \Pr(\{(h, T) = (h^k, T^k)\})$, $k = 1, 2, \dots, K$.

(1.1)

In this paper the following additional assumptions will be made about problem (1.1).

(A1) The set $X = \{x : x \in \mathbb{R}^{n_1}, Ax = b, x \geq 0\}$ is nonempty.

(A2) The set $\{y : My = w, y \geq 0\}$ is nonempty for all $w \in \mathbb{R}^{m_2}$.

(A3) The set $\{v : v \in \mathbb{R}^{m_2}, M^T v \leq q\}$ is nonempty.

(A4) $q \geq 0$.

Assumption (A1) is made to ensure that problem (1.1) is meaningful. The function Q is called the *recourse function*. Since when assumption (A2) is satisfied, $Q(x) < \infty$ for all $x \in \mathbb{R}^{n_1}$, problem (1.1) is called a *stochastic program with complete recourse*. Assumption (A3) says that the dual of the linear program that defines the recourse problem is feasible. When assumption (A3) is satisfied, there is no loss of generality in making assumption (A4) as is shown by Ariyawansa, Sorensen, and Wets [1, Appendix].

Problems of the form (1.1) occur in numerous applications. Among the examples given in [6] are the following:

- (i) The allocation of aircraft of various types among given routes to satisfy an uncertain passenger demand while minimizing operating cost plus lost revenue.
- (ii) The planning of the level of investment in sewage treatment facilities so that the expected deviations of pollutant concentration levels are minimized.
- (iii) The planning by an employer of the number of regular staff and the number of temporary staff necessary to meet an unknown demand for services at minimum

cost.

An algorithm for solving the convex program (1.1) will need the value of Q , which at $x \in X$ is given by

$$Q(x) := E[Q(x, h, T)] = \sum_{k=1}^K p^k Q(x, h^k, T^k). \quad (1.2a)$$

Such an algorithm may also need values of ∂Q , the subgradient of Q . By (1.2a) it follows that $\partial Q(x)$ for $x \in X$ is given by

$$\partial Q = - \sum_{k=1}^K p^k (T^k)^T v^*(x, h^k, T^k). \quad (1.2b)$$

In (1.2a) and (1.2b)

$$Q(x, h^k, T^k) := \min_{y \in \mathbb{R}^{m_2}} \{q^T y : My = h^k - T^k x, y \geq 0\} \quad \text{and} \quad (1.2c)$$

$$v^*(x, h^k, T^k) := \operatorname{argmax}_{v \in \mathbb{R}^{m_2}} \{[h^k - T^k x]^T v : M^T v \leq q\} \quad (1.2d)$$

for $k = 1, 2, \dots, K$. Solving (1.1) using a general algorithm for convex programs may be very time consuming because, in order to find the values and subgradients of Q at a given x , one must solve K linear programs (lps)—one for each of the K realizations of h and T . Usual applications that lead to (1.1) have ‘large’ K (i.e. in the range 10^4 to 10^6). Therefore, it is worthwhile to solve the K linear programs of (1.2c) in parallel. Solving the linear programs of (1.2c) in parallel will speed up the calculation of values of Q and ∂Q significantly, but it would be better still if it was possible to reduce the work done in parallel to something easier than solving linear programs.

Since calculating the value and subgradient of Q requires so much time, using approximations to Q and ∂Q would be attractive if the values of these approximations could be computed more quickly than the exact values. In [13], Wets has proposed an approximation Q_D of Q which serves as an upper bound for Q so that

$$Q(x) \leq Q_D(x) \quad \text{for } x \in X. \quad (1.3)$$

(See [3], [4], and [5] for related information on the above approximation.) The approximation Q_D is based on a collection $\mathcal{D}$ of nonsingular matrices in $\mathbb{R}^{m_2 \times m_2}$. Along with the upper bound Q_D , it is possible (using duality theory) to compute a corresponding lower bound, Q_V , for Q on X so that

$$Q_V(x) \leq Q(x) \leq Q_D(x) \quad \text{for } x \in X. \quad (1.4)$$

The lower bound Q_V is based on a collection V of vectors in $\mathbb{R}^{m^2}$. The collections V and $\mathcal{D}$ may be created simultaneously. See [1] for details.

The calculation of $Q_{\mathcal{D}}(x)$, $\partial Q_{\mathcal{D}}(x)$, $Q_V(x)$, and $\partial Q_V(x)$, like the calculation of $Q(x)$ and $\partial Q(x)$, may be done more quickly with the use of parallel processing. While the work required to calculate $Q_{\mathcal{D}}$ and $\partial Q_{\mathcal{D}}$ depends on the structure of $\mathcal{D}$, the hardest part of the calculation is the solution of a system of linear equations. For an 'unstructured' $\mathcal{D}$, the number of systems of linear equations that must be solved for each x is equal to $K\nu$ where $\nu = \text{card}(\mathcal{D})$; but for a collection $\mathcal{D}$ of the form detailed in [1], even fewer systems of equations need to be solved. These systems of equations, which have the form $Dt^k = h^k - T^k x$, where D is in $\mathcal{D}$, may be solved in parallel to obtain t^k , $k = 1, 2, \dots, K$. Because solving a system of linear equations is generally easier than solving a linear program, it is likely that computing $Q_{\mathcal{D}}$ and $\partial Q_{\mathcal{D}}$ using parallel processors will be faster than computing Q and ∂Q using parallel processors.

While it appears that it may be worthwhile to design the collections $\mathcal{D}$ and V and then use them to calculate the values of the approximations, $Q_{\mathcal{D}}$ and $\partial Q_{\mathcal{D}}$, instead of calculating values of Q and ∂Q exactly, only a controlled experiment will verify this. The purpose of this project is to perform such an experimental investigation. In particular, data on the amount of time involved in

calculating values of Q and ∂Q ;

developing $\mathcal{D}$ and V ; and

calculating values of $Q_{\mathcal{D}}$, $\partial Q_{\mathcal{D}}$, Q_V , and ∂Q_V

will be collected. These timing results should indicate whether or not using $Q_{\mathcal{D}}$ (and Q_V) and $\partial Q_{\mathcal{D}}$ (and ∂Q_V) is worthwhile in an algorithm used for solving (1.1). In this project the following 'speedups' will be calculated: the 'speedup' that results from computing values of Q and ∂Q with processors in parallel; the 'speedup' that results from computing values of the approximations $Q_{\mathcal{D}}$ and $\partial Q_{\mathcal{D}}$ with processors in parallel; and a 'speedup' of the approximate calculations using $Q_{\mathcal{D}}$ and $\partial Q_{\mathcal{D}}$ relative, to the exact calculation using Q and ∂Q .

The structure of the rest of the paper is as follows: section 2 describes the algorithms that generate the collections $\mathcal{D}$ and V and the way functions $Q_{\mathcal{D}}$ and Q_V are constructed; section 3 outlines the experiments that have been performed; and section 4 gives the results of the experiments. An explanation of the notation and terms used in this paper may be found in the appendix.

2. Description of the Collections $\mathcal{D}$ and $\mathcal{V}$ and the Functions $Q_{\mathcal{D}}$ and $Q_{\mathcal{V}}$

Define the function $\psi : \mathbb{R}^{m_2} \rightarrow \mathbb{R}$ by

$$\psi(w) := \inf_{y \in \mathbb{R}^{n_2}} \{q^T y : My = w, y \geq 0\}. \quad (2.1)$$

Note that $Q(x, h, T)$ of (1.1) is given by $Q(x, h, T) = \psi(w)$, $w = h - Tx$. The bounds $Q_{\mathcal{V}}$ and $Q_{\mathcal{D}}$ mentioned in the previous section for Q on X are induced by two approximants $\psi_{\mathcal{D}} : \mathbb{R}^{m_2} \rightarrow \mathbb{R}$ and $\psi_{\mathcal{V}} : \mathbb{R}^{m_2} \rightarrow \mathbb{R}$ for ψ so that $\psi_{\mathcal{V}}(w) \leq \psi(w) \leq \psi_{\mathcal{D}}(w)$. An alternate equivalent definition of the function ψ , which will be useful in the sequel is

$$\psi(w) := \inf \left\{ \alpha : \begin{pmatrix} \alpha \\ w \end{pmatrix} \in \text{pos} \begin{pmatrix} q^T \\ M \end{pmatrix} \right\}. \quad (2.2)$$

The function ψ has the following properties (see [14] and [2]):

- (i) $0 \leq \psi(w) < \infty$ for $w \in \text{pos } M$ and $\psi(0) = 0$.
- (ii) ψ is convex on $\text{pos } M$.
- (iii) The epigraph $\text{epi } \psi := \left\{ \begin{pmatrix} \alpha \\ w \end{pmatrix} : \alpha \geq \psi(w), w \in \text{pos } M \right\}$ of ψ is a convex polyhedral cone.

The following characterization for the epigraph of ψ is stated in part (v) of Theorem 3.1 of [2]

$$\text{epi } \psi = \text{pos} \begin{pmatrix} 0^T & (q^1)^T \\ M^0 & M^1 \end{pmatrix} \quad (2.3)$$

where $q^T P = [0^T, (q^1)^T]$ (with $0 \in \mathbb{R}^{n^0}$, $q^1 \in \mathbb{R}^{n^1}$, $q^1 > 0$, $n^0 + n^1 = n_2$) is the permutation of q^T so that its first n^0 components are zeroes and the next n^1 components are positive, and $MP = [M^0, M^1]$. The level set $\text{lev}_{\tau} \psi := \{w : \psi(w) \leq \tau, w \in \text{pos } M\}$ of ψ at τ ($0 \leq \tau < \infty$) is shown in part (vi) of Theorem 3.1 of [2] to be

$$\text{lev}_{\tau} \psi = \text{co} (\tau M^1 (\text{diag } q^1)^{-1}, 0). \quad (2.4)$$

The characterizations in (2.2) and (2.3) may now be illustrated geometrically for $m_2 = 2$. Based on equations (2.2) and (2.3), a typical function ψ with $m_2 = 2$ and $q > 0$ appears as in figure (2.1). The epigraph of ψ (which is a set in $\mathbb{R}^3$) includes the surface of the infinite cone along with its interior. Similarly the corresponding level set at $\tau = 1$ of that function ψ appears as in figure (2.2). (Note that the shape of the graph of the level set of ψ is the same for any value of τ .)

Given a nonsingular matrix D , it is possible to define a function ψ_D , $\psi_D : \mathbb{R}^{m_2} \rightarrow \mathbb{R}$ so that ψ_D agrees with ψ along the directions of the columns of D and the directions of

their negatives. Such a function ψ_D defined in [1] may be described formally as follows. Let $D \in \mathbb{R}^{m_2 \times m_2}$ be nonsingular and $r, s \in \mathbb{R}^{m_2}$ be defined by

$$r := \sum_{j=1}^{m_2} \psi(D_{\cdot j}) e_j, \quad s := \sum_{j=1}^{m_2} \psi(-D_{\cdot j}) e_j. \quad (2.5)$$

Then $\psi_D : \mathbb{R}^{m_2} \rightarrow \mathbb{R}$ is defined by

$$\psi_D(w) := \inf \left\{ \alpha : \begin{pmatrix} \alpha \\ w \end{pmatrix} \in \text{pos } U \right\}, \quad U := \begin{pmatrix} r^T & s^T \\ D & -D \end{pmatrix}. \quad (2.6)$$

As Ariyawansa, Sørensen, and Wets showed [1, Lemma 2.1], if ψ_D is defined by (2.5) and (2.6); $w \in \mathbb{R}^{m_2}$ is given; $t \in \mathbb{R}^{m_2}$ is the solution to the system $Dt = w$; $Y, Z \in \mathbb{R}^{m_2 \times m_2}$ are defined by

$$Y = \text{diag}(\text{sgn}(t)), \quad Z = \text{diag}(t_z); \quad (2.7)$$

and assumption (A4) is satisfied; then the value $\psi_D(w)$ and the subgradient $\partial\psi_D(w)$ of ψ_D at w can be *explicitly* given as follows:

$$\psi_D(w) = r^T t_+ - s^T t_- \quad \text{and} \quad (2.8)$$

$$\partial\psi_D(w) = \{u : D^T u = v, v = (Y_+ - Z)r + Y_- s + Zh, h \in [-s, r]\}. \quad (2.9)$$

With $m_2 = 2$, $q > 0$, and the characterization of ψ_D given in (2.6), the level set of ψ_D with $\tau = 1$ will be a convex polygon which lies inside the convex polygon for the level set of ψ with $\tau = 1$ (see figure (2.4)). Similarly, the epigraph of ψ_D will be a convex polyhedral cone which lies inside the convex polyhedral cone which defines the epigraph of ψ (see figure (2.3)).

ψ_D is an attractive upper bound for ψ because it agrees with ψ along some directions and it is convex. Moreover, it is easy to calculate the values and subgradients of ψ_D —the most difficult portion of the calculation is solving the two systems of linear equations $Dt = w$ and $D^T u = v$. Unfortunately, there exist directions along which ψ_D may be considerably larger than ψ (see for example, direction w^1 in figure (2.4)). In order to improve the approximation to ψ , it is possible to define ν functions; $\psi_{D^1}, \psi_{D^2}, \dots, \psi_{D^\nu}$; relative to a collection, $\mathcal{D}$, of ν nonsingular matrices in $\mathbb{R}^{m_2 \times m_2}$, $\mathcal{D} = \{D^1, D^2, \dots, D^\nu\}$ and then define $\psi_{\mathcal{D}} : \mathbb{R}^{m_2} \rightarrow \mathbb{R}$ by

$$\psi_{\mathcal{D}}(w) = \inf \{\psi_{D^1}, \psi_{D^2}, \dots, \psi_{D^\nu}\}. \quad (2.10)$$

For $m_2 = 2$ and $\mathcal{D} = \{D^1, D^2\}$, a typical epigraph of $\psi_{\mathcal{D}}$ and a typical level set of $\psi_{\mathcal{D}}$ with $\tau = 1$ appear as in figures (2.5) and (2.6) respectively.

$\psi_{\mathcal{D}}$ is an attractive upper bound for ψ because it agrees with ψ along many directions and it is easy to calculate since values and subgradients of each of the individual functions,

ψ^{D^i} , are easy to calculate. Unfortunately $\psi_{\mathcal{D}}$ may not be convex. For an arbitrary collection of matrices, $\mathcal{D}$, $\psi_{\mathcal{D}}$ will generally not be convex. (In figure (2.6) for example, $\text{lev}_1 \psi_{\mathcal{D}}$ is not convex.) However, it is possible to choose a collection of matrices, $\mathcal{D}$, so that $\psi_{\mathcal{D}}$ is convex.

Algorithm 3.1 of [1] creates a suitable collection of matrices, $\mathcal{D}$, so that $\psi_{\mathcal{D}}$ will be convex. The algorithm takes as input a nonsingular matrix, D_0^1 , and a set of vectors $u^k \in \mathbb{R}^{m_2}$, $u^k \neq 0$, $k = 1, 2, \dots, J$. From this input, algorithm 3.1 of [1] implicitly generates a collection of nonsingular matrices, $\mathcal{D}$, which will ensure that $\psi_{\mathcal{D}}$ is exact along all of the given directions u^k , $k = 1, 2, \dots, J$, and that $\psi_{\mathcal{D}}$ is convex. Additionally, the algorithm generates the collection of vectors $\mathcal{V}$.

The matrices in the collection, $\mathcal{D}$, which are generated by algorithm 3.1 of [1] are defined implicitly. If the function $\psi_{\mathcal{D}}$ for $\mathcal{D} = \{D_0^1\}$ agrees with the function ψ along all the directions, u^j , $j = 1, 2, \dots, J$, then algorithm 3.1 will give the quantities $Q_0^1 \in \mathbb{R}^{m_2 \times m_2}$, Q_0^1 orthogonal, $R_0^1 \in \mathbb{R}^{m_2 \times m_2}$, R_0^1 upper triangular, $r_0^1 \in \mathbb{R}^{m_2}$, and $s_0^1 \in \mathbb{R}^{m_2}$ as output. The quantities Q_0^1 and R_0^1 are the QR factors of the matrix D_0^1 . D_0^1 is implicitly stored this way so that the systems of equations in which it is the coefficient matrix may be solved in a numerically stable way. The quantities r_0^1 and s_0^1 are the r and s of (2.5) with $D := D_0^1$. If the function $\psi_{\mathcal{D}}$ for $\mathcal{D} = \{D_0^1\}$ does not agree with ψ along all of the given directions, u^j , $j = 1, 2, \dots, J$, then algorithm 3.1 of [1] will make L updates to the initial collection, $\mathcal{D} = \{D_0^1\}$, and then will give the quantities $Q^{g0} \in \mathbb{R}^{m_2 \times m_2}$, $R^{g0} \in \mathbb{R}^{m_2 \times m_2}$, $\mu^{g0} \in \mathbb{R}$, $t^g \in \mathbb{R}^{m_2}$, $r^{g0} \in \mathbb{R}^{m_2}$, $s^{g0} \in \mathbb{R}^{m_2}$, $\rho^g \in \mathbb{R}$, and $\sigma^g \in \mathbb{R}$, for $g = 1, 2, \dots, L$ as output. In this case, the collection $\mathcal{D}$ may be viewed as being made up of L groups of matrices; each group, g , has an associated 'base matrix', $D^{g0} = Q^{g0} R^{g0}$, as well as μ^g non-base matrices, D^{gm} , $m = 1, 2, \dots, \mu^g$. The quantity t^g is the solution to the equation $D^{g0} t^g = w^g$ where w^g is one of the vectors u^j . The quantities r^{g0} and s^{g0} are the r and s of (2.5) with $D := D^{g0}$. The quantities ρ^g and σ^g are the values of the function ψ at w^g and $-w^g$ respectively. If $\mu^g > 0$, the non-base member, D^{gm} , of group g is defined by $D^{gm} := D^{g0} + [w^g - D^{g0} e_m] e_m^T$ for $m = 1, 2, \dots, \mu^g$.

Relative to the above data produced by algorithm 3.1 of [1], the function $\psi_{\mathcal{D}}$ is defined by

$$\psi_{\mathcal{D}}(w) := \inf_{m,g} \{ \psi_{D^{gm}}(w); m = 0, 1, \dots, \mu^g; g = 1, 2, \dots, L \}. \quad (2.11)$$

In (2.11), the values and subgradients of $\psi_{D^{gm}}$ may be computed using $D := D^{g0}$, $r := r^{g0}$, and $s := s^{g0}$ in (2.8) and (2.9), where if $\mu^g > 0$, then $r^{gm} := r^{g0} + [\rho^g - e_m^T r^{g0}] e_m$ and $s^{gm} := s^{g0} + [\sigma^g - e_m^T s^{g0}] e_m$ for $m = 0, 1, \dots, \mu^g$; $g = 1, 2, \dots, L$. If m^* and g^* are the indices that yield the infimum in (2.11), it follows that

$$\partial \psi_{\mathcal{D}}(w) = \partial \psi_{D^{g^* m^*}}(w). \quad (2.12)$$

The reader is referred to [1, Proposition 3.1] where it is shown that for the case $L > 1$ the computational work involved in obtaining $\psi_{\mathcal{D}}(w)$ and $\partial\psi_{\mathcal{D}}(w)$ as outlined above can be further reduced.

Algorithm 3.1 of [1] also provides a collection of vectors $\mathcal{V} := \{v^j : v^j \in \mathbb{R}^{m_2}, j = 1, 2, \dots, 2(m_2 + L)\}$ as part of the output. Relative to $\mathcal{V}$, the function $\psi_{\mathcal{V}}$ is defined by

$$\psi_{\mathcal{V}}(w) := \sup_j \{(v^j)^T w; j = 1, 2, \dots, 2(m_2 + L)\}. \quad (2.13)$$

It follows that if j^* is the index that yields the supremum in (2.13) then

$$\partial\psi_{\mathcal{V}}(w) = v^{j^*}. \quad (2.14)$$

For $x \in X$ the values and the subgradients of the functions $\mathcal{Q}$, $\mathcal{Q}_{\mathcal{D}}$, and $\mathcal{Q}_{\mathcal{V}}$ may now be written in terms those of ψ , $\psi_{\mathcal{D}}$, and $\psi_{\mathcal{V}}$ as:

$$\mathcal{Q}(x) := \sum_{k=1}^K p^k \psi(h^k - T^k x), \quad (2.15)$$

$$\partial\mathcal{Q}(x) = - \sum_{k=1}^K p^k (T^k)^T \partial\psi(h^k - T^k x), \quad (2.16)$$

$$\mathcal{Q}_{\mathcal{D}}(x) := \sum_{k=1}^K p^k \psi_{\mathcal{D}}(h^k - T^k x), \quad (2.17)$$

$$\partial\mathcal{Q}_{\mathcal{D}}(x) = - \sum_{k=1}^K p^k (T^k)^T \partial\psi_{\mathcal{D}}(h^k - T^k x), \quad (2.18)$$

$$\mathcal{Q}_{\mathcal{V}}(x) := \sum_{k=1}^K p^k \psi_{\mathcal{V}}(h^k - T^k x), \quad \text{and} \quad (2.19)$$

$$\partial\mathcal{Q}_{\mathcal{V}}(x) = - \sum_{k=1}^K p^k (T^k)^T \partial\psi_{\mathcal{V}}(h^k - T^k x). \quad (2.20)$$

Figure 2.1. The function ψ and its epigraph (when $m_2 = 2$ and $q > 0$)

Figure 2.2. The level set $\text{lev}_1 \psi$ (when $m_2 = 2$ and $q > 0$)

Figure 2.3. The functions ψ and ψ_D and their epigraphs (when $m_2 = 2$ and $q > 0$)

Figure 2.4. The level sets $\text{lev}_1\psi$ and $\text{lev}_1\psi_D$ (when $m_2 = 2$ and $q > 0$)

Figure 2.5. The functions ψ , ψ_{D^1} , and ψ_{D^2} and their epigraphs (when $m_2 = 2$ and $q > 0$)

Figure 2.6. The level sets $\text{lev}_1 \psi$, $\text{lev}_1 \psi_{D^1}$, $\text{lev}_1 \psi_{D^2}$, and $\text{lev}_1 \psi_D$ (when $m_2 = 2$ and $q > 0$)

3. The Experiment

In order to decide if it is worthwhile to use the approximations, Q_D and Q_V , for Q in an algorithm to solve (1.1), a comparison of the amount of time necessary to design D and to calculate $Q_D(x)$, $\partial Q_D(x)$, $Q_V(x)$, and $\partial Q_V(x)$ to the amount of time necessary to calculate $Q(x)$ and $\partial Q(x)$ will be needed. It would be desirable to do the comparison for varying problem sizes, varying numbers of realizations of (h, T) , varying distributions for (h, T) , and varying numbers of processors.

In this project, problems of the form (1.1) were generated using the stochastic test problem generator, GENSLP, [11] which generates problems which are guaranteed to have complete recourse. GENSLP generates the variables c , A , b , q , and M so that elements of the variables are uniformly distributed in a user specified range and additionally matrices A and M have user specified 'densities' of nonzero entries. The uniform random number generator used is that of [12]. GENSLP does not generate distributions for (h, T) , but it does have a facility so that the user can generate a distribution on (h, T) . A realization (h^k, T^k) of (h, T) may be generated as follows:

$$h^k := a^0 + \chi^1 a^1 + \chi^2 a^2 + \cdots + \chi^\kappa a^\kappa \quad (3.1)$$

$$T^k := A^0 + \chi^1 A^1 + \chi^2 A^2 + \cdots + \chi^\kappa A^\kappa \quad (3.2)$$

where GENSLP generates $a^l \in \mathbb{R}^{m_2}$, $A^l \in \mathbb{R}^{m_2 \times n_1}$ for $l = 0, 1, \dots, \kappa$ once the user specifies the nonnegative integer, κ , the ranges for the elements of a^l and A^l , and the 'density' for matrices A^l ; $l = 0, 1, \dots, \kappa$.

In this project four discrete multivariate distributions on (h, T) are generated using the univariate binomial, exponential, Pascal, and Poisson distributions as follows. First, a positive integer vector, $nr \in \mathbb{N}^\kappa$, is selected, where $(nr)_j$ is the number of realizations desired for χ^j ; $j = 1, 2, \dots, \kappa$. Let ξ_{ij} be the i -th realization of χ^j and let $P_{ij} := \Pr(\{\chi^j = \xi_{ij}\})$, $i = 1, 2, \dots, (nr)_j$; $j = 1, 2, \dots, \kappa$. Note that if ξ_{ij} and P_{ij} ; $i = 1, 2, \dots, (nr)_j$; $j = 1, 2, \dots, \kappa$ are available then the following multivariate distribution on (h, T) can be described. Let i_j be any integer such that $1 \leq i_j \leq (nr)_j$; $j = 1, 2, \dots, \kappa$, then using $\chi^j = \xi_{i_j j}$; $j = 1, 2, \dots, \kappa$; in (3.1) and (3.2) a realization (h^k, T^k) of (h, T) may be generated with probability $p^k = \prod_{j=1}^\kappa P_{i_j j}$. There are $\prod_{j=1}^\kappa (nr)_j$ such realizations and it is not too difficult to verify that $\sum_{k=1}^K p^k = 1$ with $K = \prod_{j=1}^\kappa (nr)_j$. Note also that by appropriately selecting nr the value of K can be controlled.

It now remains to specify the way that ξ_{ij} and P_{ij} ; $i = 1, 2, \dots, (nr)_j$; $j = 1, 2, \dots, \kappa$ were generated. Suppose that the discrete distributions for χ^j , $j = 1, 2, \dots, \kappa$ are on a

common range $[\alpha, \beta]$, where $\alpha, \beta \in \mathbb{R}$, $\alpha < \beta$ are given. Then the four distributions are as follows:

(i) Binomial-related distribution on (h, T) :

Suppose that $p \in (0, 1)$ is given and that $(nr)_j > 1$. Then let

$$\xi_{ij} := \frac{(\beta - \alpha)}{((nr)_j - 1)}(i - 1) + \alpha \quad \text{and}$$

$$P_{ij} := \binom{(nr)_j - 1}{i - 1} (1 - p)^{((nr)_j - i)} p^{(i - 1)},$$

for $i = 1, 2, \dots, (nr)_j$ (where $(nr)_j > 1$). This is repeated for $j = 1, 2, \dots, \kappa$.

(ii) Exponential-related distribution on (h, T) :

Suppose that $\lambda \in \mathbb{R}$, $\lambda > 0$, and $\alpha \leq y^1 \leq y^2 \leq \dots \leq y^{((nr)_j - 1)} \leq \beta$, y^i uniform on (α, β) , for $i = 1, 2, \dots, ((nr)_j - 1)$ are given. Then let

$$\xi_{1j} := \frac{\alpha + y^1}{2};$$

$$\xi_{ij} := \frac{y^{i-1} + y^i}{2}, \quad \text{for } i = 2, 3, \dots, ((nr)_j - 1);$$

$$\xi_{(nr)_j j} := \frac{y^{(nr)_j - 1} + \beta}{2};$$

$$P_{1j} := F(y^1);$$

$$P_{ij} := F(y^i) - F(y^{i-1}) \quad \text{for } i = 2, 3, \dots, ((nr)_j - 1);$$

$$P_{(nr)_j j} := 1 - F(y^{(nr)_j - 1})$$

where

$$F(x) := \begin{cases} 1 - e^{-x/\lambda} & x > 0 \\ 0 & \text{otherwise} \end{cases}$$

is the cumulative distribution function of an exponential random variable with parameter λ . This is repeated for $j = 1, 2, \dots, \kappa$.

(iii) Pascal-related distribution on (h, T) :

Suppose that $p \in (0, 1)$ is given and that $(nr)_j > 0$. Then let

$$\xi_{ij} := \frac{(\beta - \alpha)}{((nr)_j - 1)}i + \left(\frac{\alpha(nr)_j - \beta}{(nr)_j - 1} \right) \quad \text{for } i = 1, 2, \dots, ((nr)_j - 1);$$

$$\xi_{(nr)_j j} := \beta;$$

$$P_{ij} := (1 - p)^{i-1} p \quad \text{for } i = 1, 2, \dots, ((nr)_j - 1); \quad \text{and}$$

$$P_{(nr)_j j} := 1 - \sum_{i=1}^{(nr)_j-1} P_{ij}.$$

This is repeated for $j = 1, 2, \dots, \kappa$.

(iv) Poisson-related distribution on (h, T) :

Suppose that $\lambda \in \mathfrak{R}$, $\lambda > 0$ is given and that $(nr)_j > 1$. Then let

$$\xi_{ij} := \frac{(\beta - \alpha)}{((nr)_j - 1)}(i - 1) + \alpha \quad \text{for } i = 1, 2, \dots, ((nr)_j - 1);$$

$$P_{ij} := \frac{e^{-\lambda} \lambda^{i-1}}{(i-1)!} \quad \text{for } i = 1, 2, \dots, ((nr)_j - 1);$$

$$\xi_{(nr)_j j} := \beta; \quad \text{and}$$

$$P_{(nr)_j j} := 1 - \sum_{i=1}^{(nr)_j-1} P_{ij}.$$

This is repeated for $j = 1, 2, \dots, \kappa$.

The description of the test problems of the form (1.1) used in this project is now complete.

The linear programming solver used in this project is the routine SPLP of Hanson and Hiebert. A good description of the code and underlying algorithms are given in [10]. Since it was important that the values and the subgradients of the recourse function and their approximations be accurate, a double precision version of SPLP was created and used for this project. SPLP is designed for portability and uses machine constants supplied by the user in determining such things as whether two quantities agree. Additionally SPLP has a flexible scheme for specifying the problem to be solved. SPLP was chosen for this project because of its portability and flexibility. The numerical tolerances that were important in obtaining the results reported in section 4 were 'TUNE', 'FACTOR', and 'TOLABS'. 'TUNE' and 'FACTOR' are used in determining if a problem is feasible with respect to a relative tolerance, while 'TOLABS' is involved in determining if a problem is feasible with respect to an absolute tolerance if the relative tolerance criteria are not met.

To obtain and update the QR factorizations of matrices in $\mathcal{D}$ as required by algorithm 3.1 of [1], double precision versions of the QRUP routines described in [7] were used.

Most of the routines used in this project contain computations involving linear algebraic operations. These computations were performed using routines from BLAS (Basic Linear Algebra Subroutines) [10] and BLAS2 [8].

The machine used for this project was the Sequent Balance 8000 of the Computer Science Department of Washington State University. This computer has ten processors, but Sequent

suggests that at most nine be used so that one processor is free to handle system work. Consequently, trials of the experiments of this project were done using one through nine processors.

To parallelize the 'DO loops' suggested in (2.15) through (2.20) Sequent's Fortran 'DOACROSS' compiler directive was used. When this directive is used, the loop is rewritten as a subroutine and then each processor executes its share of the subroutine calls. No other portions of the code were parallelized for this project.

To time the calculations of $Q(x)$, $\partial Q(x)$, $Q_D(x)$, $\partial Q_D(x)$, $Q_V(x)$, $\partial Q_V(x)$, and the creation of the collections $\mathcal{D}$ and $\mathcal{V}$, routines suggested by J.J. Dongarra of the Mathematics and Computer Science Division of Argonne National Laboratory were used. These routines, which use the system routine 'times', give relatively consistent timing values regardless of how busy the computer is. (In contrast, the Fortran library routine 'secnds' available on the Sequent gives time values which depend heavily on how busy the computer is.)

For a problem of the form (1.1), the time required to design $\mathcal{D}$ and $\mathcal{V}$ will generally depend on the data defining the particular recourse problem. In addition, the time to calculate $Q(x)$, $\partial Q(x)$, $Q_D(x)$, $\partial Q_D(x)$, $Q_V(x)$, and $\partial Q_V(x)$ will generally depend on the point x as well as the data defining the recourse problem. To minimize the effects of varying x points and varying recourse problem data, the following scheme was adopted. The creation of $\mathcal{D}$ and $\mathcal{V}$ was repeated for each of five problems generated by GENSLP and the average time required to design $\mathcal{D}$ and $\mathcal{V}$ for one problem was computed. For each of the five[†] problems generated by GENSLP, the calculations of $Q(x)$, $\partial Q(x)$, $Q_D(x)$, $\partial Q_D(x)$, $Q_V(x)$, and $\partial Q_V(x)$ were completed for five[†] different x points and an average time for one problem and one x value was computed. In the rest of this paper all reported times are average times computed as described above.

For a given problem size and for a given distribution on (h, T) let $\tau_E(K, j)$ be the time required to calculate a pair of values of Q and ∂Q with K realizations of (h, T) and j processors. Let τ_{DV} be the time to design the collections $\mathcal{D}$ and $\mathcal{V}$. Note that for all practical purposes τ_{DV} does not depend on the number of realizations, K , or the number of processors, j , because the portion of the code that designs $\mathcal{D}$ and $\mathcal{V}$ was not parallelized. Let $\tau_D(K, j)$ and $\tau_V(K, j)$ be the time to calculate a corresponding pair of values of Q_D , ∂Q_D and Q_V , ∂Q_V respectively, with K realizations and j processors. The speedups $S_1(K, j)$, $S_2(K, j)$, and $S_3(K, j)$ suggested in the introduction may now be defined more formally as

[†] The values of five problems and five x points were selected for convenience.

follows:

$$S_1(K, j) := \frac{\tau_E(K, 1)}{\tau_E(K, j)}, \quad (3.3)$$

$$S_2(K, j) := \frac{\tau_{\mathcal{D}}(K, 1)}{\tau_{\mathcal{D}}(K, j)}, \quad \text{and} \quad (3.4)$$

$$S_3(K, j) := \frac{\tau_E(K, 1)}{\tau_{\mathcal{D}\mathcal{V}} + \tau_{\mathcal{D}}(K, j)}. \quad (3.5)$$

Similar speedups may be defined for measuring the effectiveness of obtaining values of $Q_{\mathcal{V}}$ and $\partial Q_{\mathcal{V}}$. However, since $Q_{\mathcal{V}}$ is useful only when $Q_{\mathcal{D}}$ is a strict upper bound on Q , and since Q and $Q_{\mathcal{D}}$ were compared when $\mathcal{D}$ was designed so that $Q_{\mathcal{D}}$ was *exact*, such speedups are not defined here.

4. Results

The following list of the values of the various parameters described in section 3 is given here so that the experiment performed for this project may be repeated:

- (i) Problem size: $m_1 := 10$, $n_1 := 15$, $m_2 := 2$, $n_2 := 5^*$
- (ii) Problem data for GENSLP: $\kappa := 3$; A_{ij} from $U(0, 1)$; c_j from $U(0, 1)$; $|M_{ij}|$ from $U(1, 5)$; q_j from $U(0.5, 10)$; a_j^0 from $U(60, 100)$; a_j^l from $U(15/\kappa, 25/\kappa)$, for $l = 1, 2, \dots, \kappa$; A_{ij}^0 from $U(12/n_1, 20/n_1)$; A_{ij}^l from $U(12/(n_1\kappa))$, for $l = 1, 2, \dots, \kappa$; all matrices 100% dense. Here $U(\alpha, \beta)$ represents a uniform distribution on (α, β) and subscripts i and j are in the appropriate ranges suggested in (1.1).
- (iii) Number of realizations of (h, T) : For $K := 50, 500, 5000$, and 10000 , $(nr)_j \in \mathbb{R}^3$ was given by $(5, 5, 2)^T$, $(10, 10, 5)^T$, $(25, 25, 8)^T$, and $(25, 25, 16)^T$ respectively.
- (iv) The distributions (as described in section 3) with parameters as follows: $\alpha = 1.0$ and $\beta = 2.0$ for all distributions, $p = 0.4$ for the binomial-related distribution, $\lambda = 1.0$ for the exponential-related distribution, $p = 0.4$ for the Pascal-related distribution, and $\lambda = 3.0$ for the Poisson-related distribution.
- (v) The SPLP parameters: $TUNE := 1.0$, $FACTOR := 1.0$, and $TOLABS := 100\varepsilon^\dagger$ where $\varepsilon = 0.2220446049250313 \times 10^{-15}$ is the largest relative spacing for double precision floating point numbers on the Sequent.
- (vi) Input to algorithm 3.1: $D_0^1 := I$ and $u^j := M_{\cdot j}$, $j = 1, 2, \dots, n_2$. (This set of directions was chosen because when one uses the columns of the matrix M as the set of directions with any initial matrix, D_0^1 , the corresponding function $\psi_{\mathcal{D}}$ is guaranteed to agree exactly with the function ψ for all directions. This result is proved in [2, Theorem 4.1].)

The time (in seconds) required to calculate a pair of values Q and ∂Q ; to design the collections $\mathcal{D}$ and $\mathcal{V}$; to calculate a pair of values $Q_{\mathcal{D}}$ and $\partial Q_{\mathcal{D}}$; and to calculate a pair of values $Q_{\mathcal{V}}$ and $\partial Q_{\mathcal{V}}$; appear (in that order) in tables (4.1) through (4.4). The speedups $S_1(K, j)$, $S_2(K, j)$, and $S_3(K, j)$ (in that order) appear in tables (4.5) through (4.8).

* It would have been useful to time the calculations of Q and $Q_{\mathcal{D}}$ for various problem sizes as specified by m_1 , n_1 , m_2 and n_2 , but due to problems encountered in using the linear program solver, SPLP, and a shortage of time, it was only possible to complete trials on problems with $m_1 = 10$, $n_1 = 15$, $m_2 = 2$ and $n_2 = 5$.

† The only exception to $TOLABS := 100\varepsilon$ was for $K := 10000$ with the exponential-related distribution where $TOLABS := 10000\varepsilon$ was used.

$K \backslash j$	1	2	3	4	5	6	7	8	9
50	8.7	4.5	3.1	2.5	2.1	2.0	1.7	1.6	1.7
	4.6	4.6	4.6	4.6	4.6	4.6	4.6	4.6	4.6
	1.5	0.8	0.6	0.5	0.5	0.5	0.7	0.5	0.8
	0.7	0.4	0.3	0.3	0.4	0.4	0.7	0.5	0.7
500	87.2	44.1	30.0	22.9	18.7	15.5	13.4	11.8	10.9
	4.6	4.6	4.6	4.6	4.6	4.6	4.6	4.6	4.5
	14.9	7.8	5.3	3.8	3.0	3.1	2.8	2.6	2.9
	6.7	3.6	2.6	1.7	1.4	1.5	1.7	1.7	1.4
5000	872.1	441.8	295.7	224.6	188.0	154.7	135.9	124.7	99.4
	4.6	4.6	4.6	4.6	4.6	4.6	4.6	4.6	4.6
	149.3	76.6	53.0	42.9	34.9	28.3	25.8	23.9	23.0
	67.7	34.8	24.8	21.2	19.4	14.3	14.2	13.8	14.5
10000	1753.4	882.1	585.1	445.2	374.2	290.3	266.5	218.6	194.6
	4.6	4.6	4.6	4.6	4.6	4.6	4.6	4.6	4.6
	300.7	153.8	103.8	83.5	72.4	63.7	50.8	45.7	49.7
	135.5	70.9	48.6	41.8	39.4	36.3	27.3	26.2	30.7

Table 4.1. Timing values for the binomial-related distribution

$K \backslash j$	1	2	3	4	5	6	7	8	9
50	8.2	4.2	2.9	2.4	1.9	1.9	1.6	1.5	1.4
	4.9	4.9	4.9	5.0	4.9	5.0	5.0	4.9	5.0
	1.5	0.8	0.6	0.6	0.5	0.7	0.7	0.8	0.7
	0.7	0.4	0.3	0.4	0.4	0.7	0.6	0.8	0.6
500	79.3	40.3	27.3	20.9	16.8	14.0	12.1	11.0	9.9
	4.6	4.6	4.6	4.6	4.6	4.6	4.6	4.6	4.6
	15.1	7.8	5.3	4.5	3.1	3.0	2.9	2.7	2.6
	6.8	3.6	2.5	2.2	1.4	1.6	1.5	1.5	1.6
5000	776.5	389.6	261.8	201.0	159.4	133.8	116.7	105.6	88.3
	4.4	4.4	4.4	4.4	4.4	4.4	4.4	4.4	4.5
	152.3	78.0	53.5	42.2	36.6	28.6	26.1	24.4	23.8
	68.4	35.6	25.4	21.7	19.4	14.1	14.0	13.5	13.4
10000	1640.2	827.2	553.9	417.1	333.2	275.8	236.5	208.3	187.1
	4.1	4.0	4.1	4.0	4.0	4.0	4.0	4.0	4.0
	289.6	148.5	101.1	80.4	70.1	61.2	49.7	45.3	32.6
	135.0	70.2	48.4	42.2	39.1	35.7	27.5	25.6	22.8

Table 4.2. Timing values for the Exponential-related distribution

$K \backslash j$	1	2	3	4	5	6	7	8	9
50	8.7	4.5	3.1	2.3	2.0	1.8	1.6	1.5	1.4
	4.6	4.6	4.6	4.6	4.6	4.6	4.6	4.6	4.5
	1.5	0.8	0.5	0.4	0.3	0.4	0.5	0.6	0.4
	0.7	0.3	0.2	0.2	0.1	0.3	0.3	0.3	0.4
500	88.1	44.5	29.3	22.7	19.2	18.2	12.7	11.8	11.1
	4.6	4.6	4.6	4.6	4.5	4.5	4.6	4.6	4.6
	15.2	7.6	5.0	3.8	3.0	3.0	2.7	2.7	2.5
	6.8	3.5	2.3	1.7	1.4	1.4	1.5	1.6	1.5
5000	870.6	438.8	294.3	221.8	179.9	150.6	126.6	115.0	104.6
	4.5	4.6	4.6	4.6	4.6	4.6	4.6	4.6	4.6
	149.8	75.5	50.4	38.3	32.7	28.7	29.9	28.0	17.0
	67.3	34.0	23.1	17.5	15.7	14.4	18.1	16.7	11.4
10000	1751.1	880.8	591.7	446.2	359.9	292.2	264.8	229.3	206.0
	4.6	4.6	4.6	4.6	4.6	4.6	4.5	4.6	4.6
	300.8	153.6	102.2	83.6	71.8	62.3	51.8	47.4	49.9
	135.2	70.5	46.9	41.7	38.0	34.4	28.4	27.0	31.1

Table 4.3. Timing values for the Pascal-related distribution

$K \backslash j$	1	2	3	4	5	6	7	8	9
50	8.7	4.4	3.3	2.3	1.9	1.7	1.6	1.3	1.4
	4.6	4.6	4.6	4.6	4.6	4.6	4.6	4.6	4.5
	1.5	0.8	0.6	0.4	0.3	0.4	0.4	0.5	0.5
	0.7	0.3	0.2	0.2	0.1	0.3	0.2	0.4	0.5
500	87.5	43.9	30.0	22.9	19.1	15.4	13.2	11.9	11.2
	4.6	4.6	4.6	4.6	4.6	4.6	4.6	4.6	4.6
	14.9	7.5	5.1	3.8	3.1	2.7	2.8	2.7	2.8
	6.7	3.4	2.3	1.7	1.4	1.3	1.4	1.5	1.3
5000	873.1	439.4	294.6	221.8	180.0	149.3	126.0	118.4	104.6
	4.6	4.6	4.6	4.6	4.6	4.6	4.6	4.6	4.6
	149.7	76.6	51.6	38.1	32.2	28.4	30.6	28.1	17.0
	67.3	34.1	23.7	17.2	15.6	14.8	17.9	16.4	11.5
10000	1743.7	890.5	587.8	444.5	355.6	291.9	264.9	240.9	207.5
	4.6	4.6	4.6	4.6	4.6	4.6	4.6	4.6	4.6
	300.5	155.9	102.4	85.2	72.2	66.9	51.9	48.4	51.0
	135.6	71.0	46.2	42.3	38.5	38.0	28.3	26.9	31.0

Table 4.4. Timing values for the Poisson-related distribution

$K \backslash j$	1	2	3	4	5	6	7	8	9
50	1.0	2.0	2.8	3.5	4.1	4.5	5.2	5.4	5.1
	1.0	1.1	1.2	1.2	1.2	1.2	1.2	1.2	1.1
	1.4	1.6	1.7	1.7	1.7	1.7	1.7	1.7	1.6
500	1.0	2.0	2.9	3.8	4.7	5.6	6.5	7.4	8.0
	1.0	1.6	2.0	2.3	2.6	2.5	2.7	2.7	2.6
	4.5	7.1	8.8	10.4	11.4	11.4	11.8	12.1	11.8
5000	1.0	2.0	2.9	3.9	4.6	5.6	6.4	7.0	8.8
	1.0	1.9	2.7	3.2	3.9	4.7	5.1	5.4	5.6
	5.7	10.7	15.2	18.3	22.1	26.6	28.6	30.6	31.6
10000	1.0	2.0	3.0	3.9	4.7	6.0	6.6	8.0	9.0
	1.0	1.9	2.8	3.5	4.0	4.5	5.5	6.1	5.6
	5.7	11.1	16.2	19.9	22.8	25.7	31.6	34.9	32.3

Table 4.5. Speedup values for the binomial-related distribution

$K \backslash j$	1	2	3	4	5	6	7	8	9
50	1.0	2.0	2.8	3.4	4.2	4.2	5.1	5.5	5.7
	1.0	1.1	1.2	1.2	1.2	1.1	1.2	1.1	1.2
	1.3	1.4	1.5	1.5	1.5	1.4	1.5	1.4	1.5
500	1.0	2.0	2.9	3.8	4.7	5.7	6.5	7.2	8.0
	1.0	1.6	2.0	2.2	2.5	2.6	2.6	2.7	2.7
	4.0	6.4	8.0	8.7	10.2	10.5	10.6	10.9	11.0
5000	1.0	2.0	3.0	3.9	4.9	5.8	6.7	7.4	8.8
	1.0	1.9	2.7	3.4	3.8	4.7	5.1	5.4	5.5
	5.0	9.4	13.4	16.6	18.9	23.5	25.4	27.0	27.5
10000	1.0	2.0	3.0	3.9	4.9	5.9	6.9	7.9	8.8
	1.0	1.9	2.8	3.5	4.0	4.5	5.5	6.0	8.0
	5.6	10.8	15.6	19.4	22.1	25.1	30.5	33.2	44.8

Table 4.6. Speedup values for the exponential-related distribution

$K \backslash j$	1	2	3	4	5	6	7	8	9
50	1.0	1.9	2.8	3.8	4.5	5.0	5.3	5.7	6.4
	1.0	1.1	1.2	1.2	1.2	1.2	1.2	1.2	1.2
	1.4	1.6	1.7	1.7	1.8	1.8	1.7	1.7	1.8
500	1.0	2.0	3.0	3.9	4.6	4.8	7.0	7.5	7.9
	1.0	1.6	2.1	2.4	2.6	2.6	2.7	2.7	2.8
	4.4	7.2	9.2	10.6	11.6	11.7	12.2	12.2	12.4
5000	1.0	2.0	3.0	3.9	4.8	5.8	6.9	7.6	8.3
	1.0	1.9	2.8	3.6	4.1	4.6	4.5	4.7	7.2
	5.6	10.9	15.8	20.3	23.4	26.2	25.3	26.7	40.5
10000	1.0	2.0	3.0	3.9	4.9	6.0	6.6	7.6	8.5
	1.0	1.9	2.9	3.5	4.0	4.6	5.4	5.9	5.6
	5.7	11.1	16.4	19.9	22.9	26.2	31.1	33.7	32.2

Table 4.7. Speedup values for the Pascal-related distribution

$K \backslash j$	1	2	3	4	5	6	7	8	9
50	1.0	2.0	2.6	3.8	4.5	5.0	5.5	6.8	6.1
	1.0	1.1	1.2	1.2	1.2	1.2	1.2	1.2	1.2
	1.4	1.6	1.7	1.7	1.8	1.7	1.8	1.7	1.7
500	1.0	2.0	2.9	3.8	4.6	5.7	6.6	7.4	7.8
	1.0	1.6	2.0	2.3	2.5	2.7	2.7	2.7	2.6
	4.5	7.2	9.1	10.4	11.3	12.0	12.0	12.1	11.8
5000	1.0	2.0	3.0	3.9	4.9	5.8	6.9	7.4	8.4
	1.0	1.9	2.7	3.6	4.2	4.7	4.4	4.7	7.2
	5.7	10.8	15.5	20.5	23.7	26.5	24.8	26.7	40.5
10000	1.0	2.0	3.0	3.9	4.9	6.0	6.6	7.2	8.4
	1.0	1.9	2.9	3.4	4.0	4.3	5.4	5.8	5.5
	5.7	10.9	16.3	19.4	22.7	24.4	30.8	32.9	31.4

Table 4.8. Speedup values for the Poisson-related distribution

The speedup $S_3(K, j)$ is greater than one for all the trials performed for this project. This suggests that the use of Q_D to approximate Q may be profitable. For a given number of processors and a given distribution, the speedup $S_3(K, j)$ tends to increase as K increases up to a point and then stay relatively constant. This may be explained by the fact that τ_{DV} , which is approximately constant, contributes significantly to the calculation of $S_3(K, j)$ for small K , but for large enough K , τ_{DV} will be insignificant compared to τ_E and τ_D which both become very large and whose ratio remains relatively constant. The speedup $S_1(K, j)$ is larger than the speedup $S_2(K, j)$ for all distributions, all K , and all j (except, of course, $j = 1$) used in this project. This suggests that it may be worthwhile to investigate means of more effectively parallelizing the calculation of $Q_D(x)$ and $\partial Q_D(x)$; a more complex scheduling scheme than that of the ordinary "DOACROSS" compiler directive might help. The agreement between values of $Q(x)$ and values of $Q_D(x)$ was excellent in the trials of the experiments performed—agreements to 14 or 15 digits appeared to be typical.

5. Concluding Remarks

The trials of this project suggest that using Q_D and ∂Q_D as approximations to Q and ∂Q may be worthwhile, but it still should be shown that the calculation of $Q_D(x)$ and $\partial Q_D(x)$ is worthwhile for problems with larger m_2 and n_2 . One would expect that the speedup, $S_3(K, j)$, would be at least as good for larger problems because the difficulty in solving linear programs increases more with an increase in problem size than the difficulty in solving systems of linear equations. It is also important that a comparison be made of the time required by an algorithm to solve problem (1.1) using Q_D with the time required to solve (1.1) using Q .

Appendix 1

The following notational conventions are used in this paper:

$\mathbb{N} :=$ The set of nonnegative integers.

$\mathbb{N}^n :=$ the set of nonnegative integer n -tuples.

All vectors are column vectors.

For $x \in \mathbb{R}^n$, x_i denotes the i -th component of x , $i = 1, 2, \dots, n$.

For $A \in \mathbb{R}^{m \times n}$, A_i denotes the i -th row of A , $i = 1, 2, \dots, n$, while A_j denotes the j -th column of A , $j = 1, 2, \dots, m$.

$x^k \in \mathbb{R}^n$ is the k -th member of a sequence of vectors in $\mathbb{R}^n$; components of x^k are denoted by x_i^k for $i = 1, 2, \dots, n$.

$A^k \in \mathbb{R}^{m \times n}$ is the k -th member of a sequence of matrices in $\mathbb{R}^{m \times n}$; columns, rows, and elements of A^k are denoted by A_j^k , A_i^k , and A_{ij}^k respectively for $i = 1, 2, \dots, m$, $j = 1, 2, \dots, n$.

If $e_j \in \mathbb{R}^n$ then e_j is the j -th unit vector in $\mathbb{R}^n$ for $1 \leq j \leq n$.

If $e \in \mathbb{R}^n$ then e is the n -vector all of whose components are 1's.

For $\lambda \in \mathbb{R}$, $\text{sgn}(\lambda) := \begin{cases} 1 & \text{if } \lambda \geq 0 \\ -1 & \text{if } \lambda < 0 \end{cases}$.

For $x \in \mathbb{R}^n$, $\text{sgn}(x) \in \mathbb{N}^n$ is defined by $e_i^T \text{sgn}(x) := \text{sgn}(e_i^T x)$, $i = 1, 2, \dots, n$.

For $x \in \mathbb{R}^n$, $\text{diag}(x)$ denotes the diagonal matrix in $\mathbb{R}^{n \times n}$ whose ii -th element is x_i , $i = 1, 2, \dots, n$.

For $x \in \mathbb{R}^n$, x_+ , x_z , and $x_- \in \mathbb{R}^n$ are defined by:

$$(x_+)_i := \begin{cases} x_i & \text{if } x_i \geq 0 \\ 0 & \text{otherwise} \end{cases}$$

$$(x_z)_i := \begin{cases} 1 & \text{if } x_i = 0 \\ 0 & \text{otherwise} \end{cases}$$

$$(x_-)_i := \begin{cases} x_i & \text{if } x_i < 0 \\ 0 & \text{otherwise} \end{cases}$$

for $i = 1, 2, \dots, n$.

For $A \in \mathbb{R}^{m \times n}$, $\text{pos } A := \{w : Ay = w, y \in \mathbb{R}^n, y \geq 0\}$.

For $A \in \mathbb{R}^{m \times n}$, $\text{co } A := \{w : Ay = w, y \in \mathbb{R}^n, y \geq 0, e^t y = 1\}$.

For function f and $\tau \in \mathbb{R}$, $\text{lev}_\tau f := \{w : f(w) \leq \tau\}$

References

1. Ariyawansa, K.A., D.C. Sorensen and R.J.-B. Wets (1987), "Parallel schemes to approximate values and subgradients of the recourse function in certain stochastic programs", in preparation.
2. Ariyawansa, K.A., D.C. Sorensen and R. J.-B. Wets (1987), "On the convexity of an upper bounding approximant to the recourse function in a class of stochastic programs", in preparation.
3. Birge, J.R. and R. J.-B. Wets (1986), "Designing approximation schemes for stochastic optimization problems, in particular for stochastic programs with recourse", *Mathematical Programming Study*, Vol. 27, pp. 54-102.
4. Birge, J.R. and R. J.-B. Wets (1986), "On-line solution of linear programs using sublinear functions", Technical Report 86-25, Department of Industrial and Operations Engineering, The University of Michigan, Ann Arbor, Michigan.
5. Birge, J.R. and R. J.-B. Wets (1986), "Sublinear upper bounds for stochastic programs with recourse", to appear in *Mathematical Programming*.
6. King, A.J., "Stochastic Programming Problems: Examples from the Literature", in: Y. Ermoliev and R. J.-B. Wets, eds., *Numerical Methods in Stochastic Programming*, Springer, Berlin, to appear.
7. Buckley, A. (1981), "QRUP: A set of Fortran routines for updating QR factorizations", *ACM Transactions on Mathematical Software*, Vol. 7, pp. 548-549.
8. Dongarra, J.J., J.J. Du Croz, S. Hammarling, and R.J. Hanson, "An Extended Set of Fortran Basic Linear Algebra Subprograms", Algorithm No. 656, *ACM Transactions on Mathematical Software*, Vol. 14, No. 1, March 1988, pp. 18-32.
9. Hanson, R.J., K.L. Hiebert, "A Sparse Linear Programming Subprogram", Report SAND81-0297, Sandia National Laboratories, Albuquerque, New Mexico and Livermore, California.
10. Hanson, R.J., D.R. Kincaid, F.T. Krogh, and C.L. Lawson, "Basic Linear Algebra Subprograms for Fortran Usage", Algorithm No. 539, *ACM Transactions on Mathematical Software*, Vol. 5, No. 3, Sept. 1979, pp. 308-323.
11. Keller, E., "GENSLP: A program for generating input for stochastic linear programs with complete fixed recourse", Manuscript, Institut für Operations Research der Universität Zürich, Zürich, CH-8006.
12. Schrage, L., "A More Portable Fortran Random Number Generator", *ACM Transactions on Mathematical Software*, Vol. 5, No. 2, June 1979, pp. 132-138.
13. Wets, R. J.-B. (1985), "On parallel processors design for stochastic programs", Working Paper WP 85-67, International Institute for Applied Systems Analysis, A-2361 Laxenburg, Austria.
14. Wets, R. J.-B. (1966), "Programming under uncertainty: The equivalent convex program", *J. SIAM Appl. Math.*, Vol. 14, No. 1. pp. 89-105.

Appendix 4

This appendix is part of the final report of DOE Grant No. DE-FG06-87ER25045.

Performance of a Benchmark Parallel Implementation of the Van Slyke and Wets Algorithm for Two-Stage Stochastic Programs on the Sequent/Balance

K. A. Ariyawansa and D. D. Hudson
Department of Pure and Applied Mathematics
Washington State University
Pullman, WA 99164-2930

Abstract: We describe a benchmark parallel version of the Van Slyke and Wets (1969) algorithm for two-stage stochastic programs and an implementation of that algorithm on the Sequent/Balance. We also report results of a numerical experiment using random test problems and our implementation. These performance results, to the best of our knowledge, are the first available for the Van Slyke and Wets (1969) algorithm on a parallel processor. They indicate that the benchmark implementation parallelizes well, and that even with the use of parallel processing, problems with random variables having large numbers of realizations can take prohibitively large amounts of computation for solution. Thus, they demonstrate the need for exploiting *both* parallelization *and* approximation for the solution of stochastic programs.

Key words: two-stage stochastic programs, decomposition algorithms, Van Slyke and Wets algorithm, parallel processing

1. Introduction

Two-stage stochastic programs with recourse¹ are a class of optimization problems with applications in a wide variety of areas in operations research. See for example Dempster (1980), Ermoliev and Wets (1988). The functions that describe these problems involve expectations with respect to multivariate random variables. The random variable in a problem instance that is actually solved usually has a discrete distribution with a finite number of realizations. This is either because the random variable has such a distribution naturally, or if not, prior to solution, the actual distribution is usually replaced by a sequence of such distributions converging to it in some sense. The number of realizations of these random variables in meaningful problems is 'large' (say, more than 5000).

Research supported in part by DOE Grant DE-FG-06-87ER25045

¹ A precise description will be provided later in the paper.

Many algorithms for two-stage stochastic programs have been proposed. The algorithms described by Kall (1979) are good representatives based on large scale linear programming (lp) techniques. The algorithm proposed by Van Slyke and Wets (1969) and extended by Wets (1982) is a good representative of those based on decomposition—the other major approach used to derive algorithms for stochastic programs.

It is well known (see Wets (1982,85)) that when the number of realizations of the random variable describing the problem is large, even for moderate values for dimensions that determine the size of the problem, these algorithms take prohibitively large amounts of computation. Due to this, computational experience with algorithms for two-stage programs is limited to problem instances with small numbers of realizations.

Dantzig (1985) and Wets (1985) were the first to make the observation that algorithms for stochastic programs based on decomposition can take advantage of computers with parallel processing capabilities. Wets (1985) made the further point that even with the use of parallel processors, algorithms that attempt to solve stochastic programs *exactly* may take prohibitively large amounts of computation, when the number of realizations of the random variable associated with the problem is large. (See also the timing results presented in this paper.) Wets (1985) then sketched out the rudiments of an approximating scheme, in which the given two-stage stochastic program is replaced by an approximate one constructed to have the following properties: decomposition algorithms would solve the approximate problem with considerably smaller amount of computation than the given problem; and the possibility of parallelization of decomposition algorithms for the approximate problem is retained or even enhanced relative to the possible parallelization of decomposition algorithms for the given problem. The approach, then, is to sacrifice hopefully a little in accuracy to gain speed through approximation and parallelization.

Following the work of Wets (1985), Ariyawansa, Sorensen and Wets (1987) have described implementable algorithms that generate the approximate problem in the above sense for a given two-stage stochastic program. The first author of this paper is involved in a research project to design decomposition algorithms for the approximate problem generated by the algorithms of Ariyawansa, Sorensen and Wets (1987) and to implement such algorithms as well as the algorithms of Ariyawansa, Sorensen and Wets (1987) on parallel processors. One of the aims of that project is to carefully assess the performance of these approximation procedures. Since these procedures achieve speed through approximation and parallelization at the expense of accuracy, the natural basis for evaluation would be relative to the performance of a benchmark implementation of a parallelizable algorithm for the exact problem. The natural criteria for comparison would be measures of parallelization for the approximate

and exact procedures, speed of the approximate procedure relative to the speed of the exact procedure, and measures of accuracy of solution returned by the approximate procedure relative to the solution returned by the exact procedure.

As mentioned above, published computational results for algorithms for stochastic programs are limited. Moreover, the little published computational results available do not deal with problems with random variables having large numbers of realizations and do not deal with implementations on parallel processors. Therefore, in order to meet the need mentioned in the previous paragraph we have developed a parallel version of the Van Slyke and Wets (1969) algorithm (in the form given in Wets (1982)²). We have also developed an implementation of this parallel version on the Sequent/Balance.

After we developed this implementation for use as a benchmark to evaluate the performance of the approximation procedures mentioned above, it occurred to us that results of a carefully designed, repeatable computational experiment using our implementation would be of interest in its own right. The purpose of this paper, then, is to describe our benchmark parallel version of the Van Slyke and Wets (1969) algorithm, its implementation on the Sequent/Balance, and the results of a computational experiment using this implementation. Our numerical results, to the best of our knowledge, are the first available on the performance of the Van Slyke and Wets (1969) algorithm on a parallel processor.

² Henceforth in this paper we only quote the paper by Van Slyke and Wets (1969) in connection with this algorithm.

2. The Van Slyke and Wets (1969) Algorithm for the Two-Stage Stochastic Program with Recourse

The two-stage stochastic program with recourse and with a discretely distributed random variable with a finite number of realizations, is the following³:

Find $x \in \mathbb{R}^{n_1}$ such that

$z(x) := c^T x + Q(x)$ is minimized, and

$Ax = b, x \geq 0$,

where

$Q(x) := E[Q(x, \mathbf{h}, \mathbf{T})] = \sum_{k=1}^K p^k Q(x, h^k, T^k),$

$Q(x, \mathbf{h}, \mathbf{T}) := \inf_{y \in \mathbb{R}^{n_2}} \{q^T y : My = \mathbf{h} - \mathbf{T}x, y \geq 0\},$

$A \in \mathbb{R}^{m_1 \times n_1}$, $b \in \mathbb{R}^{m_1}$, $c \in \mathbb{R}^{n_1}$, $q \in \mathbb{R}^{n_2}$, $M \in \mathbb{R}^{m_2 \times n_2}$ are deterministic and given and $\mathbf{h} \in \mathbb{R}^{m_2}$, $\mathbf{T} \in \mathbb{R}^{m_2 \times n_1}$ are random with $(\mathbf{h}, \mathbf{T})$ having the probability distribution $F := \{((h^k, T^k), p^k), k = 1, 2, \dots, K\}.$

(1)

We refer the reader to survey papers by Wets (1982, 88) for general discussions on theory, applications and algorithms pertinent to (1) and its extensions. We mention however, that (1) arises in decision making contexts that fit into the following general format. Suppose that *at present* a decision x has to be chosen from the set $\{x : Ax = b, x \geq 0\}$ and that *at a future point in time* a realization (h^k, T^k) of $(\mathbf{h}, \mathbf{T})$ becomes available. At that point in time, the decision maker is allowed to take a corrective action y , if necessary, to allow for the difference $h^k - T^k x$. The recourse function Q in (1) models this correction process, and problem (1) is a well-posed version of this two-stage decision making problem. The solution of (1) yields a decision x^* that the decision maker can take *at present* and it hedges against the *presently* unknown realizations of $(\mathbf{h}, \mathbf{T})$.

In the rest of the paper we shall make the following assumptions regarding problem (1).

(A1) The set $\{x : Ax = b, x \geq 0\}$ is nonempty and bounded.

(A2) The set $\{w : Mw = w, w \geq 0\} = \mathbb{R}^{m_2}.$

(A3) The set $\{v : M^T v \leq q\}$ is nonempty.

It is easy to verify that when (A1), (A2) and (A3) hold, problem (1) has a minimizer x^* with a *finite* minimum $z(x^*) = c^T x^* + Q(x^*)$. If (A1), (A2) and (A3) are not satisfied, then it is possible for problem (1) to be infeasible or to be unbounded below. Therefore, we have

³ We use boldface letters to denote random variables and the corresponding normal-face letters to denote their realizations.

assumed that (A1), (A2) and (A3) hold in developing our benchmark parallel implementation of the Van Slyke and Wets (1969) algorithm for (1). We hasten to add that it is possible to add necessary steps to our implementation so that problems which do not satisfy (A1), (A2) or (A3) can be handled, and infeasibility or unboundedness can be detected.

We now state the Van Slyke and Wets (1969) algorithm for problem (1), when assumptions (A1), (A2) and (A3) are satisfied.

Algorithm 2.1: (Van Slyke and Wets (1969))

input:

$m_1; n_1; m_2; n_2; K; A; c; b; M; q; F := ((h^k, T^k), p^k), k = 1, 2, \dots, K).$

output:

$x; z.$

begin

/* begin initializations */

$t := 0;$

{solve the lp

find $x \geq 0$ such that

$c^T x$ is minimized, and

$Ax = b$ };

call *optimality_cut* ($m_2, n_2, K, M, q, F, x, E, e, Q$);

$t := t + 1;$

$E^t := E; e^t := e;$

call *lowerbound* ($m_1, n_1, t, A, b, (E^l, e^l, l = 1, 2, \dots, t), x, \theta, z$);

call *optimality_cut* ($m_2, n_2, K, M, q, F, x, E, e, Q$);

/* end initializations */

while $\theta < Q$ **do**

$t := t + 1;$

$E^t := E; e^t := e;$

call *lowerbound* ($m_1, n_1, t, A, b, (E^l, e^l, l = 1, 2, \dots, t), x, \theta, z$);

call *optimality_cut* ($m_2, n_2, K, M, q, F, x, E, e, Q$);

end while;

end.

optimality_cut:

input:

$m_2; n_2; K; M; q; F; x.$

output:

$E; e; Q.$

begin

$E := 0; e := 0;$

for $k := 1$ **to** K **do**

$w := h^k - T^k x;$

{solve the lp

find $y \geq 0$ such that

$q^T y$ is minimized, and

$My = w$

to obtain dual maximizer v };

$E := E + p^K (T^K)^T v;$

$e := e + p^k (h^k)^T v;$

end do;

$Q := e - E^T x;$

return;

end.

lowerbound:

input:

$m_1; n_1; t; A; b; E^l, e^l, l = 1, 2, \dots, t.$

output:

$x; \theta; z.$

begin

{solve the lp

find $x \geq 0, \theta \in \Re$ such that

```

     $c^T x + \theta$  is minimized, and
     $Ax = b$ ,
     $(E^l)^T x + \theta \geq e^l, l = 1, 2, \dots, t$ ;
     $z := c^T x + \theta$ ;
    return;
end.

```

In exact arithmetic, Algorithm 2.1 terminates after a finite number of iterations of the 'while $\theta < Q$ do' loop with $\theta = Q$. We refer the reader to Van Slyke and Wets (1969) for details of Algorithm 2.1 including this finite termination property. We mention however, that Algorithm 2.1 utilizes the fact Q has the representation

$$Q(x) = \max\{e^l - (E^l)^T x : l = 1, 2, \dots, T\}, \quad x \in \{x : Ax = b, x \geq 0\}$$

so that problem (1) could be written as

```

find  $x \in \mathbb{R}^{n_1}, \theta \in \mathbb{R}$  such that
 $c^T x + \theta$  is minimized, and
 $Ax = b$ ,
 $(E^l)^T x + \theta \geq e^l, l = 1, 2, \dots, T$ ,
 $x \geq 0$ 

```

for appropriate $E^l \in \mathbb{R}^{n_1}, e^l \in \mathbb{R}, l = 1, 2, \dots, T$ where T is finite. The routine *optimality-cut* generates the 'cuts' $(E^l)^T x + \theta \geq e^l$ one at a time. The routine *lowerbound* therefore solves increasingly less relaxed (i.e. using not all the constraints, $(E^l)^T x + \theta \geq e^l, l = 1, 2, \dots, T$) versions of (1). Consequently, x, θ returned by routine *lowerbound* are such that $\theta \leq Q(x)$ and therefore, if $\theta = Q(x)$ then x must be a minimizer of problem (1). Of course, the hope is that we shall satisfy this condition well before generating all T cuts, since T is usually large.

3. A Parallel Benchmark Version of Algorithm 2.1

We begin by noting that almost all of the computational work in Algorithm 2.1 is done in routines *lowerbound* and *optimality-cut*. The work in routine *optimality-cut* essentially involves solution of the K lp's in its 'do loop'. The work in routine *lowerbound* essentially involves the solution of a single lp. In general therefore, when the number of realizations

K is large, the computational work in Algorithm 2.1 is dominated by the computational work in routine *optimality_cut*. Routine *optimality_cut* of course is obviously parallelizable, and consequently, when K is large, we may expect to observe good parallel performance of Algorithm 2.1, simply by parallelizing routine *optimality_cut*. The purpose of this section is to state a parallel version of Algorithm 2.1, keeping in mind the need of a benchmark mentioned in Section 1.

Algorithm 2.1 needs an lp solver and certain features of the parallel version of Algorithm 2.1 that we describe here are motivated by the lp solver DPLO that we have decided to use. Before we state the parallel version of Algorithm 2.1, we therefore briefly mention certain pertinent features of DPLO. DPLO is a double precision Fortran subroutine for solving linear programs using the Revised Simplex Method. It is the current version of the subroutine DPLP (single precision versions of DPLO and DPLP are named SPLO and SPLP respectively), written by R. J. Hanson and K. L. Hiebert. The reader is referred to the paper by Hanson and Hiebert (1981) for details on DPLP and SPLP.

The ability to exploit sparsity in the constraint matrix of the lp, the availability of an optional restart procedure (for solving an lp starting with the information at optimality of a related lp solved previously), robustness, portability and availability of software in public domain were the main reasons for our choice of DPLO. Ability to exploit sparsity is important for being able to solve problems with large dimensions. DPLO uses the routines of Reid (1976) for handling sparse linear programming bases through sparse LU factors. The restart procedure is useful for handling the lp's solved by routines *lowerbound* and *optimality_cut*.

DPLO maintains the constraint matrix of a given lp, and the LU factors of the current lp basis using certain sparse data structures. Suppose that a user wishes to solve an lp, and after its solution wishes to solve another lp related to the previous one (of same dimensions). He can then use DPLO as follows. He can solve the first lp from scratch, and then use an option of DPLO that allows him to write the sparse data structure representing the constraint matrix and the column indices of the optimal basis *on to a disk file*. He can then call DPLO to solve the second lp, using another option of DPLO that would allow him to read the information saved on the disk file and to start the computations with the basis specified by the column indices saved.

We wish to emphasize three points regarding the above solution process. First, DPLO allows the possibility of the two lp's having different constraint matrices, and therefore, as part of the solution of the second lp, the sparse LU factors of the starting basis is *formed*. Of course, if the constraint matrices of the two lp's were the same, (as in the context of the lp's in routine *optimality_cut*) then this computation is unnecessary, as the LU factors of the

optimal basis of the first lp is available at the end of its solution. Secondly, the basis specified by the column indices saved may not be feasible for the second lp. DPLO uses a penalty method (see pp. 12, 13 of Hanson and Hiebert (1981)) to handle this situation. Hanson and Hiebert (1981) (see p. 13) state that this procedure "almost always will reach an optimal solution with fewer iterations", relative to a solution from scratch. Thirdly, DPLO performs disk writes and reads in connection with this restart procedure, partly to save high speed memory, and partly to allow the user to use information at optimality of the first lp, when the second lp is solved in a *different* program. If one solves the two lp's in the same program, and high speed memory limitations are not a problem, then the expensive disk writes and reads are unnecessary.

We now describe our benchmark parallel version of Algorithm 2.1. Consider the routine *optimality-cut* first. Suppose that we have $nprocs$ processors available. We parallelize the routine *optimality-cut* by dividing the K lp's equally among the $nprocs$ processors for solution in parallel. We say that a processor would handle a 'bunch' of lp's. The lp corresponding to the smallest k (loop index) value in a bunch is solved *from scratch* using DPLO. The remaining lp's in the bunch are solved using the *restart procedure* of DPLO, *modified as follows*. Since disk reads and writes are time consuming we introduced a new option to DPLO which when turned on will use data in high speed memory to retrieve information about the previously solved lp when the restart procedure is invoked, rather than reading that information from disk. Since the lp's in routine *optimality-cut* have the same constraint matrix, when using the restart procedure to solve an lp in the routine, we do not need to form the LU factorization of the starting basis. We therefore introduced a second additional option to DPLO, which when turned on will make the restart procedure retrieve the LU factorization of the starting basis from high speed memory, rather than forming it. Thus, the lp's in the bunch other than the one corresponding to the least k value are solved using the restart procedure of DPLO with both these new options turned on.

Consider now the implementation of routine *lowerbound*. Prior to the first call to *lowerbound* in Algorithm 2.1, the lp

$$\begin{aligned} &\text{find } x \in \mathbb{R}^{n_1} \text{ such that} \\ &c^T x \text{ is minimized, and} \\ &Ax = b \\ &x \geq 0 \end{aligned} \tag{2}$$

needs to be solved. Suppose that we decide to place a limit $maxcut$ on the number of cuts

to be added by Algorithm 2.1. Then (2) may be written as

$$\begin{aligned}
 & \text{find } x \in \mathbb{R}^{n_1}, \theta \in \mathbb{R} \text{ such that} \\
 & c^T x + \theta \text{ is minimized, and} \\
 & Ax + \theta = b \\
 & (E^l)^T x + \theta \geq e^l, \quad l = 1, 2, \dots, \text{maxcut} \\
 & x \geq 0
 \end{aligned} \tag{3}$$

where $E^l = 0$, $e^l = 0$ for $l = 1, 2, \dots, \text{maxcut}$. When a call is made to routine *lowerbound* with $1 \leq t \leq \text{maxcut}$, an lp of the form (3) needs to be solved, where $E^l, e^l, l = 1, 2, \dots, t$ will have values generated by calls to *optimality_cut* and $E^l = 0, e^l = 0, l = t+1, t+2, \dots, \text{maxcut}$. In our implementation we therefore solve the lp in routine *lowerbound*, using the restart procedure of DPLO as follows. We first solve lp (2), treating it in the form (3), *from scratch*. Routine *lowerbound* solves its lp using the restart procedure of DPLO with the new option that prevents disk reads turned on. The fact that the $(m_1 + t)$ -th row of the constraint matrix changes to $[(E^t)^T, 1]$ from a row of zeros can be specified to DPLO as part of this restart procedure. We however, cannot use the LU factorization of the optimal basis of the lp of the form (3) solved prior to this call to *lowerbound*. We therefore *do not* turn on our new option that prevents DPLO from forming the LU factorization of the starting basis.

For the stopping criterion in Algorithm 2.1 we use the following in our parallel version. If

$$\frac{|Q - \theta|}{\max\{1, |Q|\}} \leq \text{tol}$$

where *tol* is a prescribed tolerance, we terminate.

We can now state our benchmark parallel version of Algorithm 2.1.

Algorithm 3.1: (Benchmark Parallel Version of Algorithm 2.1)

input:

$m_1; n_1; m_2; n_2; K; A; c; b; M; q; F := ((h^k, T^k), p^k), k = 1, 2, \dots, K);$
 $nprocs; \text{maxcut}; \text{tol}.$

output:

$x; z.$

begin

```

/* begin initializations */
t := 0;
l := 1 to maxcut do
    El := 0; el := 0;
end do;
{call DPLO to solve the lp
    find  $x \geq 0, \theta \in \Re$  such that
     $c^T x + \theta$  is minimized, and
     $Ax = b$ ,
     $(E^l)^T x + \theta \geq e^l, l = 1, 2, \dots, \text{maxcut}$  from scratch};
call optimality_cut (m2, n2, K, M, q, F, x, nprocs, E, e, Q);
t := t + 1;
Et := E; et := e;
call lowerbound (m1, n1, t, A, b, (El, el, l = 1, 2, ..., t), maxcut, x, θ, z);
call optimality_cut (m2, n2, K, M, q, F, x, nprocs, E, e, Q);
/* end initializations */
while |Q - θ| / max{1, |Q|} < tol do
    t := t + 1;
    if t > maxcut then
        {report that more cuts than maxcut need to be added and stop};
    else
        Et := E; et := e;
        call lowerbound (m1, n1, t, A, b, (El, el, l = 1, 2, ..., t), maxcut, x, θ, z);
        call optimality_cut (m2, n2, K, M, q, F, x, nprocs, E, e, Q);
    end if;
end while;
end.

```

optimality_cut:

input:

m₂; n₂; K; M; q; F; x; nprocs.

output:

E; e; Q.

```

begin
   $E := 0$ ;  $e := 0$ ;
   $bnchsz := \lfloor K/nprocs \rfloor$ ;
  for  $bunch := 1$  to  $nprocs$  in parallel do
    for  $k := (bunch - 1) \cdot bnchsz + 1$  to  $\max\{bunch \cdot bnchsz, K\}$  do
       $w := h^k - T^k x$ ;
      if  $k = (bunch - 1) \cdot bnchsz + 1$  then
        {call DPLO to solve the lp
          find  $y \geq 0$  such that
             $q^T y$  is minimized, and
             $My = w$ 
          from scratch to obtain dual maximizer  $v$ };
      else
        {call DPLO with restart procedure and with new options to prevent
          disk reads, and to prevent forming LU factorization of initial basis
          to solve the lp
          find  $y \geq 0$  such that
             $q^T y$  is minimized, and
             $My = w$ 
          to obtain dual maximizer  $v$ };
      end if;
       $E := E + p^K (T^k)^T v$ ;
       $e := e + p^k (h^k)^T v$ ;
    end do;
  end do;
   $Q := e - E^T x$ ;
  return;
end.

```

lowerbound: input:

m_1 ; n_1 ; t ; b ; E^t ; e^l , $l = 1, 2, \dots, t$; $maxcut$.

output:

x ; θ ; z .

begin

{call DPLO with restart procedure, and with new option to prevent disk reads, and with a specification that $(m_1 + t)$ -th row of constraint matrix changes to $[(E^t)^T, 1]$, to solve the lp

find $x \geq 0, \theta \in \mathcal{R}$ such that

$c^T x + \theta$ is minimized, and

$Ax = b,$

$(E^l)^T x + \theta \geq e^l, l = 1, 2, \dots, maxcut\}$;

$z := c^T x + \theta;$

return;

end.

4. The Experiment

We have developed a FORTRAN implementation of Algorithm 3.1 on the Sequent/Balance. The only portion of the code that needs parallelization, as indicated in the statement of Algorithm 3.1, is that portion representing the outer 'do loop' in routine *optimality_cut*. That parallelization was done using the 'doacross' compiler directive on the Sequent, as described in Chapter 4 of Sequent Guide to Parallel Programming (1987).

In the rest of this section we describe a computational experiment that we have performed with the implementation on problems of the form (1) satisfying assumptions (A1), (A2) and (A3). All computations were done in double precision arithmetic.

The test problems for the experiments were generated using the test problem generator GENSLP of Kall and Keller (1985). Once the user specifies the problem dimensions m_1, n_1, m_2 and n_2 , ranges for elements of vectors and matrices, and densities for the matrices, GENSLP generates the deterministic data A, b, c, q and M with elements of vectors and matrices uniformly distributed in the ranges specified. GENSLP guarantees that M it generates satisfies (A2), and that the set $\{x : Ax = b, x \geq 0\}$ is nonempty. Note that we can ensure that $\{x : Ax = b, x \geq 0\}$ is bounded by selecting the range for the elements of A so that they are nonnegative. Assumption (A3) can be satisfied by simply selecting the range for the elements of q so that these elements are nonnegative. If GENSLP cannot generate A and M so that they have densities specified, it reports the densities that it has been able to achieve.

GENSLP does not generate the distribution F for (h, T) . However, it has the following mechanism so that the user can generate a distribution F for (h, T) . GENSLP treats that

$\mathbf{h}$ and $\mathbf{T}$ satisfy

$$\mathbf{h} = \mathbf{a}^0 + \sum_{j=1}^{\kappa} \mathbf{S}^j \mathbf{a}^j \quad (4.1)$$

$$\mathbf{T} = \mathbf{A}^0 + \sum_{j=1}^{\kappa} \mathbf{S}^j \mathbf{A}^j. \quad (4.2)$$

In (4.1), (4.2), $\mathbf{a}^j \in \mathbb{R}^{m_2}$, $\mathbf{A}^j \in \mathbb{R}^{m_2 \times n_1}$ for $j = 0, 1, \dots, \kappa$ are generated randomly by GENSLP once the user specifies the nonnegative integer κ , the ranges for the elements of $\mathbf{a}^j$ and $\mathbf{A}^j$, and the density for the matrices $\mathbf{A}^j$, $j = 0, 1, \dots, \kappa$. $\mathbf{S}^j$, $j = 1, 2, \dots, \kappa$ are univariate random variables for which the user can specify *univariate* distributions, thereby inducing a multivariate distributions on $(\mathbf{h}, \mathbf{T})$. In this experiment two discrete multivariate distributions on $(\mathbf{h}, \mathbf{T})$ are generated using the univariate binomial and Poisson distributions as follows. (These distributions were first described in Lessor (1988) in connection with an experiment on the performance of the algorithms of Ariyawansa, Sorensen and Wets (1987).) First a positive integer vector $\mathbf{r} \in \mathbb{N}^{\kappa}$ is selected, where r_j is the number of realizations desired for $\mathbf{S}^j$, $j = 1, 2, \dots, \kappa$. Let s_{ij} be the i -th realization of $\mathbf{S}^j$ and let $P_{ij} := \Pr(\{\mathbf{S}^j = s_{ij}\})$; $i = 1, 2, \dots, r_j$; $j = 1, 2, \dots, \kappa$. Note that if s_{ij} and P_{ij} ; $i = 1, 2, \dots, r_j$; $j = 1, 2, \dots, \kappa$ are available then the following multivariate distribution on $(\mathbf{h}, \mathbf{T})$ can be described. If i_j is any integer such that $1 \leq i_j \leq r_j$; $j = 1, 2, \dots, \kappa$, then using $\mathbf{S}^j := s_{i_j j}$; $j = 1, 2, \dots, \kappa$ in (5.1), (5.2) a realization $(\mathbf{h}^k, \mathbf{T}^k)$ of $(\mathbf{h}, \mathbf{T})$ may be generated with probability $p^k = \prod_{j=1}^{\kappa} P_{i_j j}$. There are $K := \prod_{j=1}^{\kappa} r_j$ such realizations and it is not too difficult to verify that $\sum_{k=1}^K p^k = 1$, so that $F := ((\mathbf{h}^k, \mathbf{T}^k), p^k, k = 1, 2, \dots, K)$ indeed is a distribution on $(\mathbf{h}, \mathbf{T})$. Note also that by appropriately selecting the value of $\mathbf{r}$, the value of K can be controlled.

It now remains to specify the way s_{ij} and P_{ij} ; $i = 1, 2, \dots, r_j$; $j = 1, 2, \dots, \kappa$ are generated. Suppose that the discrete distributions for $\mathbf{S}^j$, $j = 1, 2, \dots, \kappa$ are on a common range $[\alpha, \beta]$, where $\alpha, \beta \in \mathbb{R}$, $\alpha < \beta$ are given. Then two distributions are specified by selecting s_{ij} and P_{ij} , $i = 1, 2, \dots, r_j$; $j = 1, 2, \dots, \kappa$ as follows.

(i) Binomial-related distribution on $(\mathbf{h}, \mathbf{T})$:

Suppose that $p \in (0, 1)$ is given and that $r_j > 1$. Then let

$$s_{ij} := \frac{(\beta - \alpha)}{(r_j - 1)}(i - 1) + \alpha \quad \text{and}$$

$$P_{ij} := \binom{r_j - 1}{i - 1} (1 - p)^{r_j - i} p^{i - 1},$$

for $i = 1, 2, \dots, r_j$. This is repeated for $j = 1, 2, \dots, \kappa$.

(ii) Poisson-related distribution on (h, T) :

Suppose that $\lambda \in \mathbb{R}$, $\lambda > 0$ is given and that $r_j > 1$. Then let

$$s_{ij} := \frac{(\beta - \alpha)}{(r_j - 1)}(i - 1) + \alpha \quad \text{for } i = 1, 2, \dots, (r_j - 1);$$

$$P_{ij} := \frac{e^{-\lambda} \lambda^{i-1}}{(i-1)!} \quad \text{for } i = 1, 2, \dots, (r_j - 1);$$

$$s_{r_j j} := \beta; \quad \text{and}$$

$$P_{r_j j} := 1 - \sum_{i=1}^{r_j-1} P_{ij}.$$

This is repeated for $j = 1, 2, \dots, \kappa$.

The random number generator used in conjunction with GENSLP is that of Schrage (1979). The description of the test problems of the form (1) used in the experiment is now complete.

As mentioned in Section 3, the lp solver used is the subroutine DPLO of Hanson and Hiebert (1981). The numerical tolerances pertinent to DPLO that were important in obtaining the numerical results reported here are 'TUNE', 'FACTOR' and 'TOLABS'. 'TUNE' and 'FACTOR' are used in determining if an lp is feasible with respect to a relative tolerance, while 'TOLABS' is used in determining if an lp is infeasible with respect an absolute tolerance if the relative tolerance criteria cannot be met.

The experiment that we have performed consists of four parts: (a), (b), (c) and (d). Part (a) may be treated as a base case. Parts (b), (c) and (d) differ from part (a) in a single key factor that may affect the performance of Algorithm 3.1. The aim therefore is to be able to see whether conclusions on the performance of Algorithm 3.1 can be made with due consideration of these factors.

We first describe part (a) of the experiment. It involves four problem sizes: (i), (ii), (iii) and (iv). The corresponding values of problem dimensions are indicated in Table 4.1.

problem size	problem dimensions				random number seed
	m_1	n_1	m_2	n_2	
(i)	40	60	10	15	7756915
(ii)	60	88	15	22	5489721
(iii)	80	120	20	30	552119
(iv)	100	148	25	37	82353

Table 4.1 Problem dimensions and random number seeds

For each problem size, we used a binomial-related distribution with $K := 100, 1000$ and 10000 . Values $\kappa := 3$ and $p := 0.4$ were used for all three K values. Values of α, β, r_1, r_2 and r_3 used are indicated in Table 4.2.

K	α	β	r_1	r_2	r_3
100	-112.0	128.0	5	5	4
1000	-52.0	68.0	10	10	10
10000	-10.0	26.0	25	25	16

Table 4.2 Distribution parameters that depend on K

For each problem size (i), (ii), (iii) and (iv), and for each value of $K := 100, 1000, 10000$ we generated 3 problems of the form (1) using GENSLP. The ranges for elements of A, c, q, M , and $a^j, A^j, j = 0, 1, 2, 3$ were specified as in Table 4.3 below.

data variable	range for elements
A	$[0.5, 5]$
c	$[0.5, 1000/m_1]$
q	$[0.1, 10]$
M	$[1, 50]$
a^0	$[60, 180]$
$a^j, j \neq 0$	$[50/\kappa, 150/\kappa]$
A^0	$[120/n_1, 200/n_1]$
$A^j, j \neq 0$	$[120/(n_1\kappa), 200/(n_1\kappa)]$

Table 4.3 Ranges for problem data elements specified to GENSLP

The densities specified to GENSLP for the matrices A, M and $A^j, j = 0, 1, 2, 3$ were 20%, 40% and 10% respectively. Our aim is to obtain cpu time that Algorithm 3.1 and routine *optimality_cut* take, when Algorithm 3.1 is run parallelly and sequentially with the same value of $nprocs$, on the same problem. It is to allow for the variation of timing results due to randomness of data produced by GENSLP that we generated three problems for each case specified by problem size and value of K . Cpu time values that we report are average values computed using three values observed for the three problems. In Table 4.1 we also indicate the random number seed that we specified for each problem size, so that our experiment is repeatable. (We did not change this seed for different K values for the same problem size.)

In order to complete the description of part (a) of the experiment, we now specify the values of the tolerances TUNE, FACTOR and TOLABS used for DPLO, and the values

of *tol* and *maxcut* used for Algorithm 3.1. The values $TUNE := 100\epsilon$, $FACTOR := 1$, $TOLABS := \sqrt{\epsilon}$, $tol := \sqrt{\epsilon}$ and $maxcut := 15$ were used. Here ϵ is the largest relative spacing for double precision floating point numbers and is approximately 0.222×10^{-15} for the Sequent.

In Tables 4.4 and 4.5 we report the parallel and sequential cpu time values observed when $nprocs := 7$. In Table 4.6 we report the resulting speed-up values. Note that in Algorithm 3.1, we parallelize only routine *optimality_cut*, and therefore, more calls to *optimality_cut* would help improve the speed-up values. In order that the reader can put the speed-up values reported into proper perspective, in Table 4.6 we also record the number of cuts added by Algorithm 3.1 ($=$ number of calls to *optimality_cut* $- 1$). As one would expect both overall and *optimality_cut* speed-up values improve as K increases for a given problem size. The overall speed-up values around 5.5 achieved for $K := 10000$ on a 7 processor machine are encouraging.

Throughout part (a) of the experiment we kept the densities of the matrices A , M and A^j , $j = 1, 2, \dots, \kappa$ at the values stated above. Note that decreasing the density of A relative to the density of M is likely to improve the speed-up values. In part (b) of the experiment we investigated this hypothesis by repeating part (a) with density of A specified as 2.5% with all other details unchanged. Tables 4.7, 4.8 and 4.9 respectively indicate the information corresponding to Tables 4.4, 4.5 and 4.6, when this change is made. As expected, overall speed-up values improve.

Part (c) of the experiment is designed to see the effect of the distribution form on the performance. In part (c), we therefore repeated part (a) with the binomial-related distribution replaced by a Poisson-related distribution (keeping all other details unchanged) as follows. The parameter λ of the underlying Poisson distribution was set at 3.0. The performance results are as indicated in Tables 4.10, 4.11 and 4.12. Comparing these three tables with the corresponding tables for part (a), we observe that the speed-up values for parts (a) and (c) are similar.

Finally, we investigated the effect of the number processors on the performance by repeating part (a) with $nprocs := 14$ (but with all other details unchanged). At this point it should be mentioned that parts (a), (b) and (c) of the experiment were performed on the Sequent/Balance at the Department of Computer Science of Washington State University. This computer is configured with 10 processors. Therefore, part (d) was performed on the Sequent/Balance at the Advanced Computer Research Facility, Argonne National Laboratory, Argonne, Illinois. The latter computer has 24 processors. The results of part (d) are tabulated in Tables 4.13, 4.14 and 4.15.

problem size	overall parallel cpu time (sec) when K is			optimality_cut parallel cpu time (sec) when K is		
	100	1000	10000	100	1000	10000
(i)	1.3069×10^2	3.4976×10^2	2.2542×10^3	3.8683×10^1	2.6030×10^2	2.1640×10^3
(ii)	6.5846×10^2	1.3522×10^3	6.5616×10^3	1.3253×10^2	8.5357×10^2	6.1249×10^3
(iii)	1.1959×10^3	2.1727×10^3	9.5141×10^3	1.8606×10^2	1.1614×10^3	8.5694×10^3
(iv)	3.2918×10^3	7.1409×10^3	2.1286×10^4	4.3730×10^2	3.5840×10^3	1.8812×10^4

Table 4.4 Part (a) of experiment: parallel cpu time

problem size	overall sequential cpu time (sec) when K is			optimality_cut sequential cpu time (sec) when K is		
	100	1000	10000	100	1000	10000
(i)	3.1102×10^2	1.7040×10^3	1.3315×10^4	2.1906×10^2	1.6146×10^3	1.3225×10^4
(ii)	1.2751×10^3	5.8787×10^3	3.8976×10^4	7.4731×10^2	5.3800×10^3	3.8539×10^4
(iii)	1.9569×10^3	7.9039×10^3	5.1742×10^4	9.8651×10^2	6.9294×10^3	5.0839×10^4
(iv)	5.5268×10^3	2.6114×10^4	1.1802×10^5	2.4847×10^3	2.2263×10^4	1.1539×10^5

Table 4.5 Part (a) of experiment: sequential cpu time

problem size	overall speed-up when K is			optimality_cut speed-up when K is			no. of cuts added when K is		
	100	1000	10000	100	1000	10000	100	1000	10000
(i)	2.37	4.87	5.91	5.66	6.20	6.11	2, 2, 2	2, 2, 2	2, 2, 2
(ii)	1.94	4.35	5.94	5.64	6.30	6.29	6, 4, 1	4, 5, 2	5, 2, 2
(iii)	1.64	3.64	5.44	5.30	5.97	5.93	2, 2, 3	2, 2, 3	2, 2, 2
(iv)	1.68	3.66	5.54	5.68	6.21	6.13	2, 6, 2	2, 10, 2	2, 4, 2

Table 4.6 Part (a) of experiment: speed-up values and no. of cuts added

problem size	overall parallel cpu time (sec) when K is				optimality-cut parallel cpu time (sec) when K is			
	100	1000	10000		100	1000	10000	
(i)	7.6072×10^1	3.0042×10^2	3.0532×10^3		5.2394×10^1	2.8200×10^2	3.0274×10^3	
(ii)	1.5943×10^2	1.1306×10^3	1.2295×10^4		1.1391×10^2	1.0553×10^3	1.2186×10^4	
(iii)	4.7303×10^2	2.6342×10^3	8.6025×10^3		3.1238×10^2	2.4233×10^3	8.5256×10^3	
(iv)	5.9763×10^2	1.8929×10^3	1.6342×10^4		3.7702×10^2	1.7213×10^3	1.6156×10^4	

Table 4.7 Part (b) of experiment: parallel cpu time

problem size	overall sequential cpu time (sec) when K is				optimality-cut sequential cpu time (sec) when K is			
	100	1000	10000		100	1000	10000	
(i)	3.1583×10^2	1.8145×10^3	1.8953×10^4		2.9228×10^2	1.7962×10^3	1.8927×10^4	
(ii)	6.7912×10^2	6.7202×10^3	7.6310×10^4		6.3366×10^2	6.6447×10^3	7.6201×10^4	
(iii)	1.9418×10^3	1.6022×10^4	5.6620×10^4		1.7820×10^3	1.5811×10^4	5.6544×10^4	
(iv)	2.1976×10^3	1.0621×10^4	1.0084×10^5		1.9743×10^3	1.0448×10^4	1.0065×10^5	

Table 4.8 Part (b) of experiment: sequential cpu time

problem size	overall speed-up when K is				optimality-cut speed-up when K is				no. of cuts added when K is			
	100	1000	10000		100	1000	10000		100	1000	10000	
(i)	4.15	6.04	6.21		5.58	6.37	6.25		5, 2, 2	2, 3, 2	2, 2, 6	
(ii)	4.26	5.94	6.21		5.56	6.30	6.25		3, 2, 3	2, 9, 2	2, 7, 10	
(iii)	4.11	6.08	6.58		5.70	6.52	6.63		2, 10, 2	2, 13, 3	2, 2, 2	
(iv)	3.68	5.61	6.17		5.24	6.07	6.23		2, 2, 5	2, 2, 2	3, 2, 2	

Table 4.9 Part (b) of experiment: speed-up values and no. of cuts added

problem size	overall parallel cpu time (sec) when K is			<i>optimality-cut</i> parallel cpu time (sec) when K is		
	100	1000	10000	100	1000	10000
(i)	1.3341×10^2	3.5300×10^2	2.2567×10^3	3.8733×10^1	2.6030×10^2	2.1675×10^3
(ii)	4.4245×10^2	1.5515×10^3	6.0375×10^3	8.4394×10^1	9.7851×10^2	5.6459×10^3
(iii)	1.1315×10^3	2.0047×10^3	1.1561×10^4	1.6858×10^2	1.0478×10^3	1.0483×10^4
(iv)	3.5647×10^3	5.6141×10^3	1.7436×10^4	4.9032×10^2	2.7281×10^3	1.5378×10^4

Table 4.10 Part (c) of experiment: parallel cpu time

problem size	overall sequential cpu time (sec) when K is			<i>optimality-cut</i> sequential cpu time (sec) when K is		
	100	1000	10000	100	1000	10000
(i)	3.1352×10^2	1.7076×10^3	1.3347×10^4	2.1893×10^2	1.6149×10^3	1.3257×10^4
(ii)	8.3443×10^2	6.7179×10^3	3.5868×10^4	4.7706×10^2	6.1442×10^3	3.5473×10^4
(iii)	1.8030×10^3	7.1793×10^3	6.3529×10^4	8.9432×10^2	6.2723×10^3	6.2489×10^4
(iv)	6.0442×10^3	1.9933×10^4	9.5863×10^4	2.7361×10^3	1.6789×10^4	9.3696×10^4

Table 4.11 Part (c) of experiment: sequential cpu time

problem size	overall speed-up when K is			<i>optimality-cut</i> speed-up when K is			no. of cuts added when K is		
	100	1000	10000	100	1000	10000	100	1000	10000
(i)	2.35	4.84	5.91	5.65	6.20	6.12	2, 2, 2	2, 2, 2	2, 2, 2
(ii)	1.89	4.33	5.94	5.65	6.28	6.28	3, 2, 1	4, 7, 2	3, 3, 2
(iii)	1.59	3.58	5.50	5.31	5.99	5.96	2, 2, 2	2, 2, 2	3, 2, 3
(iv)	1.70	3.55	5.50	5.58	6.15	6.09	2, 7, 2	2, 6, 2	2, 2, 2

Table 4.12 Part (c) of experiment: speed-up values and no. of cuts added

problem size	overall parallel cpu time (sec) when K is				optimality_cut parallel cpu time (sec) when K is			
	100	1000	10000	100000	100	1000	10000	100000
(i)	1.1472×10^2	2.2629×10^2	1.2062×10^3	1.2062×10^3	2.2361×10^1	1.3662×10^2	1.1159×10^3	1.1159×10^3
(ii)	6.0517×10^2	9.4863×10^3	3.5677×10^3	3.5677×10^3	7.5894×10^1	4.4951×10^1	3.1299×10^3	3.1299×10^3
(iii)	1.1223×10^3	1.6211×10^3	5.3568×10^3	5.3568×10^3	1.0463×10^2	6.1158×10^2	4.4061×10^3	4.4061×10^3
(iv)	3.1156×10^3	5.4538×10^3	1.2063×10^4	1.2063×10^4	2.5897×10^2	1.9179×10^3	9.5813×10^3	9.5813×10^3

Table 4.13 Part (d) of experiment: parallel cpu time

problem size	overall sequential cpu time (sec) when K is				optimality_cut sequential cpu time (sec) when K is			
	100	1000	10000	100000	100	1000	10000	100000
(i)	3.1620×10^2	1.7282×10^3	1.3417×10^4	1.3417×10^4	2.2372×10^2	1.6384×10^3	1.3326×10^4	1.3326×10^4
(ii)	1.3222×10^3	5.9778×10^3	3.9135×10^4	3.9135×10^4	7.9166×10^2	5.4763×10^3	3.8696×10^4	3.8696×10^4
(iii)	2.0152×10^3	8.0491×10^3	5.2092×10^4	5.2092×10^4	1.0423×10^3	7.0697×10^3	5.1194×10^4	5.1194×10^4
(iv)	5.6231×10^3	2.6609×10^4	1.1882×10^5	1.1882×10^5	2.5423×10^3	2.2678×10^4	1.1617×10^5	1.1617×10^5

Table 4.14 Part (d) of experiment: sequential cpu time

problem size	overall speed-up when K is				optimality_cut speed-up when K is				no. of cuts added when K is			
	100	1000	10000	100000	100	1000	10000	100000	100	1000	10000	100000
(i)	2.76	7.64	11.12	10.00	11.94	11.99	11.94	11.94	2, 2, 2	2, 2, 2	2, 2, 2	2, 2, 2
(ii)	2.18	6.30	10.96	10.43	12.36	12.18	12.36	12.36	6, 4, 1	4, 5, 2	5, 2, 2	5, 2, 2
(iii)	1.80	4.97	9.72	9.96	11.61	11.56	11.61	11.61	2, 2, 3	2, 2, 3	2, 2, 2	2, 2, 2
(iv)	1.80	4.88	9.85	9.82	12.12	11.82	12.12	12.12	2, 6, 2	2, 10, 2	2, 4, 2	2, 4, 2

Table 4.15 Part (d) of experiment: speed-up values and no. of cuts added

5. Conclusion

We have described a simple, parallel version of the Van Slyke and Wets (1969) algorithm for two-stage stochastic programs with recourse. We have also described an implementation of that algorithm on the Sequent/Balance, and the results of a carefully designed numerical experiment with the implementation on random test problems generated by the test problem generator of Kall and Keller (1985).

The parallel implementation described in the paper was motivated by the need of a benchmark to assess the performance of parallel, *approximate* algorithms that are being developed by the first author. The numerical results presented indicate that the benchmark implementation parallelizes well. They also indicate that even with the use of parallel processing, meaningful stochastic programs can take prohibitively large amounts of computation for solution. Thus, they demonstrate the need for exploiting *both* parallelization *and* approximation for the solution of stochastic programs.

Acknowledgements

We thank the Advanced Computer Research Facility, Argonne National Laboratory, Argonne, Illinois, and the Department of Computer Science, Washington State University, for giving us permission to use their Sequent/Balance computers.

References

1. Ariyawansa, K. A., D. C. Sorensen and R. J-B. Wets (1987) "Parallel schemes to approximate values and subgradients of the recourse function in certain stochastic programs", Manuscript, Department of Pure and Applied Mathematics, Washington State University, Pullman, WA 99164-2930.
2. Dantzig, G. B. (1985) "Parallel processors for planning", Paper presented at the 12th International Symposium on Mathematical Programming, Boston, MA, August 5-9, 1985.
3. Dempster, M. A. H. (1980), *Stochastic Programming*, Academic Press, New York.
4. Ermoliev, Y. and R. J-B. Wets (1988) "Stochastic programming, an introduction", in *Numerical Techniques for Optimization*, eds., Yu. Ermoliev and R. J-B. Wets, Springer-Verlag, New York, pp. 1-32.
5. Hanson, R. J. and K. L. Hiebert (1981), "A sparse linear programming subprogram", Report SAND81-0297, Sandia National Laboratories, Albuquerque, New Mexico and

Livermore, California.

6. Kall, P. (1979), "Computational methods for solving two-stage stochastic linear programming problem", *Z. Angew. Math. Phys.*, Vol. 30, pp. 261-271.
7. Kall, P. and E. Keller (1985), "GENSLP: A program for generating input for stochastic linear programs with complete fixed recourse", Manuscript, Institut für Operations Research der Universität Zürich, Zürich CH-8006, Switzerland.
8. Lessor, K. S. (1988), "A preliminary numerical evaluation of a parallel algorithm for approximating the values and subgradients of the recourse function in a stochastic program with complete recourse", MS Project Report, Department of Computer Science, Washington State University, Pullman, WA 99164-1210.
9. Reid, J. (1976), "Fortran subroutines for handling sparse linear programming bases", Technical Report AERE-R 8269, Computer Science and Systems Division, AERE Harwell, Didcot, Oxfordshire, United Kingdom.
10. Schrage, L. (1979), "A more portable random number generator", *ACM Transactions of Mathematical Software*, Vol. 5, No. 2, pp. 132-138.
11. Van Slyke, R. and R. J-B. Wets (1969), "L-shaped linear programs with applications to optimal control and stochastic programming", *SIAM J. Appl. Math.*, Vol. 17, pp. 638-663.
12. Wets, R. J-B. (1982), "Stochastic programming: Solution techniques and approximation schemes", in *Mathematical Programming—The State of the Art*, eds., A. Bachem, M. Grötschel and B. Korte, pp. 566-603.
13. Wets, R. J-B. (1985), "On parallel processors design for stochastic programs", Working Paper WP 85-67, International Institute of Applied Systems Analysis, A-2361, Laxenburg, Austria.
14. Wets, R. J-B. (1988), "Stochastic Programming", to appear in *Handbook on Operations Research*.

END

DATE FILMED

11 / 07 / 90