A deterministic algorithm for solving multistage stochastic programming problems

Regan Baucke[a,b,*], Anthony Downward[a], Golbon Zakeri[a,b]

[a] *Electrical Power Optimization Centre at The University of Auckland, 70 Symonds Street, Auckland 1010, New Zealand*

[b] *The Energy Centre at The University of Auckland, 12 Grafton Road, Auckland 1010, New Zealand*

Abstract

Multistage stochastic programming problems are an important class of optimisation problems, especially in energy planning and scheduling. These problems and their solution methods have been of particular interest to researchers in stochastic programming recently.

Because of the large scenario trees that these problems induce, current solution methods require random sampling of the tree in order to build a candidate policy. Candidate policies are then evaluated using Monte Carlo simulation. Under certain sampling assumptions, theoretical convergence is obtained almost surely. In practice, convergence of a given policy requires a statistical test and is only guaranteed at a given level of confidence.

In this paper, we present a deterministic algorithm to solve these problems. The main feature of this algorithm is a deterministic path sampling scheme during the forward pass phase of the algorithm which is guaranteed to reduce the bound gap at all the nodes visited. Because policy simulation is no longer required, there is an improvement in performance over traditional methods for problems in which a high level of confidence is sought.

Keywords: dynamic programming, decomposition, multistage, SDDP, stochastic programming

*Corresponding author

Email address: r.baucke@auckland.ac.nz (Regan Baucke)

July 23, 2017

1. Introduction

Since the seminal work of [10], convex multistage stochastic programming problems (often implemented as stochastic dual dynamic programmes or SDDP) have been studied extensively by researchers in the field of stochastic programming [13, 3, 11, 5].

These problems are notorious in their difficulty – this is due to the exponential growth of the scenario tree that represents the filtration on the problem's probability space, and the often high-dimensional state-spaces that must be represented in each stage. Several algorithms exist that solve SDDP-type problems, readers may be familiar with [3] and [11].

The size of these problems mean that direct methods are insufficient; the problems require specialised decomposition algorithms. The essence of these decomposition methods is to break the problem into a series of stage problems which include a representation of the value/cost-to-go associated with future stages. These methods are rooted in the concepts underpinning dynamic programming.

This class of problems has many applications within the energy planning and scheduling industry. A classic example is the hydro-thermal scheduling problem – where a release policy from a hydro-electric reservoir is sought. This policy can be defined in terms of the value of water storage in the reservoir. In New Zealand, such a water valuation tool is made available to market participants by the electricity regulator. This tool (named EMI-DOASA, see [12]) provides a methodology for solving large instances of an simplified model of New Zealand's generation assets and transmission network. This model has been used to compute an optimal reservoir management for a hypothetical centrally planned system, and thereby provide a benchmark for comparison to actual releases. Moreover, the marginal value of water that defines the optimal policy can help guide reservoir management decision of individual agents.

Most implementations of SDDP algorithms sample the scenario tree at random to build piece-wise linear lower approximations of the cost function at each stage, in a nested fashion. The output of such methods are a set of lower-bound approximate cost functions. From these cost functions, a feasible policy can be computed by solving of a sequence of stage problems that incorporate the cost-to-go functions. In order to terminate the algorithm, a convergence test must be performed which requires the evaluation of the expected cost of the policy. The evaluation of the policy over the whole scenario tree can be a large computational expense and is intractable even for seemingly small problems. It was suggested by [10] that a policy evaluation could be achieved by a Monte Carlo simulation – reducing the computational expense, but forgoing a deterministic convergence criterion.

Several proofs (see [11, 5]) exist giving convergence of these methods in an almost-sure sense. However, in practice, these methods can become slow, since parts of the scenario that need to be sampled to improve the lower-bound can take an arbitrarily large number of iterations to be explored. In the meantime, linear cutting planes which bound the cost function (called 'cuts') are evaluated and included in the sub-problems at every iteration, but may not give new information.

It is the focus of this paper to present a new algorithm to solve these optimisation problems. The algorithm is similar to that of [11], but includes two additions: an upper-bound cost function and a deterministic scenario tree sampling scheme. We consider the problem class presented by [5]; that is, we make no assumptions about linearity of cost functions nor the polyhedral nature of the feasibility sets. The presence of an upper-bound function precludes the use of Monte Carlo simulation in order to evaluate the candidate policy, leading to an increase in computational performance.

Convergence proofs throughout this paper give results in the traditional sense; that is, we show that sequences of value function iterates become arbitrarily close to the true objective value, using our deterministic sampling scheme.

This paper is structured as follows: Section 2 introduces some preliminary concepts required for the algorithm; Section 3 extends Section 2 by describing the algorithm in the complete multistage stochastic setting; and Section 4 presents the results of several computational experiments to conclude the paper.

2. Algorithm preliminaries

This section introduces several aspects of our algorithm for multistage stochastic problems through simpler problems to build the reader's intuition. Firstly, we provide an alternative proof to Kelley's cutting plane algorithm from [8], which utilises a converging lower-bound and upper-bound. We will extended this idea to the two-stage and then multistage case. Initially, we are concerned with the solving following optimisation problem:

$$w^* = \min_{u \in \mathcal{U}} \quad W(u), \tag{2.1}$$

where $\mathcal{U} \subset \mathbb{R}^n$ is convex and compact, and $W(u)$ is convex and α-Lipschitz on $\mathcal{U}$. We are interested in solving this problem by forming a series of approximations to the objective function. This was first seen in [8], and works by forming a lower-bound function of W, which is defined as the point-wise maximum of linear functions over the affine hull of $\mathcal{U}$. We extend this method by including an upper-bound function.

2.1. Approximations of convex functions

For a given α-Lipschitz convex function $W : \mathcal{U} \subset \mathbb{R}^n \mapsto \mathbb{R}$ and a finite set of sample points $S \subset \mathcal{U}$, with $\bar{u} \in S$, we define the following lower-bound function of $W(u)$:

Definition 2.1.

$$\underline{W}^S(u) = \max_{\bar{u} \in S} \quad W(\bar{u}) + \langle \nabla W(\bar{u}), (u - \bar{u}) \rangle \tag{2.2}$$

where $\nabla W(\bar{u})$ is an arbitrary sub-gradient of $W(u)$ at $\bar{u}$.

The lower-bound function possesses many attractive qualities for use in decomposition algorithms. Three that we wish to highlight here are:

$$S_1 \subseteq S_2 \implies \underline{W}^{S_1}(u) \leq \underline{W}^{S_2}(u) \quad \forall u \in \mathcal{U}, \tag{2.3}$$

$$\underline{W}^S(u) \leq W(u) \quad \forall u \in \mathcal{U},\ \forall S \in [\mathcal{U}], \tag{2.4}$$

$$\underline{W}^S(\bar{u}) = W(\bar{u}) \quad \forall \bar{u} \in S,\ \forall S \in [\mathcal{U}], \tag{2.5}$$

where $[\mathcal{U}]$ is the set of all finite subsets of $\mathcal{U}$. These three properties are exploited frequently throughout this paper. Further, it can be seen that because $W(u)$ is α-Lipschitz, so is the lower-bound function $\underline{W}^S(u)$. These facts will not be proved in this paper; we contend that these properties are well known from previous works (for instance, [8]). We now introduce an upper-bound function of $W(u)$, which is defined as the objective of the following optimisation problem:

$$\begin{aligned} \tilde{W}^S(u) = \min_{\sigma_{\bar{u}}} \quad & \sum_{\bar{u} \in S} W(\bar{u}) \sigma_{\bar{u}} \\ \text{s.t.} \quad & \sum_{\bar{u} \in S} \sigma_{\bar{u}} = 1 \qquad [\mu] \\ & \sum_{\bar{u} \in S} \bar{u} \sigma_{\bar{u}} = u \qquad [\lambda] \\ & \sigma_{\bar{u}} \geq 0. \end{aligned} \tag{2.6}$$

Intuitively, $\tilde{W}^S(u)$ can be thought of as the minimal value over all convex combinations of points in S that contain u in their convex hull. When (2.6) is infeasible, we take the convention that $\tilde{W}^S(u) = \infty$. This means function $\tilde{W}^S(u)$ is an upper-bound of $W(u)$ for all $u \in \mathcal{U}$ – however we do lose the favourable property of Lipschitz continuity over $\mathcal{U}$ with this definition; $\tilde{W}^S(u)$ is Lipschitz over $\text{Conv}(S)$ only. To restore this property, we extend the definition to the following:

Definition 2.2.

$$\begin{aligned} \bar{W}^S(u) = \max_{\mu,\lambda} \quad & \mu + \langle \lambda, u \rangle \\ \text{s.t.} \quad & \mu + \langle \lambda, \bar{u} \rangle \leq W(\bar{u}), \quad \forall \bar{u} \in S, \\ & ||\lambda||_q \leq \alpha \\ & \mu \text{ free}, \end{aligned} \tag{2.7}$$

where $||\cdot||_q$ is a p-norm.

This definition is constructed from the dual of (2.6) and then further constraining the magnitude of λ. We will proceed by proving that the upper-bound function $\bar{W}^S(u)$ has the analogous properties to the lower-bound function $\underline{W}^S(u)$. We first show, in Lemma 2.1, that $\bar{W}^S(u)$ is α-Lipschitz, and then in Lemmas 2.2–2.4 we show analogues for the properties (2.3), (2.4), and (2.5), respectively.

Lemma 2.1. *$\bar{W}^S(u)$ is α-Lipschitz under the $||\cdot||_p$ norm where*

$$\frac{1}{p} + \frac{1}{q} = 1 \text{ and } p, q > 0. \tag{2.8}$$

Proof. It is easy to see that the value of $|\bar{W}^S(u_2) - \bar{W}^S(u_1)|$ is bounded above by

$$\begin{aligned} \max_{\lambda} \quad & \langle \lambda, u_2 - u_1 \rangle \\ \text{s.t.} \quad & ||\lambda||_q \leq \alpha. \end{aligned} \tag{2.9}$$

If $\alpha = 0$, then clearly $|\bar{W}^S(u_2) - \bar{W}^S(u_1)| = 0$ and so the Lipschitz condition

$$|\bar{W}^S(u_2) - \bar{W}^S(u_1)| \leq \alpha ||u_2 - u_1||_p, \quad \forall u_2, u_1 \in \mathcal{U}, \tag{2.10}$$

holds. If $\alpha > 0$ then, we can write (2.9) as

$$\begin{aligned} \alpha \times \max_{\lambda} \quad & \langle \frac{1}{\alpha}\lambda, u_2 - u_1 \rangle \\ \text{s.t.} \quad & ||\frac{1}{\alpha}\lambda||_q \leq 1. \end{aligned} \tag{2.11}$$

By the definition of the dual norm of L^p spaces, (2.11) is equal to $\alpha||u_2 - u_1||_p$ where

$$\frac{1}{p} + \frac{1}{q} = 1,$$

completing the proof. □

Lemma 2.2. *If $W(u)$ is convex and α-Lipschitz on $\mathcal{U}$ under an arbitrary but fixed $||\cdot||_p$ norm, and $\hat{u} \in S$, then $\bar{W}^S(\hat{u}) = W(\hat{u})$.*

Proof. For a given S, the points (μ, λ) in the feasible region of (2.7) represent the coefficients of a supporting hyperplane of the points $(\bar{u}, W(\bar{u}))$ $\forall \bar{u} \in S$ where μ is the intercept and λ is the gradient of the hyperplane. We note here that the feasible region is always non-empty as

$$\mu = \min_{\bar{u} \in S} W(\bar{u}), \text{ and } \lambda = \mathbf{0}, \tag{2.12}$$

is always feasible. Because the function $W(u)$ is convex and α-Lipschitz, there exists a supporting hyperplane

$$W(\hat{u}) + \langle g, u - \hat{u} \rangle \leq W(u), \ \forall u \in \mathcal{U}, \tag{2.13}$$

where $||g||_q \leq \alpha$ and g is in the set of sub-gradients of $W(u)$ at $\hat{u}$. So a feasible candidate solution for $\bar{W}^S(\hat{u})$ is $\tilde{\mu} = W(\hat{u}) - \langle g, \hat{u} \rangle$ and $\tilde{\lambda} = g$ which gives $\bar{W}^S(\hat{u}) = W(\hat{u})$. If another candidate (μ, λ) gives $\mu + \langle \lambda, \hat{u} \rangle < W(\hat{u})$, then the pair is not optimal because of the existence of $(\tilde{\mu}, \tilde{\lambda})$. If a further candidate (μ, λ) gives $\mu + \langle \lambda, \hat{u} \rangle > W(\hat{u})$, then the pair doesn't support the function at $\hat{u}$, making it infeasible. This completes the proof. □

Lemma 2.3. *If $S^1 \subseteq S^2$, then $\bar{W}^{S^1}(u) \geq \bar{W}^{S^2}(u)$.*

Proof. The function $\bar{W}^S(u)$ is defined by a maximisation problem which is feasible and bounded. $S^1 \subseteq S^2$ implies that the feasibility set of $\bar{W}^{S^2}$ is contained within $\bar{W}^{S^1}$, giving the result. □

Lemma 2.4. *If $W(u)$ is convex and α-Lipschitz on $\mathcal{U}$ under the $|| \cdot ||_p$ norm, then $\bar{W}^S(u) \geq W(u)$ for all $u \in \mathcal{U}$.*

Proof. From our monotonicty property, we have $\bar{W}^{S^*}(u) \leq \bar{W}^S(u)$, $\forall S \subseteq S^*$. Suppose there exists a $\tilde{u} \in \mathcal{U}$ for which $\bar{W}^S(\tilde{u}) < W(\tilde{u})$. By defining $S^* = S \cup \{\tilde{u}\}$, and by Lemma 2.2, we have $W(\tilde{u}) = \bar{W}^{S^*}(\tilde{u})$. So $\bar{W}^S(\tilde{u}) < \bar{W}^{S^*}(\tilde{u})$, which is a contradiction of the monotonicity of inclusion of $\bar{W}^S(u)$, completing the proof. □

Figure 1 shows the bounding functions on an arbitrary convex function $f(x)$ on $\mathcal{X} \subset \mathbb{R}^1$. We now proceed to give a new proof to Kelley's Theorem that uses both bounding functions described above. We note here that this proof does not enhance the original result, but serves as a useful introduction to the upper-bound function, and gives the format for further proofs in this paper. We require the conditions set out in (2.1) on $W(u)$ and $\mathcal{U}$ for the following lemma.

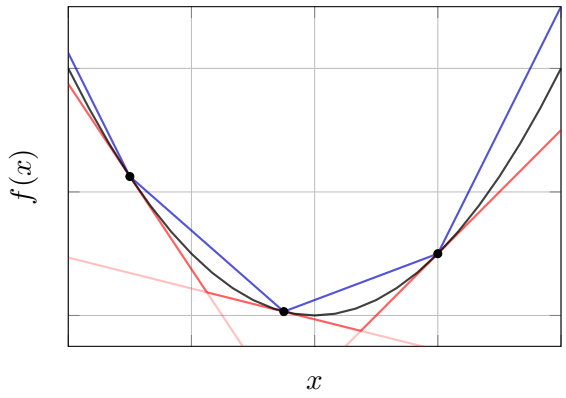

Figure 1: A convex function $f(x)$ on $\mathcal{X} \subset \mathbb{R}$, with $\bar{f}(x)$ in blue and $\underline{f}(x)$ in red.

Lemma 2.5. *Consider the sequences*

$$u^k = \arg\min_{u \in \mathcal{U}} \underline{W}^{S_{k-1}}(u), \;\; S_k = S_{k-1} \cup u^k, \tag{2.14}$$

with

$$\bar{w}^k = \min_{u \in \mathcal{U}} \bar{W}^{S_k}(u), \;\; \underline{w}^k = \min_{u \in \mathcal{U}} \underline{W}^{S_k}(u). \tag{2.15}$$

We have that

$$\lim_{k \to \infty} \left(\bar{w}^k - \underline{w}^k \right) = 0.$$

Taking the convention of u^{k+1} being the minimiser of the function $\underline{W}^{S_k}(u)$ provides notational convenience for further proofs in this paper. Further, we will use $\bar{W}^k(u)$ as a notational shorthand for $\bar{W}^{S_k}(u)$. We proceed with the proof of Lemma 2.5.

Proof. From (2.14) and (2.15), we have

$$\underline{w}^k = \underline{W}^k(u^{k+1}), \quad \forall k.$$

For $\bar{w}^k$, from (2.15) we have

$$\bar{w}^k \leq \bar{W}^k(u), \quad \forall k, \forall u \in \mathcal{U}.$$

It follows then that,

$$\bar{w}^k - \underline{w}^k \leq \bar{W}^k(u^{k+1}) - \underline{W}^k(u^{k+1}), \quad \forall k. \tag{2.16}$$

We will now show that the RHS of (2.16) converges to 0 as k tends to infinity – this will complete the proof. Suppose there exist some $\epsilon > 0$ for which

$$\epsilon \leq \bar{W}^k(u^{k+1}) - \underline{W}^k(u^{k+1}), \quad \forall k.$$

Subtracting $\bar{W}^k(u^{\hat{k}}) - \underline{W}^k(u^{\hat{k}})$ from both sides gives

$$\epsilon - \bar{W}^k(u^{\hat{k}}) + \underline{W}^k(u^{\hat{k}}) \leq \bar{W}^k(u^{k+1}) - \underline{W}^k(u^{k+1}) - \bar{W}^k(u^{\hat{k}}) + \underline{W}^k(u^{\hat{k}}), \quad \forall \hat{k}, k.$$

From the α-Lipschitz continuity of $\bar{W}^k$ and $\underline{W}^k$, we have

$$\epsilon - \bar{W}^k(u^{\hat{k}}) + \underline{W}^k(u^{\hat{k}}) \leq 2\alpha ||u^{k+1} - u^{\hat{k}}||, \quad \forall \hat{k}, k.$$

From Lemma 2.2, $\bar{W}^k(u^{\hat{k}}) - \underline{W}^k(u^{\hat{k}}) = 0, \ \forall \hat{k} \leq k$ so,

$$\frac{\epsilon}{2\alpha} \leq ||u^{k+1} - u^{\hat{k}}||, \quad \forall \hat{k} \leq k,$$

which is a contradiction of the compactness of $\mathcal{U}$. It must follow then, that no $\epsilon > 0$ exists and $\bar{w}^k - \underline{w}^k \to 0$ as $k \to \infty$. □

Because the sequences $\bar{w}^k$ and $\underline{w}^k$ bound each other and are monotonic, we remark that $\bar{w}^k, \underline{w}^k \to w^*$, as defined in (2.1). We also note that $\bar{w}^k$, the minimum of the upper-bound function at iteration k, simply collapses to the minimum of the values $W(u^k)$ sampled so far; it is not necessary in this case to store the upper-bound as a function over $\mathcal{U}$. This simpler upper-bound has been discussed by [9] in relation to its role in the Kelley Method.

2.2. The two-stage case

Suppose instead of minimising a single convex function using the above method, one was tasked with minimising the sum of several convex functions. This problem is very familiar to the stochastic programming community; the sum in question is an expectation of value functions over different scenario realisations. Without loss of generality, we wish to compute the value of

$$w^* = \min_{u \in \mathcal{U}} \quad \sum_{\omega \in \Omega} p_\omega W_\omega(u), \tag{2.17}$$

with the same requirements of convexity and α-Lipschitz continuity of $W_\omega(u)$, for all $\omega \in \Omega$ and a convex and compact $\mathcal{U}$. Here, Ω is finite. The first decomposition method for this problem class was developed in [1] and is commonly referred to as Bender's Decomposition. This decomposition method comes in two main variants: the *average-cut* technique and the *multi-cut* technique. Briefly, an average-cut method brings a single cut back to the first stage problem at every iteration. This cut has gradients equal to the average of all the second stage sub-gradients. Similarly, the constant term is the average of the second stage function evaluations. A multi-cut method includes one cut for every second stage function; the 'averaging' is done within the first stage optimisation problem itself.

We propose a different algorithm to solve this problem, which uses the upper-bound function introduced earlier, and a sampling scheme which determines the most 'useful' $\omega \in \Omega$ to sample at every iteration. Our method requires the multi-cut (as opposed to average-cut) approach in the spirit of [2]; bounding functions are computed and stored for each $W_\omega(u)$.

The algorithm is as follows: At iteration 0, define $\underline{W}_\omega^0 := -\infty, \bar{W}_\omega^0 := \infty$. Starting with $k = 1$, $x_0^k = x_0$, solve

$$u^k = \arg\min_{u \in \mathcal{U}} \quad \sum_{\omega \in \Omega} p_\omega \underline{W}_\omega^{k-1}(u), \tag{2.18}$$

Next, evaluate[1]

$$\omega^k = \arg\max_{\omega \in \Omega} \ \{p_\omega\big(\bar{W}_\omega^{k-1}(u^k) - \underline{W}_\omega^{k-1}(u^k)\big)\}. \tag{2.19}$$

We now update our approximation of $\underline{W}_{\omega^k}^{k-1}(u)$ as

$$\underline{W}_{\omega^k}^k(u) := \max\{\underline{W}_{\omega^k}^{k-1}(u), W_{\omega^k}^k(u^k) + \langle \nabla W_\omega(u^k), u - u^k \rangle\},$$

and $\bar{W}_{\omega^k}^{k-1}(u)$ as

$$\begin{aligned} \bar{W}_{\omega^k}^k(u) = \max_{\mu,\lambda} \quad & \mu + \langle \lambda, u \rangle \\ \text{s.t.} \quad & \mu + \langle \lambda, u^{\hat{k}} \rangle \le W_{\omega^{\hat{k}}}(u^{\hat{k}}) \quad \forall \hat{k} < k \\ & \mu + \langle \lambda, u^k \rangle \le W_{\omega^k}(u^k) \\ & ||\lambda|| \le \alpha \\ & \mu, \text{ free.} \end{aligned} \tag{2.20}$$

No updates are performed on the other functions. This concludes an iteration of the algorithm. The following theorem gives the convergence result for the above algorithm.

Theorem 2.1. *Consider the sequences of functions $\bar{W}_\omega^k$ and $\underline{W}_\omega^k$ and controls u^k generated by the algorithm. With*

$$\bar{w}^k = \min_{u \in \mathcal{U}} \sum_{\omega \in \Omega} p_\omega \bar{W}_\omega^k(u), \ \ \underline{w}^k = \min_{u \in \mathcal{U}} \sum_{\omega \in \Omega} p_\omega \underline{W}_\omega^k(u), \tag{2.21}$$

we have

$$\lim_{k \to \infty} \big(\bar{w}^k - \underline{w}^k\big) = 0.$$

[1]Equation (2.19) may not have an a maximal value, so the arg max would be empty. In particular, if there exists an $D \subseteq \Omega$ for which $\underline{W}_\omega^{k-1}(u^k) = -\infty, \ \forall \omega \in D$, then define ω^k as an arbitrary element of D.

Proof. From the definition of $\underline{w}^k$ in (2.21) and u^k in (2.18) we have,

$$\underline{w}^k = \sum_{\omega \in \Omega} p_\omega \underline{W}^k_\omega(u^{k+1}), \quad \forall k. \tag{2.22}$$

For $\bar{w}^k$, we have

$$\bar{w}^k \leq \sum_{\omega \in \Omega} p_\omega \bar{W}^k_\omega(u^{k+1}), \quad \forall k.$$

It follows then that,

$$\bar{w}^k - \underline{w}^k \leq \sum_{\omega \in \Omega} p_\omega \big(\bar{W}^k_\omega(u^{k+1}) - \underline{W}^k_\omega(u^{k+1})\big), \quad \forall k.$$

From the ω-criterion in (2.19), we have that

$$\bar{w}^k - \underline{w}^k \leq |\Omega| p_{\omega^{k+1}} \big(\bar{W}^k_{\omega^{k+1}}(u^{k+1}) - \underline{W}^k_{\omega^{k+1}}(u^{k+1})\big) \quad \forall k.$$

From Lemma 2.5, the RHS goes to zero as $k \to \infty$, concluding the proof. □

This demonstrates the usefulness of the upper-bound function; it provides the algorithm with the ability to sample a particular $\omega \in \Omega$ in a fashion which is guaranteed to converge. The difference between this algorithm and Benders is that only one function is sampled at every iteration, rather than the $|\Omega|$ function samples required for Benders. However, this comes at a cost of solving a multi-cut first-stage problem, and evaluating $\bar{W}^{k-1}_\omega(u^k)$ $\forall \omega \in \Omega$ at every iteration in order to determine which function $W_\omega(u)$ to sample.

3. Multistage setting

In this section, we extend the ideas in the previous section to the more difficult setting of multistage optimisation. In the previous section, our bounding functions did not converge to the true objective function, but rather their minima converged to the true function's minimum. An analogue holds for the multistage setting; we show by induction that nested cost function approximations (both upper and lower) converge at the *state trajectories* generated by the our proposed algorithm. Once again, we begin by studying the deterministic problem first, and then we move on to the stochastic version of the problem.

3.1. The deterministic case

We consider the following general deterministic multistage optimisation problem:

$$\begin{aligned} \min_{x,u} \quad & \sum_{t=0}^{T-1} C_t(x_t, u_t) + V_T(x_T) \\ \text{s.t.} \quad & x_0 \text{ is given}, \\ & x_{t+1} = f_t(x_t, u_t) \quad \forall t \in \{0, \dots, T-1\} \\ & x_t \in \mathcal{X}_t \quad \forall t \in \{0, \dots, T\} \\ & u_t \in \mathcal{U}_t(x_t) \quad \forall t \in \{0, \dots, T-1\}. \end{aligned} \tag{3.1}$$

We can write the above in a dynamic programming formulation for $t \in \{1, \dots, T-1\}$ as

$$\begin{aligned} V_t(x_t) = \min_{x_{t+1}, u} \quad & C_t(x_t, u_t) + V_{t+1}(x_{t+1}) \\ \text{s.t.} \quad & x_{t+1} = f_t(x_t, u_t) \\ & x_{t+1} \in \mathcal{X}_{t+1} \\ & u_t \in \mathcal{U}_t(x_t). \end{aligned} \tag{3.2}$$

We require several conditions on the functions and multifunctions above. Briefly, the conditions ensure that two properties hold: that all the optimisation problems defining $V_t(x_t)$ are always convex, feasible and bounded, and that a constraint qualification is satisfied (giving the existence of sub-gradients). We direct the reader to Appendix A for their full technical description. Our algorithm is as follows: at iteration 0, define $\underline{V}_t^0 := -\infty, \bar{V}_t^0 := \infty, \ \forall\, t \in 0, \dots, T-1$. Because the final cost function is given, we set $V_T^k = V_T, \bar{V}_T^k = V_T, \ \forall k \in \mathbb{N}$. Starting with $k = 1$, $x_0^k = x_0$, with $t = 0$, solve:

$$\begin{aligned} (u_t^k, x_{t+1}^k) = \arg\min_{u_t, x_{t+1}} \quad & C_t(x_t^k, u_t) + \underline{V}_{t+1}^{k-1}(x_{t+1}) \\ \text{s.t.} \quad & x_{t+1} = f_t(x_t^k, u_t) \\ & x_{t+1} \in \mathcal{X}_{t+1} \\ & u_t \in \mathcal{U}_t(x_t^k). \end{aligned} \tag{3.3}$$

Repeat the above[2] with $t \leftarrow t+1$ until $t = T-1$.

[2]Here we note that we need not make a forward pass to the final stage, we may terminate when $\bar{V}_{t+1}^k(x_{t+1}^k) - \underline{V}_{t+1}^k(x_{t+1}^k) = 0$. This occurs in the final stage automatically, but may occur earlier in the forward pass, saving computational expense.

We then begin the backward pass of the algorithm. Starting at $t \leftarrow T-1$, we solve,

$$\begin{aligned} \underline{\theta}_t^k = \min_{x_{t+1}, u_t} \quad & C_t(x_t^k, u_t) + \underline{V}_{t+1}^k(x_{t+1}) \\ \text{s.t.} \quad & x_{t+1} = f_t(x_t^k, u_t) \\ & x_{t+1} \in \mathcal{X}_{t+1} \\ & u_t \in \mathcal{U}_t(x_t^k), \end{aligned} \tag{3.4}$$

and compute the sub-gradients (β_t^k) with respect to x_t^k. Define the lower-bound cost function as

$$\underline{V}_t^k(x) := \max\{\underline{V}_t^{k-1}(x), \underline{\theta}_t^k + \langle \beta_t^k, x - x_t^k \rangle\}. \tag{3.5}$$

Solve

$$\begin{aligned} \bar{\theta}_t^k = \min_{x_{t+1}, u_t} \quad & C_t(x_t^k, u_t) + \bar{V}_{t+1}^k(x_{t+1}) \\ \text{s.t.} \quad & x_{t+1} = f_t(x_t^k, u_t) \\ & x_{t+1} \in \mathcal{X}_{t+1} \\ & u_t \in \mathcal{U}_t(x_t^k), \end{aligned} \tag{3.6}$$

in order to update the upper-bound cost function as:

$$\begin{aligned} \bar{V}_t^k(x) = \max_{\mu, \lambda} \quad & \mu + \langle \lambda, x \rangle \\ \text{s.t.} \quad & \mu + \langle \lambda, x_t^{\hat{k}} \rangle \leq \bar{\theta}_t^{\hat{k}} \quad \forall \hat{k} \leq k-1 \\ & \mu + \langle \lambda, x_t^k \rangle \leq \bar{\theta}_t^k \\ & ||\lambda||_q \leq \alpha \\ & \mu \text{ free.} \end{aligned} \tag{3.7}$$

We repeat this process with $t \leftarrow t-1$ until $t = 0$, and set $k \leftarrow k+1$. That concludes an iteration of the algorithm.

Before proceeding with the proposed proof of convergence, several important preliminary remarks need to be made. In [5], the authors demonstrate the important result of the Lipschitz-continuity of the value functions of our problem described in (3.2) and their lower-bound estimates in (3.5). This result allows the use of arguments similar to that for the proof of convergence for Kelley's Method in their own convergence proof. We take advantage of their result for our proof regarding the Lipschitz-continuity of $V_t(x_t)$ and $\underline{V}_t^k(x_t)$. Additionally, our proof requires a similar result with respect to the upper-bounding functions $\bar{V}_t^k(x_t)$. We remind the reader that the Lipschitz-continuity of our upper-bound functions is, of course, automatic and is given by (3.7) combined with Lemma 2.1; no further analysis is required.

With regards to the particular Lipschitz constant employed in the proof, [5] give an expression for a bound on the sub-gradients exhibited by the value function estimates. For the purposes of our proof, we require only that such a constant exists.

Theorem 3.1 below gives the result that the upper and lower bounds converge at the optimal state trajectories. However, we first present Lemma 3.1, which provides a useful result which is leveraged in our proof of Theorem 3.1.

Lemma 3.1. *Consider the sequences of functions $\bar{V}_t^k$ and $\underline{V}_t^k$, and state trajectories x_t^k generated by the algorithm. For all $t = 0, \ldots, T-1$, we have*

$$\lim_{k\to\infty} \left(\bar{V}_t^k(x_t^k) - \underline{V}_t^k(x_t^k)\right) = 0 \implies \lim_{k\to\infty} \left(\bar{V}_t^k(x_t^{\hat{k}(k)}) - \underline{V}_t^k(x_t^{\hat{k}(k)})\right) = 0, \tag{3.8}$$

where $\hat{k}(k) > k$.

In the following proof, we drop the dependency of $\hat{k}(k)$ on k for notational convenience.

Proof. Suppose that there exists an $\epsilon > 0$ for which

$$\epsilon \leq \left(\bar{V}_t^k(x_t^{\hat{k}}) - \underline{V}_t^k(x_t^{\hat{k}})\right), \quad \forall k.$$

Subtracting the LHS of (3.8) from both sides gives

$$\epsilon - \left(\bar{V}_t^k(x_t^k) - \underline{V}_t^k(x_t^k)\right) \leq \left(\bar{V}_t^k(x_t^{\hat{k}}) - \underline{V}_t^k(x_t^{\hat{k}})\right) - \left(\bar{V}_t^k(x_t^k) - \underline{V}_t^k(x_t^k)\right) \quad \forall k.$$

Because both $\bar{V}_t^k$ and $\underline{V}_t^k$ are α-Lipschitz on $\mathcal{X}_t$, we have

$$\epsilon - \left(\bar{V}_t^k(x_t^k) - \underline{V}_t^k(x_t^k)\right) \leq 2\alpha ||x_t^{\hat{k}} - x_t^k||, \quad \forall k.$$

Because

$$\lim_{k\to\infty} \left(\bar{V}_t^k(x_t^k) - \underline{V}_t^k(x_t^k)\right) = 0,$$

there exists some $\tilde{k}$ large enough for which $\left(\bar{V}_t^k(x_t^k) - \underline{V}_t^k(x_t^k)\right) \leq \frac{\epsilon}{2}$. This gives

$$\frac{\epsilon}{2} \leq 2\alpha ||x_t^{\hat{k}} - x_t^k||, \quad \forall k > \tilde{k}.$$

Dividing through, we have

$$\frac{\epsilon}{4\alpha} \leq ||x_t^{\hat{k}} - x_t^k||, \quad \forall k > \tilde{k},$$

which is a contradiction of the compactness of $\mathcal{X}_t$. So there exists no such $\epsilon > 0$ for which

$$\epsilon \leq \bar{V}_t^k(x_t^{\hat{k}}) - \underline{V}_t^k(x_t^{\hat{k}}) \quad \forall k,$$

concluding the proof. □

Theorem 3.1. *Consider the sequence functions $\bar{V}_t^{\hat{k}}$ and $\underline{V}_t^{\hat{k}}$, and state and control trajectories (x_t^k, u_t^k) generated by the algorithm. For all $t = 0, \ldots, T-1$, we have*

$$\lim_{k\to\infty} \left(\bar{V}_t^k(x_t^k) - \underline{V}_t^k(x_t^k)\right) = 0.$$

Proof. The proof proceeds with a backward induction. At time $t+1$, the induction hypothesis is

$$\lim_{k\to\infty} \left(\bar{V}_{t+1}^k(x_{t+1}^k) - \underline{V}_{t+1}^k(x_{t+1}^k)\right) = 0.$$

Obviously this is true for last stage by definition. Now we want to show

$$\lim_{k\to\infty} \left(\bar{V}_t^k(x_t^k) - \underline{V}_t^k(x_t^k)\right) = 0,$$

assuming the induction hypothesis. From the definition of our bounding functions in (3.7) and (3.5), we have

$$\underline{V}_t^k(x_t^k) \geq C_t(x_t^k, u_t^{k+1}) + \underline{V}_t^k(x_{t+1}^{k+1}),$$

and

$$\bar{V}_t^k(x_t^k) \leq C_t(x_t^k, u_t^{k+1}) + \bar{V}_{t+1}^k(x_{t+1}^{k+1}).$$

Subtracting these, we have

$$\bar{V}_t^k(x_t^k) - \underline{V}_t^k(x_t^k) \leq \bar{V}_{t+1}^k(x_{t+1}^{k+1}) - \underline{V}_{t+1}^k(x_{t+1}^{k+1}). \tag{3.9}$$

From Lemma 3.1 (applied at $t+1$), we have that the RHS of (3.9) goes to 0 as $k \to \infty$ completing the proof.

□

3.2. The stochastic case

In this section, we extend the multistage deterministic model by incorporating stochasticity. The problem form is very general; rather than describe a filtration process on a particular discrete probability space, we speak in terms of a general underlying scenario tree object with $\mathcal{N}$ being the set of vertices of the tree containing the leaf vertices $\mathcal{L} \subset \mathcal{N}$.

It is not the intention of the authors to discuss the philosophy of modelling multistage stochastic optimisation problems here; we direct the reader to [13] and [3] which both contain interesting discussion on differing perspectives of multistage stochastic modelling. Whether taking a sample-average approximation approach or proposing that a particular tree truly represents the problem's uncertainty is of no concern, as both viewpoints permit scenario tree representations.

Each vertex $n \in \mathcal{N} \setminus \mathcal{L}$ has a cost function $C_n(x, u)$ and an associated state feasibility set $\mathcal{X}_n$ and control set $\mathcal{U}_n$. Arc weights p_n give the probability of arriving at vertex n from the root node. Finally, the leaves have final value functions $V_m(x_m)$ $\forall m \in \mathcal{L}$ defined over $\mathcal{X}_m$. Of course, the deterministic case can be thought of as a special case of the above; one where the scenario tree is simply a path graph and all value functions have unit probabilities. The complete description of the optimisation problem is given by the following:

$$\begin{aligned} V(\hat{x}_0) = \min_{x,u} \quad & \sum_{n \in \mathcal{N} \setminus \mathcal{L}} p_n C_n(x_n, u_n) + \sum_{m \in \mathcal{L}} p_m V_m(x_m) \\ \text{s.t.} \quad & x_0 = \hat{x}_0, \\ & x_m = f_m(x_n, u_n), \quad \forall m \in R(n), \quad \forall n \in \mathcal{N} \setminus \mathcal{L} \\ & x_n \in \mathcal{X}_n, \quad \forall n \in \mathcal{N} \\ & u_n \in \mathcal{U}_n(x_n), \quad \forall n \in \mathcal{N} \setminus 0. \end{aligned} \tag{3.10}$$

The associated dynamic programming equations are given by

$$\begin{aligned} V_n(x_n) = \min_{x_m, u_n} \quad & C_n(x_n, u_n) + \sum_{m \in R(n)} \frac{p_m}{p_n} V_m(x_m) \\ \text{s.t.} \quad & x_m = f_m(x_n, u_n), \quad \forall m \in R(n) \\ & x_m \in \mathcal{X}_m, \quad \forall m \in R(n) \\ & u_n \in \mathcal{U}_n(x_n). \end{aligned} \tag{3.11}$$

where $R(n)$ is the set of immediate children of vertex n. The vertex $P(n)$ is defined as the parent of vertex n. The symbol n here refers to a vertex in the scenario tree rather than the dimension of an associated Euclidean space. Figure 2 depicts the staging structure used for the problem description. We note that this is not restrictive; differing staging frameworks can be reconciled with an appropriate transformation of variables. We require the same conditions as

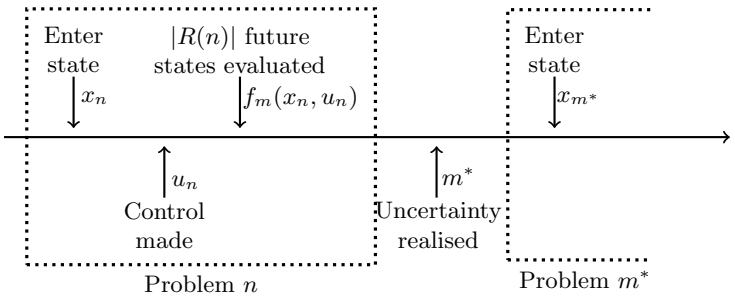

Figure 2: The staging considered in our problem formulation.

those laid out for the deterministic problem (3.1), but rather than the conditions holding for every stage of the problem, we have them for every vertex in the tree.

Much like the deterministic multistage algorithm, every iteration of the algorithm has two phases; a forward and backward pass. Our algorithm is as follows: At iteration 0, define $\underline{V}_n^0 := -\infty, \bar{V}_n^0 := \infty,\ \forall n \in \mathcal{N}\backslash\mathcal{L}$. We set $\underline{V}_m^k = V_m(x), \bar{V}_n^k(x) = V_n(x),\ \forall n \in \mathcal{L}$. We introduce a new object Φ_n^k, which keeps track of child vertex of vertex n with the maximal difference in bounds at iteration k during the forward pass. Starting with $k = 1$, $x_0^k = x_0$, with $n = 0$, we begin an iteration. For the current node n, solve

$$\begin{aligned}(\dot{u}_n, \dot{x}_m) = \underset{u_n, x_m}{\arg\min} \quad & C_n(x_n^k, u) + \sum_{m \in R(n)} \frac{p_m}{p_n} \underline{V}_m^{k-1}(x_m) \\ \text{s.t.} \quad & x_m = f_m(x_n^k, u_n), \quad \forall m \in R(n), \\ & x_m \in \mathcal{X}_m, \quad \forall m \in R(n), \\ & u_n \in \mathcal{U}_n(x_n^k).\end{aligned} \tag{3.12}$$

Next, update Φ_n^k as:

$$\Phi_n^k = \underset{m \in R(n)}{\arg\max} \ \frac{p_m}{p_n}\big(\bar{V}_m^{k-1}(x_m^k) - \underline{V}_m^{k-1}(x_m^k)\big). \tag{3.13}$$

Further, update u_n^k as $\dot{u}_n$, and $x_{\Phi_n^k}^k$ as $\dot{x}_{\Phi_n^k}$. Set $n := \Phi_n^k$ – if Φ_n^k is not unique, then select one arbitrarily. If Φ_n^k is a leaf node, terminate the forward pass. Otherwise, continue from (3.12) with the updated vertex n. We now begin the backward pass of the algorithm. Starting with the last vertex visited in the forward pass n, we solve

$$\begin{aligned}\underline{\theta}_n^k = \min_{x_m, u_n} \quad & C_n(x_n^k, u_n) + \sum_{m \in R(n)} \frac{p_m}{p_n} \underline{V}_m^k(x_m) \\ \text{s.t.} \quad & x_m = f_m(x_n^k, u_n), \quad \forall m \in R(n), \\ & x_m \in \mathcal{X}_m, \quad \forall m \in R(n), \\ & u_n \in \mathcal{U}_n(x_n).\end{aligned} \tag{3.14}$$

Compute the sub-gradients (β_n^k) with respect to x_n^k then define the lower-bound value function as

$$\underline{V}_n^k(x) := \max\{\underline{V}_n^{k-1}(x), \underline{\theta}_n^k + \langle \beta_n^k, x - x_n^k \rangle\}. \tag{3.15}$$

Solve

$$\begin{aligned}\bar{\theta}_n^k = \min_{x_m, u_n} \quad & C_n(x_n^k, u_n) + \sum_{m \in r(n)} \frac{p_m}{p_n} \bar{V}_m^k(x_m) \\ \text{s.t.} \quad & x_m = f_m(x_n^k, u_n), \quad \forall m \in R(n), \\ & x_m \in \mathcal{X}_m, \quad \forall m \in R(n), \\ & u_n \in \mathcal{U}_n(x_n),\end{aligned} \tag{3.16}$$

in order to update the upper-bound value function as

$$\begin{aligned} \bar{V}_n^k(x) = \max_{\mu,\lambda} \quad & \mu + \langle \lambda, x \rangle \\ \text{s.t.} \quad & \mu + \langle \lambda, x_n^{\hat{k}} \rangle \leq \bar{\theta}_n^{\hat{k}} \quad \forall \hat{k} \leq k-1 \\ & \mu + \langle \lambda, x_n^k \rangle \leq \bar{\theta}_n^k \\ & ||\lambda||_q \leq \alpha \\ & \mu, \text{ free.} \end{aligned} \tag{3.17}$$

We repeat this process from 3.14 with $n \leftarrow P(n)$ until $n = 0$. This concludes an iteration of the algorithm.

In the following theorem, we prove that the algorithm that we have detailed above for solving (3.10) is convergent.

Theorem 3.2. *Consider the sequences of functions $\bar{V}_n^k$ and $\underline{V}_n^k$ and state, control and maximal-child sets, (x_n^k, u_n^k, Φ_n^k) generated by the algorithm. For all $n \in \mathcal{N} \backslash \mathcal{L}$, we have*

$$\lim_{k\to\infty} \left(\bar{V}_n^k(x_n^k) - \underline{V}_n^k(x_n^k)\right) = 0.$$

Proof. The proof proceeds with a backwards induction. At nodes $m \in R(n)$, the induction hypothesis is

$$\lim_{k\to\infty} \left(\bar{V}_m^k(x_m^k) - \underline{V}_m^k(x_m^k)\right) = 0.$$

Obviously this is true for $m \in \mathcal{L}$ by definition. Now we want to show

$$\lim_{k\to\infty} \left(\bar{V}_n^k(x_n^k) - \underline{V}_n^k(x_n^k)\right) = 0,$$

assuming the induction hypothesis. From the definition of our bounding functions in (3.15) and (3.17), we have

$$\underline{V}_n^k(x_n^k) \geq C_t(x_n^k, u_n^{k+1}) + \sum_{m\in R(n)} \frac{p_m}{p_n} \underline{V}_m^k(x_m^{k+1}),$$

and

$$\bar{V}_n^k(x_n^k) \leq C_t(x_n^k, u_n^{k+1}) + \sum_{m\in R(n)} \frac{p_m}{p_n} \bar{V}_m^k(x_m^{k+1}),$$

Subtracting these, we have

$$\bar{V}_n^k(x_n^k) - \underline{V}_n^k(x_n^k) \leq \sum_{m\in R(n)} \frac{p_m}{p_n} \left(\bar{V}_m^k(x_m^{k+1}) - \underline{V}_m^k(x_m^{k+1})\right).$$

From (3.13) we have

$$\bar{V}_n^k(x_n^k) - \underline{V}_n^k(x_n^k) \leq |R(n)| \frac{p_{\Phi_n^{k+1}}}{p_n} \big(\bar{V}^k_{\Phi_n^{k+1}}(x^{k+1}_{\Phi_n^{k+1}}) - \underline{V}^k_{\Phi_n^{k+1}}(x^{k+1}_{\Phi_n^{k+1}}) \big). \tag{3.18}$$

From Theorem 3.1, the RHS of (3.18) approaches zero as $k \to \infty$, completing the proof. □

3.3. Discussion

The algorithm detailed above is deterministic, it requires no random sampling. Furthermore, there is no requirement to form an upper-bound through policy simulation. The algorithm agrees with a certain intuition: that if a path which is guaranteed to have 'disagreement' in its bounds is sampled and improved, then the bound gaps over the whole tree must converge in the limit.

In [5], the authors present an algorithm *class*. For instance, the CUPPS and DOASA algorithms are both in this class, as they only differ in when cuts are formed and included in the value function estimates per major iteration. In the CUPPS approach of [3], simulation and update are performed at the same time on the same vertex; whereas in the DOASA algorithm, a complete forward pass is performed, followed by a complete backward pass (as illustrated in Figure 3) – both algorithms are convergent. We have refrained from including this generalisation into our algorithm (we present only a DOASA approach) lest we obfuscate our algorithm's upper-bound and tree sampling aspects.

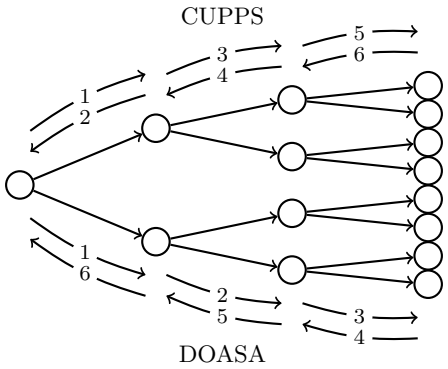

Figure 3: CUPPS and DOASA algorithms acting on a scenario tree.

The algorithm in general does not exploit any particular structure in the scenario tree; the tree object is simply used as a means to describe a proof in the most general setting. In practice, what makes these problems tractable is the effective compression of the scenario trees brought on by a particular structure in the

uncertainty. In any one iteration, the information we learn at a particular vertex could be valid for many vertices in our tree. This idea is commonly referred to as *cut sharing* and is discussed extensively in [7].

Our algorithm provides two major advantages. Firstly, the forward pass is guided and visits vertices in the tree which are guaranteed to 'discover' the most about the value functions. Secondly, a deterministic upper-bound can be computed in a way which takes advantage of the structure of scenario tree in exactly the same way as described in [7] for the lower-bounds. We refer to this as *column sharing*; it is the key to forming a deterministic upper-bound for a given policy without enumeration.

For random sampling methods, Philpott and Guan in [11] show that almost-sure convergence can be achieved after a finite number of iterations if value functions are polyhedral. We believe that under the same assumptions, their finite-time convergence arguments hold analogously for our algorithm as well.

4. Computational experiments

In this section, we present the results of several computational experiments. Two algorithms are considered in this section; the first being the algorithm described in this paper, and the second being a multi-cut DOASA algorithm. We note that these algorithms share many similarities. Indeed, the former algorithm becomes the latter when the upper-bound information is neglected and the forward pass is conducted with random sampling.

We present a scalable test problem used to benchmark the implementations of our algorithms. Our implementations are written in Julia, making use of the JuMP library (see [4]) to interface with Gurobi (see [6]) to solve the sub-problems.

4.1. A test problem

In line with SDDP tradition, we will present a multistage hydro-thermal scheduling problem to benchmark the different solution techniques. Our problem consists of a three-reservoir two-chain system meeting demand at two nodes with a transmission line. The reservoirs receive stage-wise independent inflows. Demand can be met by hydro generation or by a mix of thermal generation whose fuel costs are deterministic but may differ between stages. Many other hydro-thermal models with varying degrees of complexity exist – see [10] and [12] as examples.

One such stage problem is shown in (4.1) where: l_h is the reservoir level, r_h and s_h are the release and spill from reservoir h respectively; $a_{f,i}$ is the supply of type f at node i which has a stage-dependent cost $c^t_{f,i}$ per unit; $\mathcal{U}(l)$ represents

the reservoir dynamics and reservoir capacities; and $f_m(r_h, s_h, l_h^t)$ are the linear dynamics of the system (containing a random inflow term).

$$
\begin{aligned}
V_n(l_h^t) = \min_{a, l_{h,m}^{t+1}, r_h, s_h} \quad & \sum_{i \in \mathcal{I}} \sum_{f \in \mathcal{F}} c_{f,i}^t a_{f,i} + \mathbb{E}[V_m^{t+1}(l_{h,m}^{t+1})] \\
\text{s.t.} \quad & 0 \leq a_{f,i} \leq K_{f,i} \\
& \sum_{f \in \mathcal{F}} a_{f,1} + \sum_{h \in \mathcal{H}} r_{h,1} + t = d_1^t \\
& \sum_{f \in \mathcal{F}} a_{f,2} + \sum_{h \in \mathcal{H}} r_{h,2} - t = d_2^t \\
& -T \leq t \leq T \\
& (r_h, s_h) \in \mathcal{U}_n(l_h^t) \\
& l_{h,m}^{t+1} = f_m(r_h, s_h, l_h^t), \quad \forall m \in R(n).
\end{aligned}
\tag{4.1}
$$

Because of the independence structure of the uncertainty, we are able to share cuts (and columns!) between vertices in our scenario tree. The norm considered for this problem (which appears in (3.17)) is the $||\cdot||_\infty$ norm; this allows the upper-bound function to be computed as a linear programme. Our implementation involves embedding the dual of (3.17) directly into (3.16) – similar to what is done for the lower-bound functions in many SDDP implementations.

4.2. Results

Problem 1 is a 20-stage, 9-scenario per stage instance of the problem above. Table 1 shows the results of several runs of the deterministic algorithm with a decreasing relative ϵ, where $\epsilon^k = \frac{\bar{V}_0^k(x_0) - \underline{V}_0^k(x_0)}{\bar{V}_0^k(x_0)}$. As expected, in relatively few iterations, the

Table 1: Results for Problem 1

Relative ϵ	Lower	Upper	Time (s)	Iterations	LP Solves
20.0%	406.86	506.89	48.02	295	28025
10.0%	413.10	457.63	137.26	533	50605
5.0%	416.25	438.06	396.39	929	88180
1.0%	419.01	423.24	7461.57	3653	346080

algorithm is able produce a 'good' policy; similar to policies generated by methods that use random sampling. Much more computational effort is required to reduce the relative bound gap as the gap begins to close. We will discuss these ideas further in relation to Figures 4 and 5. It is worth mentioning here that without our method, a policy evaluation (in order to form a *deterministic* upper-bound)

for this problem would require 1,519,708,182,382,116,100 ($\approx 1.52 \times 10^{18}$) linear programme evaluations.

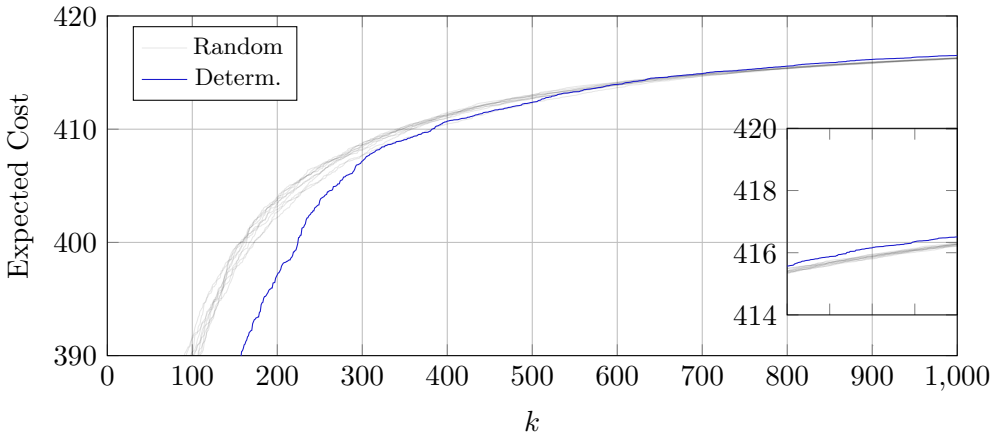

Figure 4: Lower-bounds on the Expected Cost between the two algorithms.

Comparing results between between our method and other methods requires careful consideration. We will first discuss the results shown in Figure 4. This figure plots the the cost of the the lower-bound at each iteration i.e. $V_0^k(x_0)$. Note that because DOASA uses random sampling, we have plotted the results of several runs of the algorithm on this problem, each with different random seeds. Here we can see that initially our lower-bound lags behind the lower-bounds generated by DOASA with random sampling. However, at iteration 600, we can see the lower-bound of our method begins to match the others. Beyond this, our method delivers a higher lower-bound, for the same number of iterations. We believe the early relatively poor lower-bound performance is due to the fact that the algorithm's sampling is prioritising bringing the upper-bound down. We conclude here that a random sampling scheme under stage-wise independence provides a coarse lower-bound estimate quite quickly, but struggles to make the marginal gains towards the later portion of the algorithm.

We conjecture that one reason for the good early performance of the random sampling method is the relatively simple uncertainty structure (stage-wise independence) in our example. Moreover, the reason the random sampling method cannot make those small gains towards the end, is that these may be associated with specific sample paths that have not been sampled, whereas our method identifies a path to search which guarantees and improvement at each iteration.

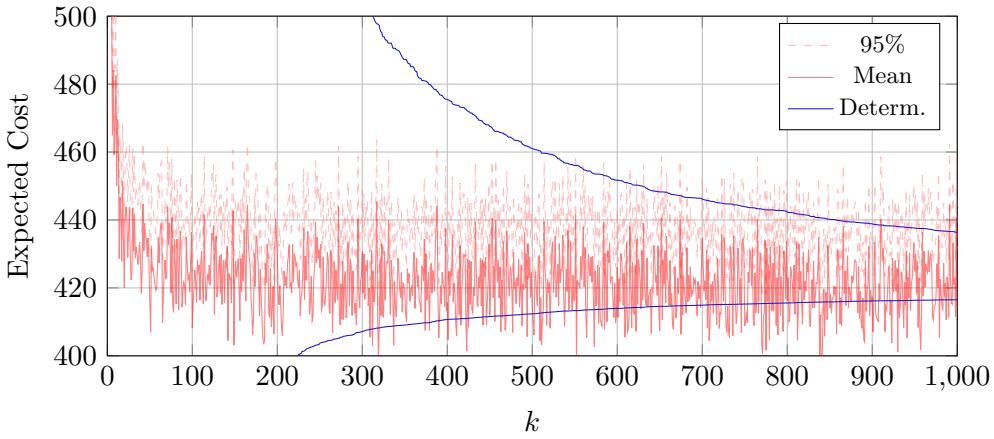

Figure 5: Convergence plot with Monte Carlo estimates for an upper bound.

4.3. Solution verification

Figure 5 shows both the upper-bound and lower-bound of the deterministic method, and an 800 sample Monte Carlo estimate of the expected cost of a policy generated from random sampling[3]. The dashed line is the upper-bound of a 95% confidence interval constructed about the mean i.e. we are 95% sure that the expected cost of the current policy lies below this line and above the 95% confidence interval lower bound (not shown in the plot). The construction

Table 2: Monte Carlo estimates required for convergence tests

Gap	Confidence Level			
	95%	97.5%	99%	99.9%
10%	440	574	762	1238
5%	1832	2393	3174	5161
1%	48690	63595	84366	137189

of these confidence intervals (to check for convergence) can be computationally expensive and requires the user to determine how often this check is performed, and to what confidence level. We contrast this with our deterministic algorithm: for each iteration of our algorithm, we have a bound on the maximum distance the current policy is from optimality.

[3]The authors would like to acknowledge Oscar Dowson for kindly providing the code which performed the Monte Carlo simulation from a policy generated from a random sampling DOASA implementation.

In Table 2 above, we investigate the computational expense required to build confidence intervals to various tolerance levels (% gap) assuming the variance given by the policy in Figure 5 at iteration 1000. Specifically, we compute the number of Monte Carlo samples required to deliver a confidence interval with the same width as 1%, 5% or 10% of the best lower-bound for the objective function. For example, almost 140,000 Monte Carlo samples would have to be carried out in order to form a 99.9% confidence interval with a width equal to the difference of the deterministic upper and lower bound at 1%. Because each sample requires the evaluation of 20 stage problems, this number of samples requires more than six times the number of LP solves (simply to evaluate an confidence interval) than our method would take to give a solution with a guaranteed 1% gap.

Finally, we note that convergence of a stochastic upper-bound is sensitive to the variance of the distribution of costs given by a particular policy. The higher the policy's variance, the wider the confidence interval will be for a given sample-size and confidence level. This means that random sampling algorithms may find no evidence that the policy has not converged, even when the policy is far from optimal. However, since our method that we propose in this paper is deterministic, we do not need to simulate to estimate an upper-bound and so never encounter cases where a convergence test gives either false positives or false negatives.

Acknowledgements

The authors would like to thank the members of the Electrical Power Optimisation Centre at the University of Auckland for stimulating discussiosn on the topic. Secondly, we thank Oscar Dowson for kindly providing the code used to simulate a stochastic upper-bound for Figure 5. Finally, the authors would like to recognise the support of the Energy Education Trust of New Zealand over the period of the development of this work.

Appendix A. Technical conditions

We consider the following general deterministic multistage optimisation problem:

$$\begin{aligned} \min_{x,u} \quad & \sum_{t=0}^{T-1} C_t(x_t, u_t) + V_T(x_T) \\ \text{s.t.} \quad & x_0 = \hat{x}_0, \\ & x_{t+1} = f_t(x_t, u_t) \quad \forall t \in \{0, \dots, T-1\} \\ & x_t \in \mathcal{X}_t \quad \forall t \in \{0, \dots, T\} \\ & u_t \in \mathcal{U}_t(x_t) \quad \forall t \in \{0, \dots, T-1\} \end{aligned} \tag{A.1}$$

with the following conditions:

1. for all t, $\mathcal{X}_t$ is a non-empty subset of $\mathbb{R}^n$;
2. for all t, multifunctions $\mathcal{U}_t : \mathbb{R}^n \mapsto \mathbb{R}^m$ are convex and non-empty compact valued;
3. the final cost function V_T is a convex lower semicontinous proper function. The cost functions $C_t(x, u)$ $\forall t < T$ are convex lower semicontinous proper functions of x and u;
4. for all $t < T$, functions f_t are affine;
5. the final cost function V_T is finite-valued and Lipschitz-continuous on $\mathcal{X}_T$;
6. for all $t < T$, there exists $\delta_t > 0$, defining $\mathcal{X}'_t := \mathcal{X}_t + B_t(\delta_t)$, where

$$B_t(\delta) = \{y \in \text{Aff}(\mathcal{X}_t) \mid ||y|| \leq \delta\}$$

such that

(a) $\forall x \in \mathcal{X}'_t,\ \forall u \in \mathcal{U}_t(x),\ C_t(x, u) < \infty$;

(b) $\forall x \in \mathcal{X}'_t$

$$f_t(x, \mathcal{U}_t(x))) \cap \mathcal{X}_{t+1} \text{ is non-empty.}$$

Conditions 1-5 ensure that (A.1) and its value function definitions are convex optimisation problems. Condition 6 is designed to give a constraint qualification condition for the optimisation problem (as it is non-linear in general). In [5], the authors call this condition *extended relatively complete recourse*, which is a more general class of recourse (which includes complete recourse). In their analysis, the existence of δ_t allows the construction of a Lipschitz constant to give their Lipschitz-continuity of $V_t(x_t)$ and $\underline{V}_t(x_t)$ result. We note that these conditions are identical to those required by [5] in their paper.

References

[1] J.F. Benders. Partitioning procedures for solving mixed-variables programming problems. *Numerische Mathematik*, 4:238–252, 1962/63. URL http://eudml.org/doc/131533.

[2] John R. Birge and François V. Louveaux. A multicut algorithm for two-stage stochastic linear programs. *European Journal of Operational Research*, 34(3):384–392, 1988. ISSN 03772217. doi: 10.1016/0377-2217(88)90159-2.

[3] Z. Chen and W. Powell. Convergent Cutting-Plane and Partial-Sampling Algorithm for Multistage Stochastic Linear Programs with Recourse 1. *Optimization*, 102(3):497–524, 1999. ISSN 0022-3239. doi: 10.1023/A:1022641805263.

[4] Iain Dunning, Joey Huchette, and Miles Lubin. JuMP: A Modeling Language for Mathematical Optimization. *arXiv:1508.01982 [math.OC]*, pages 1–25, 2015. URL http://arxiv.org/abs/1508.01982.

[5] P Girardeau, V Leclere, and A B Philpott. On the Convergence of Decomposition Methods for Multistage Stochastic Convex Programs. *Mathematics of Operations Research*, 40(1):130–145, 2014. ISSN 0364-765X. doi: 10.1287/moor.2014.0664. URL http://dx.doi.org/10.1287/moor.2014.0664.

[6] Inc. Gurobi Optimization. Gurobi Optimizer Reference Manual, 2016. URL http://www.gurobi.com.

[7] G Infanger and D P Morton. Cut Sharing for Multistage Stochastic Linear Programs with Interstage Depency. *Mathematical Programming*, 75(241-256):241–256, 1996.

[8] JE Jr Kelley. The cutting-plane method. *Journal of the Society for Industrial and Applied Mathematics*, 8(4):703–712, 1960.

[9] Yu Nesterov. Introductory Lectures on Convex Programming Volume I: Basic course. *Lecture Notes*, I:1–211, 1998. doi: 10.1007/978-1-4419-8853-9. URL http://enpub.fulton.asu.edu/cseml/Fall2008_ConvOpt/book/Intro-nl.pdf.

[10] M. V F Pereira and L. M V G Pinto. Multi-stage stochastic optimization applied to energy planning. *Mathematical Programming*, 52(1-3):359–375, 1991. ISSN 00255610. doi: 10.1007/BF01582895.

[11] A. B. Philpott and Z. Guan. On the convergence of stochastic dual dynamic programming and related methods. *Operations Research Letters*, 36(4):450–455, 2008. ISSN 01676377. doi: 10.1016/j.orl.2008.01.013.

[12] Andy Philpott and Geoff Pritchard. EMI-DOASA. 2013.

[13] Alexander Shapiro. Analysis of stochastic dual dynamic programming method. *European Journal of Operational Research*, 209(1):63–72, 2011. ISSN 03772217. doi: 10.1016/j.ejor.2010.08.007. URL http://dx.doi.org/10.1016/j.ejor.2010.08.007.