ReSa: A method for solving multistage stochastic linear programs

Magnus Hindsberger *

*Australian Energy Market Operator, Brisbane, Australia

Abstract. This paper presents a new sampling scheme for solving large multistage stochastic linear programs using Benders decomposition. The approach is compared with two alternative sampling approaches by applying the methods to a large hydrothermal scheduling model with stochastic inflows. For this problem the new scheme performs better.

Keywords: stochastic programming; decomposition; sampling

Received November 2013. Accepted February 2014

Introduction

Multistage stochastic linear programming is a popular technique for solving convex optimisation problems involving planning under uncertainty, where stochastic outcomes are revealed over time. Using stochastic Benders decomposition allows large such problems to be solved. Benders decomposition was introduced by Benders (1962) within the context of mixed integer programming. It was discovered independently by van Slyke and Wets (1969), who extended it into two-stage stochastic linear programming. A multistage version was presented by Birge (1985).

Stochastic Benders decomposition approximates the future cost function at each stage by a piecewise affine function of the decision variables that is computed using the optimal dual variables computed from the linear programs that are solved in the immediately subsequent stage. Even with this approximation, many linear programs must be solved to solve such a problem, and recent efforts in this area have been focussed on using sampling to reduce this computational effort.

The first method to use sampling in a multistage stochastic programming application was the Stochastic Dual Dynamic Programming (SDDP) algorithm of Pereira and Pinto (1991). Donohue and Birge (2006) argued that SDDP is most suited for problems with many stages and few random outcomes per stage. They presented an alternative, the Abridged Nested Decomposition (AND) method, which outperformed SDDP for a problem with few stages and many possible random outcomes per stage. In this paper we investigate a variation of SDDP that we call Reduced Sampling (ReSa). This method incorporates some of the ideas from AND into a sampling scheme suitable for problems with many stages and few random outcomes per stage. A proof of convergence for all algorithms was given in Linowsky and Philpott (2005).

In this paper prior knowledge to the Benders decomposition technique has been assumed. However, the next section will briefly present the concepts of the method to ease the presentation of the three sampling methods,
SDDP, AND, and ReSa, which will be presented in the sections 3-5. Section 6 will present some computational results of the performance of the algorithms applied to a hydro reservoir optimisation problem while in section 7 some concluding remarks will be given.

**Benders decomposition**

This section will give a brief introduction to the stochastic Benders decomposition method. For details the reader can refer to the text by Birge and Louveaux (1997). It is a nested decomposition algorithm. For a problem with \( T \) stages, the stage one problem in the nested problem structure can be written as:

\[
P_1: \quad z = \text{minimize} \quad c_1 x_1 + Q_2(x_1) \quad (1)
\]

subject to

\[
A_1 x_1 = b_1 \quad (2)
\]

\[
x_1 \geq 0 \quad (3)
\]

with

\[
Q_2(x_{t-1}) = \sum_{\omega_{t,k} \in \Omega_t} p(\omega_{t,k}) Q_t(x_{t-1}, \omega_{t,k}) \quad (4)
\]

where \( p(\omega_{t,k}) \) is the probability of realisation of random outcome \( \omega_{t,k} \) from the finite set \( \Omega_t = [\omega_{t,1}, \ldots, \omega_{t,K}] \) and \( Q_t(x_{t-1}, \omega_{t,k}) \) is the optimal solution value for the stage \( t \) problem \( P_{t,k} \) given by:

\[
P_{t,k}: \quad Q_{t,k}(\hat{x}_{t-1}, \omega_{t,k}) = \text{minimize} \quad c_{t,k}(\omega_{t,k}) x_{t,k} + Q_{t+1}(x_{t,k}) \quad (5)
\]

subject to

\[
A_{t,k} x_{t,k} = b_{t,k}(\omega_{t,k}) - W_{t-1}(\omega_{t,k}) \hat{x}_{t-1} \quad (6)
\]

\[
x_{t,k} \geq 0. \quad (7)
\]

Here \( \hat{x}_{t-1} \) denotes a feasible solution to the problem \( P_{t-1,k} \). Observe the nested structure where a master problem at stage \( t-1 \) has a number of subproblems at stage \( t \), which each are master problems for a number of stage \( t+1 \) subproblems.

The Benders decomposition algorithm will during each iteration solve all subproblems for all stages, \( t \), and random realisations, \( k \). The algorithm will add linear constraints, denoted cuts, to the subproblems of all stages. Two types of cuts can be added; optimality cuts and feasibility cuts. An optimality cut is a linear approximation of \( Q_{t+1}(x_t) \), also denoted the future cost function, and is based on the dual variables of the (6) constraints of the stage \( t+1 \) problems. As such, each optimality cut provides a lower bound on \( Q_{t+1}(x_t) \). This is illustrated in Figure 1 where two cuts \( \delta_1 \) and \( \delta_2 \) together form an approximation of \( Q_{t+1}(x_t) \). It can be seen that the function is a piecewise linear, convex function of \( x_t \). In the following, it is assumed that \( Q_{T+1}(x_T) = 0 \).

![Fig. 1. An example of a future cost function (left) and an approximation by two cuts \( \delta_1 \) and \( \delta_2 \) (right)](image)
Feasibility cuts are used to penalize values of $x_t$, which otherwise would make subproblems at later stages infeasible. Throughout this paper we assume relatively complete recourse, meaning that a stage $t$ subproblem will have a feasible solution irrespective of the value of $\hat{x}_{t-1}$. Hence, no feasibility cuts will have to be considered.

Solving the stage 1 subproblem (1)-(3) with a set of cuts approximating $Q_2(x_1)$ will give a lower bound $z$ of the problem since this is a relaxation of the problem. Also, given a feasible solution vector $\hat{x}^* = \{x_1^*, ..., x_T^*\}$ to the problem (1)-(7), the objective function value, $z^*$, where

$$z^* = \sum_{t=1}^{T} c_t x_t^*$$

(8)

will form an upper bound $\bar{z}$ to the objective function value $z$ since it is a minimization problem. The Benders decomposition algorithm terminates when $\bar{z}$ equals $z^*$, in which case the problem is solved to optimality. Alternatively, the user can select to terminate the algorithm when the two are sufficiently close in terms of the quality of the solution required.

Problems with a large number of stages e.g. in combination with a large number of possible random outcomes at each stage will be time consuming to solve even with Benders decomposition. The scenario tree is too big. A scenario is here defined as a sequence of random outcomes leading from the stage 1 subproblem to a stage $t$ subproblem. A full scenario is scenario covering all $T$ stages. The scenario tree is then the set of all full scenarios. The methods described in the following sections are all designed for solving problems with large scenario trees.

**Stochastic dual dynamic programming**

The SDDP method, see Pereira and Pinto (1999) and Velásquez, Restrepo and Campo (1999), is based on Benders decomposition, but instead of solving all subproblems in the scenario tree at each iteration in order to calculate the cuts, only a subset of scenarios of length $T$ are sampled (independent identically distributed) and the associated subproblems solved. An important assumption in SDDP and similar sampling techniques (such as AND and ReSa) in relation to this is that serial independence exists. This allows cuts calculated from the solved stage $t+1$ subproblems to be shared amongst all stage $t$ subproblems regardless of whether they were sampled or not. Serial independence exists when the probabilities $p(o_{h,k})$ and the realisations of the random variables are independent of history, i.e. the realisations of $o_{h,k}$, $o_{h,k+1}$, $o_{h,k+2}$...

1. Initialization of parameters and dynamic sets

   **Forward pass:**
   2. Sample $S$ full scenarios forming the set $SS$
      
      \[
      \text{FOR } t=1 \text{ TO } T \\
      \text{FOR } s=1 \text{ TO } S \\
      \text{Solve stage } t \text{ subproblem } s \text{ of } SS \\
      \text{END} \\
      \text{END} \\
      \]

   3. Calculate $\bar{z}'$ and $z^*$
   4. Terminate if converged

   **Backward pass:**
   5. \[
      \text{FOR } t=T-1 \text{ TO } 1 \\
      \text{FOR } s=1 \text{ TO } S \\
      \text{Solve all stage } t+1 \text{ subproblems of problem } s \text{ of } SS \\
      \text{Calculate and add cut to all stage } t \text{ subproblems} \\
      \text{END} \\
      \text{END} \\
   \]

   6. GOTO 2

Fig. 2. The Stochastic Dual Dynamic Programming algorithm
The algorithm is outlined in Figure 2. If the full scenario tree was used in step 2 instead of the sampled set, the algorithm would be similar to the Benders decomposition algorithm.

The idea behind using the sampling is illustrated by in Figure 3, where the left figure is a full 5-stage scenario tree with 3 possible outcomes each stage. The figure in the middle shows 3 randomly chosen full scenarios being the sequences of black dots from the stage one problem (the leftmost dot) to the three dots denoted $S_1$ through $S_3$. The black dots represent subproblems that will be solved during the forward pass while both black and grey subproblems will be solved during the backward pass of the algorithm. For each black dot in stages $t<T$, a cut will be calculated and shared among all other stage $t$ subproblems.

![Fig. 3. Full scenario tree (left), SDDP (middle) and AND (right) samplings](image)

**Convergence criterion**

In SDDP, the lower bound $\bar{z}$ is computed as in Benders decomposition after adding cuts from the dual solutions of the sampled scenarios. While $\bar{z}$ is a true lower bound, only an estimate $\overline{z'}$ of the upper bound can be found since only a subset of all full scenarios are solved during each iteration. Given the feasible solutions $\bar{x}^s = \{x_1^s, ..., x_T^s\}$ of scenarios $s=1,...,S$ the objective values

$$z^s = \sum_{t=1}^{T} c_t x_t^s$$  \hspace{1cm} (9)

provide independent identically distributed samples of the (random) objective value $z$, with mean, say $\mu$ and variance, say $\sigma^2$, so the estimate

$$\overline{z'} = \frac{1}{S} \sum_{s=1}^{S} z^s$$  \hspace{1cm} (10)

is asymptotically normally distributed with mean $\mu$ and variance $\sigma^2 = \sigma^2 / S$. Typically, $\sigma^2$ is not known and so $\sigma^2$ is estimated using

$$\hat{\sigma}^2 = \frac{1}{S(S-1)} \sum_{s=1}^{S} (\overline{z'} - z^s)^2.$$  \hspace{1cm} (11)

This can be used to construct a confidence interval e.g. $[\overline{z'} - 1.96 \hat{\sigma}, \overline{z'} + 1.96 \hat{\sigma}]$, which is the 95 percent confidence interval for the value of $\overline{z'}$. If the lower bound $\bar{z}$ lies within this confidence interval, i.e.

$$\bar{z} \geq \overline{z'} - 1.96 \hat{\sigma}$$  \hspace{1cm} (12)

at the end of a forward pass then the algorithm is stopped, viz. step 4 in the algorithm given in Figure 2. More on stopping criteria for this kind of sampling based algorithms can be found in Morton (1998) and Hindsberger (2003).
Abridged nested decomposition

Abridged Nested Decomposition (AND), Donohue and Birge (2006), is a more recent sampling method using the same basic idea as SDDP. Like in SDDP, the assumption of serial independence is needed in AND, and this method also uses a sample of all subproblems at each stage for calculating an approximation of the future cost function.

The main difference between the two sampling techniques can be seen by examining the sampled scenario trees from the forward pass. Looking at Figure 3, it can be seen that the SDDP technique creates scenarios spanning $T$ stages, like the scenarios $S_1$, $S_2$, and $S_3$ from the figure in the middle. In the AND technique some of the scenarios end before stage $T$ is reached, like the upper branch on the right figure.

The AND algorithm is outlined in Figure 4. It requires integer values of $n_t$ for $t = 2, \ldots, T - 1$ and $m_t$ for $t = 2, \ldots, T - 2$. In the algorithm $m_t$ denotes the branching solutions, namely the subset of solved stage $t$ subproblems for each of which $n_{t+1}$ stage $t+1$ subproblems will be sampled and solved. The stage one subproblem is always considered to be a branching solution. Returning to the AND example in Figure 3 it can be seen that $n_2 = 3$, $m_2 = 2$, $n_3 = 2$, $m_3 = 2$, and $n_4 = 2$. The values should initially be chosen small, in the order 1-5.

Donohue (1996) argues that the structure of the scenarios solved during the forward and backward passes in AND allows information in the form of cuts to be obtained from more parts of the scenario tree than in SDDP while having similar computation time. Therefore AND is more suited for solving problems with bushier (few stages, many possible random outcomes) scenario trees, while SDDP may perform better on long narrow scenario trees. Another difference, which speeds up the algorithm, is that AND will initially start with a low sampling size, increasing the number of subproblems to be sampled for each iteration if the algorithm has not converged.

1. Initialization of parameters and dynamic sets
   - **Forward pass:**
     2. Solve stage 1 subproblem and add this to $SS_1$
        - FOR $t = 2$ to $T - 1$
          - Sample and solve $n_t$ stage $t$ subproblems of each subproblem in $SS_{t-1}$.
          - Sample $m_t$ of the solved stage $t$ problems and add to $SS_t$
        - END
   - **Backward pass:**
     3. FOR $t = T - 1$ to 1
        - Solve all stage $t+1$ subproblems of each subproblem in $SS_t$
        - Calculate and add cut to all stage $t$ subproblems
        - END
   4. Calculate $z$
   5. IF no improvement of $z$
   6. **Sampling step:**
      6. Sample $S$ full scenarios forming the set $SS$
         - FOR $t = 1$ TO $T$
         - FOR $s = 1$ TO $S$
         - Solve stage $t$ subproblem $s$ of $SS$
         - END
         - END
      7. Calculate $z'$
      8. Terminate if converged
      9. Increase $n_t$ and $m_t$
      10. GOTO 2

Fig. 4. The Abridged Nested Decomposition algorithm

A lower bound, $z$, is calculated as for Benders decomposition and SDDP. However, in the SDDP algorithm $S$ full scenarios are sampled and solved during the forward pass and these can be used for calculating a statistical
valid $\bar{z}'$ estimate. The forward pass in the AND algorithm does not allow such an estimate to be made. Instead a sampling step equal to the forward pass in SDDP is performed after the backward pass to allow $\bar{z}'$ to be computed, see section 3.1. The sampling step is purely a device to enable the calculation of a statistical valid upper bound. Observe that this might be computational expensive, so it need not be computed at every pass. In step 5 of the algorithm, one can specify a criterion for going on with the sample step. One such criterion could be if the lower bound $\bar{z}$ has not been increased with more than 0.1 percent since the previous iteration. This indicates that the algorithm is getting close to convergence and hence, the sampling step should be run to check for termination.

**Reduced sampling method**

In this section we outline a new approach called Reduced Sampling (ReSa). Also this algorithm is based on the assumption of serial independence. The idea behind ReSa is to improve the efficiency of the SDDP method using some of the innovations of AND, in the hope of getting better performance than AND when solving long scenario trees, and better performance than SDDP when solving bushier scenario trees. The main feature of AND that we use is to limit the number of subproblems to be solved during the early iterations. The algorithm is presented in Figure 5. Compared with AND, the ReSa method like SDDP calculates the $\bar{z}'$ estimate based on the forward pass, so no sampling step is needed. Calculation of the lower bound $\bar{z}$ is like in Benders decomposition.

Otherwise, comparing the ReSa algorithm with the SDDP algorithm, it can be seen that the algorithms are very similar. The main difference is the introduction of $B_t$, the number of randomly selected stage $t$ problems, which were solved during the forward pass, for which a cut is calculated during the backward pass.

Normally, the values of $B_t$ should be low in the beginning—in the order 1 to 5. They can be increased (though for any $t$, $B_t$ cannot be higher than $S$) if improvements from iteration to iteration are insignificant. This is checked by the criterion in step 6. For example, one such rule could be to increase $B_t$ if the $\bar{z}'$ estimate is higher than the previous one. The idea is to add more sampled problems from $SS$ to $BS_t$ to solve as convergence slows down. On the other hand large initial gains can be obtained with very few cuts, so there is no reason to use too much computation power on the backward pass. For $B_t = 2$, the scenario trees illustrating the idea are shown in Figure 6.

---

1. Initialization of parameters and dynamic sets

**Forward pass:**

2. Sample $S$ scenarios forming the set $SS$
   
   FOR $t=1$ TO $T$
   
   FOR $s=1$ TO $S$
   
   Solve stage $t$ subproblem $s$ of $SS$
   
   END
   
   END

3. Calculate $\bar{z}'$ and $\bar{z}$

**Backward pass:**

4. Terminate if converged

5. FOR $t=T-1$ TO 1

   Sample $B_t$ stage $t$ subproblems from $SS$ forming the set $BS_t$
   
   FOR $b=1$ TO $B_t$
   
   Solve all stage $t+1$ subproblems of problem $b$ of $BS_t$
   
   Calculate and add cut to all stage $t$ subproblems
   
   END
   
   END

6. IF (no improvements) AND ($B_t < S$)

   $B_t = B_t + 1$

   ENDIF

7. GOTO 2

---

Fig. 5. The Reduced Sampling algorithm
In Figure 6 the middle graph shows the forward pass of 3 sampled full scenarios used for estimating $\pi$ while the right graph shows the backward pass. This can be compared with the one for SDDP in Figure 3, and the use of black and grey colors is the same though a white dot here indicates a subproblem solved during the forward pass, which is not solved during the backward pass.

Firstly, two stage $T$-$1$ subproblems of those solved during the forward pass are randomly selected. The associated stage $T$ subproblems are solved and cuts for all stage $T$-$1$ subproblems are calculated and added. Next two stage $T$-$2$ subproblems of $SS$ are randomly picked and the subproblems of these are solved adding cuts to all stage $T$-$2$ subproblems, etc. Compared with SDDP fewer cuts are added, but also fewer scenarios are solved during the backward pass—in this case 33% less.

Fig. 6. Full scenario tree (left), ReSa forward pass (middle), and ReSa backward pass (right)

Based on the results in the next section, we believe the ReSa algorithm is very well suited for long narrow scenario trees. For problems where the scenario trees are bushier, the $B_t$ values will probably end up being equal to $S$ before the stopping criterion is met. From then on the algorithm will perform just as SDDP. It may improve the speed of convergence if during step 6 the number of $S$ (and thus also the $B_t$ values) are increased at this point.

**Computational results**

As a computational test case the three sampling based methods were applied to a hydro reservoir management problem. This is based on the Nordic power system and is modelled as a stochastic linear program with 12 stages, each corresponding to one month in time. Hydropower is the dominant technology producing on average about 50% of the total production in the region. Nuclear power plants, thermal cogeneration and condensing plants and gas turbines make up the remaining half. The objective is to minimize the expected cost of production of the hydro-thermal system. The power system is split into 8 regions with different mixtures of production capacity and with transmission constraints in between. In 5 of the 8 regions hydro reservoirs exist, which are operated independently, but the inflows to them all are controlled by a single stochastic parameter, which is the only stochastic parameter in the model. Having stage 1 as the present situation, three possible inflow outcomes are
defined for each of the stages 2 and 3 as well as the last 3 stages while the 6 remaining stages had 9 possible outcomes each. This gives a total of nearly 130 million possible full scenarios. The size of each LP-subproblem to be solved at each stage is $1089 \times 786$ not including the cuts that will be generated by the algorithms. The model is described in more detail in Hindsberger (2005), including the arguments behind the serial independence assumption.

The model was formulated in GAMS, see Brooke, Kendrick and Meeraus (1992). All the presented algorithms for solving the problem were also written in GAMS with the subproblems being solved using CPLEX 6.5.2 on a 500 MHz Pentium computer. To indicate whether a sufficient level of convergence had been achieved, the stopping criterion given as (12) was used.

**SDDP method**

We first applied SDDP to the hydro-thermal model using different sampling sizes, $S$. Values of 8, 12, 16, and 20 scenarios to be sampled have been tested. Since the method is stochastic, the number of iterations needed to achieve convergence can vary, giving varying computation times. Therefore 10 computations have been made for each case. The boxplot in Figure 7 shows that SDDP gives very different computation times depending on the number of sampled scenarios. In a boxplot, the maximum and minimum values obtained are marked as the end points of the vertical lines. The upper and lower edges of a box show the 75% and 25% percentiles respectively, while the cross indicates the average value.

![Boxplot of computation times of SDDP for different numbers of sampled scenarios](image)

**Fig. 7.** Boxplot of computation times of SDDP for different numbers of sampled scenarios

It can be seen that with fewer scenarios to be sampled, less computation time is needed to meet the stopping criterion. This is partly explained by the fact that in particular the backward pass takes considerably longer with more scenarios sampled. The additional information obtained from generating more cuts may not speed up the algorithm similar. Contrary, the runs with fewer samples tend to converge faster.

Figure 8 shows the average standard deviation estimates (right axis) of $E'$ for the 4 test cases. As comparison, the objective function value is around 2700. It can be seen that the fewer samples, the bigger is the standard deviation. This is part of the stopping criterion given in (3.4) so runs with high standard deviations in general need less iterations before the stopping criterion is met. So SDDP 8 is faster than SDDP 20 not only because fewer subproblems are to be solved during the iterations, but also because it terminates more easily with a solution that allows for less confidence than that of SDDP 20.

Hence, by using the chosen stopping criterion, it is necessary to trade off the increase in computation time against the higher quality of the solution. This tradeoff is shown by the two graphs in Figure 8.
AND method

Four experiments have been carried out with the AND implementation. In all cases, the number of sampled scenarios, $S$, in the sampling step was 12 and initially $n_t = 2$ meaning two subproblems were solved for each branching solution $m_t$. Either the initial number of branching solutions, $m_t$, was 1 (cases denoted B1) or 2 (cases denoted B2). Also, two different criteria for going on with the sampling step were tried. One was a normal approach, denoted N, where the sampling step was run if the change in the lower bound was less than 0.1 percent compared with the previous iteration. The other option tried was running the sampling step at the end of each iteration. These cases are denoted E. Note that apart from calculating the upper bound estimate and checking for convergence, all $m_t$ and $n_t$ values are increased as part of the sampling step.

In Figure 9 the results are shown. It can be seen that AND B1-N gives the fastest convergence on average for this particular problem. Cases starting with fewer branching solutions (i.e. B1 vs B2) and increasing the number slowly over the course of the algorithm (i.e. N vs E), tend to perform better than a more rapid increase. Especially, the N cases are substantially better than the E cases where the computation time grows exponentially from iteration to iteration. Also, while going from B1 to B2 in the E cases increases computation time significantly, the impact from going from B1 to B2 in the N cases is much less.

ReSa method

Using the ReSa implementation, the use of three different start values for $B_t$ has been tested for a fixed forward sample size, $S$, of 12. The values $B_t$ was set to respectively 2, 4, and 6 for all $t$. The $B_t$ values were increased by one at the end of an iteration either if the variance estimate of $\bar{z}$ increases compared with the previous iteration, or if the $\bar{z}$ estimate itself is higher than the previous one. If both conditions apply then $B_t$ is increased by two for all $t$.

The results of the three test cases, denoted ReSa 2, ReSa 4 and ReSa 6, are shown in Figure 10. It can be seen that starting with the lower initial $B_t$ values gives the best performance for this problem.

Using initial $B_t$ values of 2, Figures 11 and 12 show the results for runs with different sample sizes, $S =$12, 25, 37 and 50. As for SDDP (see Figure 8), it can be seen that the computation time increases with increasing number of samples while the standard deviation of the upper bound estimate decreases. The latter forces the algorithm to run longer before the stopping criterion is met. Hence, the growth in computation time is not linear.
Fig. 9. Boxplot of computation times of AND with different parameter settings

Fig. 10. Boxplot of computation times of ReSa with different initial values of $B_i$. 
Fig. 11. Boxplot of computation times of ReSa with different sample sizes, $S$

Fig. 12. Average computation time (full line - left axis) and average standard deviation of the upper bound estimates (dashed line - right axis) for ReSa with different sample sizes, $S$

Comparison

Figure 13 shows the results of comparing each algorithm for a sample size, $S$, of 12 when it has the best performing parameter settings, namely SDDP 12, AND B1-N and ReSa 2. Again, boxplots are used to indicate the computation times achieved in the 10 test runs of each algorithm. Of the three implementations, the better algorithm for this model looks like ReSa. We will now look into why.
Compared with SDDP, the AND and ReSa algorithms should initially have faster iterations since fewer cuts are calculated to approximate the future cost function. The average running time per iteration is graphed in Figure 14 (right axis) for the three methods. It can be seen that the running time for the first iteration is lowest for AND but it grows rather quickly to end higher than those of the other methods. Looking at ReSa the computation time at iteration 1 is slightly higher than for AND but significantly lower than the one for SDDP. Growth is slower than AND so from iteration 4, ReSa is fastest. Also, it can be observed that the computation time for SDDP increases too, though very little. This is due to the extra cuts being added to each subproblem, making it a little harder to solve each time.

In Figure 14 also the average gaps between the upper bound estimate and the lower bound at the end of each iteration have been plotted (left axis). While SDDP and ReSa have approximately the same gaps, the gap for AND is considerably lower indicating that it is likely to be closer to the optimal solution. No gap is defined for the first two iterations, as AND B1-N never performed the sampling step before iteration 3.

While the AND method in general has a smaller gap, the fast growth in computation time makes it inferior to ReSa as it was seen in Figure 13. However, this indicates that an even slower increase of the number of branching solutions for AND may make it perform better overall. To analyse whether this was true, a new AND experiment was performed. Here, the \( m_t \) values were increased the first time the sampling step was performed, the \( n_t \) values the second time, then the \( m_t \) values, etc. Hence, the growth in the number of subproblems to be solved was much slower than in the normal implementation. The case is denoted AND B1-S.

The results are shown in Figure 15. It can be observed that the computation time of AND is further improved with the effort to reduce the growth in subproblems to be solved as iterations are done. However, the computation still takes slightly longer on average if compared with ReSa 2. Making the growth in the number of subproblems to be solved even slower for the AND method may however increase its performance even more. One way could be the hardening the criterion for when to do the sampling step. Also, in the AND B1-S example all \( m_t \) values are increased at the same time. By only increasing \( m_t \) for certain stages \( t \) at the same time, the growth in computation time can be kept further down, which could lead to faster computation of the overall result. Similar applies for increasing the \( n_t \) values. Similar, the performance of ReSa might be improved by using another scheme for increasing the \( B_t \) values. Overall, we believe that the computation time for AND for longer, narrow scenario trees can be reduced down to that similar to ReSa. However, compared with the straightforward design of ReSa, more time will be needed to find the best initial values of \( m_t \) and \( n_t \) as well as how and how often it they should be increased to reach the optimal growth rate for the particular problem being solved.
Fig. 14. Gap between UB estimate and LB (bars, left axis) for SDDP, AND, and ReSa compared with computation time in minutes for each iteration (lines, right axis)

Fig. 15. Boxplot of computation times of SDDP 12, AND B1-N, AND B1-S, and ReSa 2

Conclusions

This paper has presented a new algorithm for solving multistage stochastic linear programs. The performance of the algorithm has been compared with those of similar existing algorithms: SDDP and the AND algorithm. The test case has been a hydro-thermal model of the Nordic power system. From the results, it can be seen that ReSa outperforms SDDP as well as AND for solving this specific model. Compared with SDDP the ReSa method looks much superior in terms of speed. Furthermore, as it is identical with SDDP for fixed \( B_t = S \), any better choice of \( B_t \) will result in ReSa performing better. Hence, ReSa will always be able to perform at least as good as
SDDP regardless of the problem. Compared with AND, ReSa also performed better, but other parameter settings for AND, basically slowing down the expansion of the scenario tree to be analysed, may improve the performance of AND to a level equal to or better than ReSa. An optimal growth strategy for \( n \) and \( m \) values in AND to facilitate this is an obvious future research topic.

Using only 12 samples for estimating the upper bound is too low a value to confidently invoke the Central Limit Theorem, and thus the computation times vary considerably. Future trials with more samples should be made to see if the results are consistent with the ones presented here. As shown in Figure 12, a sample size of 50 reduces the variance by a factor of 4. A final observation is that choosing the right parameters, e.g. the number of samples for SDDP, branching solutions for AND, etc., is very important as the choice of these parameters highly affects the performance of the algorithms. The ReSa 2 was on average 31 percent faster than ReSa 6 while AND B1-S on average runs 33 percent faster than AND B1-N.

Acknowledgments. The author would like to express his deepest thanks to Professor Andrew B. Philpott, University of Auckland, for his helpful suggestions during the course of this work. Furthermore, the comments from two anonymous referees helped to improve the paper considerably.

References