Improving the Performance of Stochastic Dual Dynamic Programming

de Matos, Vitor L. * and Philpott, Andy B. † and Finardi, Erlon C. ‡

January 6, 2015

Abstract

This paper is concerned with tuning the Stochastic Dual Dynamic Programming algorithm to make it more computationally efficient. We report the results of some computational experiments on a large-scale hydrothermal scheduling model developed for Brazil. We find that the best improvements in computation time are obtained from an implementation that increases the number of scenarios in the forward pass with each iteration and selects cuts to be included in the stage problems in each iteration. This gives an order of magnitude decrease in computation time with little change in solution quality.

Key words: stochastic programming; stochastic dual dynamic programming; cut selection; hydrothermal scheduling.

*Corresponding author. Plan4 Engenharia, Rua Maria Luiza Agostinho, n 92, Itacorubi, Florianopolis, Brazil, phone: +554899812565, vitor@plan4.com.br

†Electric Power Optimization Centre, University of Auckland, New Zealand, a.philpott@auckland.ac.nz

‡Laboratório de Planejamento de Sistemas de Energia Elétrica, Universidade Federal de Santa Catarina, Brazil, erlon.finardi@ufsc.br

1 Introduction

The Stochastic Dual Dynamic Programming (SDDP) algorithm of Pereira and Pinto [1] is a technique for attacking multi-stage stochastic linear programs that have a stage-wise independence property that makes them amenable to dynamic programming. This method approximates the future cost function of dynamic programming using a piecewise linear outer approximation, defined by cutting planes computed by solving linear programs. This helps mitigate the curse of dimensionality that arises from discretizing the state variables. The intractability arising from a branching scenario tree is avoided by essentially assuming stage-wise independent uncertainty. This allows cuts to be shared between different states, effectively collapsing the scenario tree. Although it was developed over twenty years ago, and has been cited over the years in many applied papers, SDDP has received some recent attention in the mathematical programming literature ([2],[3],[4],[5],[6],[7]) that explores the mathematical properties of this method, in some cases extending it to deal with risk-averse objective functions.

This paper is concerned with some of the implementation details of SDDP algorithms. By carrying out some computational tests on a real application, we attempt to draw some conclusions about how the basic method might be improved by tuning it to build a near optimal policy in the shortest computation time. There are basically two techniques for tuning the algorithm that we shall investigate. The first concerns how one should visit the scenarios in the SDDP algorithm. The classical version of SDDP [1] samples a fixed number of scenarios for each "forward pass". We compare this with alternative strategies that traverse one scenario at a time [5], as well as one that increases the number of scenarios per pass as the algorithm proceeds (as discussed in [7]).

In tandem with scenario selection, we investigate several strategies for selecting cuts to include in the linear programming problems that are solved at each stage. This can lead to a dramatic decrease in computation time with little degradation in the performance of the policies obtained. As a consequence, cut selection strategies are crucial for solving real-life

hydrothermal scheduling problems using SDDP-type algorithms within a computation time that must be kept modest so that models can be solved frequently by the Independent System Operator and by energy companies to provide support to their decisions.

The paper is laid out as follows. In Section 2 we recall the basic algorithm for SDDP. Section 3 describes the three different tree-traversal strategies we shall test and Section 4 describes methods we use for selecting cuts. Section 5 then shows the results of applying these strategies to a test problem that is derived from a long-term model of the Brazilian electricity system. We make our final remarks in Section 6.

2 Stochastic Dual Dynamic Programming

To describe how SDDP works, we consider a class of stochastic linear programs that have T stages, denoted $t = 1, 2, \ldots, T$, in each of which a random right-hand-side vector $b_t(\omega_t) \in \mathbb{R}^m$ has a finite number of realizations defined by $\omega_t \in \Omega_t$. We assume that the outcomes ω_t are stage-wise independent, and that Ω_1 is a singleton, so the first-stage problem is

$$\begin{aligned} z = \min \quad & c_1^\top x_1 + \mathbb{E}[Q_2(x_1, \omega_2)] \\ \text{s.t.} \quad & A_1 x_1 = b_1, \\ & x_1 \geq 0, \end{aligned} \tag{1}$$

where $x_1 \in \mathbb{R}^n$ is the first stage decision and $c_1 \in \mathbb{R}^n$ a cost vector, A_1 is a $m \times n$ matrix, and $b_1 \in \mathbb{R}^m$.

We denote by $Q_2(x_1, \omega_2)$ the second stage costs associated with decision x_1 and realization $\omega_2 \in \Omega_2$. The problem to be solved in the second and later stages t, given state x_{t-1} and realization ω_t, can be written as

$$\begin{aligned} Q_t(x_{t-1}, \omega_t) = \min \quad & c_t^\top x_t + \mathbb{E}[Q_{t+1}(x_t, \omega_{t+1})] \\ \text{s.t.} \quad & A_t x_t = b_t(\omega_t) - E_t x_{t-1}, \quad [\pi_t(\omega_t)] \\ & x_t \geq 0, \end{aligned} \tag{2}$$

where $x_t \in \mathbb{R}^n$ is the decision in stage t, c_t its cost, and A_t and E_t denote $m \times n$ matrices. Here $\pi_t(\omega_t)$ denotes the dual variables of the constraints. In stochastic control terminology $\mathbb{E}[Q_{t+1}(x_t, \omega_{t+1})]$ represents a Bellman function. In the last stage we assume either that $\mathbb{E}[Q_{T+1}(x_T, \omega_{T+1})] = 0$, or that there is a convex polyhedral function that defines the expected future cost after stage T.

The information structure of the problem defined by (1) and (2) is illustrated in Figure 1 in the form of a scenario tree. A scenario tree starts from a root node that defines the first stage decisions (1), and as we move forward in time it branches in the set of possible realizations for the random variable. As one can notice from the figure, the number of variables and constraints in the multistage stochastic programming problem grows very fast with the number of children nodes (branches) and stages, which makes it impossible to solve in most real applications. In order to overcome this difficulty, the SDDP algorithm relies on a stage-wise independence assumption to enable a dynamic programming simplification. By sampling scenarios, a policy for this dynamic program is computed that is close to optimal for the values of the state variables that are visited by this policy.

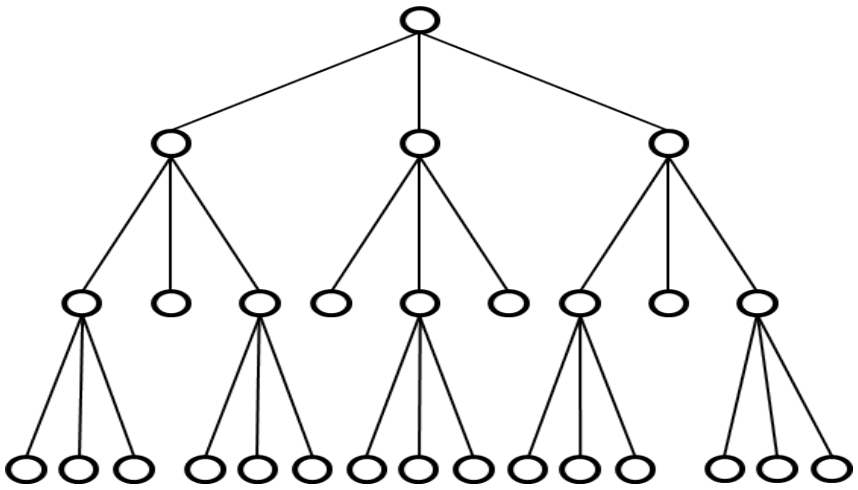

Figure 1: Scenario Tree.

Before defining the sampling procedure, it is important to understand that the algorithm aims at building a policy that is defined at stage t by a polyhedral outer approximation of $\mathbb{E}[Q_{t+1}(x_t, \omega_{t+1})]$ resulting in an approximate value function $\mathcal{Q}_t(x_{t-1}, \omega_t)$. The outer ap-

proximation is constructed using cutting planes called Benders cuts, or just *cuts*. In other words in each tth-stage problem, $\mathbb{E}[Q_{t+1}(x_t, \omega_{t+1})]$ is replaced by the variable θ_{t+1} which is constrained by the set of linear inequalities

$$\theta_{t+1} + \bar{\pi}_{t+1,k}^{\top} E_{t+1} x_t \geq \bar{g}_{t+1,k} \qquad \text{for } k = 1, 2, ...K, \tag{3}$$

where K is the number of cuts. Here $\bar{\pi}_{t+1,k} = \mathbb{E}[\pi_{t+1}(\omega_{t+1})]$, which defines the gradient $-\bar{\pi}_{t+1,k}^{\top} E_{t+1}$ and the intercept $\bar{g}_{t+1,k}$ for cut k in stage t, where

$$\bar{g}_{t+1,k} = \mathbb{E}[\mathcal{Q}_{t+1}(x_t^k, \omega_{t+1})] + \bar{\pi}_{t+1,k}^{\top} E_{t+1} x_t^k.$$

As a consequence, problem (1) is approximated by

$$\begin{aligned} z = \min \quad & c_1^{\top} x_1 + \theta_2 \\ \text{s.t.} \quad & A_1 x_1 = b_1, \\ & \theta_2 + \bar{\pi}_{2,k}^{\top} E_2 x_1 \geq \bar{g}_{2,k}, \\ & x_1 \geq 0, \theta_2 \geq 0 \end{aligned} \tag{4}$$

and (2) is approximated by

$$\begin{aligned} \mathcal{Q}_t(x_{t-1}, \omega_t) = \min \quad & c_t^{\top} x_t + \theta_{t+1} \\ \text{s.t.} \quad & A_t x_t = b_t(\omega_t) - E_t x_{t-1}, \qquad [\pi_t(\omega_t)] \\ & \theta_{t+1} + \bar{\pi}_{t+1,k}^{\top} E_{t+1} x_t \geq \bar{g}_{t+1,k}, \\ & x_t \geq 0, \theta_{t+1} \geq 0. \end{aligned} \tag{5}$$

The approximately optimal policy defined by the cuts is computed through a sequence of major iterations each consisting of a *forward pass* and a *backward pass*. In each forward pass, a set of N scenarios is sampled from the scenario tree, so instead of visiting all scenarios defined in Figure 1 we visit a small subset as shown in Figure 2. Decisions are taken for each node of those N scenarios, starting in the first stage and moving forward up to the last stage. In each stage, the observed values of the state variables x_t, and the costs of each node in all scenarios are saved.

At the end of the forward pass, a convergence criterion is tested, and if it is satisfied then the algorithm is stopped, otherwise it starts the backward pass, which is defined below. In

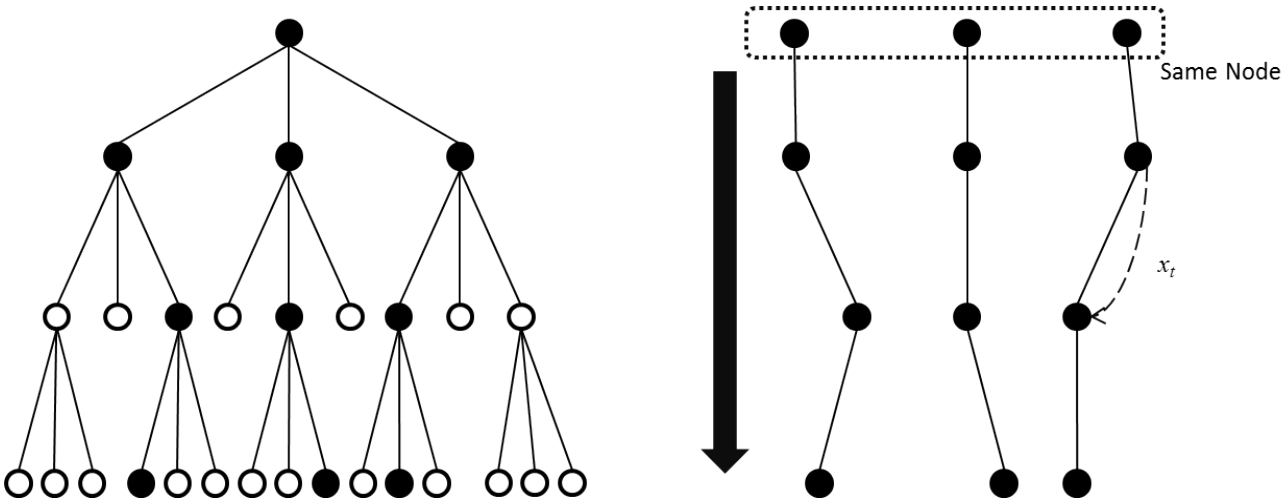

Figure 2: Forward Pass.

the standard version of SDDP [1], the convergence test is satisfied when z, the lower bound on the expected cost at the first stage (called the *Lower Bound*), is statistically close to an estimate of the expected total operation cost (called the *Upper Bound*) obtained by averaging the cost of the policy defined by the cuts when applied to the N sampled scenarios. In this simulation the total operation cost for each scenario is the sum of the present cost $(c_t^\top x_t)$ over all stages t.

For completeness we have included this test in our mathematical description of SDDP, but in our computational experiments we adopt a different approach in which the algorithm is terminated after a fixed number of iterations. This has proved to be more reliable than the standard test for the problems we are solving (see [3], [6] for a discussion of the drawbacks of the standard convergence criterion).

In the backward pass SDDP amends the current policy by adding N cuts to each stage problem, starting at the last stage and working backwards to the first. In each stage t we solve the next stage problems for all possible realizations (Ω_{t+1}), so we assume that we are moving backwards to improve our policy we take into account all nodes in the tree that are related to the sampled scenarios as depicted in Figure 3. The values of the objective functions and dual variables at optimality are taken as expectations over all realizations to define a cut

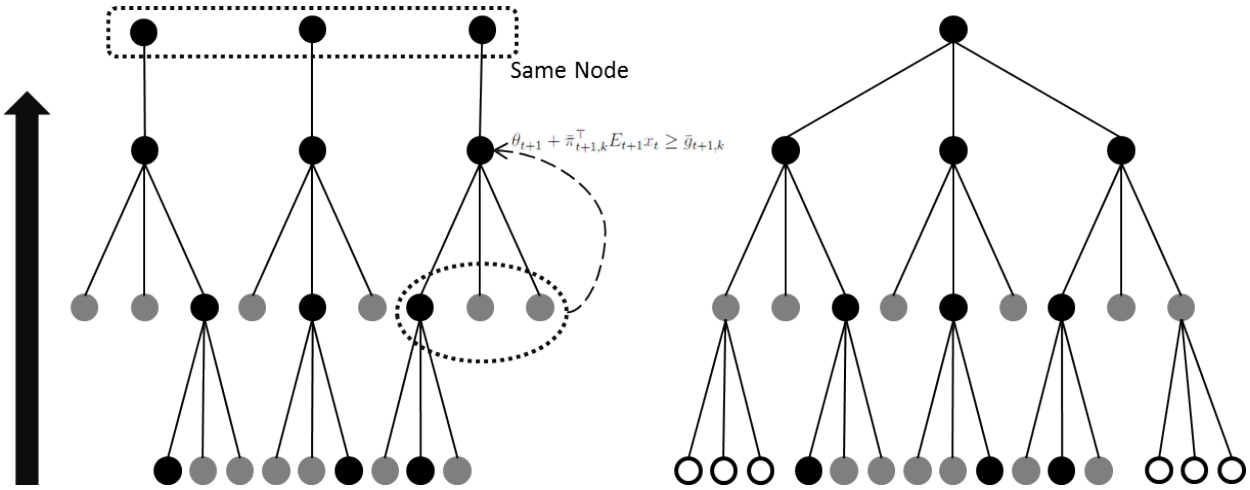

Figure 3: Backward Pass.

that is added to all problems at stage t.

In summary, the SDDP algorithm performs the following three steps repeatedly until the convergence criterion is satisfied.

1. Sampling

   Sample scenarios $s = 1, ..., N$;

2. Forward Pass

   For $t = 1$ solve (4) and save x_1 and z;

   For $t = 2, ..., T$ and $s = 1, ..., N$,

   Solve (5), where ω_t is defined by s, and save $x_t(s)$ and $\mathcal{Q}_t(x_{t-1}, \omega_t)$.

3. Standard Convergence Test (at 90% confidence level)

   Calculate the Upper Bound: $z_u = c_1^{\top} x_1 + \frac{1}{N} \sum_{s=1}^{N} \sum_{t=2}^{T} c_t^{\top} x_t(s)$

   $$\sigma_u = \sqrt{\frac{1}{N-1} \sum_{s=1}^{N} \left(c_1^{\top} x_1 + \sum_{t=2}^{T} c_t^{\top} x_t(s) - z_u \right)^2}.$$

   Calculate the Lower Bound: $z_l = z$;

   Stop if

$$z_u - \frac{1.96}{\sqrt{N}}\sigma_u < z_l < z_u + \frac{1.96}{\sqrt{N}}\sigma_u,$$

otherwise go to the Backward Pass.

4. Backward Pass

   For $t = T, ..., 2$, and $s = 1, ..., N$,

   For $\omega_t \in \Omega_t$, solve (5) using $x_{t-1}(s)$ and save $\pi_t(\omega_t)$ and $\mathcal{Q}_t(x_{t-1}, \omega_t)$;

   Calculate a cut (3) and add it to all subproblems in stage $t - 1$.

   Go to step 1.

3 Tree Traversing Strategies

In the classical description [1] of SDDP, N scenarios are visited at the same time in the forward pass and then a backward pass is performed to build N cuts for each stage. Philpott and Guan [5] describe a variation of this method in which $N = 1$, that appears to perform well on problems when the algorithm performs only a small number of iterations before terminating. In this tree-traversing strategy we visit only one scenario at a time in the forward pass and then a backward pass is performed to build one cut at each stage. This is described formally in the following procedure.

1. Sampling

   Sample scenarios $s = 1, ..., N$;

2. Visiting Scenarios

   For $s = 1, ..., N$,

   For $t = 1$ solve (4) and save $x_1(s)$ and $z(s)$;

   Forward Pass

   For $t = 2, ..., T$,

Solve (5), where ω_t is defined by s, and save $x_t(s)$ and $\mathcal{Q}_t(x_{t-1}, \omega_t)$.

Backward Pass

For $t = T, ..., 2$,

For $\omega_t \in \Omega_t$, solve (5) using $x_{t-1}(s)$ and save $\pi_t(\omega_t)$ and $\mathcal{Q}_t(x_{t-1}, \omega_t)$;

Calculate a cut (3) and add it to all subproblems in stage $t-1$.

3. Convergence Test

Calculate the Upper Bound: $z_u = \frac{1}{N} \sum_{s=1}^{N} \sum_{t=1}^{T} c_t^\top x_t(s)$.

Calculate the Lower Bound: $z_l = z(N)$;

Stop if z_l and z_u are sufficiently close;

otherwise go to 1.

This strategy performs well in the early iterations of the algorithm, where a small number of cuts in the first few scenarios leads to a reasonable approximation of the Bellman function. In contrast, computing N cuts in each early iteration might be wasting computational effort on parts of the state space that will not be visited with high probability by an optimal policy or on states that have already been visited.

Observe that the convergence test differs from the classical SDDP convergence test. The value $\sum_{t=1}^{T} c_t^\top x_t(s)$ is an unbiased estimate of the value of the approximate policy defined by the cuts obtained so far, and so its expectation is an upper bound on the expected cost C of the optimal policy. Thus

$$\begin{aligned} \mathbb{E}[z_u] &= \mathbb{E}\left[\frac{1}{N} \sum_{s=1}^{N} \sum_{t=1}^{T} c_t^\top x_t(s)\right] \\ &= \frac{1}{N} \sum_{s=1}^{N} \mathbb{E}[\sum_{t=1}^{T} c_t^\top x_t(s)] \\ &\geq \frac{1}{N} \sum_{s=1}^{N} C \\ &= C. \end{aligned}$$

However $\sum_{t=1}^{T} c_t^\top x_t(s)$ is obtained using cuts computed from scenarios $1, 2, \ldots, s-1$, and so it is not independent of $\sum_{t=1}^{T} c_t^\top x_t(r)$, $r = 1, 2, \ldots, s-1$. The consequence is that the variance of the estimator z_u cannot be proved to decrease linearly with N. Furthermore, we cannot invoke a central limit theorem to obtain a confidence interval for C. Of course, any candidate policy can be simulated with re-sampled scenarios to give a confidence interval of its expected cost, but it is not possible for this interval estimation to occur during the course of the algorithm. This makes it difficult to reliably terminate the method, and in practice we typically run it for a pre-determined number of iterations.

It is also important to point out that using one scenario at a time will become less advantageous if we exploit parallel processing. In our case, each processor will handle one scenario at a time, so to make best use of these one should visit a number of scenarios that is a multiple of the number of processors.

An alternative approach to the one scenario per pass strategy is to increment the number of scenarios visited in each pass as we proceed through the iterations. A version of this *scenario incrementation* strategy was discussed by [7]. In our implementation, we proceed in batches of N scenarios at a time. In the first batch we use one scenario in each of N forward passes, then in the second batch we use two scenarios in each of $N/2$ forward passes, and so on. The number of scenarios per pass is determined by a parameter k, which we choose to be a factor of N. Thus for $N = 200$, $k \in \mathcal{K} = \{1, 2, 4, 5, 8, 10, 20, 25, 40, 50, 100, 200\}$.

In summary, the algorithm is as below.

1. $k = 1$;
2. Sampling

   Sample scenarios $s = 1, ..., N$;
3. Visiting Scenarios

   For $i = 1, ..., N/k$;

Forward Pass

For $t = 1$ solve (4) and save x_1 and z;

For $t = 2, ..., T$, $j = 1, ..., k$, $s = (i-1)k + j$;

Solve (5), where ω_t is defined by s, and save $x_t(j)$ and $\mathcal{Q}_t(x_{t-1}, \omega_t)$.

Backward Pass

For $t = T, ..., 2$, $j = 1, ..., k$, $s = (i-1)k + j$;

For $\omega_t \in \Omega_t$, solve (5) using $x_{t-1}(j)$ and save $\pi_t(\omega_t)$ and $\mathcal{Q}_t(x_{t-1}, \omega_t)$;

Calculate a cut (3) and add it to each subproblem at stage $t-1$.

4. Convergence Test (at 90% confidence level) applied only when $k = N$.

Calculate the Upper Bound: $z_u = c_1^\top x_1 + \frac{1}{N} \sum_{j=1}^{N} \sum_{t=2}^{T} c_t^\top x_t(j)$

$$\sigma_u = \sqrt{\frac{1}{N-1} \sum_{j=1}^{N} \left(c_1^\top x_1 + \sum_{t=2}^{T} c_t^\top x_t(j) - z_u \right)^2}.$$

Calculate the Lower Bound: $z_l = z$;

Stop if

$$z_u - \frac{1.96}{\sqrt{N}} \sigma_u < z_l < z_u + \frac{1.96}{\sqrt{N}} \sigma_u,$$

otherwise (or if test has not been applied) set k to be the next highest integer in $\mathcal{K}$ and go to 3.

Scenario incrementation has some advantages over the one scenario per pass approach. First of all it admits a convergence test, because the forward pass at iterations where k is sufficiently large gives k independent samples of operation cost that can be used to compute a confidence interval. (We actually perform this test only when $k = N$). The second advantage is that when we wish to improve a reasonably good policy, it is better to visit more scenarios than one per pass. The reason for this comes from the fact that the cuts created by all scenarios visited will be considered in each stage problem solved by the backward pass, and will thus enhance the quality of the cuts created after these solves.

4 Cut Selection

As the SDDP algorithm proceeds each stage problem involves an increasing number of cuts, not all of which are active at each solve. It makes sense to select some subset of these to include in each subproblem to make it solve faster. Since all cuts computed might eventually be useful for some sequence of states visited, we do not actually *delete* any cuts that are created, rather we *select* from a set of cut indices those cuts that are to be used at each stage. At the completion of the algorithm, all cuts that have been computed can be included and used to define the policy.

We digress briefly here to mention that our cut selection strategies have a fundamentally different goal from their use in mixed integer programming. In that setting one seeks a cut for separating a fractional point from the convex hull of integer solutions, and so a method for selecting this to be as deep as possible is desirable. The reader is referred to [8] for an account of this selection procedure when Benders decomposition is used.

The selection strategies that we employ are closest in spirit to those used in regularized decomposition methods applied to two-stage stochastic programming [9] and bundle-trust methods for non-smooth convex optimization [10]. Both classes of problem seek a single optimal decision (the first-stage decision) which is determined by a finite set of cutting planes, so these methods store a finite set from which inactive cuts may be discarded during the course of the algorithm. In a multistage stochastic program we seek a set of cuts for every possible vector of state variables that might be visited by an optimal policy. As a result, it is not possible to guarantee that a cut that is inactive in a given iteration will be inactive for some other state variables that might be visited in a different scenario at a later iteration. Discarding such a cut without careful thought might entail recalculating it later in the algorithm. As mentioned above we never delete a cut entirely but seek to select a subset that might guide the policy in a promising direction without requiring a large computational effort for each solve.

Hence, the key question we address here is, "what cuts should be selected to be included at

each iteration?" For simplicity of notation we define

$$\begin{aligned} \beta_k^\top &= -\bar{\pi}_{t+1,k}^\top E_{t+1} \\ \alpha_k &= \bar{g}_{t+1,k} \end{aligned}$$

where the stage t is suppressed. Suppose the cuts are computed at values x^i, for $i = 1, 2, ...K$. We can evaluate each cut at all the values of x^i to give the table

	x^1	x^2	...	x^K
1	$\alpha_1 + \beta_1^\top x^1$	$\alpha_1 + \beta_1^\top x^2$	...	$\alpha_1 + \beta_1^\top x^K$
2	$\alpha_2 + \beta_2^\top x^1$	$\alpha_2 + \beta_2^\top x^2$	...	$\alpha_2 + \beta_2^\top x^K$
$\vdots$	$\vdots$	$\vdots$		$\vdots$
K	$\alpha_K + \beta_K^\top x^1$	$\alpha_K + \beta_K^\top x^2$		$\alpha_K + \beta_K^\top x^K$

which simplifies to a square matrix $A = (a_{ij})$ where

$$a_{ij} = \alpha_i + \beta_i^\top x^j.$$

We now describe several methodologies for selecting cuts for a stage problem based on the values in A.

4.1 Last-cuts strategy

The *last-cuts* strategy selects to include in the stage problem the H cuts most recently added. This strategy can perform poorly, because as we proceed we may ignore cuts that are important for the convergence of the process. In addition, the set of retained cuts may not be large enough to include all the important cuts. An improvement in this strategy has been discussed in [7] in which not only the last H cuts are selected, but they also consider a set of the last H_2 active cuts. Nevertheless in this paper we restrict attention to the first strategy, and compare it to some alternative cut selection strategies that we now describe.

4.2 Level of Dominance

Given a set of K cuts at stage t we say that cut l is *dominated* if for every x that is feasible for the stage problem there is at least one $k \neq l$ with

$$\alpha_l + \beta_l^\top x \leq \alpha_k + \beta_k^\top x.$$

This is illustrated in Figure 4. In each stage problem it makes sense to include only those cuts that are not dominated. However, it is too expensive computationally to determine this exactly at each iteration, so we resort to heuristics.

The simplest of these is called *Level 1 Dominance* (or just *Level 1*). For every j, we compute $i(j) = \arg\max_i\{a_{ij}\}$, and select every row i of A for which $i \in \{i(j) \mid j = 1, 2, \ldots, K\}$. This chooses to ignore every cut that is not the highest cut at some x^j.

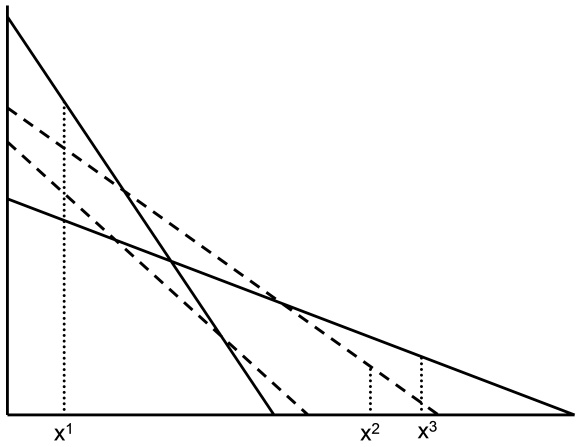

Figure 4: The lower dashed cut is dominated. The remaining cuts are computed at points x^1, x^2 (upper dashed), and x^3. The upper dashed cut is not dominated by the solid cuts but would not be included in a Level 1 dominance selection as it is not the highest cut at x^1, x^2 or x^3.

The Level 1 strategy can be implemented very easily by storing a vector v with jth component $v(j) = \max_i\{a_{ij}\}$, and a vector i with jth component $i(j)$, and updating these every time a cut is added to the problem. If for any $j = 1, 2, \ldots, K$,

$$\alpha_{K+1} + \beta_{K+1}^\top x^j > v(j)$$

then we set $i(j) = K + 1$, and $v(j) = \alpha_{K+1} + \beta_{K+1}^{\top} x^j$. Then we compute

$$v(K+1) = \max_i \{a_{i,K+1}\}$$

and

$$i(K+1) = \arg\max_i \{a_{i,K+1}\}.$$

The list of cuts to retain in the stage problem is then determined by indices in $i(j)$ which can be selected in any call to the linear programming solver.

The Level 1 strategy will not select cuts that are not binding at some x^j, but such a cut might be binding at some other feasible point (see Figure 4). Thus it might make sense to keep a list of the best and second best cuts for each j. If one kept the H highest cuts at each point x^j then we call this the *Level H Dominance* strategy. This can be implemented with a simple modification of the Level 1 algorithm that stores vectors v^h, $h = 1, 2, \ldots, H$, and i^h, $h = 1, 2, \ldots, H$, with $v^h(j)$ giving the hth largest value of the jth column of A, and $i^h(j)$ giving the index i that yields this value.

4.3 Dynamic cut selection

A promising approach to cut selection is to enable the algorithm to select active cuts automatically from the list of cuts that have already been generated. We call this *dynamic cut selection* (DCS). In DCS the SDDP solution strategy is exactly the same, but instead of using all cuts at once we add the cuts iteratively as needed. This means that every time we solve (5) and for the given decision x_t we evaluate $v(j) = \alpha_j + \beta_j^{\top} x_t$ for $j = 1, 2, ..., K$. If the cut $j = \arg\max_j \{v(j)\}$ has not been added to (5) yet, then we add it to the problem and solve it again.

In order to avoid the need to always start without cuts and having to solve several LPs to build the desired policy, we suggest retaining the set of cuts that were added to subproblems solved at stage t in the previous scenarios. So, we only clear all cuts from the LPs at the beginning of a new iteration. This form of warm starting identifies a broad set of cuts to

include that are likely to be important in all subproblems at a given stage.

The following algorithm describes how the DCS is included in the SDDP algorithm when scenario incrementation is used.

1. $k = 1$;

2. Sampling

   Sample scenarios $s = 1, ..., N$;

   Remove all cuts from the stage problems.

3. Visiting Scenarios

   For $i = 1, ..., (N/k)$,

   For $t = 1$ solve (4) and save x_1 and z;

   Forward Pass

   For $t = 2, ..., T$ and $j = 1, ..., k$,

   $s = (i-1)k + j$;

   New cut just added = true;

   While (New cut just added)

   New cut just added = false;

   Solve (5), where ω_t is defined by s, and save $x_t(s)$ and $\mathcal{Q}_t(x_{t-1}, \omega_t)$.

   If cut $j = \arg\max_j\{\alpha_j + \beta_j^\top x_t(s)\}$ not in (5), add cut j to (5) and set New cut just added = true.

   Backward Pass

   For $t = T, ..., 2$ and $j = 1, ..., k$,

   $s = (i-1)k + j$;

   New cut just added = true;

While (New cut just added)

New cut just added = false;

For $\omega_t \in \Omega_t$, solve (5) using $x_{t-1}(s)$ and save $\pi_t(\omega_t)$ and $\mathcal{Q}_t(x_{t-1}, \omega_t)$;

If cut $j = \arg\max_j\{\alpha_j + \beta_j^\top x_t(s)\}$ not in (5), add cut j to (5) and set New cut just added = true.

Calculate a cut (3) and save it to stage $t-1$.

4. Convergence Test (at 90% confidence level) applied only when $k = N$

Calculate the Upper Bound: $z_u = c_1^\top x_1 + \frac{1}{k}\sum_{j=1}^{k}\sum_{t=2}^{T} c_t^\top x_t(j)$

$$\sigma_u = \sqrt{\frac{1}{k-1}\sum_{j=1}^{k}\left(c_1^\top x_1 + \sum_{t=2}^{T} c_t^\top x_t(j) - z_u\right)^2}.$$

Calculate the Lower Bound: $z_l = z$;

Stop if

$$z_u - \frac{1.96}{\sqrt{N}}\sigma_u < z_l < z_u + \frac{1.96}{\sqrt{N}}\sigma_u,$$

otherwise (or if test has not been applied) set k to be the next highest integer in $\mathcal{K}$ and go to 2.

5 Computational Results

The computational testing of the tuning strategies discussed in this paper was carried out on the model of the Brazilian electricity system described in [11]. The Brazilian power system comprises 158 Hydro power plants and 151 Thermal Power Plants. The model we consider has a 10-year horizon with monthly time stages, giving a total of 120 stages. The hydro power plants are aggregated to form four energy equivalent reservoirs, as discussed in [11]. The uncertainty in this problem is assumed to be only in the inflows, which we assume to be stage-wise independent with a known lognormal probability distribution function. It is important to point out that each month of the year has a specific lognormal distribution,

i.e. there are a total of 12 distributions for each energy equivalent reservoir. It is important also to mention that results are reported only for this single case. The efficiency of tuning strategies will vary from problem to problem, but we expect a similar pattern of results should emerge from experiments on hydrothermal scheduling problems of commensurate size.

We use Monte Carlo sampling to select a finite set of 20 equally likely inflow outcomes at each stage. This amounts to a sample-average approximation (SAA) problem with 20^{119} scenarios. The scenarios for the SDDP algorithm are sampled within the SAA problem by means of Monte Carlo sampling. Since our intention is to identify the best algorithm for solving the SAA problem, we also estimate the value of any candidate policy by sampling from the SAA problem rather than the underlying lognormal distribution. All results were obtained in a 2-processor Intel Xeon X5690, with 32 GB RAM 1333MHz, high-performance 300 GB SAS 15000 rpm disk and Windows Server 2008 R2 Standard Edition, using 10 parallel processes. To solve each stage problem we use Gurobi 4.61.

In this paper we compare 21 cases, which are outlined in Table 1. We will first give the results of each cut selection strategy for the three tree traversing strategies, and then we will compare the three best cases. In all cases we will compute 10,000 cuts and we will simulate over 5,000 scenarios sampled within the scenario tree. The simulation was used to compute

	One Scenario	Scenario Incrementation	Traditional
600 Cuts	1	8	15
1000 Cuts	2	9	16
Level 1	3	10	17
Level 2	4	11	18
Level 3	5	12	19
DCS	6	13	20
No Selection	7	14	21

Table 1: Cases to be considered

the expected operation cost over 5,000 scenarios, which will give us a better approximation

to the operation cost of each policy than an upper bound estimated with only 200 scenarios. The simulation is performed 11 times for each case to give an approximation of the operation cost after computing 200, 400, 600, 800, 1000, 2000, 3000, 4000, 6000, 8000 and 10000 cuts. Due to the parallel processing implementation, it is important to point out that, in the one-scenario-per-pass cases, we have 10 scenarios, one for each processor.

To begin with, we discuss cut selection strategies in the one-scenario-per-pass cases 1-7. These were implemented and run until 10,000 cuts had been computed. Table 2 and Table 3 present the values of the Lower Bound and the expected operation cost with increasing numbers of iterations.

Case	Cut Selection	4,000 Cuts	6,000 Cuts	8,000 Cuts	10,000 Cuts
1	600 Cuts	24.412	24.525	24.585	24.624
2	1000 Cuts	24.412	24.527	24.589	24.632
3	Level 1	24.413	24.527	24.591	24.634
4	Level 2	24.415	24.528	24.591	24.635
5	Level 3	24.415	24.529	24.592	24.635
6	DCS	24.414	24.528	24.591	24.635
7	No Selection	24.412	24.527	24.591	24.634

Table 2: Lower Bound for cases with one scenario per pass (10^9 BRL).

Observe that after 10,000 cuts the strategy of keeping the most recent 600 cuts gives the smallest Lower Bound and the most expensive policy on average. The strategy of keeping the most recent 1000 cuts was better but still worse than the more sophisticated cut selection strategies. One important aspect to point out from Table 3 is the fact that after 8,000 cuts per stage had been computed, the cases with the dominance levels, dynamic cut selection and no cut selection had an expected operation cost smaller than the 600 Cuts case when it had computed 10,000 cuts per stage.

From the perspective of expected operation cost of the policy at termination, there is little to choose between dominance levels, dynamic cut selection and no selection. Figure 5 shows the

Case	Cut Selection	4,000 Cuts	6,000 Cuts	8,000 Cuts	10,000 Cuts
1	600 Cuts	25.165	25.104	25.051	25.027
2	1000 Cuts	25.146	25.078	25.032	25.016
3	Level 1	25.136	25.062	25.014	24.995
4	Level 2	25.139	25.058	25.025	24.994
5	Level 3	25.128	25.062	25.011	24.998
6	DCS	25.134	25.061	25.027	25.001
7	No Selection	25.136	25.069	25.015	24.989

Table 3: Expected operation cost for cases with one scenario per pass (10^9 BRL).

computation time in hours (solid line) and the speedup (bars) of each cut selection strategy compared to the no selection case. From Figure 5, one can notice that all cut selection strategies reduce the computation time, and the best speedup is obtained when keeping only the last 600 cuts. Of the strategies that produced little degradation in solution quality, the Level 1 dominance strategy was quickest, producing a policy 11 times quicker than the no selection case.

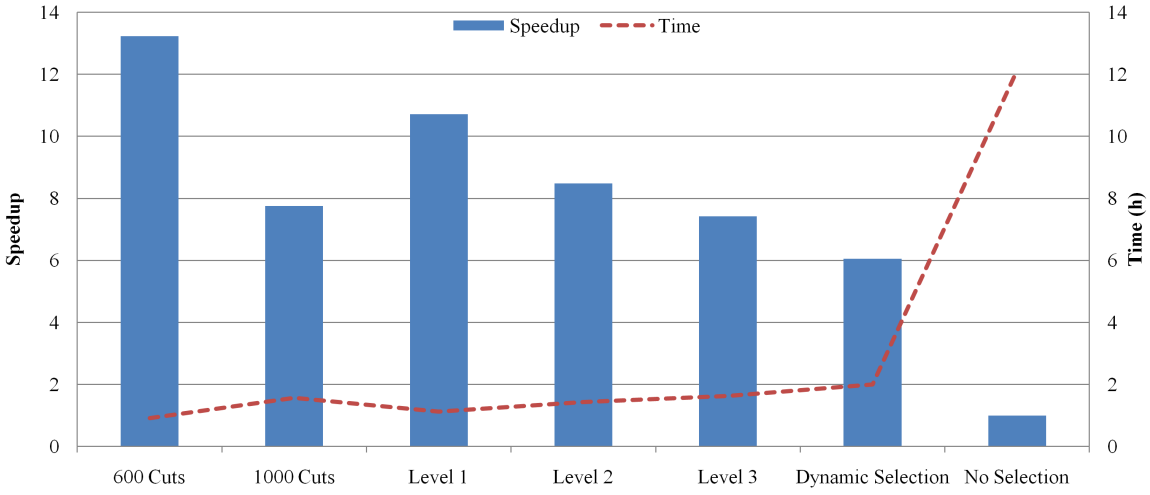

Figure 5: Computation time for cases with 1 scenario per pass.

We now proceed to analyze cases 8-14 that use scenario incrementation. Here the SDDP

Case	Cut Selection	4,000 Cuts	6,000 Cuts	8,000 Cuts	10,000 Cuts
8	600 Cuts	24.584	24.637	24.661	24.678
9	1000 Cuts	24.590	24.644	24.674	24.695
10	Level 1	24.582	24.643	24.677	24.700
11	Level 2	24.585	24.644	24.677	24.700
12	Level 3	24.590	24.644	24.677	24.700
13	DCS	24.585	24.644	24.678	24.701
14	No Selection	24.593	24.645	24.678	24.701

Table 4: Lower Bound for cases with scenario incrementation (10^9 BRL).

algorithm starts with one scenario per pass for each of 10 processors and at the end of every iteration (after N scenarios have been visited) we increase k the number of scenarios per pass in each processor to be the next highest factor of N/(number of processes). Thus when there are 10 processes and $N = 200$, k is chosen from the set $\mathcal{K} = \{1, 2, 4, 5, 10, 20\}$. This gives 10, 20, 40, 50, 100, and 200 scenarios in each forward pass.

The lower bound for the scenario incrementation case with all seven cut selection strategies is shown in Table 4, in which we find that the lower bound is very similar for all cut-selection strategies throughout the iterative process. It is possible to see that from 6,000 to 10,000 cuts, the lower bound from the 600 Cuts strategy is slightly smaller than all other cases.

Table 5 presents the expected operation cost for the scenario incrementation case, where one can observe that the expected operation cost is very similar for all cut selection strategies, but the policies with 600 and 1000 Cuts are more expensive than the others.

Based on solution quality alone, it is hard to discriminate between Level 1 dominance, Level 2 dominance and dynamic cut selection. Figure 6 shows that Level 1 dominance was slightly quicker than the competing methods.

The last tree-traversing strategy we study is the traditional SDDP algorithm in which all scenarios are visited in a single pass. In these cases (15-21) we have 20 scenarios per pass

Case	Cut Selection	4,000 Cuts	6,000 Cuts	8,000 Cuts	10,000 Cuts
8	600 Cuts	25.125	25.053	25.012	24.999
9	1000 Cuts	25.112	25.050	25.007	24.991
10	Level 1	25.085	25.025	24.990	24.969
11	Level 2	25.092	25.032	25.001	24.977
12	Level 3	25.087	25.024	24.990	24.972
13	DCS	25.083	25.031	24.993	24.974
14	No Selection	25.077	25.030	24.992	24.976

Table 5: Expected operation cost for cases with scenario incrementation (10^9 BRL).

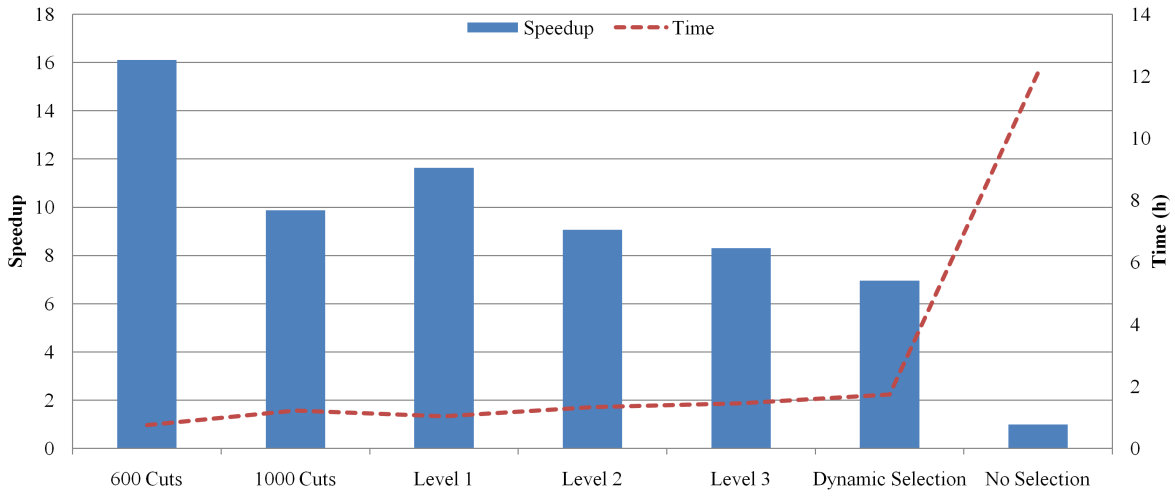

Figure 6: Computation time for cases with scenario incrementation.

Case	Cut Selection	4,000 Cuts	6,000 Cuts	8,000 Cuts	10,000 Cuts
15	600 Cuts	24.557	24.624	24.654	24.673
16	1000 Cuts	24.535	24.632	24.668	24.691
17	Level 1	24.551	24.630	24.669	24.694
18	Level 2	24.547	24.632	24.670	24.695
19	Level 3	24.550	24.633	24.671	24.696
20	DCS	24.544	24.628	24.669	24.695
21	No Selection	24.545	24.635	24.673	24.697

Table 6: Lower Bound for cases with traditional tree traversing (10^9 BRL).

Case	Cut Selection	4,000 Cuts	6,000 Cuts	8,000 Cuts	10,000 Cuts
15	600 Cuts	25.119	25.057	25.016	24.996
16	1000 Cuts	25.140	25.066	25.007	24.990
17	Level 1	25.126	25.042	24.999	24.973
18	Level 2	25.124	25.037	24.996	24.976
19	Level 3	25.144	25.034	24.998	24.975
20	DCS	25.125	25.049	24.991	24.979
21	No Selection	25.112	25.036	24.999	24.972

Table 7: Expected operation cost for cases with traditional tree traversing (10^9 BRL).

for each process. Table 6 shows the Lower Bound over the course of the algorithm. Here one can observe at the end of the iterative process that all values are very similar, except for the Lower Bound for 600 Cuts that is smaller.

The expected operation cost is shown in Table 7.

The computational speedups for cut selection strategies for traditional SDDP are shown in Figure 7. As in the previous analysis, the Level 1 dominance strategy provides a good policy with the best speedup.

As already discussed in [12] and [7], the traditional tree traversing strategy has a poorer

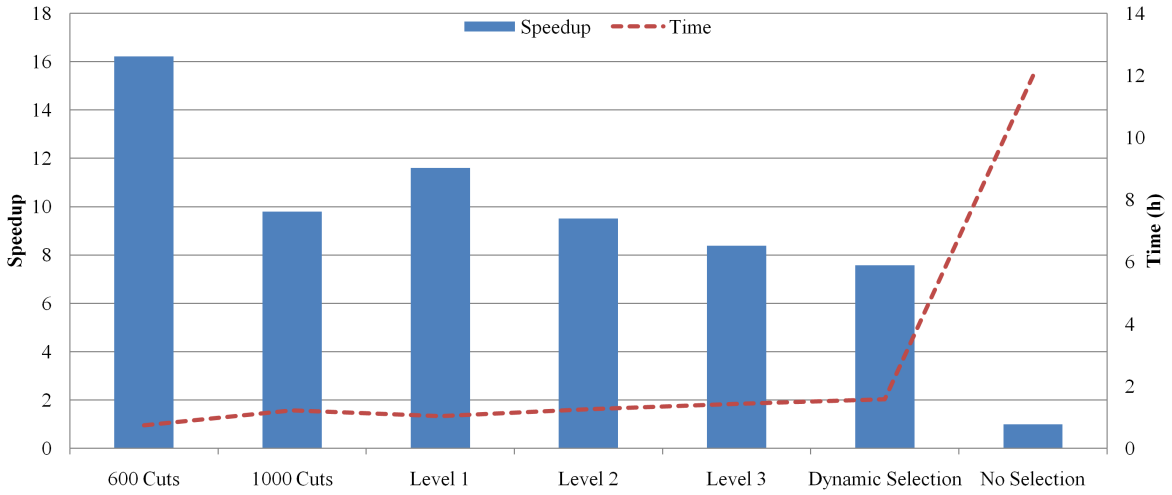

Figure 7: Computation time for cases with traditional tree traversing.

performance in the early iterations when compared to the case in which we start our problem with one scenario per pass. This is because in early iterations the current policy tends to be myopic, so when you visit many scenarios in one single pass this policy is likely to make similar decisions in several scenarios, adding a number of cuts that are closely related. In contrast, for the one scenario-per-pass algorithm, every scenario visited will use a Bellman function approximation updated in all the previous scenarios. For example, in the first iteration the traditional tree traversing strategy performs a forward pass with no cuts so all water is used in all 200 scenarios, whilst in the one-scenario-per-pass case only the first scenario performs a forward pass with no cuts because from the second scenarios onwards there will be cuts available from the previous scenarios. This improves the approximation of the Bellman function more rapidly in the early iterations.

However this advantage does not persist if the algorithm runs for many iterations. Cases 7, 14, and 21 are compared in tables 8 and 9. Table 8 shows that without cut selection, the traditional approach (case 21) gives better results with 10000 cuts than one scenario per pass, but comparable results to scenario incrementation (case 14). The reason for this behaviour is a clear benefit of visiting more than one scenario per pass as we enhance the quality of our policy. This is due to the fact that when we have policies of better quality, the forward pass

Case	200 Cuts	400 Cuts	600 Cuts	...	6,000 Cuts	8,000 Cuts	10,000 Cuts
7	21.944	22.864	23.254	...	24.527	24.591	24.634
14	21.944	22.912	23.571	...	24.645	24.678	24.701
21	8.532	20.133	20.143	...	24.635	24.673	24.697

Table 8: Lower Bound for tree traversing strategies (10^9 BRL).

Case	200 Cuts	400 Cuts	600 Cuts	...	6,000 Cuts	8,000 Cuts	10,000 Cuts
7	28.795	26.847	26.308	...	25.069	25.015	24.989
14	28.795	26.913	26.639	...	25.030	24.992	24.976
21	69.181	32.092	28.868	...	25.036	24.999	24.972

Table 9: Expected operation cost for tree traversing strategies (10^9 BRL).

provides different states, and in the backward pass all scenarios will receive all 200 cuts at once. As a consequence, when creating a cut for the previous stages all scenarios will have access to all 200 new cuts created in the next stage at the same iteration. On the other hand, when using one scenario per pass in the backward pass the first scenario will add just one cut per stage and, as a result, this scenario will not take into account the other 199 scenarios. This is also the reason for the extremely good performance of scenario incrementation, which benefits in the first few iterations from not visiting too many scenarios, but when we have a better policy, visits all 200 scenarios at once.

The plots in figures 5, 6 and 7 show that all three methods without cut selection take about 12 hours to compute 10,000 cuts (with traditional tree traversal slightly less). This is about ten times slower than these methods take with a Level 1 dominance cut selection strategy.

We conclude by examining the Level 1 dominance cut selection strategy in a bit more detail. As shown in Table 10, one can see that the traditional tree traversing strategy eventually overtakes the strategy in which we keep one scenario per pass throughout the iterative process.

In Table 11 we compare the estimated expected costs of the policies. The least expensive

Case	200 Cuts	400 Cuts	600 Cuts	...	6,000 Cuts	8,000 Cuts	10,000 Cuts
3	21.733	22.828	23.264	...	24.527	24.591	24.634
10	21.733	22.959	23.454	...	24.643	24.677	24.700
17	8.532	20.587	20.597	...	24.630	24.669	24.694

Table 10: Lower Bound for tree traversing strategies with Level 1 cut selection (10^9 BRL).

Case	200 Cuts	400 Cuts	600 Cuts	...	6,000 Cuts	8,000 Cuts	10,000 Cuts
3	29.119	26.799	26.280	...	25.062	25.014	24.995
10	29.119	26.726	26.240	...	25.025	24.990	24.969
17	69.181	32.012	28.206	...	25.042	24.999	24.973

Table 11: Expected operation cost for tree traversing strategies with Level 1 cut selection (10^9 BRL).

policy is yielded by scenario incrementation, athough after 10000 cuts the traditional tree-traversing strategy and scenario incrementation achieved a very similar lower bound and expected operation cost. This indicates that we are very close to the optimal policy in both cases.

Finally, Figure 8 presents the computation time for each tree-traversing strategy with the Level 1 dominance cut selection strategy, in which one can see that the computation time is very similar. The scenario incrementation and traditional tree-traversing strategies are about 10% faster since the cut selection algorithm is run less often.

6 Conclusions

The experimental results in this paper show that cut selection strategies do not have a significant impact on the policy in terms of lower bound and expected operation cost at the end of the iterative process, but they may reduce the computation time by an order of magnitude. The best combination of tuning strategies, at least to solve this problem

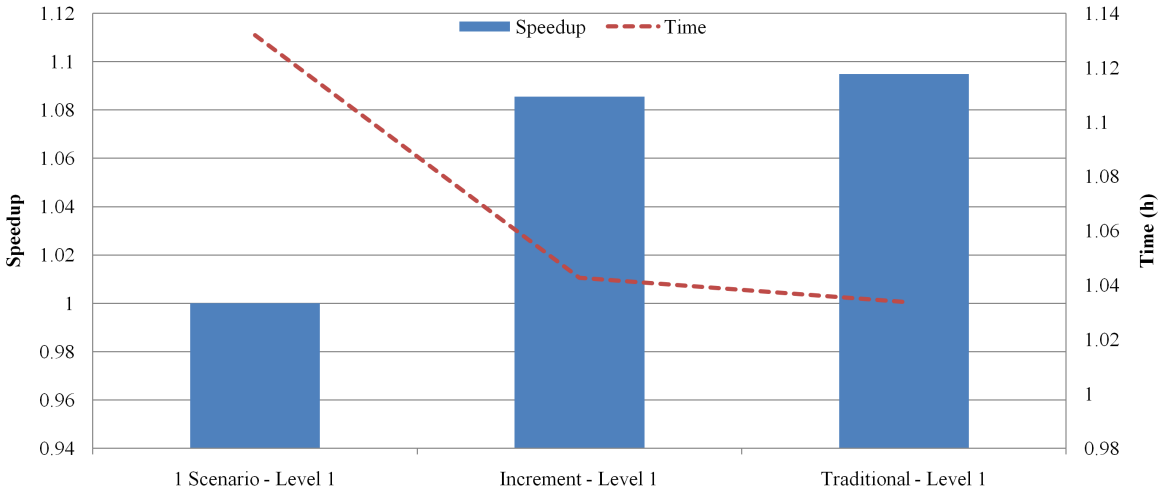

Figure 8: Computational time for tree traversing strategies.

instance, is to start the process with one scenario per pass, increment the number of scenarios throughout the iteration, and use a Level 1 dominance cut selection strategy. Although the traditional tree traversing strategy was quicker than scenario incrementation and produces a similar policy after 10,000 cuts, it has the drawback of being worse if the algorithm is terminated early.

References

[1] M. V. F. Pereira, L. M. V. G. Pinto, Multi-stage stochastic optimization applied to energy planning, Mathematical Programming 52 (1991) 359–375.
URL http://dx.doi.org/10.1007/BF01582895

[2] V. Guigues, W. Romisch, Sampling-based decomposition methods for risk-averse multistage stochastic programs, available at http://www.optimization-online.org/DB_HTML/2010/10/2763.html. Date of access: 04/July/2012 (2010).
URL http://www.optimization-online.org/DB_HTML/2010/10/2763.html

[3] T. Homem-de Mello, V. de Matos, E. Finardi, Sampling strategies and stopping criteria for stochastic dual dynamic programming: a case study in long-term hydrothermal scheduling, Energy Systems 2 (2011) 1–31.
URL http://dx.doi.org/10.1007/s12667-011-0024-y

[4] A. Philpott, V. de Matos, Dynamic sampling algorithms for multi-stage stochastic programs with risk aversion, European Journal of Operational Research 218 (2) (2012) 470 – 483. doi:10.1016/j.ejor.2011.10.056.
URL http://www.sciencedirect.com/science/article/pii/S0377221711010332

[5] A. Philpott, Z. Guan, On the convergence of stochastic dual dynamic programming and other methods, Operations Research Letters 36 (2008) 450–455.

[6] A. Shapiro, Analysis of stochastic dual dynamic programming method, European Journal of Operational Research 209 (1) (2011) 63 – 72. doi:10.1016/j.ejor.2010.08.007.
URL http://www.sciencedirect.com/science/article/pii/S0377221710005448

[7] A. Shapiro, W. Tekaya, J. P. da Costa, M. P. Soares, Risk neutral and risk averse stochastic dual dynamic programming method, European Journal of Operational Research 224 (2) (2013) 375–391.

[8] M. Fischetti, D. Salvagnin, A. Zanette, A note on the selection of Benders cuts, Mathematical Programming 124 (1-2) (2010) 175–182.

[9] A. Ruszczyński, A regularized decomposition method for minimizing a sum of polyhedral functions, Mathematical Programming 35 (3) (1986) 309–333.

[10] H. Schramm, J. Zowe, A version of the bundle idea for minimizing a nonsmooth function: Conceptual idea, convergence analysis, numerical results, SIAM Journal on Optimization 2 (1) (1992) 121–152.

[11] V. de Matos, E. Finardi, A computational study of a stochastic optimization model for long term hydrothermal scheduling, International Journal of Electrical Power and Energy Systems In Press, Corrected Proof. doi:http://dx.doi.org/10.1016/j.ijepes.2012.06.021.

[12] V. de Matos, A. Philpott, E. Finardi, Z. Guan, Solving long-term hydrothermal scheduling problem, in: 17th Power Systems Computation Conference 2011, Vol. 1, 2011, pp. 1355–1362.