Abstract

Multi-stage stochastic linear programs arise in many operational research situations, when some optimal plan is sought subject to uncertainty on some or all of the problem data. When expressed as deterministic equivalent models, these problems result in very large scale linear programs, even for a relatively coarse representation of the probability distributions governing the parameters. In this paper we review the research being carried out at Auckland into solution techniques for solving these large-scale problems. We describe some recent experiments with interior-point algorithms and stochastic Benders' decomposition. The test problems we use come from long-term hydro-electricity generation planning.

1 Introduction

In recent years there has been renewed interest in mathematical programming models which have parameters subject to uncertainty. The importance of these stochastic programming models has always been recognized due to their wide applicability (see e.g. [5] for a survey), but their practical use has been limited by the difficulty of obtaining optimal solutions. The rapid increases in computing power available to operations researchers has resulted in the development of a number of new algorithmic approaches to solving stochastic programming problems. In this paper we report on some of the research which we are carrying out at the University of Auckland into new computational techniques for these problems.

For a comprehensive introduction to the field of stochastic programming we recommend the monograph [4]. Here we give a brief (and incomplete) account starting with the two-stage stochastic programming problem with recourse. This model arises when a first-stage decision must be made subject to uncertainty in the parameters, followed by a recourse decision at a later time when all the uncertainty has been resolved and the decision maker is assumed to have perfect information. We shall assume that the uncertainty is defined by a random variable
ξ, which has a finite number of realizations ξ_k, k = 1, 2, ..., N, each with probability p_k, k = 1, 2, ..., N. Each of these realizations gives a different set of matrices \{T_{2k}, W_{2k}, d_{2k}, b_{2k}\}.

The two-stage stochastic programming problem with recourse has the following form:

\[
\text{TSSP: } \text{minimize } \mathbf{c}_1^T x_1 + \mathcal{Q}(x_1, \xi) \\
\text{subject to } W_1 x_1 = b_1, \\
x_1 \geq 0,
\]

where

\[
\mathcal{Q}(x_1, \xi) = \sum_{k=1}^{k=N} p_k \mathcal{Q}(x_1, \xi_k),
\]

and

\[
\mathcal{Q}(x_1, \xi_k) = \min \{c_2^T x_{2k} \mid W_{2k} x_{2k} = b_{2k}, T_{2k} x_1, x_{2k} \geq 0\}.
\]

Here \(x_1\) is the set of first-stage decisions and \(x_{2k}, k = 1, 2, ..., N\) are the recourse decisions. The objective function \(c_1^T x_1 + \mathcal{Q}(x_1, \xi)\) measures the expected cost of the plan.

The two-stage stochastic programming problem with recourse can be expressed as a deterministic equivalent linear problem:

\[
\text{TSDE: } \text{minimize } \mathbf{c}_1^T x_1 + \sum_{k=1}^{k=N} p_k c_2^T x_{2k} \\
\text{subject to } W_1 x_1 = b_1, \\
T_{2k} x_1 + W_{2k} x_{2k} = b_{2k}, \quad k = 1, 2, ..., N \\
x_1 \geq 0, \quad x_{2k} \geq 0, \quad k = 1, 2, ..., N
\]

In most practical situations, all uncertainty is not resolved prior to the recourse decisions being made, and the second-stage problems of determining \(\mathcal{Q}(x_1, \xi_k)\) for each \(k\) must account for the expected cost of later recourse decisions. This results in the multi-stage stochastic program with recourse. For convenience we shall illustrate this using a three-stage model. We obtain:

\[
\text{MSSP: } \text{minimize } \mathbf{c}_1^T x_1 + \mathcal{Q}(x_1, \xi) \\
\text{subject to } W_1 x_1 = b_1, \\
x_1 \geq 0,
\]

where

\[
\mathcal{Q}(x_1, \xi) = \sum_{k=1}^{k=N} p_k \mathcal{Q}(x_1, \xi_k);
\]
Now the number of scenarios for MSSP is equal to \( N_1 N_2 \). The general multi-stage stochastic program with recourse has a deterministic equivalent form which is difficult to write down in a simple form. We again restrict attention to a three stage model:

**MSDE:**
\[
\text{minimize } \quad c_1^T x_1 + \sum_{k=1}^{N_1} p_k \left( c_2^T x_2k + \sum_{l=1}^{N_2} q_l c_3^T x_3kl \right) \\
\text{subject to } \quad W_1 x_1 = b_1, \\
\quad T_2k x_1 + W_2k x_2k = b_2k, \\
\quad S_{3kl} x_1 + T_{3kl} x_2k + W_{3kl} x_3kl = b_{3kl}, \\
\quad x_1 \geq 0, \quad x_2k \geq 0, \quad x_{3kl} \geq 0.
\]

Here \( k = 1, 2, \ldots N_1 \) and \( l = 1, 2, \ldots N_2 \). When \( N \) is large it easy to see that the deterministic equivalents of stochastic programs become very large linear programming problems. For example in the case of MSDE if \( W_1, T_2k, W_2k, S_{3kl}, T_{3kl}, \) and \( W_{3kl} \) are all 10 by 20 matrices, a model with \( N = 1 \) over three time periods would have 30 constraints and 60 variables. By contrast, if \( \xi_k \) and \( \xi_l \) are random vectors with 10 components, each of which has 2 possible realizations then \( N_1 = N_2 = 2^{10} = 1024 \), and MSDE has \( 10 \times 1024^2 + 10 \times 1024 + 10 = 10,496,010 \) constraints, and twice as many variables.

Such large problems present a daunting prospect for mathematical programming software, and most recent approaches have favoured some form of decomposition (see [4]). It is generally recognized in the mathematical programming folklore that because of very efficient sparse matrix techniques, decomposition algorithms for linear programming are slower than direct methods, when it is possible to store the entire basis matrix in memory. It is also becoming accepted that primal-dual interior point methods for linear programming will outperform the simplex method when applied to large problems.

Our purpose in this paper is to investigate to what extent these suppositions are true, by applying the methods to instances of MSSP of increasing size. This was achieved by taking a multi-stage model of a single-reservoir long-term hydroelectricity generation problem and systematically increasing the number of decision stages. These models are described in detail in the paper [3]. The problems in each stage have eight constraints and nine variables, and the sizes of the problems we
Table 1: Test Problem sizes

<table>
<thead>
<tr>
<th>Problem id</th>
<th>decision stg</th>
<th>time prd</th>
<th>Ni</th>
<th>MSDE # var</th>
<th>MSDE # constr</th>
</tr>
</thead>
<tbody>
<tr>
<td>P1</td>
<td>3</td>
<td>4</td>
<td>3</td>
<td>117</td>
<td>104</td>
</tr>
<tr>
<td>P2</td>
<td>4</td>
<td>5</td>
<td>3</td>
<td>360</td>
<td>320</td>
</tr>
<tr>
<td>P3</td>
<td>5</td>
<td>6</td>
<td>3</td>
<td>1089</td>
<td>968</td>
</tr>
<tr>
<td>P4</td>
<td>6</td>
<td>7</td>
<td>3</td>
<td>3276</td>
<td>2912</td>
</tr>
<tr>
<td>P5</td>
<td>6</td>
<td>7</td>
<td>5</td>
<td>35,154</td>
<td>31,248</td>
</tr>
</tbody>
</table>

Figure 1: Test Problem sizes

generated are shown in Figure 1. In the next section of the paper we describe some experiments on MSDE with the barrier optimizer in CPLEX 3.0. The final section describes the method known as multi-stage stochastic Benders' decomposition, and the results of some experiments with this method in CPLEX.

2 Interior-point methods

The commercial linear programming code CPLEX 3.0 contains an option to solve linear programs using a primal-dual predictor-corrector interior-point method referred to in our tables as barrier. The implementation of this method in CPLEX is based on the paper by Mehrotra (see [7], [6]). Upon termination this yields an optimal solution to within some tolerance. A basic solution may be obtained from this by applying a purification scheme. This combination results in what is known as a hybrid barrier method which we denote by hybbar.

Figure 2 contains the results of our experiments with primal simplex, dual simplex, barrier and hybbar. The term real time refers to elapsed time and includes the time to load the problem. The CPU time taken by the solver is referred to as user time. The column headed sys time refers to time spent on solving the problem which is not counted in the CPU time. This is mainly composed of time devoted to accessing memory, and becomes large if the problem requires memory paging.

The number of iterations of the barrier method is low as expected, and one can see that when the problems reach the size of P3, the CPU time for barrier begins to overtake both the primal and dual simplex methods.
<table>
<thead>
<tr>
<th>Problem id</th>
<th>Method</th>
<th>real time</th>
<th>sys time</th>
<th>user time</th>
<th># of iterations</th>
</tr>
</thead>
<tbody>
<tr>
<td>P1</td>
<td>simplex</td>
<td>0.7</td>
<td>0.4</td>
<td>0.1</td>
<td>97</td>
</tr>
<tr>
<td>P1</td>
<td>barrier</td>
<td>0.7</td>
<td>0.4</td>
<td>0.1</td>
<td>11</td>
</tr>
<tr>
<td>P1</td>
<td>hybbar</td>
<td>0.7</td>
<td>0.4</td>
<td>0.1</td>
<td>–</td>
</tr>
<tr>
<td>P1</td>
<td>dual</td>
<td>0.7</td>
<td>0.4</td>
<td>0.1</td>
<td>68</td>
</tr>
<tr>
<td>P2</td>
<td>simplex</td>
<td>0.9</td>
<td>0.4</td>
<td>0.3</td>
<td>337</td>
</tr>
<tr>
<td>P2</td>
<td>barrier</td>
<td>0.9</td>
<td>0.4</td>
<td>0.3</td>
<td>14</td>
</tr>
<tr>
<td>P2</td>
<td>hybbar</td>
<td>1.0</td>
<td>0.4</td>
<td>0.4</td>
<td>–</td>
</tr>
<tr>
<td>P2</td>
<td>dual</td>
<td>0.9</td>
<td>0.4</td>
<td>0.2</td>
<td>205</td>
</tr>
<tr>
<td>P3</td>
<td>simplex</td>
<td>1.9</td>
<td>0.5</td>
<td>1.2</td>
<td>1065</td>
</tr>
<tr>
<td>P3</td>
<td>barrier</td>
<td>1.7</td>
<td>0.5</td>
<td>1.0</td>
<td>16</td>
</tr>
<tr>
<td>P3</td>
<td>hybbar</td>
<td>1.9</td>
<td>0.5</td>
<td>1.1</td>
<td>–</td>
</tr>
<tr>
<td>P3</td>
<td>dual</td>
<td>3.0</td>
<td>0.5</td>
<td>1.2</td>
<td>634</td>
</tr>
<tr>
<td>P4</td>
<td>simplex</td>
<td>9.4</td>
<td>0.7</td>
<td>8.3</td>
<td>3350</td>
</tr>
<tr>
<td>P4</td>
<td>barrier</td>
<td>4.6</td>
<td>0.6</td>
<td>3.8</td>
<td>20</td>
</tr>
<tr>
<td>P4</td>
<td>hybbar</td>
<td>5.2</td>
<td>0.7</td>
<td>4.2</td>
<td>–</td>
</tr>
<tr>
<td>P4</td>
<td>dual</td>
<td>8.3</td>
<td>0.7</td>
<td>7.3</td>
<td>1871</td>
</tr>
<tr>
<td>P5</td>
<td>simplex</td>
<td>1046</td>
<td>3.8</td>
<td>949</td>
<td>34577</td>
</tr>
<tr>
<td>P5</td>
<td>barrier</td>
<td>103</td>
<td>2.4</td>
<td>98</td>
<td>40</td>
</tr>
<tr>
<td>P5</td>
<td>hybbar</td>
<td>123</td>
<td>3.6</td>
<td>110</td>
<td>–</td>
</tr>
<tr>
<td>P5</td>
<td>dual</td>
<td>820</td>
<td>3.5</td>
<td>803</td>
<td>20813</td>
</tr>
</tbody>
</table>

Figure 2: Timings for MSDE (seconds).

3 Stochastic Benders’ Decomposition

The *L-Shaped* method was developed by VanSlyke and Wets [8] for two-stage stochastic linear programming following a similar decomposition technique for integer programming due to Benders[1]. Thus it is also referred to as *Stochastic Benders’ Decomposition* or SBD. This method was extended to multi-stage problems by Birge (see e.g. [2]).

There are many clear accounts of this technique and its properties (see e.g. [4] or [3]). We shall give an algorithm description for three stages only. For simplicity we assume that all the matrices $S_{3k\ell}$ are zero.

Multi-Stage Stochastic Benders’ Decomposition

1. Set $F_1 = F_{2k} = \mathbb{R}^n \times \{-M\}$ where $M$ is some large number.

2. Solve

\[
P(1): \quad \text{minimize} \quad c_1^T x_1 + \theta_1 \\
\text{subject to} \quad W_1 x_1 = b_1,
\]
\[ (x_1, \theta_1) \in F_1, \]
\[ x_1 \geq 0, \]

to give optimal primal variables \((\bar{x}_1, \bar{\theta}_1)\).

3. For \( k = 1 \) to \( N_1 \) solve

\[
P(2, k): \quad \text{minimize} \quad c_k^T x_{2k} + \theta_{2k} \]
\[
\text{subject to} \quad W_{2k} x_{2k} = b_{2k} - T_{2k} x_1,
\]
\[ (x_{2k}, \theta_{2k}) \in F_{2k}, \]
\[ x_{2k} \geq 0, \]

to give optimal primal variables \((\bar{x}_{2k}, \bar{\theta}_{2k})\).

4. For \( k = 1 \) to \( N_1 \), \( l = 1 \) to \( N_2 \) solve

\[
P(3, k, l): \quad \text{minimize} \quad c_{3kl}^T x_{3kl} \]
\[
\text{subject to} \quad W_{3kl} x_{3kl} = b_{3kl} - T_{3kl} x_{2k},
\]
\[ x_{3kl} \geq 0, \]

to give optimal primal variables \(\bar{x}_{3kl}\) and dual variables \(\bar{\pi}_{3kl}\).

5. \( F_{2k} = F_{2k} \cap \{(x_{2k}, \theta_{2k}) \mid \sum_{i=1}^{N_2} q_i \bar{\pi}_{3ki}^T (b_{3ki} - T_{3ki} x_{2k}) \leq \theta_{2k}\} \)

6. For \( k = 1 \) to \( N_1 \) solve \( P(2, k) \) to give optimal dual variables \(\bar{\pi}_{2k}\).

7. \( F_1 = F_1 \cap \{(x_1, \theta_1) \mid \sum_{k=1}^{N_1} p_k \bar{\pi}_{2k}^T (b_{2k} - T_{2k} x_1) \leq \theta_1\} \)

8. If converged then stop; otherwise go to step 2

The convergence criterion is usually determined by the gap between the upper bound \( c_1^T \bar{x}_1 + \sum_{k=1}^{N_1} p_k [c_k^T \bar{x}_{2k} + \sum_{i=1}^{N_2} q_i \bar{\pi}_{3ki}^T (b_{3ki} - T_{3ki} \bar{x}_{2k})] \) on the optimal value of MSSP, and a lower bound given by \( c_1^T \bar{x}_1 + \bar{\theta}_1 \). The problems \( P(1) \), \( P(2, k) \), \( P(3, k, l) \) form a tree structure. In our description a single pass of the algorithm above sends solutions from \( P(1) \) to each of its children \( P(2, k) \) and then solutions from these problems to each of their children. The problems \( P(3, k, l) \) then send back \( N_1 \) extra constraints, one for each problem \( P(2, k) \), which are then re-solved before passing a single constraint back to \( P(1) \). (Other strategies for passing information between stages are discussed in [3].

We have implemented this method using CPLEX 3.0 as the subproblem solver. The CPLEX subroutine library makes it straightforward to pass information between problems. Figure 3 gives the solution times for Multi-Stage SBD applied to
Problem id | real time | sys. time | user time | # of passes
---|---|---|---|---
P1 | 0.6 | 0.2 | 0.3 | 1
P2 | 2.3 | 1.0 | 1.1 | 5
P3 | 5.8 | 2.4 | 3.1 | 4
P4 | 25 | 10.5 | 13.8 | 7
P5 | 1680 | 214 | 168 | 9

Figure 3: Timings for Multi-Stage SBD (seconds)

the test problems. Observe that the system time grows quite rapidly with the number of stages in the problem because of the large communication overhead involved. It is interesting also to note that, for all problems except P5, solving a deterministic equivalent linear program is faster than applying the Multi-Stage SBD method. Thus when the underlying deterministic models are small, it makes sense to aggregate some of the problems to form larger deterministic equivalent problems. In the context of the three-stage model we might aggregate the problems $P(3, k, l)$, $l = 1, 2, \ldots, N_2$ so as to form a two stage model with each second stage problem being a deterministic equivalent for $P(2, k)$. We call this process stage aggregation.

Suppose that the deterministic equivalent problem takes $\tau$ seconds less CPU time than the Multi-Stage SBD method on a two-stage model of a given size. At the leaves of the scenario tree there are $L$ such problems to solve. One might expect a large saving in CPU time at each iteration of the algorithm but since each problem is solved from a warm start after the first pass, this savings estimate is optimistic. Assuming that the speedup is obtained only in the first solve gives a conservative estimate of $\tau L$ seconds improvement.

Problem id | real time | sys. time | user time | # of passes
---|---|---|---|---
P1 | 0.3 | 0.1 | 0.1 | 1
P2 | 0.8 | 0.3 | 0.3 | 2
P3 | 1.9 | 0.5 | 1.2 | 2
P4 | 9.1 | 3.3 | 5.4 | 6
P5 | 76 | 29 | 45 | 8

Figure 4: Timings for stage aggregation (seconds).

Figure 4 shows solution times for our test problems. These are solved using stage aggregation over the last two stages with CPLEX 3.0 as the subproblem solver. For problem P4, we have $L = 81$, and the saving is about 8 seconds indicating that $\tau$ is of the order of 0.1. For P5, $L = 625$, and the time saving is 120 seconds. It seems clear that when $L$ is large at the extremities of the scenario tree it makes sense to use stage aggregation. We note that some of the advantages shown in Figure 4 might be decreased by scenario bunching whereby basis information is distributed amongst similar problems in the same stage, although it is likely that will lead to
further system time overheads which are already becoming relatively large for the SBD method applied to P4 and P5.

We conclude with the observation that very fast solution times of the barrier methods raise the possibility that these might be profitably used as solvers as part of a stage-aggregation strategy in a Multi-Stage SBD. One attractive feature of feasible primal-dual methods is that one might terminate them prematurely and obtain feasible primal and dual solutions. These can be used in the Multi-Stage SBD without endangering the upper and lower bounding properties which are used to determine convergence, so this approach shows considerable promise for further investigation.

References


