CORE
Metadata, citation and similar papers at core.ac.uk
Provided by RERO DOC Digital Library

Ann Oper Res (2006) 142: 147–164
DOI 10.1007/s10479-006-6166-y

Some insights into the solution algorithms for SLP problems

Peter Kall · János Mayer

© Springer Science + Business Media, Inc. 2006

Abstract We consider classes of stochastic linear programming problems which can be efficiently solved by deterministic algorithms. For two–stage recourse problems we identify two such classes. The first one consists of problems where the number of stochastically independent random variables is relatively low; the second class is the class of simple recourse problems. The proposed deterministic algorithm is successive discrete approximation. We also illustrate the impact of required accuracy on the efficiency of this algorithm. For jointly chance constrained problems with a random right–hand–side and multivariate normal distribution we demonstrate the increase in efficiency when lower accuracy is required, for a central cutting plane method. We support our argumentation and findings with computational results.

1. Introduction

Stochastic linear programming (SLP) problems are generally considered as being numerically hard to solve. Despite their name, SLP problems are nonlinear programming problems, with the special feature that they involve either integrals or sums usually containing a large number of terms.

Recently excellent and interesting computational results have been achieved for large–scale two–stage complete recourse problems with a finite discrete distribution, by Linderoth, Shapiro, and Wright (2002), and Linderoth and Wright (2002). In the first paper the authors use sampling methods; for the theoretical background of these sampling methods see Shapiro and Homem–de–Mello (2000) and the references therein. In the second paper the authors develop an L-shaped method combined with trust region techniques. Nevertheless, implicitly this paper relies also on sampling, because sampled versions of the original large–scale test problems are solved. In both papers the authors have obtained their computational results by utilizing a powerful computational grid.

P. Kall · J. Mayer (✉)
IOR University of Zurich
e-mail: mayer@ior.unizh.ch

For general complete recourse problems, involving both a large number of discretely distributed random variables and a large number of joint realizations, the presently available solution approaches are all sample–based, i.e. they are stochastic methods. For solving large–scale problems, the stochastic decomposition method of Higle and Sen (1991, 1996), and the algorithms described in Linderoth, Shapiro, and Wright (2002) have been successfully applied.

Algorithms, which do not belong to the class of stochastic methods, will be called deterministic algorithms. For a detailed discussion, concerning the distinction between stochastic and deterministic methods see Kall and Wallace (1994) Section 3.8.

In this paper we argue that for some subclasses of two–stage complete recourse problems, specialized deterministic algorithms can successfully be used to solve large–scale problems. The high efficiency of these solvers makes it possible to solve those problems on an ordinary PC. In this context, by large–scale we mean problems with a large number of joint realizations. We support our argumentation with computational results. We will show, that specialized deterministic algorithms may perform significantly better than deterministic methods designed for the general case.

Another speciality of SLP problems, which we would like to stress, is the fact that part of the model specification is the probability distribution of the random entries. In many practical modeling cases the distribution can only be specified as a rough approximation of the true distribution. In such cases it seems to be questionable, whether it makes sense to solve the SLP problem with high accuracy concerning the optimal objective value. Some SLP algorithms provide in each iteration lower and upper bounds on the optimal objective value, which the currently available best feasible solution fulfills. These bounds can be utilized in stopping criteria.

Our computational results presented in this paper, both for two–stage complete recourse problems and for jointly chance constrained problems, show that requiring lower accuracy with respect to the objective value in the stopping rule may result in a dramatic reduction in the computing time.

As a motivation for the importance of solving large–scale SLP problems, we refer to some recent large–scale SLP applications in practice: a finance application in Consigli and Dempster (1998), planning logistic operations in Dempster et al. (2000), a vehicle routing problem by Laporte, Louveaux, and Mercure (1992), and a capacity planning application by MirHassani et al. (2000).

We will consider two types of SLP problems: two stage complete recourse problems with finite discrete distribution of the random entries and jointly chance constrained problems (problems with a probabilistic constraint), where only the right hand side (RHS) is stochastic and has a multivariate normal distribution.

Our discussions concerning algorithms will be concentrated on deterministic methods. Although we also present computational results with our implementation of the stochastic decomposition method, this is merely for illustrative purposes. A computational comparison of stochastic– and deterministic algorithms, which ensures consistent quality of solutions obtained by the corresponding solvers, requires an extensive computational study. This is beyond the scope of this paper.

Please note, that we do not intend to rank algorithmic approaches or solvers. On the one hand, this could only be done on the basis of a much more extensive testing. On the other hand, some of our solvers are not the latest available implementations of the underlying algorithms.

2. Two stage recourse problems

Two stage recourse problems can be formulated as follows:

$$\begin{cases} \min & [c^{\mathrm{T}}x + \mathbb{E}\,[Q(x,\xi)]] \\ \text{s.t.} & Ax = b \\ & x \geq 0 \end{cases} \tag{1}$$

where A is an $(m_1 \times n_1)$ matrix, b and c have corresponding dimensions, and $Q(x,\xi)$ is the optimal objective value of the *recourse subproblem*

$$\begin{cases} Q(x,\xi) = & \min & q^{\mathrm{T}}(\xi)y \\ & \text{s.t.} & W(\xi)y = h(\xi) - T(\xi)x \\ & & y \geq 0 \end{cases} \tag{2}$$

where $W(\xi)$ is an $(m_2 \times n_2)$ random matrix, $T(\xi)$ is an $(m_2 \times n_1)$ random matrix, $h(\xi)$ denotes an m_2–dimensional random vector, and $q(\xi)$ stands for an n_2–dimensional random vector. $W(\xi)$ is called the recourse matrix.

ξ denotes an r–dimensional random vector on a probability space. Let us denote by Ξ the support of ξ. We restrict our attention to the case when ξ has a finite discrete distribution ("scenarios") with N being the number of the joint realizations of ξ. In the special case, when the components of ξ are stochastically independent, we denote by R_i the number of realizations of ξ_i, $i = 1, \ldots, r$. In this case the number of joint realizations of ξ will be $N = \prod_{i=1}^{r} R_i$. When each of the components of ξ has the same number of realizations R, we obtain for the number of joint realizations $N = R^r$.

We assume that the random entries are modeled as:

$$\begin{cases} h(\xi) = h^0 + \sum_{j=1}^{r} h^j \xi_j \\ q(\xi) = q^0 + \sum_{j=1}^{r} q^j \xi_j \\ T(\xi) = T^0 + \sum_{j=1}^{r} T^j \xi_j \\ W(\xi) = W^0 + \sum_{j=1}^{r} W^j \xi_j \end{cases} \tag{3}$$

where h^j and q^j are deterministic vectors, T^j and W^j are deterministic matrices, $j = 0, \ldots, r$, with dimensions corresponding to the dimension of the random array on the left–hand–side of the equations.

For general properties of the recourse problem (1) see e.g. (Birge and Louveaux, 1997; Kall and Wallace, 1994).

Problem (1) as formulated above is called a *random recourse problem*. This means that the recourse matrix $W(\xi)$ is stochastic, ($\exists\, j \geq 1$ such that $W^j \neq 0$ holds in (3)). As already mentioned in the Introduction, in this paper we advocate the use of specialized algorithms for some subclasses of two stage recourse problems. In Table 1 we list the subclasses which will be considered. We assume $q(\xi) \equiv q$ throughout, that is, we consider only the case when the objective function in the second stage is deterministic.

Introducing the notation $\bar{\xi} = \mathbb{E}[\,\xi\,]$, the *expected value problem* corresponding to (1) is defined as the deterministic LP problem

Table 1 Subclasses of two stage recourse problems

Fixed recourse:	$W(\xi) \equiv W$, the recourse matrix is deterministic
Relatively complete recourse:	For $\forall\, x \in \{x \mid Ax = b,\ x \geq 0\}$ and for $\forall \xi \in \Xi$ the second stage problem (2) is feasible, i.e. $\{y \mid Wy = h(\xi) - T(\xi)x,\ y \geq 0\} \neq \emptyset$.
Complete recourse:	$\forall\, \hat{h} \in \boldsymbol{R}^{m_2}$ holds $\{y \mid Wy = \hat{h},\ y \geq 0\} \neq \emptyset$.
Simple recourse:	$T(\xi) \equiv T$ and $W = (I, -I)$, with I denoting the identity matrix.

$$\begin{cases} \min & cx + q^{\mathrm{T}}(\bar{\xi})y \\ \text{s.t.} & T(\bar{\xi})x + W(\bar{\xi})y = h(\bar{\xi}) \\ & x \geq 0 \\ & y \geq 0 \end{cases} \tag{4}$$

As mentioned above, we assume that ξ has a finite discrete distribution with N being the number of joint realizations of ξ. In this case (1) can equivalently be formulated as a deterministic LP problem of the size $(m_1 + m_2 \cdot N) \times (n_1 + n_2 \cdot N)$. Denoting the realizations of the random arrays by $\hat{T}^{(k)}$, $\hat{W}^{(k)}$, $\hat{h}^{(k)}$ and $\hat{q}^{(k)}$, respectively, with probabilities p_k, $k = 1, \ldots, N$, this LP is the following:

$$\begin{cases} \min & cx + \displaystyle\sum_{k=1}^{N} p_k (\hat{q}^{(k)})^{\mathrm{T}} y^k \\ \text{s.t.} & Ax = b \\ & \hat{T}^{(k)}x + \hat{W}^{(k)}y^k = \hat{h}^{(k)}, \quad k = 1, \ldots, N \\ & x \geq 0 \\ & y^k \geq 0, \quad k = 1, \ldots, N \end{cases} \tag{5}$$

The LP problem (5) has a special, so called dual block angular structure, where the number of diagonal blocks equals N. The size of the equivalent LP grows rapidly with the number of realizations. To see this, let us consider the case of stochastically independent components of ξ, with each of the components having the same number of realizations R. As we have discussed above, the number of joint realizations will be $N = R^r$, which grows exponentially with r. Note, that large scale equivalent LP's can easily arise on the basis of small–size expected value problems (4), by just having a large number of joint realizations of ξ.

Due to the existence of the equivalent LP, the difficulty of the numerical solution of (1) is frequently judged by considering the size of this LP. While this might be true in the general random recourse case, we argue that this notion of "large scale SLP" is inappropriate when considering subclasses of the general setting. Structural properties of (1), like those listed in Table 1, and the probability distribution of ξ have a significant impact on the numerical difficulty of recourse problems.

As an analogy, let us consider combinatorial optimization which also operates in the realm of the combinatorial explosion, and many combinatorial optimization models can also be formulated equivalently as LP's. In the field of combinatorial optimization, utilizing special structure of the original formulation frequently leads to radically better algorithms than those based on the equivalent LP. Our point is that the situation is in some respects similar in SLP.

We would like to point out the usefulness of the affine relations (3) in the modeling phase. The case, when there is a one–to–one correspondence between the random entries of the model and the random variables (the components of ξ), is obviously included by choosing appropriate unit arrays in the sums. Beyond this, the affine relations offer the possibility to express the random entries in terms of fewer random variables ("factors") which may –depending on the particular application– even be assumed of being stochastically independent. This feature means no advantage to algorithms which aim at the solution of the equivalent LP (5), because this LP depends directly on the joint distribution of the random array entries. For approximation algorithms, however, the reduction of the number of random variables via (3) may extend the scope of applications far beyond the capabilities of methods dealing with the equivalent LP.

Considering algorithms for two–stage recourse problems, we would like to point out, that the solvers CPA, MSLiP, and FortSP are now also available as Internet resources at the Neos Server for Optimization, at http://www-neos.mcs.anl.gov/neos/. For the documentation of CPA visit the Neos Server, MSLiP will be discussed below, and FortSP is part of the SPInE integrated environment for stochastic programming; for the latter see (Valente, Mitra, and Poojari, 2005).

As mentioned in the Introduction, our discussions are concentrated on deterministic algorithms. Nevertheless, we also include a single stochastic method, for illustrative purposes. We will only take into account algorithms, for which we also have a solver, i.e., an implementation of the method. Below we list the algorithms and corresponding solvers which will be considered. The years indicated at the implementation of the solvers are dates corresponding to the development year of our version; for some solvers there exist also newer versions.

- Algorithms for the deterministic equivalent LP

  - General purpose LP algorithms
    - The simplex method as implemented in general–purpose LP solvers; we utilized the commercial solver **GAMS/OSL1** (Brooke et al., 1998); for detailed information please visit http://www.gams.com.
    - The primal–dual interior point method of Gondzio (1995).
      Solver: **HOPDM**, implemented by Gondzio in 1995.
  - Algorithms utilizing the structure of the LP
    - The augmented system interior point method of Mészáros (1997), see also Maros and Mészáros (1998); the structure is utilized in sparsity reduction. The algorithm also works for random recourse.
      Solver: **BPMPD**, implemented by Mészáros in 1998.
    - A variant of the nested Benders decomposition method of Birge (1985), developed by Gassmann (1990). This algorithm specializes basically in the two stage case to the L–shaped method of Van Slyke and Wets (1969). The method is for random recourse; for fixed recourse special techniques like "trickling down" are included for solving the similar diagonal subproblem, see e.g. (Kall and Wallace, 1994). Solver: **MSLiP**, implemented by Gassmann in 1994.
    - The regularized decomposition method of Ruszczyński (1986) for fixed recourse problems. This algorithm is in essence a regularized Benders decomposition method with cut dropping. Solver **QDECOM**, implemented by Ruszczyński in 1985.

- Algorithms for complete recourse aiming at the original problem (without building the equivalent LP)

- The successive discrete approximation method of Kall (1974), Kall and Stoyan (1982), Kall (1988), Kall (1998), Frauendorfer and Kall (1988). The idea is the following: The set of realizations is covered with an interval. Starting with the expected value problem, a sequence of approximate problems is generated by successively subdividing the starting interval and taking the coarser realizations corresponding to the subdivision. At each step lower and upper bounds on the optimal objective value are provided. Solver: **DAPPROX**, implemented by Kall and Mayer in 2001. For details concerning the implementation see also Mayer (1998).
- The stochastic decomposition method of Higle and Sen (1991, 1996). This method can be viewed as a stochastic version of Benders decomposition. The algorithm itself is stochastic and supplies a solution in the probabilistic a.s. sense. The implementation is based on the guidelines of Higle and Sen, see (Higle and Sen, 1996). Solver: **SDECOM**, implemented by Kall and Mayer in 1995. For details concerning the implementation see also (Mayer, 1998).

• Algorithm for simple recourse

- The successive discrete approximation method of Kall and Stoyan (1982), which is the successive approximation method as discussed above, specialized to the structure. The algorithm utilizes the fact that the nonlinear part in the objective of (1) is separable w.r. to $\chi = Tx$. Solver: **SRAPPROX**, implemented by Kall and Mayer in 1994. For details concerning the implementation see also (Mayer, 1998).

Bounds–based stopping rules are available for BPMPD, DAPPROX and SRAPPROX.

3. SLP problems with joint chance constraint

SLP problems with joint chance constraint (probabilistic constraint) can be formulated as follows:

$$\left.\begin{array}{ll} \min & c^{\mathrm{T}}x \\ \text{s.t.} & \mathbb{P}(Tx \geq h(\xi)) \geq \alpha \\ & Ax = b \\ & x \geq 0 \end{array}\right\} \tag{6}$$

where A is an $m_1 \times n_1$ matrix and T is an $m_2 \times n_1$ matrix; ξ is a random vector on a probability space with a joint multinormal distribution. Please notice that (6) has been formulated with a deterministic technology matrix. Under these assumptions (6) is a convex programming problem, see e.g. (Kall and Wallace, 1994; Prékopa, 1995).

Jointly chance constrained problems are considered "large scale" in the SLP sense, if the dimension of the random RHS (the number of random variables) is large. A couple of years ago problem (6), with an 8–dimensional random RHS already counted as large scale (Kall and Mayer, 1998). This has recently been extended to approximately 30 dimensions, see (Mayer, 2000).

The algorithms for solving (6) are constructed as follows: a general nonlinear programming algorithm is taken and is adapted to the special structure, see e.g. (Kall and Wallace, 1994; Mayer, 1998; Prékopa, 1995). In this paper we will only consider one solver, which has the capability of bounds–based stopping.

- A central cutting plane method of Mayer (1998). Solver: PROBALL, implemented by Mayer in 2001.

4. Testing environment

Our basic tool for performing the tests was our model management system SLP–IOR (Kall and Mayer, 1992, 1996; Mayer, 1998; Kall and Mayer, 2005). The new Windows32 version of SLP–IOR (developed in Borland Delphi 6) has been utilized. We have used the workbench facilities of the system, which support the dealing with test problem batteries. The following facilities have been used:

- The further developed version of the test problem generator GENSLP of Kall and Keller (see Mayer 1998) has been used for randomly generating test problem batteries of the following type: batteries containing recourse problems with existing optimal solution and test problem batteries with jointly chance constrained problems having a known solution.
- Test problem batteries can be handled by SLP–IOR as a whole; various operations can be selected, which are performed afterwards on each element of the battery in turn. The following facilities have been used:
  - Discretizing the probability distribution;
  - endowing the test problems with a normally distributed RHS.
  - A set of solvers can be selected to the battery; each element in the battery is then solved in turn with the selected solvers and the computational results summary tables are provided as LaTeX tableaus.

The solver library of SLP–IOR contains several solvers. The solvers which have participated in the tests are listed above with the algorithms. Except of GAMS/OSL, all solvers listed are part of the SLP–IOR distribution.

- LP–subproblems are solved with Minos 5.4.
- For computing the multivariate normal distribution function and its gradient the subroutine package PCSPNOR3 of Szántai (1987) has been used.
- The solvers have been developed in Fortran, the compiler is Compaq Visual Fortran 6.1.
- Computer: IBM PC/Pentium 660 MHz, 256 MB RAM;
- Operating system: Windows NT 4.0.

The computational results reported in the next section can completely be reproduced, because SLP–IOR is available free of charge for academic purposes and the test problem batteries are also available in SLP–IOR's input data format.

The test runs have generally been carried out with the default control parameter settings of all solvers, see (Mayer, 1998). The default parameter settings have been chosen on the basis of our extensive experience with the solvers, they are supposed to correspond to best average behavior, as observed so far.

The stopping tolerance for the bounds–based solvers DAPPROX and SRAPPROX has been chosen as relatively large, in order to demonstrate the effect mentioned in the Introduction. The tolerance values, which have been used in the tests, are indicated with the test run reports. The stopping rule for these solvers is as follows: The algorithm is stopped when

$$\frac{UB^k - LB^k}{1 + |LB^k|} < \varepsilon \tag{7}$$

is fulfilled, where LB^k, UB^k are the lower and upper bounds on the optimal objective value at iteration k, respectively. The stopping rule for SDECOM is based on average estimation for incumbent solutions (Higle and Sen, 1991), see also (Mayer, 1998). In the meantime much better stopping rules are available based on bootstrapping (Higle and Sen, 1996); these are not yet implemented. For BPMPD, being a primal–dual interior point method, the stopping rule is essentially based on the duality gap, see (Mészáros, 1997). QDECOM solves the equivalent LP problem by regularized decomposition, for the stopping criterion see (Ruszczyński, 1986).

5. The test problem batteries

Test problems from the literature are available nowadays as test problem collections, downloadable via the Internet. The benchmark problems in the collections are in SMPS standard format, for this format see (Birge et al., 1987). The presently available collections are the following:

- The POSTS test–set by Birge and Holmes, available at http://users.iems.nwu.edu/∼jrbirge//html/dholmes/post.html;
- the Ariyawansa–Felt test problem collection, see (Ariyawansa and Felt, 2001), available at http://www.uwsp.edu/math/afelt/slptestset.html;
- the WATSON benchmark collection of Dempster, available at http://www-cfr.jims.cam.ac.uk/research/stprog.html.

Apart from the first test problem battery in Section 6.1, we illustrate our point with randomly generated test problem batteries. Let us explain shortly the reason for this. For problems with joint chance constraints the reason is simple: the few test problems available in the literature have just up to 4 random variables in the right–hand–side. All of them can be solved by PROBALL within 0.1 seconds, in the above specified computing environment.

For two–stage recourse problems the reason is not that obvious. On the one hand, there are just a few test problems in the benchmark–collections, which could serve for illustrating our point concerning the advantages of using the special structure of simple recourse. On the other hand, we also wish to demonstrate that, under certain circumstances, solvers based on successive discrete approximation are suitable for solving complete recourse problems with a huge number of joint realizations. Again, the test problem collections contain just a few test problems of this type. In addition, these test problems do not belong the class of SLP problems, for which the successive discrete approximation method can be applied, see the remark below concerning SSN.

For test problems, where an underlying continuous distribution is known, or for which the number of joint realizations is already high, test problem batteries with an increasing number of realizations can be generated by sampling. This method has been used by several authors, for recent computational results of this type see, for example, (Linderoth and Wright, 2002). In this approach the arrays of the underlying deterministic LP (in our denotation the arrays A, b, c, h^j, q^j, T^j, W^j, $\forall j$) remain fixed and only the probability distribution changes.

In our experiments we wished to work with a substantially higher population of genuinely different test problems. We have chosen to work with test problem batteries, each consisting of 10 test problems, where the above listed arrays of the underlying LP are different across test problems in the battery. On this basis we have then constructed further batteries with an increasing number of realizations, essentially by utilizing the method as outlined above. Our computational results complement in this sense the computational results in the literature.

Table 2 Test problems from the literature: computing time in seconds

Problem	Dapprox	BPMPD	HOPDM	Sdecom	MSLiP	OSL
4node16	11.3	0.9	4.0	60.0	0.8	4.7
4node32	38.3	2.4	37.5	1.4	1.2	17.7
4node64	145.0	6.2	301.8	104.3	2.7	100.1
4node256	355.4*	37.5		131.2	11.1	1372.7
Stochfor2	—	2.7	14.6	—	3.4	63.3

Finally let us comment on our avoidance of two large–scale test problems, SSN of Sen, Doverspike, and Cosares (1994) and the full–scale storm problem of Mulvey and Ruszczyński (1995). The number of random variables in these problems is 86 and 118, respectively, with an astronomical number of joint realizations. As mentioned in the Introduction, these problems can presently only be solved by solvers based on sampling. We only have a single solver, SDECOM, of this type in SLP–IOR, therefore we have not included the above–mentioned test problems into our comparative investigations. Let us remark, that for DAPPROX the difficulty is not the huge number of joint realizations, but the dimension of ξ.

6. Computational results: Complete recourse

6.1. Some test problems from the literature

A small selection of test problems from the literature serves for the purpose of illustrating the performance of SLP solvers, compared to the performance of some general–purpose LP solver. The selection was motivated by the requirement, that the selected SLP problems should be tractable for the general–purpose solvers, when applied to the LP–equivalent (5).

Battery 1: *Literature*

There are 5 test problems in this battery, all of them are from the Ariyawansa–Felt collection.

The first 4 have their origin in a cargo network scheduling problem of Mulvey and Ruszczyński (1995); they are instances of a general problem, also known as the storm problem. The dimensions are $m_1 = 14, n_1 = 52, m_2 = 74, n_2 = 186$. Only the RHS is stochastic and contains 12 stochastically independent random entries. The number of realizations varies between 16 and 256 and is indicated in the test problem name. The motivation for selecting these problems is that they fit into our general framework of illustrating how the increasing number of realizations affects solver performance.

The fifth test problem, STOCHFOR2, originates in a forest planning model of Gassmann (1989). The dimensions are: $m_1 = 15, n_1 = 15, m_2 = 102, n_2 = 96$. Both $T(\xi)$ and $W(\xi)$ are stochastic (random recourse) and have altogether 168 random entries with 64 joint realizations. This problem has been selected to illustrate that SLP problems with a large number of random variables may still be tractable for general–purpose LP solvers, provided that the number of overall realizations is small.

For DAPPROX the maximum number of subintervals of the support had to be changed from the default value (2000) to 100 in order to fit the problems into the available memory on our PC. The computing times can be seen in Table 2.

For 4node256, DAPPROX ran out of available memory; the achieved relative accuracy at termination was 1.2%. For this problem the memory capacity was insufficient for HOPDM,

the solver terminated in the preprocessing phase. For MSLiP the *scaling* option had to be switched on in order to solve the 4node* problems.

We think that DAPPROX was able to solve these problems on our PC just by good luck. With 12 random variables there was memory space just for 100 potential subintervals in the approximation scheme; this is in general far too small for 12 random variables. Despite the fact that the problems 4node* have no complete recourse, they could be solved both with DAPPROX and SDECOM. These problems seem to have relatively complete recourse. Stochfor2 is a random recourse problem, therefore DAPPROX and SDECOM could not be applied.

On this test battery, the general–purpose LP solvers HOPDM and OSL have clearly been outperformed by BPMPD and MSLiP. SDECOM had relatively large fluctuations in the computing time across test problems with a small number of realizations, but outperformed OSL for the number of realizations 256.

6.2. Scaling up

In this section we present our computational results concerning complete recourse problems with an increasing number of joint realizations of the random vector ξ.

Battery 2: *Trier–V*

1. By utilizing the test problem generator facility (GENSLP) of SLP–IOR we have generated a test problem battery consisting of 10 complete recourse problems with $m_1 = 10, n_1 = 20$, $m_2 = 5$, $n_2 = 10$. There are 5 independent random variables ($r = 5$) which correspond directly to the RHS components of the second stage (unit arrays in (3)) and only the RHS is stochastic. The nonzero density of the matrices A, T and W is 10%.
2. Each of the 10 test problems have been endowed with a normally distributed RHS, with expected value equal to the randomly generated RHS components and standard deviation being 10% of the magnitude of the expected value.
3. The distribution has been discretized with increasing number of realizations thus resulting in a sequence of 10 test problem batteries each consisting of 10 test problems leading altogether to 100 test problems. The number of coordinatewise subintervals were 4, 5, 6, 7, 8, 10, 15, 20, 50 and 100.

As a result, 10 test problem batteries arised, containing altogether 100 test problems.

For the solvers BPMPD, DAPPROX the stopping tolerance has been chosen as $\varepsilon = \varepsilon_{\text{DAPPROX}} = 0.001$ (for DAPPROX see (7)). For these solvers this assures consistent solution quality concerning the solution time.

For SDECOM the maximum sample size has been 5000. We have made no attempt to ensure consistent quality of the solution delivered by SDECOM, w.r. to the other solvers. As mentioned in the Introduction, comparing the performance of sample–based solvers with deterministic solvers is beyond the scope of this paper. For SDECOM we have chosen the best parameter setting we have found so far; the computing times displayed in the Tables just serve for illustrative purposes.

The computational times are summarized in Table 3. In this table, the first column contains the number of joint realizations of ξ for the corresponding test problem battery consisting of 10 test. The other columns contain the average solution time for the 10 test problems in the corresponding battery.

Table 3 Battery Trier–V: average computing time in seconds for the batteries. Blank fields indicate that the solver has terminated in the setup phase due to memory limitation

#realiz.	Qdecom	BPMPD	Dapprox	Sdecom
1024	1.1	1.5	1.0	10.1
3125	10.1	6.1	1.9	9.0
7776	53.9	18.9	2.9	6.3
16807	253.2		3.7	7.4
32768	†		5.5	5.2
100000	†		9.6	5.4
759000			16.6	12.1
3200000			44.8	9.6
312500000			146.2	9.7
10^{10}			139.0	13.0

BPMPD has ran out of available memory for 16807 realizations; for QDECOM available memory has been exceeded for 759000 realizations. The symbol † means that although QDECOM succeeded in setting up the problem, the computing time turned out to be exceedingly high in comparison with the solution times of DAPPROX and SDECOM. For the 32768–realizations–battery the average solution time for QDECOM, for the first 7 problems, has been ∼17.8 minutes. After this the run has been aborted. Considering the goals of this paper, we found it pointless to solve these batteries with QDECOM.

For SDECOM the maximum sample size has been exceeded for 2 test problems in 7 batteries, and for 3 problems in the last battery. This is most probably due to the out of date stopping rule, see the remark above.

For DAPPROX, with the last battery corresponding to 10^{10} realizations, the maximum number for subintervals has been exceeded for 1 problem and the master problem became too large for Minos for another test problem. The latter run is not included in the average.

Note, that QDECOM and BPMPD, both being based on the equivalent LP, are already out of business for a not too high number of realizations. The SLP problems in the lower part of Table 3 are far beyond the capabilities of the LP–based approach, probably even for more powerful computers than our PC. DAPPROX and SDECOM works fine also with huge numbers of realizations. The type of problems represented by the battery Trier–V are, according to our experience also with other test problem batteries, easy problems for these solvers.

Table 4 shows the average computing time for DAPPROX, with increasing required accuracy. We have considered the accuracy range between 0.1% and 5%, which should be sufficient for most SLP applications. The Table shows that, for this battery, a significant speedup occurs when less accurate solutions are required. The reason is clear: for the successive discrete approximation method a lower required accuracy implies that usually a coarser subdivision is already sufficient.

Table 5 shows the average computing time for QDECOM and BPMPD, again with increasing required accuracy. The solvers are indicated by appending "Q" and "B" to the number of realizations, respectively. Let us observe, that the required accuracy has no significant impact on solver performance, at least in the range, which we consider.

Battery 3: *Trier–D*

The differences between this battery and Trier–V are as follows. Both T and h are stochastic and the arrays in the affine sums (3) are randomly generated. The nonzero pattern in the terms for $T(\xi)$ in (3) is the same for each term. For $h(\xi)$ they have 100% nonzero density and for

Table 4 Battery Trier–V: average computing time in seconds for DAPPROX, with increasing required accuracy

ε	0.05	0.01	0.005	0.001
1024	0.3	0.7	0.8	1.0
3125	0.4	0.9	1.1	1.9
7776	0.5	1.4	1.7	2.9
16807	0.6	1.5	2.2	3.7
32768	0.7	2.3	3.0	5.5
100000	1.2	3.7	5.1	9.6
759000	1.3	5.9	8.0	16.6
3200000	2.5	12.5	19.8	44.8
312500000	3.4	33.1	55.9	146.2
10^{10}	3.8	38.0	74.8	139.0

Table 5 Battery Trier–V: average computing time in seconds for QDECOM and BPMPD, with increasing required accuracy

ε	0.05	0.01	0.005	0.001
1024Q	0.9	1.0	1.0	1.1
3125Q	8.7	9.2	9.7	10.1
7776Q	49.1	50.1	51.9	53.9
1024B	1.3	1.3	1.4	1.5
3125B	5.5	5.7	5.7	6.1
7776B	17.7	18.0	18.4	18.9

Table 6 Battery Trier–D: average computing time in seconds. Blank fields indicate that the solver has terminated in the setup phase due to memory limitation

No. of joint realiz.	Qdecom	BPMPD	Dapprox	Sdecom
1024	1.0	2.2	1.8	5.0
3125	9.9	9.7	3.9	13.6
7776	55.2	28.0	6.5	13.8
16807	282.5		11.8	21.7
32768	†		21.7	17.8
100000	†		56.8	10.9
759000			171.7	17.0

$T(\xi)$ the nonzero density of the terms is 40%. This way the model instances contain altogether 25 highly dependent random entries, with stochastic interdependence between $T(\xi)$ and $h(\xi)$.

This battery contains altogether 70 test problems.

The stopping tolerance for QDECOM, BPMPD, and DAPPROX has been chosen as $\varepsilon = 0.01$. The maximum sample size for SDECOM has been 5000; concerning this solver the same comments apply, as for the previous battery.

The mean computing times are displayed in Table 6. The meaning of the † symbols is the same as for Table 3.

Table 7 displays the computing time for different values of required accuracy. Running times marked with "*" or double "*" mean, that for the corresponding battery and precision DAPPROX has run out of available memory for one or two test problems, respectively. The aborted runs are not included into the average. The symbol † indicates that we have not run DAPPROX on the battery with the largest test problems and the highest accuracy requirement, for obvious reasons.

Comparing the results to those for the previous battery, we observe that for DAPPROX the problems in this battery turn out to be definitely more difficult. This corresponds to our general experience, that SLP problems, having random entries also in the technology

Table 7 Battery Trier–D: average computing time in seconds for DAPPROX, with increasing required accuracy

ε	0.05	0.01	0.005	0.001
1024	0.6	1.8	2.7	5.1
3125	1.2	3.9	5.5	11.6
7776	1.6	6.5	11.2	22.5
16807	2.9	11.8	19.1	44.8
32768	3.6	21.7	32.0	92.0
100000	9.0	56.8	53.5*	163.9**
759000	17.7	171.7	33.1**	†

Table 8 Battery BerlCR: computing time in seconds for Dapprox, with increasing required accuracy

ε	0.05	0.01	0.005
min	0.64	10.61	19.0
max	10.7	144.31	394.2
mean	5.9	63.24	132.3
sdev	3.5	46.21	113.5

matrix, which stochastically depend on random variables in the right–hand–side, are usually numerically quite difficult.

Battery 4: *BerlCR*

This battery has been generated in the same manner as Trier–V. Instead of showing what happens when increasing the number of realizations, it serves for illustrating the effect of a large numbers of random entries in $T(\xi)$.

The battery consists of 10 test problems. The dimensions are: $m_1 = 50, n_1 = 100, m_2 = 5$, $n_2 = 10$. The nonzero density of the matrices A, T and W is 10%. There are 5 independent random variables ($r = 5$), the number of joint realizations is 32768. Both T and h are stochastic and the arrays in the affine sums (3) are randomly generated with varying nonzero pattern in the terms for $T(\xi)$ in (3). For $h(\xi)$ they have 100% nonzero density and for $T(\xi)$ the nonzero density of the terms is 20%. The model instances generated this way contain altogether ~ 340 (this number slightly varies across test problems in the battery) highly dependent random entries with stochastic interdependence between $T(\xi)$ and $h(\xi)$.

We present computational results with Dapprox, with different values of the stopping tolerance ε. Table 8 shows the basic statistics concerning the solution time with DAPPROX.

7. Computational results: simple recourse

According to our experience, simple recourse problems are not substantially more difficult, when solving them with SRAPPROX, than the corresponding expected value LP problems (4). The runs below have been carried out with the stopping tolerance (see (7)) $\varepsilon = \varepsilon_{\text{SRAPPROX}} = 10^{-6}$.

Battery 5: *SR1D*

This is a randomly generated test problem battery consisting of 10 problems. The dimensions are $m_1 = 200$, $n_1 = 400$, $m_2 = 90$, $n_2 = 180$. The nonzero density is both for A and for T 2%. We have $h(\xi) = \xi$, $r = 90$, there are 90 independent random entries in the RHS. The battery has been generated similarly as Trier–V, by first injecting a normal distribution and discretizing afterwards with 10 subintervals componentwise. The resulting discretely

Table 9 Battery SR1D: computing time in seconds

min	1.0
max	1.5
mean	1.2
sdev	0.2

Table 10 Battery Trier–G: computing time in seconds

min	26.3
max	32.6
mean	29.0
sdev	1.9

distributed RHS has 10^{90} joint realizations. Table 9 shows the computing time statistic with SRAPPROX; sdev stands for standard deviation.

Battery 6: *Trier–G*

The battery contains 10 randomly generated problems, generated in the same way as SR1D. The dimensions are: $m_1 = 200, n_1 = 500, m_2 = 300, n_2 = 600$; The nonzero density for both A and for T is 3%. For the random right–hand–side $h(\xi) = \xi$ holds, with $r = 300$, i.e., the random RHS contains 300 independent random variables with the astronomical number of 10^{300} joint realizations. The computing time statistic with SRAPPROX is shown in Table 10.

Comparing both tables, the successive approximation method seems to be quite insensitive both with respect to increasing the number of random variables and concerning the increase in the number of joint realizations.

The discrete approximation algorithm starts by solving the expected value problem (4). In the runs with our batteries and SRAPPROX, the solution time for the expected value problem makes out about 99% of the overall solution time for the simple recourse problem.

Let us remark that multiple simple recourse models, introduced and first studied by KleinHaneveld (1986) behave quite similarly when applying the transformation method of van der Vlerk (2002) and afterwards applying SRAPPROX. The reason is the insensitivity of SRAPPROX with respect to an increase in the number of realizations.

8. Computational results: Joint chance constraint

In Mayer (2000) computational results have been published for the test problem batteries CAM and CBJ, which have been obtained, with one exception, in the same computational environment as that reported in this paper. The exception is the Fortran compiler as specified in Section 4. The stopping tolerance for PROBALL had in Mayer (2000) the default value 10^{-5}. We present computational results for the battery CAM with different required accuracy levels. For the other two batteries in this section, we have chosen $\varepsilon = \varepsilon_{\text{PROBALL}} = 0.01$ (1%) in the stopping rule (7), which may be appropriate in many situations.

Battery 7: *CAM*

This is a randomly generated test problem battery containing 10 problems with dimensions: $m_1 = 80$, $n_1 = 160$, $m_2 = 20$; A and T have nonzero densities 10% and 5%, respectively. The random RHS is 20–dimensional; the deterministic part is inequality constrained and 10 of the deterministic constraints are active at the solution; $\alpha = 0.9$. The average computing

Table 11 Battery CAM: computing time in seconds

ε	0.05	0.01	0.005	0.001
min	2.1	2.0	2.1	2.1
max	26.3	26.4	86.3	86.4
mean	5.3	5.5	29.7	48.7
sdev	7.4	7.4	34.8	27.2

Table 12 Battery CBJ: computing time in seconds

min	7.0
max	478.3
mean	79.7
sdev	158.5

Table 13 Battery BERJC: computing time in seconds

min	7.5
max	558.4
mean	111.0
sdev	180.4

time reported in Mayer (2000) is 157 sec. Table 11 shows the basic statistics concerning the computational time for PROBALL.

Battery 8: *CBJ*

The battery has been randomly generated and has similar characteristics as CAM. The dimensions are $m_1 = 50$, $n_1 = 120$, $m_2 = 30$; A and T both have density 5%. The random RHS is 30–dimensional; $\alpha = 0.9$. The average computing time in Mayer (2000) is 19.13 min, ~ 1140 sec. The computational results with PROBALL are shown in Table 12.

Comparing the computational times for CAM and CBJ with those reported in Mayer (2000) one can observe that the computational time has dramatically reduced when lower accuracy was taken. Part of this reduction might however be attributed to the new Fortran compiler.

Battery 9: *BERJC*

The battery is randomly generated in a similar manner as the previous batteries. The dimensions in the deterministic part (and consequently also for T) are slightly increased: $m_1 = 100$, $n_1 = 200$, $m_2 = 30$; the random RHS is 30–dimensional; $\alpha = 0.9$. The results with PROBALL are displayed in Table 13.

9. Conclusions

For two–stage complete recourse problems we have identified two subclasses, for which the successive discrete approximation method turns out to be an efficient solution method for large–scale problems.

The first class consists of SLP problems with a relatively low dimensioned random vector ξ, with stochastically independent components. For this class large scale means a large number of joint realizations. We have supported this finding by presenting computational results for altogether 180 test problems, grouped into 3 test problem batteries. In these test problems the dimension of ξ is 5, the highest number of joint realizations is 10^{10}. According

to our experience, which is not documented in this paper, the discrete approximation solver can successfully be utilized for this class of problems up to a 10–dimensional random vector ξ on our PC. Scale up properties of algorithms are particularly interesting and useful as progressively larger models are investigated for real applications. An examination of Tables 3 and 6 clearly indicates that the structure exploiting algorithms DAPPROX and SDECOM have the desirable scale up properties. This is in contrast with the deterministic equivalent solvers. Other investigators (Valente, Mitra, and Poojari, 2005) also report comparable scale up results. We have also illustrated for this problem class, that solvers aiming at the solution of the equivalent LP become inappropriate from a certain number of realizations upwards. The reason is that, given the limited storage capacity of the PC, either the equivalent LP cannot be set up at all, or the solver runs out of available storage. Our results also show, that the computing time dramatically reduces when requiring lower accuracy for the solution.

The second class consists of simple recourse problems. This type of problems turned out to be as not substantially more difficult to solve, than the expected value problem. According to our experience, this holds when both the number of random variables and the number of joint realizations are large. We have illustrated this point by computational results for two batteries, each of which containing 10 test problems. The dimension of ξ for the batteries is 90 and 300, with number of joint realizations 10^{90} and 10^{300}, respectively.

For jointly chance constrained problems we have considered the case, when only the right–hand–side is stochastic and the distribution is non–degenerate multivariate normal. We have illustrated, that using a central cutting algorithm, such problems can be solved on our PC with up to 30–dimensional ξ. Similarly to two–stage problems, the computing time reduces significantly, when requiring lower accuracy. We have solved altogether 30 test problems, grouped into three batteries, with the dimension of ξ being 20, 30, and 30, respectively.

In presenting our computational results, we have confined ourselves to presenting the computing time. Our findings are fully reproducible and our model management system SLP–IOR, which has been utilized for obtaining the computational results, is freely available for academic purposes.

References

Ariyawansa, K.A. and A.J. Felt. (2001). "On a New Collection of Stochastic Linear Programming Test Problems." WEB–Publication at www.optimization-online.org.

Birge, J.R. (1985). "Decomposition and Partitioning Methods for Multistage Stochastic Linear Programs." *Oper. Res.* 33, 989–1007.

Birge, J.R., M.A.H. Dempster, H. Gassmann, E. Gunn, A.J. King, and S.W. Wallace. (1987). "A Standard Input Format for Multiperiod Stochastic Linear Programs." Working Paper WP-87-118, IIASA, Laxenburg, Austria.

Birge, J.R. and F. Louveaux. (1997). *Introduction to Stochastic Programming*. Springer-Verlag.

Brooke, A., D. Kendrick, A. Meeraus, and R. Raman. (1998). GAMS. A user's guide. Technical report, GAMS Development Corporation, Washington DC, USA. It can be downloaded from http://www.gams.com/docs.

Consigli, G. and M.A.H. Dempster. (1998). "The CALM Stochastic Programming Model for Dynamic Asset–Liability Management." In W.T. Ziemba and J.M. Mulvey (eds.), *Worldwide asset and liability management*, pp. 464–500. Cambridge University Press.

Dempster, M.A.H., N. Hicks Pedron, E. Medova, J.E. Scott, and A. Sembos. (2000). "Planning Logistic Operations in the Oil Industry." *Journal of Operational Research Society* 51, 1271–1288.

Frauendorfer, K. and P. Kall. (1988). "A Solution Method for SLP Recourse Problems with Arbitrary Multivariate Distributions—the Independent Case." *Problems of Control and Information Theory* 17, 177–205.

Gassmann, H.I. (1989). "Optimal Harvest of a Forest in The Presence of Uncertainty." *Canadian Journal of Forest Research* 19, 1267–1274.

Gassmann, H.I. (1990). "MSLiP: A Computer Code for the Multistage Stochastic Linear Programming Problem." *Math. Prog.* 47, 407–423.

Gondzio, J. (1995). "HOPDM (version 2.12) – A Fast LP Solver Based on a Primal–Dual Interior Point Method." *Eur. J. Oper. Res.* 85, 221–225.

Higle, J.L. and S. Sen. (1991). "Stochastic Decomposition: An Algorithm for Two-Stage Linear Programs with Recourse." *Math. of Oper. Res.* 16, 650–669.

Higle, J.L. and S. Sen. (1996). "Stochastic Decomposition. A Statistical Method for Large Scale Stochastic Linear Programming." Kluwer Academic Publ.

Kall, P. (1974). "Approximations to Stochastic Programs with Complete Fixed Recourse." *Numer. Math.* 22, 333–339.

Kall, P. (1988). "Stochastic Programming with Recourse: Upper Bounds and Moment Problems – a Review." In J. Guddat, B. Bank, H. Hollatz, P. Kall, D. Klatte, B. Kummer, K. Lommatzsch, K. Tammer, M. Vlach, and K. Zimmermann (eds.), *Advances in Mathematical Optimization (Dedicated to Prof. Dr.hc. F. Nožička)*, pp. 86–103. Akademie-Verlag, Berlin.

Kall, P. (1998). "Bounds for and Approximations to Stochastic Linear Programs with Recourse –Tutorial." In K. Marti and P. Kall (eds.), *Stochastic Programming Methods and Technical Applications*, pp. 1–21. Springer-Verlag.

Kall, P. and J. Mayer. (1992). "SLP-IOR: A Model Management System for Stochastic Linear Programming—System Design." In A.J.M. Beulens and H.-J. Sebastian (eds.), *Optimization-Based Computer-Aided Modelling and Design*, pp. 139–157. Springer-Verlag.

Kall, P. and J. Mayer. (1992). "A Model Management System for Stochastic Linear Programming." In P. Kall (ed.), *System Modelling and Optimization*, pp. 580–587. Springer-Verlag.

Kall, P. and J. Mayer. (1996) "SLP–IOR: An Interactive Model Management System for Stochastic Linear Programs." *Math. Prog.* 75, 221–240.

Kall, P. and J. Mayer. (1998). "On Testing SLP Codes with SLP–IOR." In F. Giannessi, T. Rapcsák, and S. Komlósi (eds.), *New Trends in Mathematical Programming*, pp. 115–135. Kluwer Academic Publ.

Kall, P. and J. Mayer. (2005). "Building and Solving Stochastic Linear Programming Models with SLP–IOR." In S.W. Wallace and W.T. Ziemba (eds.), *Applications of Stochastic Programming*. MPS–SIAM–Series in Optimization, pp. 79–93.

Kall, P. and D. Stoyan. (1982), "Solving stochastic programming problems with recourse including error bounds." *Math. Operationsforsch. Statist., Ser. Opt.* 13, 431–447.

Kall, P. and S.W. Wallace. (1994). *Stochastic Programming*. John Wiley & Sons.

Klein Haneveld, W. (1986). *Duality in Stochastic Linear and Dynamic Programming*. Springer-Verlag, Lecture Notes in Economics and Mathematical Systems 274.

Laporte, G., F.V. Louveaux, and H. Mercure. (1992). "The Vehicle Routing Problem with Stochastic Travel Times." *Transportation Science* 26, 161–170.

Linderoth, J. and S. Wright. (2002). "Decomposition Algorithms for Stochastic Programming on a Computational Grid." Optimization Technical Report 02–07, Computer Sciences Department, University of Wisconsin–Madison.

Linderoth, J., A. Shapiro, and S. Wright. (2002). "The Empirical Behavior of Sampling Methods for Stochastic Programming." Optimization Technical Report 02–01, Computer Sciences Department, University of Wisconsin–Madison.

Maros, I. and Cs. Mészáros. (1998). "The Role of the Augmented System in Interior Point Methods." *European Journal of Operational Research* 107(3), 720–736.

Mayer, J. (1998). *Stochastic Linear Programming Algorithms: A Comparison Based on a Model Management System*. Gordon and Breach.

Mayer, J. (2000). "On the Numerical Solution of Jointly Chance Constrained Problems." In S. Uryasev (ed.), *Probabilistic Constrained Optimization: Methodology and Applications*, pp. 220–233. Kluwer Academic Publ.

Cs. Mészáros. (1997). "The Augmented System Variants of IPM's in Two Stage Stochastic Programming." *Eur. J. Oper. Res.* 101, 317–327.

MirHassani, S.A., C.A. Lucas, G. Mitra, E. Messina, and C.A. Poojari. (2000). "Computational Solution of Capacity Planning Models Under Uncertainty." *Parallel Computing* 26, 511–538.

Mulvey, J.M. and A. Ruszczyński. (1995). "A New Scenario Decomposition Method for Large–Scale Stochastic Optimization." *Oper. Res.* 43, 477–490.

Prékopa, A. (1995). *Stochastic Programming*. Kluwer Academic Publ.

Ruszczyński, A. (1986). "A Regularized Decomposition Method for Minimizing a Sum of Polyhedral Functions." *Math. Prog.* 35, 309–333.

Sen, S., R.D. Doverspike, and S. Cosares. (1994). "Network Planning with Random Demand." *Telecommunications Systems* 3, 11–30.

Shapiro, A. and T. Homem–de–Mello. (2000). "On the Rate of Convergence of Optimal Solutions of Monte Carlo Approximations of Stochastic Programs." *SIAM J. Opt.* 11, 70–86.
Szántai, T. (1987). "Calculation of the Multivariate Probability Distribution Function Values and their Gradient Vectors." Working Paper WP-87-82, IIASA.
Valente, P., G. Mitra, and C.A. Poojari. (2005). "A Stochastic Programming Integrated Environment (SPInE)." In S.W. Wallace and W.T. Ziemba (eds.), *Applications of stochastic programming*, MPS–SIAM–Series in Optimization, pp. 115–136.
Van Slyke, R., and R.J-B. Wets. (1969). "L–Shaped Linear Program with Applications to Optimal Control and Stochastic Linear Programs." *SIAM J. Appl. Math.* 17, 638–663.
Van der Vlerk, M.H. (2002). "On Multiple Simple Recourse Models." SOM Research Report 02A06, University of Groningen.

Web References

www-neos.mcs.anl.gov/neos
www.gams.com
users.iems.nwu.edu/∼jrbirge//html/dholmes/post.html
www.uwsp.edu/math/afelt/slptestset.html
www-cfr.jims.cam.ac.uk/research/stprog.html
www.optimization-online.org