Fully Polynomial Time (Σ, Π)-Approximation Schemes for Continuous Stochastic Convex Dynamic Programs

Nir Halman * Giacomo Nannicini †

July 7, 2015

Abstract

We develop fully polynomial time (Σ, Π)-approximation schemes for stochastic dynamic programs with continuous state and action spaces, when the single-period cost functions are convex Lipschitz-continuous functions that are accessed via value oracle calls. That is, for every given additive error parameter $\Sigma > 0$ and multiplicative error factor $\Pi = 1 + \epsilon > 1$, the scheme returns a feasible solution whose value is no more that $\Sigma + \Pi v^*$, where $v^* \geq 0$ is the optimal value, and it runs in time polynomial in the input size and in $\log(\frac{1}{\Sigma}) + \frac{1}{\epsilon}$. We show that this result is best possible in the sense that in general it is not possible to compute in polynomial time (Σ, Π)-approximations for stochastic dynamic programs with continuous state and action spaces, whenever either $\Sigma = 0$ or $\Pi = 1$. Our results are achieved by extending to the continuous setting the technique of K-approximation sets and functions introduced by Halman et al. [*Math. Oper. Res.*, 34, (2009), pp. 674–685]. This extension requires a completely new set of analytical and algorithmic tools that may be useful in their own right for further development of approximation algorithms. A preliminary computational evaluation shows that the proposed approximation scheme is orders of magnitude faster than solving the discretized problem exactly, and finds solutions of better quality than approximate solution approaches taken from the literature for comparable computational effort.

Keywords: Approximation algorithms, stochastic dynamic programming, K-approximation sets and functions, (Σ, Π)-approximation sets and functions.

*Hebrew University of Jerusalem, E-mail: halman@huji.ac.il

†Engineering Systems & Design, Singapore University of Technology and Design, Singapore, E-mail: nannicini@sutd.edu.sg

1 Overview of the paper

Dynamic Programming (DP) is an algorithmic technique used for solving sequential, or multi-stage, decision problems and is a fundamental tool in combinatorial optimization (see, e.g., [Hoc97], [Vaz01, Ch. 8]). We consider a discrete-time finite-horizon stochastic dynamic program (DP, to be distinguished from "dynamic programming" by context), as defined in [Ber05]. The system dynamics are of the form: $I_{t+1} = f(I_t, x_t, D_t)$ for $t = 1, \ldots, T$, where t is the discrete time index, $I_t \in \mathcal{S}_t$ is the state of the system at time t ($\mathcal{S}_t$ is the *state space* at stage t), $x_t \in \mathcal{A}_t(I_t)$ is the action or decision to be selected at time t ($\mathcal{A}_t(I_t)$ is the *action space* at stage t and state I_t), D_t is a discrete random variable over the sample space $\mathcal{D}_t$, and T is the number of time periods. In the context of this paper, D_t represents an exogenous information flow. The cost function $g_t(I_t, x_t, D_t)$ gives the cost of performing action x_t from state I_t at time t for each possible realization of the random variable D_t. In our context, I_t and x_t are one-dimensional, while D_t is a fixed-dimensional vector. The random variables are assumed independent but not necessarily identically distributed. Costs are accumulated over all time periods: the total incurred cost is equal to $\sum_{t=1}^{T} g_t(I_t, x_t, D_t) + g_{T+1}(I_{T+1})$. In this expression, $g_{T+1}(I_{T+1})$ is the cost paid if the system ends in state I_{T+1}, and the sequence of states is defined by the system dynamics. The problem is that of choosing a sequence of actions $x_1, \ldots, x_T$ that minimizes the expectation of the total incurred cost. This problem is called a *stochastic dynamic program*. Formally, we want to determine:

$$z^*(I_1) = \min_{x_1,\ldots,x_T} E\left[g_1(I_1, x_1, D_1) + \sum_{t=2}^{T} g_t(f_t(I_{t-1}, x_{t-1}, D_{t-1}), x_t, D_t) + g_{T+1}(I_{T+1})\right], \quad (1)$$

where I_1 is the initial state of the system and the expectation is taken with respect to the joint probability distribution of the random variables D_t. It is well known that such a problem can be solved through a recursion.

Theorem 1.1 (Bellman optimality equation [BD62]) *For every initial state I_1, the optimal value $z^*(I_1)$ of the DP is given by $z_1(I_1)$, where z_1 is the function defined by $z_{T+1}(I_{T+1}) = g_{T+1}(I_{T+1})$ together with the recursion:*

$$z_t(I_t) = \min_{x_t \in \mathcal{A}_t(I_t)} E_{D_t}\left\{g_t(I_t, x_t, D_t) + z_{t+1}(f_t(I_t, x_t, D_t))\right\}, \quad t = 1, \ldots, T. \quad (2)$$

The function $z_t(I_t)$ in the recursion above is called the *cost-to-go function* or *value function*.

We distinguish between *discrete* and *continuous* state and action spaces. In the former case, which we call *discrete DP*, the state and action spaces are finite sets of elements, in the latter case, which we call *continuous DP*, they are compact intervals on the real line. Combinatorial optimization problems such as the knapsack problem are naturally cast as discrete DPs. In other contexts, e.g., economics and finance, the state and action may be best represented as continuous variables, for instance when they represent money; examples are provided in [AC03, Ch. 5], [NP10, Phe62]. This paper deals with continuous DPs, and we assume that our goal is that of computing $z_1(I)$ for the entire initial state space $\mathcal{S}_1$, as is customary in DP. Several specific applications of our continuous DP framework are discussed in Section 3. We note that computing the value function at a single point, i.e. $z^*(I_1)$, is already computationally intractable: Corollary 2.6 of this paper shows that even an *approximation* of $z^*(I_1)$ within any constant ratio is not possible in finite time.

Given a minimization problem and a bound $\epsilon > 0$ on the relative error, a $(1+\epsilon)$-approximation algorithm finds a solution in time polynomial in the input size, where the value of the objective function is at most $(1+\epsilon)$-times the optimal value, i.e., with *multiplicative error* of at most $1+\epsilon$. A common alternative error measure is additive error. Given a bound $\Sigma > 0$ on the additive error, an additive Σ-approximation algorithm finds a solution in time polynomial in the input size, where the value of the objective function differs from the optimal value by at most Σ. If ϵ (resp. Σ) are parameters given explicitly to the algorithm and the running time of the approximation algorithm is also polynomial in $1/\epsilon$ (resp. $\log(1/\Sigma)$), then we call it a *fully polynomial approximation scheme (FPTAS)*. While the factor $\log(1/\Sigma)$ can be negative, it is implicitly assumed that the polynomials are intended to be nonnegative.

Our results. A common approach to deal with continuous state and action spaces is to discretize them with a stepsize $\delta > 0$, and then solve (or approximate) the resulting discrete problem. A finer discretization results in smaller accuracy loss. If the multiplicative error introduced could be bounded as a function of δ, and if the corresponding discrete problem admits a K-approximation algorithm for some constant $K > 1$, then one could get a KL-approximation for the continuous problem by approximating the discretized problem, for any constant $L > 0$. A similar argument works for additive approximations. In this paper we show that such a scheme may fail:

Theorem 1.2 *There does not exist, in general, a polynomial discretization for a continuous DP with which one can bound either the relative or the absolute error between the value of an optimal solution of the continuous DP and that of the discrete DP.*

The discussion above leads to the question of how to approximately solve a continuous DP with rigorous error bounds. We propose to combine the additive and multiplicative error measures into a single 2-dimensional error measure that can be infinitely small. That is, we introduce a new type of approximation scheme. For a bound $\Pi = 1 + \epsilon > 1$ on the *multiplicative* error and a bound $\Sigma > 0$ on the *absolute* error, a (Σ, Π)-approximation algorithm finds a feasible solution with value at most $\Pi v^* + \Sigma$, where $v^* \geq 0$ is the value of an optimal solution. If such a (Σ, Π)-approximation algorithm runs in time polynomial in the input size, we call it a *polynomial time* (Σ, Π)*-approximation scheme*. If the running time is also polynomial in $1/\epsilon$ and $\log(1/\Sigma)$, we call it a *fully polynomial time* (Σ, Π)*-approximation scheme*, abbreviated as (Σ, Π)-FPTAS. We argue that because additive and relative approximations are widely accepted measures of approximation, the same should be true for (Σ, Π)-approximations. We propose (Σ, Π)-FPTASs as efficient approximation algorithms, and show how to construct such an algorithm for continuous DPs with the following structure: the transition function f_t is linear and separable in its variables, while the cost function g_t has a convex structure (we give a formal definition in Section 3). We call this a *convex DP*. Convex DPs find many applications, some of which will be discussed in this paper (see also [HNO15] and the references therein). It is possible to extend our scheme to continuous *monotone* DPs similar to [HKL$^+$14], but this is not discussed in the present paper for lack of space. We show that a (Σ, Π)-FPTAS is in some sense "best possible" for continuous DPs, because additive or multiplicative approximations alone may not exist in general.

We distinguish between an *explicit stochastic DP*, where the distribution of each random variable is given explicitly as a list of scenarios $(d, \text{Prob}(D = d))$, and an *implicit stochastic DP*, where the distribution is given as a value oracle to its CDF. An advantage of the implicit stochastic DP model is that the assumptions are so weak that they encompass various ways of specifying random variables, in particular (truncated) well-known distributions with given parameters. We are now ready to state our second main result.

Theorem 1.3 *Every implicit stochastic convex continuous DP satisfying some technical conditions (Conditions 1, 2 and 3 in this paper) admits a fully polynomial time* (Σ, Π)*-approximation scheme.*

Relevance to existing literature. We are aware of six constructive frameworks that give FPTASs for classes of optimization problems. The ones in [Woe00, PW07, MS13] deal with deterministic discrete DPs only. The one in [SS06] deals with stochastic linear programs (LPs). The one in [HKL$^+$14] deals with explicit stochastic discrete DPs. The one in [HNO15] works under the same DP model, but provides a faster FPTAS from both theoretical and practical standpoints. Our approach draws on the idea of K-approximation sets and functions of [HKL$^+$14, HNO15], but we deal with continuous domains *directly*, rather than discretizing and approximating the discretized problem (which would fail due to Theorem 1.2). We are not aware of any FPTAS framework for implicit stochastic DP in general, although [HOSL12] provides problem-specific FPTASs for the nonlinear newsvendor and lotsizing problems when expressed as implicit stochastic DPs. While some (Σ, Π)-approximation algorithms are known, e.g., [EP04], to the best of our knowledge none of these permits both Σ and Π to be infinitely close to $0, 1$, respectively (i.e., at least one of Σ, Π is assumed to be constant). In this sense, the notion and the construction of a (Σ, Π)-*FPTAS* are novel.

Continuous DP models are frequently employed in several domains, see e.g. [AC03]. Discretization (possibly with refinement) is a standard solution approach. Typically, error bounds with respect to the optimal value function are not provided. For example, the well-known approximate linear programming approach [dFVR03] provides a bound on the approximation error as compared to the best value function approximation that can be obtained as a linear combination of a given choice of basis functions. This type of error bound is clearly different in spirit from the natural concepts of additive and multiplicative error with respect to the optimal solution. We are not aware of any approximate DP (ADP) approach that provides error bounds of the type discussed in this paper in polynomial time; thus, for space reasons instead of surveying the abundant ADP literature we refer to the comprehensive references [Ber05, Pow11]. However, algorithms that asymptotically converge to the optimal value have been proposed in the literature: the most relevant for this paper are SPAR and CAVE, see [GP01, PRT04, NP13]. In particular, [NP13] studies a class of continuous DPs very similar to ours, and proposes an algorithm that is shown to converge to the optimal policy as the number of iterations goes to infinity. It is important to note that the ADP methodologies discussed above require fewer structural properties than our approach and can be applied to a broader set of problems. We argue that (Σ, Π)-FPTASs are interesting despite the more limited applicability because they provide much stronger guarantees on the approximation error and running time than is typically found in the ADP literature.

Contributions and organization. This paper makes several contributions. First, we formally establish that continuous DPs are strictly harder to approximate than their discrete counterparts. To overcome this hardness result we introduce a novel measure of error, i.e., (Σ, Π)-approximations where each one of the parameters Σ, Π can be infinitely close to $0, 1$, respectively. We design (Σ, Π)-FPTASs for implicit stochastic continuous DPs, thus extending the framework of [HKL$^+$14] to both implicit stochastic and continuous settings. As a consequence, some of the explicit stochastic discrete DPs for which [HKL$^+$14] give an FPTAS, also admit a (Σ, Π)-FPTAS when the distributions are given implicitly and the state and action spaces are continuous. Finally, we extend the technique of K-approximation sets and functions to continuous functions and (Σ, Π)-approximations: we believe that this technique will be useful for further development of approximation algorithms. In the Appendix, we provide a preliminary computational evaluation of our approximation scheme, comparing it to solving or approximating the discretized DP, and to an ADP approach taken from the literature. The results show that our method is superior in terms of running time and solution quality, indicating that the methodology proposed in this paper has the potential of being successful in practice on the class of problems for which it applies. In particular, while discretization works quite well as a heuristic, working with the continuous problem directly yields better solutions for comparable computational effort, as suggested by Thm. 1.2.

The paper is organized as follows. In Section 2 we define our model of computation and establish our hardness results. Then, we introduce a new type of approximation functions and sets, and algorithms to compute them. In Section 3 we state the implicit-stochastic continuous DP model to which our (Σ, Π)-approximation schemes apply, and our main result for convex DPs that shows how to construct the approximation scheme. The Appendix contains the proofs and the computational evaluation.

Notation. Let $\mathbb{R}, \mathbb{R}^+, \mathbb{Z}, \mathbb{Z}^+, \mathbb{N}$ denote the set of real numbers, nonnegative real numbers, integers, nonnegative integers and positive integers, respectively. For every $a, b \in \mathbb{R}$ with $a < b$, let $[a, b] = \{x \mid a \leq x \leq b\}$. Given a finite set $D \subset \mathbb{R}$, we denote by $D^{\min}$ and $D^{\max}$ the minimum and maximum element in D, respectively. For every $A, B \in \mathbb{Z}$ with $A < B$, let $[A, \dots, B] = \{A, A+1, \dots, B\}$. For simplicity, we assume throughout that $B - A > 1$. Let X be a set, and let $Y(x)$ be a set for every $x \in X$. We denote by $X \otimes Y$ the set $\bigcup_{x \in X} Y(x)$. We write $\log z$ to denote $\log_2 z$.

When a function $\varphi : D \to \mathbb{R}$ over a linearly ordered set D is called either *increasing* or *decreasing*, it is meant in the weak sense. Let $\varphi^{\max} = \sup_{x \in D} \varphi(x)$, $\varphi^{\min} = \inf_{x \in D} \varphi(x)$, and denote by t_φ the time needed to evaluate φ at a single point in its domain. Let $D' \subseteq D \subset \mathbb{R}$ with D' finite. We define the *piecewise linear extension of φ induced by D'* as the function obtained by making φ linear between successive values of D'. A function φ over D is called *convex over D'* if its piecewise linear extension induced by D' is convex. We define the *convex extension of φ induced by D'* as the function defined as the lower envelope of the convex hull of $\{(x, \varphi(x)) \mid x \in D'\}$, which is piecewise linear.

2 Hardness results and a new type of approximation

2.1 Model of computation and black-box-oriented methods

In this paper we face the problem of efficiently constructing a succinct representation for a function $\varphi : D \to \mathbb{R}^+$, having access only to a (zero-order) value oracle that for every point $x \in D$ returns $\varphi(x)$. By a representation of a function we mean a way to compute the function values using information stored on the computer memory, e.g., store its explicit formula. A representation of a function is *succinct* if it occupies a "small" space in the computer memory, e.g., a univariate d-piecewise linear function over an interval has a representation in $d + 1$ space, consisting of its values on the breakpoints and endpoints. In many cases, a succinct representation is either not possible or very costly, so an efficient succinct representation for an approximation of φ is of interest.

To discuss approximation algorithms, we need to define the model of computations that we use. We use the real arithmetic model as defined in [BN13, Sec. 5.1.1]. In this model, the computations are carried out by an idealized version of the usual computer that is capable of storing countably many reals and can perform the standard *exact* real arithmetic operations. Consider a problem (P) over a function $\varphi : D \to \mathbb{R}$. For example, in [BN13] (P) is the problem of finding the minimum of φ over D; in our paper, it is constructing a succinct representation of φ over D. Let us fix a family $\mathcal{F}(D)$ of problems (P) with D common for all problems from the family, so that the family itself is nothing but a certain family of functions over domain D. We define the information-based complexity of $\mathcal{F}(D)$ as in [BN13, Sec 5.1.1]; here we provide a brief summary, referring to [BN13] for the details. We use the same notation: $\delta > 0$ is an accuracy parameter to which the problem should be solved according to some accuracy measure, and $\mathcal{B}$ is an algorithm that solves (P) by generating a sequence of search points and calling the oracle to get the values of φ on these points. (Note that in [BN13] subgradients are also computed, while in our case we do not use first-order information.) $T_{\mathcal{B}}(\varphi, D, \delta)$ denotes the number of steps in which $\mathcal{B}$ is capable of solving within accuracy δ a problem φ over D. The complexity of $\mathcal{F}(D)$ w.r.t. a solution method $\mathcal{B}$ is the function $\mathrm{Compl}_{\mathcal{B}}(D, \delta) = \max_{\varphi \in \mathcal{F}(D)} T_{\mathcal{B}}(\varphi, D, \delta)$. Finally, the *information-based complexity* of the family $\mathcal{F}(D)$ of problems is defined as $\mathrm{Compl}(D, \delta) = \min_{\mathcal{B}} \mathrm{Compl}_{\mathcal{B}}(D, \delta)$, the minimum being taken over all solution methods.

2.2 Additive and multiplicative approximation functions

We now discuss the natural concepts of additive and multiplicative approximation functions, and their limitations in the context of finding succinct approximations of functions over continuous domains. We are concerned with determining which functions φ over continuous intervals on the real line admit efficient approximations, and with finding approximations for $\varphi^{\min}$ efficiently. We show that Lipschitz-continuous functions are hard to approximate additively or multiplicatively.

Definition 2.1 (Additive Σ-approximation functions) *Let $\Sigma > 0$ and $r, \tilde{r} \geq 0$ be arbitrary real numbers. We say that $\tilde{r}$ is an* additive Σ-approximation of r *if $r \leq \tilde{r} \leq r + \Sigma$. Let $\varphi, \tilde{\varphi} : D \to \mathbb{R}^+$ be real-valued functions over a continuous domain D. The function $\tilde{\varphi}$ is said to be an* additive Σ-approximation of φ *if $\varphi(x) \leq \tilde{\varphi}(x) \leq \varphi(x) + \Sigma$, $\forall x \in D$. Let $D = [A, B]$ be an interval on the real line. A function $\tilde{\varphi}$ is called a* succinct Σ-approximation of φ *if it admits a representation in space polylogarithmic in $\varphi^{\max} + B - A + 1/\Sigma$, and is an additive Σ-approximation of φ. Such a function $\tilde{\varphi}$ is said to be* efficient *if it can be constructed in time polylogarithmic in $\varphi^{\max} + B - A + 1/\Sigma$.*

Proposition 2.2 *For every positive real numbers $\Sigma, \kappa > 0$ and a κ-Lipschitz continuous convex function $\varphi : [A, B] \to \mathbb{R}^+$, the function* ApxArgMin *finds a point D' for which $\varphi(D')$ is an additive Σ-approximation of $\varphi^{\min}$ in $O(\log(\kappa(B - A)/\Sigma))$ queries to φ.*

Remark. In Proposition 2.2, as well as in all subsequent algorithmic results in this paper, the value of κ need not be given explicitly.

Proposition 2.3 *Let $\Sigma > 0$ be an arbitrary real number, A be an arbitrary positive integer number, and let $\kappa := 2A\Sigma$. A κ-Lipschitz continuous monotone convex function $\varphi : [0, A] \to \mathbb{R}^+$ does not necessarily admit a succinct additive Σ-approximation.*

Function ApxArgMin(φ, A, B, Σ)
Let $A' \leftarrow A,\ B' \leftarrow B,\ \kappa' \leftarrow 3\Sigma/(B' - A')$
while $\kappa'(B' - A') > 2\Sigma$ **do**
  Divide $[A', B']$ into 4 equal-size subintervals and perform queries to φ on the corresponding 5 endpoints
  let D' be an argmin of these values
  if φ is not monotone on these endpoints **then** let $C' \leftarrow D'$
  elseif φ is increasing on these points **then** let C' be the neighbor of A'
  else let C' be the neighbor of B'
  let A' be the neighbor of C' to the left and B' be its neighbor to the right
  Let l_{AC} be the line that passes through $(A', \varphi(A'))$ and $(C', \varphi(C'))$, and let Δ_{AC} be its slope
  Let l_{CB} be the line that passes through $(C', \varphi(C'))$ and $(B', \varphi(B'))$, and let Δ_{CB} be its slope
  $\kappa' \leftarrow \max\{|\Delta_{AC}|, |\Delta_{CB}|\}$
end while
return D'

Algorithm 1: FUNCTION APXARGMIN(φ, A, B, Σ)

Propositions 2.2 and 2.3 imply that constructing a succinct additive approximation function is strictly harder than finding an additive approximation of $\varphi^{\min}$.

We now discuss multiplicative approximations. A definition of multiplicative approximation functions for functions over finite discrete domains is given in [HKL$^+$14]. The notation used there is K-approximation functions.

Definition 2.4 (Multiplicative Π-approximation functions) *Let $\Pi = 1 + \epsilon > 1$ and $r, \tilde{r} \geq 0$ be arbitrary reals. We say that $\tilde{r}$ is a* multiplicative Π-approximation of r *or* relative ϵ-approximation of r *if $r \leq \tilde{r} \leq \Pi r$. Let $\varphi, \tilde{\varphi} : D \to \mathbb{R}^+$ be real-valued functions over a continuous domain D. The function $\tilde{\varphi} : D \to \mathbb{R}^+$ is said to be a* multiplicative Π-approximation of φ *if $\varphi(x) \leq \tilde{\varphi}(x) \leq \Pi\varphi(x),\ \forall x \in D$. Let $D = [A, B]$ be an interval on the real line. We call a function $\tilde{\varphi}$ a* succinct Π-approximation of φ *if it admits a representation in space polynomial in $\log \varphi^{\max} + \log(B - A) + 1/\epsilon$, and is a Π-approximation of φ. Such a function $\tilde{\varphi}$ is said to be* efficient *if it can be constructed in time polynomial in $\log \varphi^{\max} + \log(B - A) + 1/\epsilon$.*

Proposition 2.5 *Let $\Pi > 1$ be an arbitrary real number, A be an arbitrary positive integer number, and $\varphi : [-A, A] \to \mathbb{R}^+$ be a 1-Lipschitz continuous convex function. It is not possible, in general, to find a Π-approximation of $\varphi^{\min}$ in any finite number of queries to φ.*

Corollary 2.6 *For any fixed multiplicative error $\Pi > 1$, it is not possible, in general, to Π-approximate in finite time a continuous convex DP, even if the time horizon consists of a single period and the cost function is 1-Lipschitz continuous.*

Corollary 2.7 *Let $\Pi > 1$ be an arbitrary real number and A be an arbitrary positive integer number. A convex function $\varphi : [-A, A] \to \mathbb{R}^+$ does not necessarily admit a succinct Π-approximation, even if it is 1-Lipschitz continuous.*

2.3 (Σ, Π)-approximation functions

Let $\mathcal{F}([\mathcal{A}, \mathcal{B}])$ be the family of either convex or monotone functions φ over interval $[A, B]$ that are κ-Lipschitz continuous. Let (P) be the problem of the construction a succinct representation for a δ-approximation of φ. The previous subsection shows that Compl$([A, B], \delta)$ is not polynomial in $\log(B - A) + 1/\delta$, when the measure of approximation is either δ-additive or δ-relative error. In other words, the notions of efficient additive succinct approximation and efficient multiplicative approximation are too strong to be used for the family $\mathcal{F}([\mathcal{A}, \mathcal{B}])$. In this section we introduce a measure of approximation that is "natural" to use and for which Compl$([A, B], \delta)$ is polynomial in $\log(B - A) + 1/\delta$, as will be shown in Section 2.4.

Definition 2.8 ((Σ,Π)-approximation functions) *Let $\Sigma \geq 0,\ \Pi = 1+\epsilon \geq 1$, and $r, \tilde{r} \geq 0$ be arbitrary real numbers. We say that $\tilde{r}$ is a* (Σ,Π)-approximation *of r if $r \leq \tilde{r} \leq \Pi r + \Sigma$. Let $\varphi, \tilde{\varphi} : D \to \mathbb{R}^+$ be real-valued functions over continuous domain D. The function $\tilde{\varphi} : D \to \mathbb{R}^+$ is said to be a* (Σ,Π)-approximation *of φ if $\varphi(x) \leq \tilde{\varphi}(x) \leq \Pi\varphi(x) + \Sigma$ for all $x \in D$. Let $D = [A,B]$ be an interval on the real line. We call a function $\tilde{\varphi}$ a* succinct (Σ,Π)-approximation *of φ if it admits a representation in space polynomial in $\log \varphi^{\max} + \log(B-A) + \log(1/\Sigma) + 1/\epsilon$ and is a (Σ,Π)-approximation of φ. Such a function $\tilde{\varphi}$ is said to be* efficient *if it can be constructed in time polynomial in $\log \varphi^{\max} + \log(B-A) + \log(1/\Sigma) + 1/\epsilon$.*

(Σ,Π)-approximation generalizes additive and multiplicative approximations in the sense that $(\Sigma,1)$-approximation is additive Σ-approximation and $(0,\Pi)$-approximation is Π-multiplicative approximation. We will see that (Σ,Π)-approximations of Lipschitz-continuous (convex or monotone) functions can be computed efficiently. However, as the next proposition tells us, it is still too strong a measure of approximation when dealing with "ordinary" (as opposed to Lipschitz) continuous functions.

Proposition 2.9 *Let $1 > \Sigma > 0,\ \Pi > 1$ be arbitrary real numbers, and $\varphi : [0,1] \to [0,1]$ be a continuous monotone function. φ does not necessarily admit a succinct (Σ,Π)-approximation.*

The following proposition provides a set of general computational rules of (Σ,Π)-approximation functions. Its validity follows directly from the definition of (Σ,Π)-approximation functions. Computational rules of (Σ,Π)-approximations specialized to *convex* functions can be constructed as well, see e.g. Appendix A.2.

Proposition 2.10 (Calculus of (Σ,Π)-approximation Functions) *For $i = 1,2$ let $\Sigma_i \geq 0,\ \Pi_i \geq 1$, let $\varphi_i : D \to \mathbb{R}^+$ be an arbitrary function over continuous domain D, and let $\tilde{\varphi}_i : D \to \mathbb{R}^+$ be a (Σ_i,Π_i)-approximation of φ_i. Let $\psi_1 : D \to D$, and let $\alpha_i \in \mathbb{R}^+$. The following properties hold:*

1. *φ_1 is a (0,1)-approximation of itself,*
2. *(linearity of approximation) $\alpha_1\tilde{\varphi}_1 + \alpha_2$ is a $(\alpha_1\Sigma_1, \Pi_1)$-approximation of $\alpha_1\varphi_1 + \alpha_2$,*
3. *(summation of approximation) $\tilde{\varphi}_1 + \tilde{\varphi}_2$ is a $(\Sigma_1+\Sigma_2, \max\{\Pi_1,\Pi_2\})$-approximation of $\varphi_1 + \varphi_2$,*
4. *(composition of approximation) $\tilde{\varphi}_1(\psi_1)$ is a (Σ_1,Π_1)-approximation of $\varphi_1(\psi_1)$,*
5. *(minimization of approximation) $\min\{\tilde{\varphi}_1,\tilde{\varphi}_2\}$ is a $(\max\{\Sigma_1,\Sigma_2\}, \max\{\Pi_1,\Pi_2\})$-approximation of $\min\{\varphi_1,\varphi_2\}$,*
6. *(maximization of approximation) $\max\{\tilde{\varphi}_1,\tilde{\varphi}_2\}$ is a $(\max\{\Sigma_1,\Sigma_2\}, \max\{\Pi_1,\Pi_2\})$-approximation of $\max\{\varphi_1,\varphi_2\}$,*
7. *(approximation of approximation) If $\varphi_2 = \tilde{\varphi}_1$ then $\tilde{\varphi}_2$ is a $(\Sigma_2 + \Pi_2\Sigma_1, \Pi_1\Pi_2)$-approximation of φ_1.*

2.4 Computing (Σ,Π)-approximation functions via (Σ,Π)-approximation sets

The basic idea underlying succinct (Σ,Π)-approximation functions is to keep only a number of points in the domain that is logarithmic in $\frac{\varphi^{\max}}{\Sigma}$, in what is called a (Σ,Π)-approximation set, and then perform linear interpolation between these points to approximate the original function φ.

Definition 2.11 ((Σ,Π)-approximation sets) *Let $\Sigma \geq 0$ and $\Pi \geq 1$, let $\varphi : [A,B] \to \mathbb{R}^+$ be a convex function, and let $W \subseteq [A,B]$ be a finite set containing the endpoints A, B. Denote by $\hat{\varphi}$ the piecewise linear extension of φ induced by W. We say that W is a* (Σ,Π)-approximation set *of φ if $\hat{\varphi}$ is a (Σ,Π)-approximation function of φ.*

Algorithm 2 takes as input a convex or monotone increasing Lipschitz continuous function $\varphi : [A,B] \to \mathbb{R}^+$, and returns a point that is "sandwiched" between two given functions $\text{low}(\cdot), \text{high}(\cdot)$.

Proposition 2.12 *Given a κ-Lipschitz continuous function φ, a domain $[A,B]$ on which φ is either convex or monotone increasing, and functions low$(\cdot)$ and high$(\cdot)$ such that low$(\cdot) <$ high$(\cdot)$ and $\varphi(A) \leq$ low(A),* FUNCSEARCHINC *returns in $O((1+t_\varphi)(\log \frac{(B-A)\kappa}{\text{high}(B)-\text{low}(A)}))$ time a point x for which low$(x) \leq \varphi(x) \leq$ high(x), if such a point exists, and the point B otherwise.*

1: **Function FuncSearchInc**($\varphi, [A, B], \text{low}(\cdot), \text{high}(\cdot)$)
2: **if** $\varphi(B) < \text{high}(B)$ **then return** B
3: **while** $\varphi((A+B)/2) > \text{high}((A+B)/2)$ or $\varphi((A+B)/2) < \text{low}((A+B)/2)$ **do**
4: **if** $\varphi((A+B)/2) > \text{high}((A+B)/2)$ **then** $B \leftarrow (A+B)/2$ **else** $A \leftarrow (A+B)/2$
5: **end while**
6: **return** $(A+B)/2$

Algorithm 2: FUNCTION FUNCSEARCHINC($\varphi, [A, B], \text{low}(\cdot), \text{high}(\cdot), C$) returns a point $x \in [A, B]$ that satisfies $\text{low}(x) \le \varphi(x) \le \text{high}(x)$, if such a point exists, and point B otherwise.

1: **Function ApxSetConvInc**($\varphi, [A, B], \Sigma, \Pi$)
2: $x \leftarrow$ FUNCSEARCHINC$(\varphi, [A, B], \varphi(A) + \Sigma/3, \varphi(A) + 2\Sigma/3)$, $W \leftarrow \{A, x, B\}$
3: **while** $x < B$ **do**
4: $x \leftarrow$ FUNCSEARCHINC$(\varphi, [x, B], \frac{1}{2}(\Pi+1)(\varphi(x) + \sigma_\varphi(x)(\cdot - x)), \Pi(\varphi(x) + \sigma_\varphi(x)(\cdot - x)))$
5: $W \leftarrow W \cup \{x\}$
6: **end while**
7: **return** W

Algorithm 3: FUNCTION APXSETCONVINC($\varphi, [A, B], \Sigma, \Pi$)

1: **Function ApxSetConv**($\varphi, [A, B], \Sigma, \Pi$)
2: $\tilde{x} \leftarrow$ APXARGMIN$(\varphi, A, B, \Sigma/3)$
3: **return** APXSETCONVDEC$(\varphi, [A, \tilde{x}], \Sigma, \Pi) \cup$ APXSETCONVINC$(\varphi, [\tilde{x}, B], \Sigma, \Pi)$

Algorithm 4: FUNCTION APXSETCONV($\varphi, [A, B], \Sigma, \Pi$)

Algorithm 3 (called APXSETCONVINC) constructs a (Σ, Π)-approximation set for a convex function φ that decreases by at most $\Sigma/3$ before being monotone increasing. It is based on the idea of estimating the slope of φ around its minimum, and increase it by roughly a factor Π at every step.

We define function APXSETCONVDEC similarly for convex functions that increase by at most $\Sigma/3$ from their argmin, and define function APXSETCONV that constructs a (Σ, Π)-approximation sets for convex functions, see Algorithm 4.

Let $\Delta_{\bar{x}} := x - A$, where x is the value computed in step 2 of Algorithm 3. Let $\varphi : [A, B] \to \mathbb{R}$ be a convex function. If φ is monotone increasing, we define $\sigma_\varphi(x) := \frac{\varphi(x) - \varphi(x - \Delta_{\bar{x}})}{\Delta_{\bar{x}}}$ as a lower bound on the slope of φ at x for any $x \ge A + \Delta_{\bar{x}}$, and $\sigma_\varphi^{\max} := \sigma_\varphi(B)$ (we never use the term $\sigma_\varphi(x)$ if $x < A + \Delta_{\bar{x}}$). If φ is monotone decreasing, we define $\sigma_\varphi(x) = -\sigma_{-\varphi}(x)$ for any $x \ge A + \Delta_{\bar{x}}$ and $\sigma_\varphi^{\max} := -\sigma_{-\varphi}^{\max}$.

Proposition 2.13 (Approximation of a convex function with direct access) *Let $\varphi : [A, B] \to \mathbb{R}^+$ be a κ-Lipschitz continuous convex function. Then for every constants $\Sigma > 0$ and $\Pi = 1 + \epsilon > 1$ the following holds. (i)* APXSETCONV *constructs a (Σ, Π)-approximation set W for φ of cardinality $O(\log_\Pi \min\{\frac{\sigma_\varphi^{\max}}{\sigma_\varphi(A+\Delta_{\bar{x}})}, \frac{\varphi^{\max}}{\Sigma}\})$ in $O((1+t_\varphi)(\log_\Pi \min\{\frac{\sigma_\varphi^{\max}}{\sigma_\varphi(A+\Delta_{\bar{x}})}, \frac{\varphi^{\max}}{\Sigma}\}) \log((B-A)\kappa))$ time. (ii) The piecewise linear extension of φ induced by W, $\hat{\varphi}$, is a convex piecewise linear $(\kappa\Pi + \Sigma)$-Lipschitz (Σ, Π)-approximation of φ, whose value at any point in $[A, B]$ can be determined in $O(\log |W|)$ time if W is stored in a sorted array $(x, \varphi(x)), x \in W$.*

Sometimes, direct access to a convex function $\breve{\varphi} : [A, B] \to \mathbb{R}^+$ is time-consuming, therefore it is beneficial to construct a succinct approximation to that function. We define a function COMPRESSCONV for this purpose: COMPRESSCONV$(\breve{\varphi}, [A, B], \Sigma, \Pi)$ computes $W \leftarrow$ APXSETCONV$(\breve{\varphi}, [A, B], \Sigma, \Pi)$, and returns the piecewise linear extension of $\breve{\varphi}$ induced by W[1]. Applying Propositions 2.10 and 2.13 yields the following result.

Proposition 2.14 *Let $\Pi_1, \Pi_2 \ge 1$, $\Sigma_1, \Sigma_2 \ge 0$ be real numbers, and let $\varphi : [A, B]$ be a convex function over $[A, B]$. Let $\breve{\varphi}$ be a κ-Lipschitz continuous convex (Σ_2, Π_2)-approximation function of*

[1] We are grateful to Jim Orlin for suggesting this function, as well as the term "Compress".

φ. *Then Function* CompressConv*($\breve{\varphi}, [A, B], \Sigma_1, \Pi_1$) returns in* $O((1 + t_{\breve{\varphi}})(\log_{\Pi_1} \min\{\frac{\sigma^{\max}_{\breve{\varphi}}}{\sigma_{\breve{\varphi}}(A+\Delta_{\bar{x}})}, \frac{\breve{\varphi}^{\max}}{\Sigma_1}\}) \log((B-A)\kappa))$ *time an* $O(\log_{\Pi_1} \min\{\frac{\sigma^{\max}_{\breve{\varphi}}}{\sigma_{\breve{\varphi}}(A+\Delta_{\bar{x}})}, \frac{\breve{\varphi}^{\max}}{\Sigma_1}\})$ *piecewise linear* $(\kappa\Pi_1 + \Sigma_1)$*-Lipschitz continuous convex* $(\Sigma_2 + \Pi_2\Sigma_1, \Pi_1\Pi_2)$*-approximation of* φ.

3 Approximation scheme for implicit stochastic DPs

To derive a (Σ, Π)-FPTAS for a DP as described in Section 1, the following conditions are needed:

Condition 1 *$\mathcal{S}_{T+1}, \mathcal{S}_t, \mathcal{A}_t(I_t)$ are intervals on the real line, for $I_t \in \mathcal{S}_t$ and $t = 1, \ldots, T$. For each $t = 1, \ldots, T$, D_t is a discrete random variable with a finite support $\mathcal{D}_t$ which is contained in an interval on the real line. The kth largest element in $\mathcal{D}_t$ can be identified in constant time for any $1 \leq k \leq |\mathcal{D}_t|$.*

Condition 2 *For every $t = 1, \ldots, T + 1$, the functions f_t, g_t and the CDF of D_t are either given explicitly (e.g., as explicit formulae) or accessed via value oracles. Moreover, f_t and g_t are Lipschitz continuous and g_t is nonnegative.*

Condition 3 *The function g_{T+1} is convex over $\mathcal{S}_{T+1}$. For $t = 1, \ldots, T$, the set $\mathcal{S}_t \otimes \mathcal{A}_t$ is convex. The function g_t can be expressed as $g_t(I, x, d) = g^I_t(f'_t(I, d)) + g^x_t(f''_t(x, d)) + g^u_t(f_t(I, x, d))$, where $g^I_t(\cdot), g^x_t(\cdot), g^u_t(\cdot)$ are univariate Lipschitz continuous convex functions and f_t, f'_t, f''_t are linear and separable in their variables.*

The input data of the problem includes the number of time periods T, the initial state I_1, the constant $\gamma = \min\{\text{Prob}(D^{\max}_t), \text{Prob}(D^{\min}_t) \mid t = 1, \ldots, T\}$, the maximum cost in a time period U_g, the maximum lengths U_S and U_A of the state and action spaces, and the sizes n_t of the supports of the random variables. The input size is therefore $\log I_1 + \log 1/\gamma + \log U_g + \log U_S + \log U_A + T \log(\max_t n_t)$. We call a DP formulation (2) *convex* whenever it satisfies Condition 3; we call it *implicit* if $D_t, t = 1, \ldots, T$ are given via a value oracle to the CDF.

We briefly highlight the main differences between the implicit stochastic continuous DP model discussed in this paper and the explicit stochastic discrete DP model of [HKL$^+$14, HNO15]. In our present model of computation, the computer is capable of storing countably many reals and performing real arithmetic operations, therefore we do not require that the function values or the support points be rational numbers. On the other hand, due to Proposition 2.9 we require that the functions f_t and g_t be Lipschitz continuous. A further difference is that in the present paper we include the maximum length of the state and action spaces as part of the input data of the problem; in the discrete model this is not necessary, because [HKL$^+$14, Condition 1] implies that these values are polynomially bounded by the input size. Considering Condition 3, our present continuous model relaxes the assumption that the coefficients of x in the transition functions are in $\{-1, 0, 1\}$. This requirement was necessary in order to preserve discrete convexity of the cost-to-go functions, see [HKL$^+$14, Thm. 9.2]. For "ordinary" convexity, it is sufficient to require that the transition functions are linear functions that are separable in their variables. We require that g^I_t, g^x_t are univariate so that we can apply Proposition 3.2.

Our main result (Theorem 1.3) is that every stochastic DP satisfying Conditions 1, 2 and 3 admits a (Σ, Π)-FPTAS. Moreover, the approximation scheme succinctly (Σ, Π)-approximates z_1 and returns an approximate policy. [HKL$^+$14] presented seven stochastic problems that fit into their FPTAS framework. The distribution of the random variables in these problems was assumed to be given explicitly. Six of these problems (single-item inventory control, single-item batch dispatch, single-resource revenue management, growth models, lifetime consumption of risky capital and cash management) seem to be more naturally cast in a framework with continuous state and action spaces. It is easy to verify that these problems, as described in [HKL$^+$14], under the less restrictive assumption of implicitly-defined random variables, satisfy Conditions 1, 2 and 3, and therefore admit a (Σ, Π)-FPTAS as a consequence of Theorem 1.3.

We prove Theorem 1.3 by efficiently constructing succinct (Σ, Π)-approximations of the cost-to-go functions z_t by backward induction. Proposition 2.3 tells us that it is not possible in general to build a succinct additive Σ-approximation to z_t. Corollary 2.7 tells us that it is not possible in general to

build a succinct multiplicative Π-approximation for z_t. This implies that the proof approach of Theorem 1.3 cannot yield additive or multiplicative approximations to stochastic continuous DPs. In fact, such approximations cannot be obtained in general, as is stated below.

Theorem 3.1 *An implicit stochastic continuous convex DP satisfying Conditions 1, 2 and 3 does not necessarily admit an additive or a multiplicative approximation, regardless of the scheme used to build the approximation.*

We note that the proof of Theorem 3.1 also implies Theorem 1.2.

We now discuss how to deal with implicitly described random variables in the context of (Σ, Π)-approximation sets. [HOSL12] uses the notion of K-approximation sets and functions in order to approximate CDFs as well as expectations of monotone functions via CDFs. [HOSL12, Prop. 2] shows how to approximate $E_D(\xi(f(x, D)))$ if ξ is nonnegative increasing and $\text{Prob}(f(x, D) \geq a_i)$ is monotone in x. We generalize this result to piecewise linear convex functions and arbitrary discrete random variables in the following way.

Proposition 3.2 *Let $\xi : [A, B] \to \mathbb{R}^+$ be a (not necessarily monotone) convex function. Let $\Sigma_1 \geq 0$ and $\Pi_1, \Pi_2 \geq 1$. Let $\psi : [A, B] \to \mathbb{R}^+$ be an increasing piecewise linear convex function with breakpoints $A = a_1 < \ldots < a_n < B$ and slopes $0 = \Delta_0 \leq \Delta_1 < \ldots < \Delta_n$ that (Σ_1, Π_1)-approximates ξ. Let D be a (not necessarily nonnegative) discrete random variable. Let $\tilde{F}(\cdot)$ be a monotone increasing step multiplicative Π_2-approximation of the CDF of D with breakpoints at $d_1 < \ldots < d_m$ (see [HKL+14, Sec. 4.1] for an algorithm to compute $\tilde{F}$) and let $\tilde{F}(d_0) = 0$. Let $f(x, d) = bx + e - d$ for some given b, e, and $m_i = \arg\max_j \{d_j \mid d_j \leq bx + e - a_i\}$. Then*

$$\tilde{\xi}(x) = \psi(A) + \sum_{i=1}^{n} (\Delta_i - \Delta_{i-1}) \left((bx + e - d_{m_i} - a_i)\tilde{F}(d_{m_i}) + \sum_{k=1}^{m_i - 1} (d_{k+1} - d_k)\tilde{F}(d_k) \right)$$

is a convex piecewise linear $(\Pi_2\Sigma_1, \Pi_1\Pi_2)$-approximation of $E_D(\xi(f(x, D)))$ with $O(nm)$ pieces that can be computed in $O((n + m)(t_\psi + t_{\tilde{F}}))$ time for each $x \in [A, B]$.

The proof of the above proposition relies on the following two facts. First, by the calculus of (Σ, Π)-approximation functions, $E_D(\psi(f(x, D))) = \sum_{j=1}^{m} \psi(f(x, d))\text{Prob}(D = d_j)$ is a (Σ_1, Π_1)-approximation of $E_D(\xi(f(x, D)))$. Second, it exploits a representation of $\psi(\cdot)$ as $\psi(x) = \psi(A) + \sum_{i=1}^{n} \psi_i(x)$, where $\psi_i(x) = (\Delta_i - \Delta_{i-1})(x - a_i)$ if $x \geq a_i$, $\psi_i(x) = 0$ if $x < a_i$, and $\Delta_0 = 0$.

The above proposition can also be applied to separable linear transition functions $f(x, d) = bx + e + cd$ with the coefficient c being other than -1. This can be achieved using a transformed random variable D' with $D' := D/(-c)$. We also note that we can get a similar result for a convex function that is approximated by a *decreasing* piecewise linear convex function. We therefore get the following corollary.

Corollary 3.3 *Let $\xi : [A, B] \to \mathbb{R}^+$ be a (not necessarily monotone) convex function. Let $\Sigma_1 \geq 0$, $\Pi_1, \Pi_2 \geq 1$, and let $C \in \mathbb{R}$ with $A \leq C \leq B$. Let $\psi^- : [A, C] \to \mathbb{R}^+$ be a decreasing piecewise linear convex function with n^- breakpoints that (Σ_1, Π_1)-approximates ξ over $[A, C]$. Let $\psi^+ : [C, B] \to \mathbb{R}^+$ be an increasing piecewise linear convex function with n^+ breakpoints that (Σ_1, Π_1)-approximates ξ over $[C, B]$. Let D be a (not necessarily nonnegative) discrete random variable. Let $\tilde{F}(\cdot)$ be a monotone increasing step multiplicative Π_2-approximation of the CDF of D with m breakpoints. Last, let $f(\cdot, \cdot)$ be a function separable and linear in its variables. Applying Proposition 3.2 twice, we can compute a convex piecewise linear value oracle with $O((n^- + n^+)m)$ pieces that is a $(\Pi_2\Sigma_1, \Pi_1\Pi_2)$-approximation of $E_D(\xi(f(x, D)))$. Each of its values can be computed in $O((n^- + n^+ + m)(t_{\psi^-} + t_{\psi^+} + t_{\tilde{F}}))$ time .*

The goal of the remainder of this section is to present an approximation scheme (called SCHEME-CONVDP) to compute a succinct $(\delta, 1 + \epsilon)$-approximation for the value function $z_1(\cdot)$ of a continuous convex DP, thereby proving Theorem 1.3. To obtain the values of the cost-to-go function $\tilde{z}_t(\cdot)$, we will use Proposition 3.4 below.

1: **Procedure SchemeConvDP**(δ, ϵ)
2: $\Sigma \leftarrow \frac{\delta}{(6T+1)\sqrt[3]{1+\epsilon}}$, $\Pi \leftarrow \sqrt[3T+1]{1+\epsilon}$, $\Pi' \leftarrow \Pi$, $\Pi'' \leftarrow \sqrt[3]{\Pi}$, $W_{T+1} \leftarrow$ **ApxSetConv**$(g_{T+1}, A, B, \Pi''\Sigma, \Pi)$
3: Let $\breve{z}_{T+1}$ be the piecewise linear extension of g_{T+1} induced by W_{T+1}
4: **for** $t := T$ **downto** 1 **do**
5: Let $\tilde{F}_t$ be a succinct monotone increasing step multiplicative Π-approximation of the CDF of D_t
6: For $\diamond \in \{I, x, u\}$: $\breve{g}_t^\diamond \leftarrow$COMPRESSCONV$(g_t^\diamond, \mathcal{S}_t, \Pi''\Sigma/2, \Pi')$
7: For $\diamond \in \{I, x, u\}$: Let $\bar{G}_t^\diamond(\cdot)$ be a convex $(\Pi''\Pi\Sigma/2, \Pi'\Pi)$-approximation for $G_t^\diamond(\cdot) := E_{D_t}(g_t^\diamond(f'(\cdot, D_t)))$
8: Let $\bar{Z}f_{t+1}(\cdot)$ be a convex $((6(T-t)+1)\Pi''\Pi\Sigma, \Pi'\Pi)$-approximation for $Zf_{t+1}(\cdot) := E_{D_t}(z_{t+1}(\cdot + c_t D_t))$
9: For $\diamond \in \{I, x, u\}$: $\breve{G}_t^\diamond \leftarrow$COMPRESSCONV$(\bar{G}_t^\diamond, \mathcal{S}_t, \Pi''\Sigma/(2\Pi'), \Pi)$
10: $\breve{Z}f_{t+1} \leftarrow$COMPRESSCONV$(\bar{Z}f_{t+1}, \mathcal{S}_t, \Pi''\Sigma/\Pi', \Pi)$
11: $\breve{z}_t \leftarrow$COMPRESSCONV$(\bar{z}_t, \mathcal{S}_t, \Pi''\Sigma/(\Pi'\Pi), \Pi)$ /* $\bar{z}_t$ *is as defined in* (3) */
12: $\Pi' \leftarrow \Pi'\Pi^3$, $\Pi'' \leftarrow \Pi''\Pi$
13: **end for**

Algorithm 5: $(\delta, 1+\epsilon)$-FPTAS for a continuous convex DP with $f_t(I_t, x_t, D_t) = a_t I_t + b_t x_t + c_t D_t$.

Proposition 3.4 *Suppose the DP formulation* (2) *satisfies Condition 3 with* $f_t(I_t, x_t, D_t) = aI_t + bx_t + cD_t$. *Let* $\Sigma_I, \Sigma_x, \Sigma_u, \Sigma_z > 0$; $\Pi_I, \Pi_x, \Pi_u, \Pi_z > 1$, t *and* I_t *be fixed values, where* $\Pi_z \geq \Pi_I, \Pi_x, \Pi_u$, $I_t \in \mathcal{S}_t$ *and* $t \in [1, \ldots, T]$. *Let* $\breve{G}_t^I(\cdot)$ *be a convex* (Σ_I, Π_I)*-approximation of* $G_t^I(\cdot) := E_{D_t}(g_t^I(f_t'(\cdot, D_t)))$. *Let* $\breve{G}_t^x(\cdot)$ *be a convex* (Σ_x, Π_x)*-approximation of* $G_t^x(\cdot) := E_{D_t}(g_t^x(f_t''(\cdot, D_t)))$. *Let* $\breve{G}_t^u(y)$ *be a convex* (Σ_u, Π_u)*-approximation of* $G_t^u(y) := E_{D_t}(g_t^u(y + cD_t))$. *Let* $\breve{Z}f_{t+1}(y)$ *be a convex* (Σ_z, Π_z)*-approximation of* $Zf_{t+1}(y) := E_{D_t}(z_{t+1}(y + cD_t))$. *Let* κ *be the largest Lipschitz constant among the functions* $G_t^I(\cdot), G_t^x(\cdot), G_t^u(\cdot), Zf_{t+1}(\cdot)$. *Let*

$$\bar{z}_t(I_t) = \breve{G}_t^I(I_t) + \min_{x_t \in \mathcal{A}_t(I_t)} \left\{ \breve{G}_t^x(x_t) + \breve{G}_t^u(aI_t + bx_t) + \breve{Z}f_{t+1}(aI_t + bx_t) \right\}. \tag{3}$$

Then, for every given $\Sigma > 0$, $\bar{z}_t(\cdot)$ *is a convex* $(\Sigma + \Sigma_I + \Sigma_x + \Sigma_u + \Sigma_z, \Pi_z)$*-approximation of* $z_t(\cdot)$. *Moreover, each value* $\bar{z}_t(I_t)$ *can be determined in* $O(t_{\breve{G}_t^I} + (t_{\breve{G}_t^x} + t_{\breve{G}_t^u} + t_{\breve{Z}f_{t+1}}) \log(b\kappa(\max \mathcal{A}(I_t) - \min \mathcal{A}(I_t))/\Sigma))$ *time.*

Before stating our concluding result, we briefly explain the idea behind and the main steps of Algorithm 5 (called SCHEMECONVDP). The algorithm basically follows the Bellman optimality equation, proceeding backwards from stage T to stage 1. In stage t the algorithm first constructs a succinct approximation $\bar{F}_t$ to the CDF of the random event D_t. (This is needed because in our model the random event D_t is given implicitly, so computing expectations exactly may require nonpolynomial time in general.) The algorithm then produces an approximated value oracle $\bar{z}_t$ to the cost-to-go function z_t via Proposition 3.4. Last, using this value oracle it produces a succinct representation $\breve{z}_t$ of $\bar{z}_t$ via Function COMPRESSCONV. The error parameters are chosen such that the total accumulated error is bounded by $(\delta, 1+\epsilon)$ (via the calculus of (Σ, Π)-approximation functions). We give now a more specific explanation to the main steps. Step 5 is realized by calculating a multiplicative Π-approximation set W_t' for the CDF F_t of D_t, e.g., by function APXSET in [HKL$^+$14, Sec. 4.1], and then setting $\tilde{F}$ to be the monotone extension of F induced by W_t' (see Definition A.1). In steps 7–8 we construct convex approximations to $G_t^I(\cdot), G_t^x(\cdot), G_t^u(\cdot), Zf_{t+1}(\cdot)$ via Corollary 3.3. Step 11 gives a succinct approximation $\breve{z}_t$ of the cost-to-go function at stage t. In each iteration of the algorithm the multiplicative error of $\breve{z}$ increases by Π^3 (one factor of Π is added in step 8, one in step 10, and one in step 11).

Theorem 3.5 (Approximation scheme for continuous convex DPs) *Consider a DP that satisfies Conditions 1, 2, and 3, and any pair of reals* $0 < \epsilon < 1$ *and* $\delta > 0$. *Then the function* $\breve{z}_1$ *generated at step 11 of the last iteration of Algorithm 5 is a convex* $(\delta, 1+\epsilon)$*-approximation of the optimal cost* z^* *as in* (1). *Moreover, Algorithm 5 runs in time polynomial in* $\frac{1}{\epsilon} + \log \frac{1}{\delta}$ *and the (binary) input size.*

References

[AC03] Jerome Adda and Russel W. Cooper. *Dynamic Economics: Quantitative Methods and Applications*. The MIT Press, Boston, MA, 2003.

[BD62] Richard Bellman and Stuart Dreyfus. *Applied Dynamic Programming*. Princeton University Press, Princeton, NJ, 1962.

[Ber05] Dimitri Bertsekas. *Dynamic Programming and Optimal Control, Volume I*. Athena Scientific, Belmont, MA, 2005.

[BN13] Aharon Ben-Tal and Arkadi Nemirovski. Lectures on convex optimization, 2013.

[dFVR03] Daniela Pucci de Farias and Benjamin Van Roy. The Linear Programming approach to approximate Dynamic Programming. *Operations Research*, 51(6):850–865, 2003.

[DM02] Elizabeth D. Dolan and Jorge J. Moré. Benchmarking optimization software with performance profiles. *Mathematical Programming*, 91(2):201–213, 2002.

[EP04] Michael Elkin and David Peleg. $(1+\epsilon, \beta)$-spanner constructions for general graphs. *SIAM Journal on Computing*, 33(3):608–631, 2004.

[GP01] Gregory A. Godfrey and Warren B. Powell. An adaptive, distribution-free algorithm for the newsvendor problem with censored demands, with applications to inventory and distribution. *Management Science*, 47(8):1101–1112, 2001.

[HKL+14] Nir Halman, Diego Klabjan, Chung-Lun Li, Jim Orlin, and David Simchi-Levi. Fully polynomial time approximation schemes for stochastic dynamic programs. *SIAM Journal on Discrete Mathematics*, 28(4):1725–1796, 2014.

[HNO15] Nir Halman, Giacomo Nannicini, and James Orlin. A computationally efficient FPTAS for convex stochastic dynamic programs. *SIAM Journal on Optimization*, 25(1):317–350, 2015.

[Hoc97] Dorit Hochbaum. Various notions of approximations: Good, better, best, and more. In D. Hochbaum, editor, *Approximation Algorithms for NP-hard Problems*. PWS Publishing Company, Boston, MA, 1997.

[HOSL12] Nir Halman, James B. Orlin, and David Simchi-Levi. Approximating the nonlinear newsvendor and single-item stochastic lot-sizing problems when data is given by an oracle. *Operations Research*, 60(2):429–446, 2012.

[MS13] Shashi Mittal and Andreas S. Schultz. A general framework for designing approximation schemes for combinatorial optimization problems with many objectives combined into one. *Operations Research*, 61(2):386–397, 2013.

[NP10] Juliana M. Nascimento and Warren B. Powell. Dynamic programming models and algorithms for the mutual fund cash balance problem. *Management Science*, 56(5):801–815, 2010.

[NP13] Juliana M. Nascimento and Warren B. Powell. An optimal approximate dynamic programming algorithm for concave, scalar storage problems with vector-valued controls. *IEEE Transactions on Automatic Control*, 58(12):2995–3010, 2013.

[Phe62] Edmund S. Phelps. The accumulation of risky capital: A sequential utility analysis. *Econometrica*, 30:729–743, 1962.

[Pow11] Warren B. Powell. *Approximate Dynamic Programming: Solving the Curses of Dimensionality, 2nd Edition*. Wiley, 2011.

[PRT04] Warren B. Powell, Andrzej Ruszczyński, and Huseyin Topaloglu. Learning algorithms for separable approximations of discrete stochastic optimization problems. *Mathematics of Operations Research*, 29(4):814–836, 2004.

[PW07] Kirk Pruhs and Gerhard J. Woeginger. Approximation schemes for a class of subset selection problems. *Theoretical Computer Science*, 382(2):151–156, 2007.

[SS06] David B. Shmoys and Chaitanya Swamy. An approximation scheme for stochastic linear programming and its application to stochastic integer programs. *Journal of the ACM*, 53(6):978–1012, 2006.

[Vaz01] Vijay Vazirani. *Approximation Algorithms*. Springer, Berlin, 2001.

[Woe00] Gerhard J. Woeginger. When does a dynamic programming formulation guarantee the existence of a fully polynomial time approximation scheme (FPTAS)? *INFORMS Journal on Computing*, 12(1):57–74, 2000.

A Proofs

Given $x \in [D^{\min}, D^{\max}]$, for $x < D^{\max}$ let $\text{next}(x, D) := \min\{y \in D \mid y > x\}$, and for $x > D^{\min}$ let $\text{prev}(x, D) := \max\{y \in D \mid y < x\}$. In our analysis, the following notation will be used.

$D^{\min} = \min\{d \mid \text{Prob}(D = d) > 0\}$: smallest element in the support of D;

$D^{\max} = \max\{d \mid \text{Prob}(D = d) > 0\}$: greatest element in the support of D;

$n_t = |\mathcal{D}_t|$: maximum number of different values that D_t can take;

$n^* = \max_t n_t$: maximum number of different values that D_t can take over the entire time horizon;

$U_{\mathcal{S}} = \max_{t=1,\dots,T+1}(\max \mathcal{S}_t - \min \mathcal{S}_t)$: maximum length of the state space;

$U_{\mathcal{A}} = \max_t \max_{I_t \in \mathcal{S}_t}(\max \mathcal{A}_t(I_t) - \min \mathcal{A}_t(I_t))$: maximum length of the action space;

$g_t^{\max} = \max_{I \in \mathcal{S}_t,\, x \in \mathcal{A}_t(I),\, d \in \mathcal{D}_t} g_t(I, x, d)$: maximum cost value in time period t, $t = 1, \dots, T$;

$g_{T+1}^{\max} = \max_{I \in \mathcal{S}_{T+1}} g_{T+1}(I)$

$U_g = \max_{t=1,\dots,T+1} g_t^{\max}$: maximum cost value in a time period;

$\gamma = \min\{\text{Prob}(D_t^{\max}), \text{Prob}(D_t^{\min}) \mid t = 1, \dots, T\}$.

Definition A.1 (Monotone extensions) *Let $\varphi : [A, B] \to \mathbb{R}^+$ be a real-valued function. Let $W \subseteq [A, B]$ be a finite set that contains A, B. The* monotone extension of φ induced by W *is:*

$$\hat{\varphi}(x) = \begin{cases} \varphi(x) & \text{if } x \in W; \\ \max\{\varphi(\textit{prev}(x, W)), \varphi(\textit{next}(x, W))\} & \text{otherwise.} \end{cases}$$

A.1 Proposition 2.2

Proof. Due to the convexity of φ, a minimum of φ is located in the interval $[A', B']$, where C' is its middle point.

Due to the convexity of φ, its minimum lies either (i) not below the intersection of the half plane $x \geq A'$ and the half plane above the line l_{CB}, or (ii) not below the intersection of the half plane $x \leq B'$ and the half plane above the line l_{AC}. After step 12, the value of κ' is an upper bound to the Lipschitz constant on $[A', B']$ because of convexity. Therefore, the maximum difference between the true minimum and $\varphi(D')$ is at most $\kappa'(B' - A')/2$. If this term is less than Σ, the algorithm exits the while loop and reports D'.

The number of queries done throughout the execution of the algorithm derives from the fact that in each iteration the length of the interval is at most half of the length of its predecessor. □

A.2 Proposition 2.3

Proof. Let $\varphi^* : [0, A] \to \mathbb{R}^+$ be an A-piecewise linear convex function with breakpoints in every integer number in $[0, A]$, with $\varphi^*(0) = 0, \varphi^*(1) = 4\Sigma$ and such that in every piece the slope of φ^* increases by 4Σ. Hence, $\varphi^{*\max} = \varphi^*(A) = 2A(A + 1)\Sigma$, φ^* is κ-Lipschitz continuous, and has an exact representation in $A + 1$ space. For every $i = 1, \dots, A - 1$, let φ_i be an $(A - 1)$-piecewise linear convex function that coincides with φ^* everywhere excepts for the interval $(i - 1, i + 1)$, in where the function is linear increasing from $\varphi^*(i - 1)$ to $\varphi^*(i + 1)$, so it has an exact representation in A space. Note that the maximum (additive) difference between φ_i and φ^* is in i and is 2Σ. Let $\Phi = \{\varphi^*, \varphi_1, \dots, \varphi_{A-1}\}$. Thus, Φ is a family of convex monotone functions, in which each pair φ^*, φ_i has a difference of at least 2Σ in exactly one integer point in $[0, A]$, i.e., in i. This implies that every additive Σ-approximation

of φ^* and of φ_i must distinguish between φ^* and φ_i. But φ^* and φ_i differ in an interval of length 2. Therefore, if the space of each approximated representation of $\varphi \in \Phi$ is bounded by $A/2-1$, then there exists at least one interval of length at least 2 in which there is no information on the exact values of φ^*, and therefore it is not possible to distinguish between φ^* and some other function in Φ. □

A.3 Proposition 2.5

Proof. Consider the family Φ consisting of all nonnegative convex functions $\varphi : [-A, A] \to [0, A/2]$ that are 1-Lipschitz continuous with $\varphi(-A) = \varphi(A) = A/2$. We develop a resisting (adversarial) oracle that imitates a function φ in Φ to which it is hard to decide whether it is strictly positive. The oracle starts from the entire set Φ and tries to answer each call of the approximation algorithm in a somewhat "worst" possible way. Let $\Phi' \subseteq \Phi$ be the set of functions compatible with all answers of the oracle so far. The oracle answers queries so that the property that Φ' contains at least one function that attains value zero, as well as at least one strictly positive function remains invariant throughout the entire process of answering queries.

In order to produce "bad" answers, the oracle keeps record of (i) the set Q consisting of all query points $(x, \varphi(x))$ it answered so far (we let Q_X denote the projection of the set Q on the X-axis), (ii) the largest interval $[B, C]$ on which the value of at least one function in Φ' is always zero, (iii) the rightmost query point L to the left of B, i.e. $L = \max\{x \leq B \mid x \in Q_X\}$, and (iv) the leftmost query point R to the right of C, i.e., $R = \min\{x \geq C \mid x \in Q_X\}$. At the beginning of the process Q is set to $\{(-A, A/2);\ (A, A/2)\}$. Because φ is 1-Lipschitz continuous, we get that $[B, C] = [-A/2, A/2]$. Therefore, $L = -A$ and $R = A$. We denote by $\varphi_D : [-A, L] \to [0, A/2]$ the piecewise linear function whose breakpoints are $\{(x, y) \in Q \mid x \leq L\}$. By the construction of Q as explained below, it is easy to see that φ_D is monotone decreasing. Similarly, we denote by $\varphi_I : [R, A] \to [0, A/2]$ the piecewise linear function whose breakpoints are $\{(x, y) \in Q \mid x \geq R\}$. By the construction of Q as explained below, it is easy to see that φ_I is monotone increasing.

In subsequent stages, when the oracle gets a query point $x \leq L$, it returns the value $\varphi_D(x)$, so the set Φ' does not change. If the oracle gets a query point $x \geq R$, it returns the value $\varphi_I(x)$, so again the set Φ' does not change. Otherwise, suppose w.l.o.g that $x \leq (B + C)/2$. The oracle returns then the value v of the linear function between $(L, \varphi(L))$ and $(2(B + C)/3, 0)$ on x. The following "bookkeeping" is done. (i) $Q \leftarrow Q \cup \{(x, v)\}$, (ii) φ_D is updated by adding to it the linear piece between $(L, \varphi(L))$ and $(x, \varphi(x))$, (iii) $B \leftarrow 2(B + C)/3$, (iv) $L \leftarrow x$, (v) Φ' is updated to consist of all nonnegative convex functions $\varphi : [-A, A] \to [0, A/2]$ that are 1-Lipschitz continuous and compatible with the query points in Q. In this way, the size of the updated interval $[B, C]$ is the third of the size of the old interval. Note that φ_D concatenated with the linear function between $(L, \varphi(L))$ and $(R, \varphi(R))$, and with φ_I is a strictly positive convex function that belongs to Φ', while φ_D concatenated with the linear function between $(L, \varphi(L))$ and $(B, 0)$, and with the zero function over $[B, C]$, and with φ_I is a convex function that attains the value 0 and also belongs to Φ'. Therefore, the invariant that Φ' contains at least one function that attains value zero, as well as at least one strictly positive function remains valid. Consequently, it is not possible to find, at any step of the process, a single Π-approximation value to all the minimum values of the functions in Φ' for any value of Π, and the process must continue forever. □

A.4 Proposition 2.9

Proof. Consider the family $\Phi = \{\varphi_n \mid n = 1, 2, \ldots\}$, where

$$\varphi_n(x) = \begin{cases} 0 & \text{if } x \leq \frac{1}{2^n} \\ (x - \frac{1}{2^n})2^n & \text{if } \frac{1}{2^n} < x < \frac{1}{2^{n-1}} \\ 1 & \text{otherwise.} \end{cases}$$

Suppose by negation that there is an algorithm that builds in finite time a (Σ, Π)-approximation function $\tilde{\varphi}$ for any function in Φ, and that x^* is the smallest query point greater than zero generated by the algorithm. Since this algorithm is finite, x^* exists and is well defined. We define an adversarial oracle that returns $\varphi(x) = 1$ for all $x \in (0, 1]$ and $\varphi(0) = 0$. Let $n^* = 1 + \lceil \log(1/x^*) \rceil$. Note that $\varphi_{n^*}(x) = \varphi_{n^*+10}(x) = 1$ for all $x \geq x^*$ so $\tilde{\varphi}$ cannot distinguish between these two functions, but $\varphi_{n^*}(x^*/4) = 0$ and $\varphi_{n^*+10}(x^*/4) = 1$. The fact that $\Sigma < 1$ implies that there does not exist a single value v such that $\tilde{\varphi}(x^*/4) = v$ and v is a (Σ, Π)-approximation for both 0 and 1. This contradiction arises from our assumption that it is possible to build $\tilde{\varphi}$ in finite time. □

A.5 Proposition A.2

Proposition A.2 serves as a computational rule specialized to (Σ, Π)-approximations of *convex* functions.

Proposition A.2 (Approximation of summation of composition of convex functions) *Let $\Sigma_i \geq 0,\ \Pi_i \geq 1$ for $i = 1, \ldots, n$. Let D be a compact interval and let $C(x)$ be a compact interval for every $x \in D$. Let $\varphi_i : D \to \mathbb{R}^+$ be convex over D, let $\breve{\varphi}_i$ be a convex (Σ_i, Π_i)-approximation of φ_i, and let $\psi_i : D \otimes C \to D$ be an affine function for $i = 1, \ldots, n$. Define functions $\varphi, \bar{\varphi} : D \to \mathbb{R}^+$ as $\varphi(x) = \min_{y \in C(x)} \{\sum_{i=1}^n \varphi_i(\psi_i(x, y))\}$ and $\bar{\varphi}(x) = \min_{y \in C(x)} \{\sum_{i=1}^n \breve{\varphi}_i(\psi_i(x, y))\}$. Then, $\bar{\varphi}$ is a convex $(\Sigma_1 + \ldots + \Sigma_n, \max\{\Pi_1, \ldots, \Pi_n\})$-approximation of φ. Moreover, if $\breve{\varphi}_i$ are all κ-Lipschitz continuous functions and $[A(x), B(x)] := C(x)$ then for every $\Sigma > 0$ and $x \in D$ it is possible to find a $(\Sigma + \Sigma_1 + \ldots + \Sigma_n, \max\{\Pi_1, \ldots, \Pi_n\})$-approximation of $\varphi(x)$ in $O(\log(n\kappa(B(x) - A(x))/\Sigma))$ queries to each $\breve{\varphi}_i$.*

The proof exploits the following well-known result.

Proposition A.3 *Let $a, b, c : A \to \mathbb{R}$ be functions over a convex set A, let b, c be convex over A, and let $f(\cdot, \cdot)$ be an affine function over its variables. For all $y \in A$ let $a(y) = \min_{x \in A}\{b(x) + c(f(x, y))\}$. Then a is a convex function over A.*

Proof. Let us fix $x \in D$. By summation of approximation and composition of approximation (i.e., properties 3 and 4 of Proposition 2.10), $\sum_{i=1}^n \breve{\varphi}_i(\psi_i)(x, \cdot)$ is a $(\Sigma_1 + \ldots + \Sigma_2, \max\{\Pi_1, \ldots, \Pi_n\})$-approximation function of $\sum_{i=1}^n \varphi_i(\psi_i)(x, \cdot)$. Moreover, since $\breve{\varphi}_i$ are convex and ψ_i are affine, $\sum_{i=1}^n \breve{\varphi}_i(\psi_i)(x, \cdot)$ is a convex function. By Proposition 2.2 it is possible to find an additive Σ-approximation, i.e., $(\Sigma, 1)$-approximation, $\tilde{v}(x)$ of $\tilde{\varphi}(x)$ in the claimed number of queries. Therefore, $\tilde{v}(x)$ is a $(\Sigma + \Sigma_1 + \ldots + \Sigma_n, \max\{\Pi_1, \ldots, \Pi_n\})$-approximation of $\varphi(x)$ due to approximation of approximation (i.e., property 7 of Proposition 2.10 applied with parameters set to $n = 2,\ \varphi_1 = \tilde{\varphi},\ \varphi_2 = \tilde{v}, (\Sigma_1, \Pi_1) = (\Sigma_1 + \ldots + \Sigma_2, \max\{\Pi_1, \ldots, \Pi_n\});\ (\Sigma_2, \Pi_2) = (\Sigma, 1)$). The convexity of $\bar{\varphi}(\cdot)$ is due to Proposition A.3. □

A.6 Proposition 2.12

Proof. Suppose first that the condition on line 2 is satisfied. If $\varphi(B) \geq \text{low}(B)$, we found a point satisfying the desired condition and we are done. Otherwise, $\varphi(A), \varphi(B) \leq \text{low}(B)$, so either the convexity or the monotonicity of φ implies that no point x satisfying $\text{low}(x) \leq \varphi(x) \leq \text{high}(x)$ exists, and we are done returning B as the failure condition.

If the condition on line 2 is not satisfied, the algorithm keeps the inequalities $\varphi(B) > \text{high}(B)$ and $\varphi(A) < \text{low}(A)$ invariant throughout the execution of the while loop. Because φ is κ-Lipschitz continuous, the minimum size of an interval where such invariants hold is $\frac{\text{high}(B) - \text{low}(A)}{\kappa}$. Therefore, running time is $O(\log \frac{(B-A)\kappa}{\text{high}(B) - \text{low}(A)})$. Note that φ need not be monotone. □

A.7 Proposition 2.13

Proof. By Proposition 2.2, $\varphi(\tilde{x})$ is an additive $\Sigma/3$-approximation of $\varphi^{\min}$. Therefore, $\varphi : [A, \tilde{x}]$ increases by at most $\Sigma/3$ after its true argmin over $[A, \tilde{x}]$, and $\varphi : [\tilde{x}, B]$ decreases by at most $\Sigma/3$ between $\tilde{x}$ and its true argmin. Therefore, the calls to APXSETCONVINC and APXSETCONVDEC are well defined. We next prove (i) and (ii) for the restriction of φ over $[\tilde{x}, B]$. The proof for the restriction of φ over $[A, \tilde{x}]$ is similar. Considering APXSETCONVINC, after the first call to FUNCSEARCHINC (step 2), φ is assured to be monotone increasing and convex over $[x, B]$. Moreover, $\min\{\varphi(\tilde{x}), \varphi(x)\} - \min\{\varphi(y) \mid \tilde{x} \le y \le x\} \le \Sigma$, so $\{\tilde{x}, x\}$ is a $(\Sigma, 1)$-approximation set of φ over $[\tilde{x}, x]$. The rest of the proof follows from [HNO15, Thm. 3.2]. □

A.8 Theorems 1.2 and 3.1

Proof. Consider first multiplicative approximations with the following deterministic convex single-period DP. Let A be an arbitrary positive integer number, $T = 1,\ \mathcal{S}_1 = \mathcal{S}_2 = [-A, A],\ \mathcal{A}_1(I) = [-A, A],\ \forall I \in \mathcal{S}_1,\ D_1 \equiv 1,\ f_1(I, x, D) \equiv 0,\ g_2(I) \equiv 0$ and $g_1(I, x, d) = \varphi(x)$, where $\varphi \in \Phi$ is a family of 1-Lipschitz continuous convex functions over $[-A, A]$ as defined in the proof of Proposition 2.5. It is easy to verify that Conditions 1 and 2 are satisfied. By defining $g_1^I = g_1^u \equiv 0,\ f_1''(x, d) = x$ and $g_1^x(x) = \varphi(x)$ we get that Condition 3 is also satisfied, so we get indeed a continuous DP which fits into our model. Note that $z_1(I) = 0$ if and only if $\varphi^{\min} = 0$ for any $I \in [-A, A]$. Since any multiplicative approximation of $z_1(I)$ is 0 if and only if $z_1(I) = 0$, this cannot be computed in finite time due to Proposition 2.5. As each call to the oracle described in the proof of Proposition 2.5 can be identified with a discretization point, Theorem 1.2 follows with relative errors

We next consider additive approximations with the following implicit convex single-period DP. Let A be an arbitrary positive integer number, $T = 1, \mathcal{S}_1 = [0, 0], \mathcal{S}_2 = [0, A], \mathcal{A}_1(0) = [0, 0], D_1 \in [0, \ldots, A]$ with $p_0 = p_1 = \cdots = p_A = 1/(A+1)$, $f_1(I, x, d) = d, g_1(I, x, d) = 0, g_2(I) = (A+1)\varphi(I)$ with $\varphi \in \Phi$, where $\varphi^*, \varphi_i \in \Phi$ are $2A(A+1)\Sigma$-Lipschitz continuous nonnegative convex functions as defined in the proof of Proposition 2.3. Assume that D_1 is given implicitly via an oracle to the CDF. It is easy to verify that Conditions 1, 2, 3 are satisfied and this is an implicit convex DP. Call $\beta = \sum_{j=0}^{A} \varphi^*(j)$. Suppose that $\varphi \equiv \varphi^*$. Because D_1 is uniformly distributed, it is easy to verify that $z_1(0) = \beta$, and this is the optimum of the DP. Now suppose that $\varphi \equiv \varphi_i$ for any $i \in [1, A-1]$. In this case, as φ^* differs from φ_i only in i by a value of 2Σ, we have $z_1(0) = \beta + 2\Sigma$ and this is the optimum of the DP. Suppose we were able to compute a Σ-approximation to the optimum of the DP. But then we would be able to distinguish between the case $\varphi \equiv \varphi^*$ and the case $\varphi \equiv \varphi_i$ for any i. Due to to Proposition 2.3, this cannot be done in time polynomial in $\log A$.

We give an alternative proof for additive approximations that uses a deterministic DP; however, this is a *monotone* DP (the continuous version of the monotone DPs of [HKL$^+$14]) rather than convex. Let A be an arbitrary positive integer number, $T = 1,\ \mathcal{S}_1 = \mathcal{S}_2 = [0, A],\ \mathcal{A}_1(I) = [0, A],\ \forall I \in \mathcal{S}_1,\ D_1 \equiv 1,\ f_1(I, x, D) = x,\ g_2(I) = A + \varphi^*(I)$ and $g_1(I, x, d) = A - \varphi(x)$, where φ^* and $\varphi \in \Phi$ are Lipschitz continuous nonnegative monotone functions as defined in the proof of Proposition 2.3. It is easy to verify that Conditions 1 and 2 are satisfied, and g_1 is monotone in the sense of [HKL$^+$14], so we get indeed a continuous DP which fits into an extension of our model to monotone DPs. Note that $z_1(I) = 2A$ if $\varphi \equiv \varphi^*$ and $z_1(I) = 2A - 2\Sigma$ if $\varphi = \varphi_i$ as defined in the proof of Proposition 2.3 by choosing action $x = i$. Since any additive Σ-approximation value of $z_1(I)$ is at most $2A - \Sigma$ if $z_1(I) = 2A - 2\Sigma$, and is at least $2A$ if $z_1(I) = 2A$ we get a separation between these two cases. But due to Proposition 2.3 this cannot be computed in time polynomial in $\log A$. As each point in the representation described in Proposition 2.3 can be identified with a discretization point, Theorem 1.2 follows with additive errors.

□

A.9 Proposition 3.2

Proof. Let $\chi = E_D(\psi(f(x,D)))$ and notice that:

$$\begin{aligned}
\chi(x) &= E_D(\psi(f(x,D))) \\
&= \textstyle\sum_{j=1}^{m} \psi(f(x,d_j))\text{Prob}(D=d_j) \\
&= \psi(A) + \textstyle\sum_{j=1}^{m}\sum_{i=1}^{n} \psi_i(f(x,d_j))\text{Prob}(D=d_j) \\
&= \psi(A) + \textstyle\sum_{i=1}^{n}\left(\sum_{j=1}^{m} \psi_i(f(x,d_j))\text{Prob}(D=d_j)\right) \\
&= \psi(A) + \textstyle\sum_{i=1}^{n}(\Delta_i-\Delta_{i-1})\sum_{j=1\mid f(x,d_j)\geq a_i}^{m}(f(x,d_j)-a_i)\text{Prob}(D=d_j) \\
&= \psi(A) + \textstyle\sum_{i=1}^{n}(\Delta_i-\Delta_{i-1})\sum_{j=1\mid bx+e-d_j\geq a_i}^{m}(bx+e-d_j-a_i)\text{Prob}(D=d_j) \\
&= \psi(A) + \textstyle\sum_{i=1}^{n}(\Delta_i-\Delta_{i-1})\left((bx+e-d_{m_i}-a_i)F(d_{m_i}) + \sum_{k=1}^{m_i-1}(d_{k+1}-d_k)F(d_k)\right).
\end{aligned} \tag{4}$$

Then

$$\begin{aligned}
E_D(\xi(f(x,D))) &= \textstyle\sum_{j=1}^{m} \xi(f(x,d_j))\text{Prob}(D=d_j) \\
&\leq \chi(x) \\
&= \psi(A) + \textstyle\sum_{i=1}^{n}(\Delta_i-\Delta_{i-1})\left((bx+e-d_{m_i}-a_i)F(d_{m_i}) + \sum_{k=1}^{m_i-1}(d_{k+1}-d_k)F(d_k)\right) \\
&\leq \psi(A) + \textstyle\sum_{i=1}^{n}(\Delta_i-\Delta_{i-1})\left((bx+e-d_{m_i}-a_i)\tilde{F}(d_{m_i}) + \sum_{k=1}^{m_i-1}(d_{k+1}-d_k)\tilde{F}(d_k)\right),
\end{aligned}$$

where the first inequality is due to ψ being a (Σ_1,Π_1)-approximation function of ξ, the second equality is due to (4) and the second inequality is due to $\tilde{F}(y)$ being a Π_2-approximation function of $\text{Prob}(D\leq y)$. On the other hand, by using similar arguments we have

$$\begin{aligned}
E_D(\psi(f(x,D))) &= \sum_{j=1}^{m}\psi(f(x,d_j))\text{Prob}(D=d_j) \\
&\leq \Sigma_1 + \Pi_1\sum_{j=1}^{m}\xi(f(x,d))\text{Prob}(D=d_j) = \Sigma_1 + \Pi_1 E_D(\xi(f(x,D))),
\end{aligned}$$

and

$$\begin{aligned}
\tilde{\xi} &\leq \psi(A) + \Pi_2\sum_{i=1}^{n}(\Delta_i-\Delta_{i-1})\left((bx+e-d_{m_i}-a_i)F(d_{m_i}) + \sum_{k=1}^{m_i-1}(d_{k+1}-d_k)F(d_k)\right) \\
&\leq \Pi_2 E_D(\psi(f(x,D))).
\end{aligned}$$

We get the desired approximation ratio by combining the above two inequalities. From the one before last line in (4), it is easy to verify that $\tilde{\xi}$ is a convex piecewise linear increasing function where each slope increase belongs to the set $\{b\text{Prob}(D=d_k)(\Delta_i-\Delta_{i-1}) \mid k=1,\ldots,m;\ i=1,\ldots,n\}$. We conclude the proof by noting that one can compute $\tilde{\xi}$ in the claimed running time by first preprocessing the $m-1$ partial sums $\sum_{k=1}^{m-1}(d_{k+1}-d_k)\tilde{F}(d_k)$. □

A.10 Proposition 3.4

Proof. We apply Proposition A.2 with the following parameter setting: Let $D=\mathcal{S}_t$, $C(\cdot)=\mathcal{A}_t(\cdot)$, $n=3$, $x=I_t$ and $y=x_t$. Let $\varphi_1(\cdot)=G_t^x(\cdot)$, $\breve{\varphi}_1(\cdot)=\breve{G}_t^x(\cdot)$ and $\psi_1(I,x)=x$. Let $\varphi_2(\cdot)=G_t^u(\cdot)$, $\breve{\varphi}_2(\cdot)=\breve{G}_t^u(I_t,\cdot)$ and $\psi_2(I,x)=aI+bx$. Let $\varphi_3(\cdot)=Zf_{t+1}(\cdot)$, $\breve{\varphi}_3(\cdot)=\breve{Z}f_{t+1}(\cdot)$ and $\psi_3(I,x)=aI+bx$. Let $\Sigma_1=\Sigma_x$, $\Sigma_2=\Sigma_u$, $\Sigma_3=\Sigma_z$ and $\Pi_1=\Pi_x$, $\Pi_2=\Pi_u$, $\Pi_3=\Pi_z$. Hence, by Proposition A.2, $\bar{\varphi}$ is a convex $(\Sigma+\Sigma_x+\Sigma_u+\Sigma_z, \max\{\Pi_x,\Pi_u,\Pi_z\})$-approximation of $\sum_{i=1}^{3}\varphi_i$. By the calculus of approximation (property summation of approximation) we conclude that $\breve{G}_t^I+\bar{\varphi}$ is a convex $(\Sigma+\Sigma_I+\Sigma_x+\Sigma_u+\Sigma_z,\Pi_z)$-approximation of z_t. The computational time follows from Proposition 2.2. □

A.11 Theorem 3.5

Proof. Note first that Proposition 3.4 assures us that $\bar{z}_t$ is a convex function. Hence, the call to function CompressConv in step 11 is valid.

Next, we prove that $\breve{z}_1$ constructed by Algorithm 5 is a succinct $(\delta, 1+\epsilon)$-approximation of z_1. To do so, we first show by induction that $\breve{z}_t$ is a convex $((6(T+1-t)+1)\Pi^{(T+1-t)+1/3}\Sigma, \Pi^{3(T+1-t)+1})$-approximation function of z_t for every $t = 1, \ldots, T+1$. For the base case of $t = T+1$, we apply Proposition 2.13 with $\varphi = g_{T+1}$, and $W = W_{T+1}$, and get that $\breve{z}_{T+1}$ is convex $(\Pi''\Sigma, \Pi)$-approximation of z_{T+1}. Thus, the base case is valid. The induction hypothesis is that $\breve{z}_{t+1}$ is a convex $((6(T-t)+1)\Pi^{(T-t)+1/3}\Sigma, \Pi^{3(T-t)+1})$-approximation of z_{t+1}.

Note that in steps 5-11 in the iteration that calculates $\breve{z}_t$ we have $\Pi' = \Pi^{3(T-t)+1}$ and $\Pi'' = \Pi^{(T-t)+1/3}$, so we will show that $\breve{z}_t$ is a convex $((6(T+1-t)+1)\Pi''\Pi\Sigma, \Pi'\Pi^3)$-approximation of z_t. Applying CompressConv with $\Sigma_1 = \Pi''\Sigma/2,\ \Pi_1 = \Pi',\ \Sigma_2 = 0, \Pi_2 = 1$ we get from Proposition 2.14 that $\breve{g}_t^I, \breve{g}_t^x, \breve{g}_t^u$ are succinct $(\Pi''\Sigma/2, \Pi')$-approximations of g_t^I, g_t^x, g_t^u, respectively. In step 7 we apply Corollary 3.3 with $\xi = g_t^I$ (resp. g_t^x, g_t^u), $\psi = \breve{g}_t^I$ (resp. $\breve{g}_t^x, \breve{g}_t^u$), $\Sigma_1 = \Pi''\Sigma/2, \Pi_1 = \Pi'$ and $\Pi_2 = \Pi$, to get convex $(\Pi''\Pi\Sigma/2, \Pi'\Pi)$-approximations of G_t^I (resp. G_t^x, G_t^U), respectively. In step 8 we apply Corollary 3.3 with $\xi = z_{t+1}, \psi = \breve{z}_{t+1}, \Sigma_1 = (6(T-t)+1)\Pi''\Sigma, \Pi_1 = \Pi'$ and $\Pi_2 = \Pi$, to get that $\bar{Z}f_{t+1}$ is a convex $((6(T-t)+1)\Pi''\Pi\Sigma, \Pi'\Pi)$-approximation value oracle for Zf_{t+1}.

Applying CompressConv with $\Sigma_1 = \Pi''\Sigma/(2\Pi'),\ \Pi_1 = \Pi,\ \Sigma_2 = \Pi''\Pi\Sigma/2, \Pi_2 = \Pi'\Pi$ we get by Proposition 2.14 that $\breve{G}_t^I, \breve{G}_t^x, \breve{G}_t^u$ are succinct $(\Pi''\Pi\Sigma, \Pi'\Pi^2)$-approximations of G_t^I, G_t^x, G_t^u, respectively. Moreover, due to the same proposition, this time applied with $\Sigma_1 = \Pi''\Sigma/\Pi',\ \Pi_1 = \Pi,\ \Sigma_2 = (6(T-t)+1)\Pi''\Pi\Sigma, \Pi_2 = \Pi'\Pi$, we get that $\breve{Z}f_{t+1}$ is a succinct $((6(T-t)+2)\Pi''\Pi\Sigma, \Pi'\Pi^2)$-approximation of Zf_{t+1}.

Considering step 11, in order to calculate a value oracle for $\bar{z}_t$ we apply Proposition 3.4 with $\Sigma_I = \Sigma_x = \Sigma_u = \Pi''\Pi\Sigma,\ \Sigma_z = (6(T-t)+2)\Pi''\Pi\Sigma$ and $\Pi_I = \Pi_x = \Pi_u = \Pi_z = \Pi'\Pi^2$ to get that $\bar{z}_t$ is a convex $((4+(6(T-t)+2))\Pi''\Pi\Sigma, \Pi'\Pi^2)$-approximation function of z_t.

Applying CompressConv with $\varphi = z_t$, $\tilde{\varphi} = \bar{z}_t$, $\Sigma_1 = \Pi''\Sigma/(\Pi'\Pi),\ \Pi_1 = \Pi$, $\Sigma_2 = 6(T+1-t)\Pi''\Pi\Sigma,\ \Pi_2 = \Pi'\Pi^2$, we get due to Proposition 2.14 that $\breve{z}_t$ is a convex $((6(T+1-t)+1)\Pi''\Pi\Sigma, \Pi'\Pi^3)$-approximation of z_t, or equivalently, a $((6(T+1-t)+1)\Pi^{(T+1-t)+1/3}\Sigma, \Pi^{3(T+1-t)+1})$-approximation function of z_t. This completes the proof by induction. Recalling that $\Pi = \sqrt[3T+1]{1+\epsilon}$, and $\Sigma = \frac{\delta}{(6T+1)\sqrt[3]{1+\epsilon}}$ and taking $t = 1$, we have that $\breve{z}_1$ is indeed a $(\delta, 1+\epsilon)$-approximation of z_1 as requested.

It remains to prove that the running time of Algorithm 5 is polynomial in both the input size and $\frac{1}{\epsilon} + \log\frac{1}{\delta}$. Recall that $\log U_{\mathcal{S}}$, $\log U_{\mathcal{A}}$, and $\log U_g$ take all part in the input size. Note that

$$O(\log\Pi) = O(\epsilon/T), \qquad O(\log\Pi') = O(\log\Pi'') = O(\log(1+\epsilon)) = O(\epsilon).$$

Clearly, the running time of the algorithm is dominated by the for-loop, which has T iterations. We analyze hereby the running time of a single iteration. Due to [HKL$^+$14, Prop. 4.5, Prop. 4.6], the running time of step 5 is $O(\log_\Pi(1/\gamma)\log n_t)$ and the query time of $\tilde{F}_t$ is

$$t_{\tilde{F}_t} = O(\log\log_\Pi(1/\gamma)) = O\left(\log\frac{T\log(1/\gamma)}{\epsilon}\right). \tag{5}$$

We note that the running time of step 6 is dominated by that of step 9 because while we do have direct access to $\breve{g}_t^I, g_t^x, g_t^u$ we have no direct access to $\bar{G}_t^I, \bar{G}_t^x, \bar{G}_t^u$. Due to Proposition 2.13, the query time of $\breve{g}_t^\diamond$ is

$$t_{\breve{g}_t^\diamond} = O(\log\log_{\Pi'}\frac{g_t^{\diamond\max}}{\Pi''\Sigma}) = O\left(\log\frac{\log(TU_g/\delta)}{\epsilon}\right). \tag{6}$$

(Note that this is an overestimation. For the sake of simplicity we omit the term $\frac{\sigma_\varphi^{\max}}{\sigma_\varphi(A+\Delta_{\bar{x}})}$ from the $\min$ term throughout the proof.) Step 7 is performed by applying Corollary 3.3 with parameters set to

$\xi = g_t^\diamond,\ \psi = \breve{g}_t^\diamond,\ \Sigma_1 = \Pi''\Sigma/2,\ \Pi_1 = \Pi',\ n^- + n^+ = n = O(\log_{\Pi'} \frac{g_t^{\diamond\max}}{\Pi''\Sigma}),\ D = D_t,\ \Pi_2 = \Pi$ and $m = O(\log_\Pi(1/\gamma))$. By (5)-(6) we get that the query time of $\bar{G}_t^\diamond$ is therefore

$$t_{\bar{G}_t^\diamond} = O\left(\frac{1}{\epsilon}(\log(TU_g/\delta) + T\log(1/\gamma))(\log\frac{\log(TU_g/\delta)}{\epsilon} + \log\frac{T\log(1/\gamma)}{\epsilon})\right).$$

Step 8 is also performed by applying Corollary 3.3, this time with parameters set to $\xi = z_{t+1},\ \psi = \breve{z}_{t+1},\ \Sigma_1 = (6(T-t)+1)\Pi''\Pi\Sigma,\ \Pi_1 = \Pi'\Pi^3,\ n^- + n^+ = n = \log_{\Pi'} \frac{z_{t+1}^{\max}}{T\Pi''\Sigma},\ D = D_t,\ \Pi_2 = \Pi$ and $m = \log_\Pi(1/\gamma)$. As $O(z_{t+1}^{\max}) = O(TU_g)$ we get that the query time of $\bar{Z}f_{t+1}$ is therefore $t_{\bar{Z}f_{t+1}} = O(t_{\bar{G}_t^\diamond})$

We consider next the query time of $\bar{z}_t$. By Proposition 3.4, the query time of $\bar{z}_t$ is $t_{\bar{z}_t} = O(t_{\breve{G}_t^I} + (t_{\breve{G}_t^x} + t_{\breve{G}_t^u} + t_{\breve{Z}f_{t+1}})\log(b\kappa U_{\mathcal{A}}/(\Pi''\Pi\Sigma)))$. Note that by Proposition 2.13,

$$t_{\breve{G}_t^I} + t_{\breve{G}_t^x} + t_{\breve{G}_t^u} + t_{\breve{Z}f_{t+1}} = O(\log\log_\Pi \frac{\Pi' U_g}{\Pi''\Sigma}) = O(\log\frac{TU_g}{\epsilon\delta}),$$

so

$$t_{\bar{z}_t} = O(\log\frac{TU_g}{\epsilon\delta}\log\frac{\kappa TU_A}{\delta}).$$

Due to Proposition 2.14, the total running time of steps 9-11 (during a single iteration of the for-loop) is

$$O(\log_\Pi \frac{\Pi' TU_g}{\Pi''\Sigma}\log(b\kappa U_{\mathcal{S}})(t_{\bar{G}_t^\diamond} + t_{\bar{Z}f_{t+1}} + t_{\bar{z}_t})) = \frac{T}{\epsilon}\log\frac{TU_g}{\delta}\log(\kappa U_{\mathcal{S}})(t_{\bar{G}_t^\diamond} + t_{\bar{Z}f_{t+1}} + t_{\bar{z}_t}))$$

Therefore, the running time of the entire algorithm is

$$O\left(\frac{T^2}{\epsilon}\left[\log\frac{TU_g}{\delta}\log(\kappa U_{\mathcal{S}})(t_{\bar{G}_t^\diamond} + t_{\bar{Z}f_{t+1}} + t_{\bar{z}_t}) + \log n^* \log\frac{1}{\gamma}\right]\right),$$

which is polynomial in both $\frac{1}{\epsilon}, \log\frac{1}{\delta}$ and the input size. □

B Computational experiments

We provide a preliminary computational evaluation of the proposed approximation scheme. We compare it to an EXACT DP solution to the discretized problem (recall by Theorem 1.2 that the error could be large, though in practice this is typically not the case), and to our implementation of the SPAR algorithm of [NP13]. SPAR as described in [NP13] requires linear costs, but we can apply it to nonlinear convex problems because our action space is univariate and we can minimize over it via binary search. We also remark that the convergence proof of [NP13] relies on knowing the minimum distance between breakpoints of the value function; in our case, this distance is unknown, and we resort to applying SPAR approximately, using a small value for the minimum distance between breakpoints. This forsakes convergence in theory, but it is a commonly used approach in practice. Despite being a value iteration algorithm, SPAR converges to the optimal policy, but not necessarily to the optimal objective value: it computes the value functions up to a constant, and therefore it does not provide an approximation of the objective value at termination. We use the number of iterations without improvement as a stopping critearion; notice that if this number is set too small, it is possible that sample paths with low probability are never explored and the algorithm exits with a relatively low quality policy. Finally, we remark that SPAR is more general than our algorithm as it can be applied to problems with large-dimensional action spaces, as long as some linearity conditions are satisfied. SPAR converges asymptotically but does not provide an approximation guarantee in finite time. We implemented SPAR as described in [NP13], without additional fine-tuning.

In Appendix B.3, we compare the (Σ, Π)-FPTAS discussed in this paper with the (discrete) FPTAS presented in [HNO15]; the comparison with the discrete FPTAS is reported separately because of the

need to test several discretization steps and approximation factors, resulting in a large number of curves that would clutter the graphs in the other sections.

We made the following choices for our experimental setup. First, we test the algorithms on problems with a small uncertainty set, i.e., each discrete random variable can only take on a small set of values. This choice is dictated by the fact that otherwise computing the solution with EXACT would not be practically feasible even for very small instances. Since we are testing a new methodology, comparing with the standard method to compute exact solutions is important. As a consequence of the small uncertainty sets, it is better to compute expectations exactly rather than approximate them, for all algoritms. Second, we only compare the value of the policies computed by the algorithms, i.e. the cost of the sequence of actions implicitly defined by the value functions z_t for $t = 1, \ldots, T$ via the $\arg\min$ operator, rather than the approximation of the objective function value $z^*(I_1)$ of equation (1). This is because SPAR tries to approximate the value functions up to a constant (i.e. it tries to approximate their slopes only, not their values), therefore it does not provide any approximation of $z^*(I_1)$, but it yields a policy. Note that to compute the value of a policy, we would have to explore all sample paths and determine a sequence of actions for each of them. If this is computationally infeasible, i.e. there are more than 20000 possible realizations of the vector of random variables $(D_1, \ldots, D_T)$, we estimate the value of a policy through Monte Carlo simulation over 20000 sample paths (we use the same samples for all algorithms).

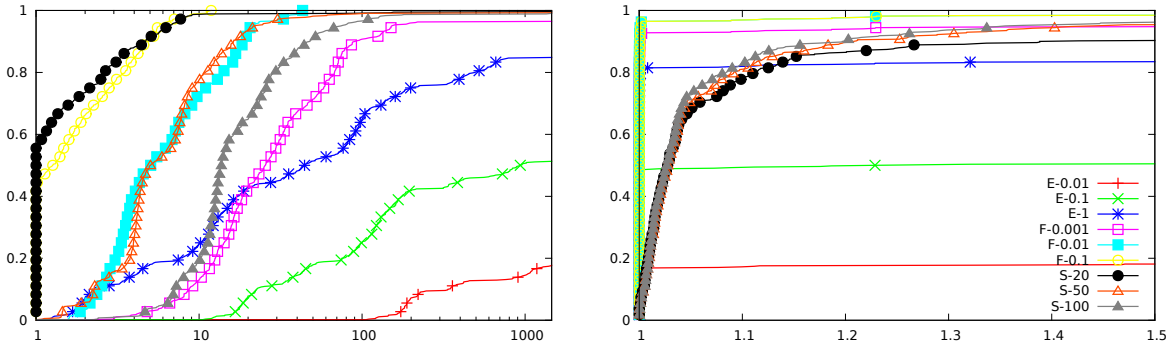
All algorithms are implemented in Python 3 using SciPy when appropriate, and tested on an Intel Xeon E5-4620 @ 2.20 Ghz (HyperThreading and TurboBoost disabled). The time limit is set to 1000 seconds per instance, and if an algorithm does not converge before the time limit, that particular run is considered a failure; however, since SPAR returns a policy even when it does not converge, we consider the corresponding policy when comparing policy values.

We test the algorithms on randomly generated instances of a continuous single-item inventory control problem, which is a classical problem in supply chain management: we have to manage a warehouse over a finite time horizon to satisfy uncertain demand, minimizing procurement costs, holding costs, and backlogging costs. Results are discussed in the next section. We additionally tested our algorithms on the problem of managing a finite-capacity battery to satisfy an uncertain energy demand from a household in a context with an intermittent but free energy source (e.g. wind turbine) and a reliable but expensive energy source (e.g. power grid). Results for this additional problem confirm the analysis for the inventory control problem and are reported in Appendix B.4 more briefly.

B.1 Experiments with the continuous inventory control problem

We generate inventory control instances similarly to [HNO15], but the inventory is assumed to take continuous values (e.g. if the goods are liquid) and the demand is allowed to be fractional. An instance of the stochastic single-item inventory control problem is defined by the number of time periods T, the maximum demand in each period M, and the type of cost functions. At each time period t, the set $\mathcal{D}_t$ of the $N = 5$ demand values is generated as follows: first we draw the maximum demand w_t at stage t uniformly at random in $[\lfloor M/2 \rfloor, M]$, then we draw the remaining $N - 1$ values in $[1, w_t]$. This method ensures that none of the instances we generate is too easy and that the difficulty scales with M. The probabilities with which demands occur are randomly assigned by generating N integers $q_i, i = 1, \ldots, N$ in $[1, \ldots, 10]$, and computing the probability of the k-th value of the demands as $q_k / (\sum_{i=1}^{N} q_i)$. We must also define the cost functions, namely: procurement, storage and backlogging costs. For each of these functions, we select a coefficient c uniformly at random in $[1, \ldots, 20]$ for unit procurement cost, in $[1, \ldots, 10]$ for storage costs, in $[10, \ldots, 50]$ for backlogging costs. Then we consider three different types of cost functions: linear ($f(x) = cx$), quadratic ($f(x) = cx^2$) and cubic ($f(x) = cx^3/1000$).

In our experiments, the number of stages is $T \in \{5, 10, 20\}$, the maximum demand is $M \in \{100, 1000, 10000\}$ (the size of the state and action spaces is proportional to M), and the various cost functions are convex polynomials of degree $d \in \{1, 2, 3\}$. At each stage, the uncertain demand can take on exactly 5 values, as mentioned above. We generate 20 instances for each combination of the parameters T, M, d, for a total of 540 instances.



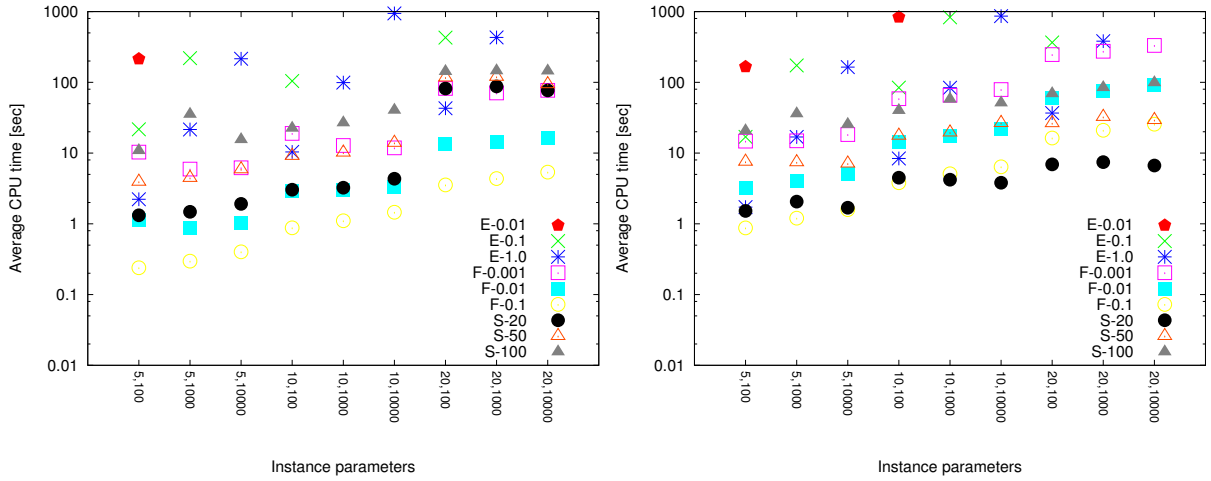
(a) CPU time.

(b) Policy value.

Figure 1: Performance profile of the running time (left) and policy value (right) on the inventory control problem for different algorithms. The x-axis of the left graph is a on a log scale.

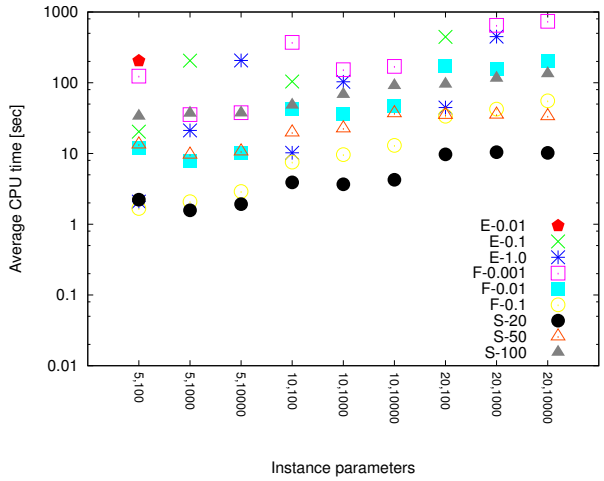
We compare the algorithms in terms of CPU time and quality of the computed policies using performance profiles [DM02], see Fig. 1. A performance profile reports the fraction (y axis) of instances for which an algorithm is at most k times (x axis) worse than the best performing algorithm on that instance. We report results for the following algorithms: EXACT with discretization $0.01, 0.1, 1.0$ (labeled E-0.01, E-0.1, E-1 respectively); the (Σ, Π)-FPTAS with $\Sigma = 10$ and $\Pi = 1.001, 1.01, 1.1$ (labeled F-0.001, F-0.01, F-0.1 respectively); and SPAR with minimum distance between breakpoints 0.01 and stopping criterion set to $20, 50, 100$ iterations without improvement (labeled S-20, S-50, S-100 respectively). We tried both smaller and larger discretization steps for EXACT and SPAR; we find this particular set of parametrizations to be representative of the overall picture. The performance profiles show very clearly that the FPTAS is better than EXACT under every respect, except when Π is chosen too small $(= 1.001)$. Not only it is significantly faster, sometimes by more than two orders of magnitude, but it also finds policies of similar quality. As can be seen from Fig. 1 (a), F-0.1, F-0.01 and S-20 solve all instances within the time limit, while E-0.01 solves only $\approx 18\%$. The quality of the policies found by the FPTAS is matched only by the much slower EXACT: E-0.01, which we expect to be the best algorithm in terms of solution quality, finds better policies than F-0.1 in 90 instances out of the 98 it solves, but the policy value of F-0.1 is worse by 3.8% only on average. This suggests that even the FPTAS with $\Pi = 1.1$ finds near-optimal policies. The fastest version of EXACT is E-1, but on the instances on which E-1 finds better policies than F-0.1, the improvement in policy value is only 0.6%, and this comes at the cost of a much larger solution time. The FPTAS seems to outperform SPAR on these problems: S-20 is generally faster than F-0.1, but it is slower on average (12.89 seconds vs. 10.98 seconds), and the quality of the policies is much better for F-0.1. S-50 has comparable speed to F-0.01, but returns worse policies. For completeness, in addition to the performance profiles of Fig. 1 we give a plot of the CPU times for different classes of instances in Fig. 2 (a)-(c).

Since EXACT with fine discretization typically provides better policies, it is natural to ask which algorithm finds the best policy for equal CPU time. To this end, we plotted a graph of the average policy values versus average CPU times for each group of instances solved by EXACT: an example of such graph is reported in Fig. 2 (d), where it appears evident that the FPTAS is faster than EXACT for equal policy value. The plot does not show SPAR because it yields considerably larger policy values ($\approx$ 309000) and including it would make the differences between FPTAS and EXACT invisible on the graph. Of course, SPAR has broader applicability (e.g. action spaces described by polyhedra, the capability of returning a policy even after a small number of iterations) that would make it more suitable in a different setting; a more comprehensive comparison requires a different experimental setup, and is left for future research.

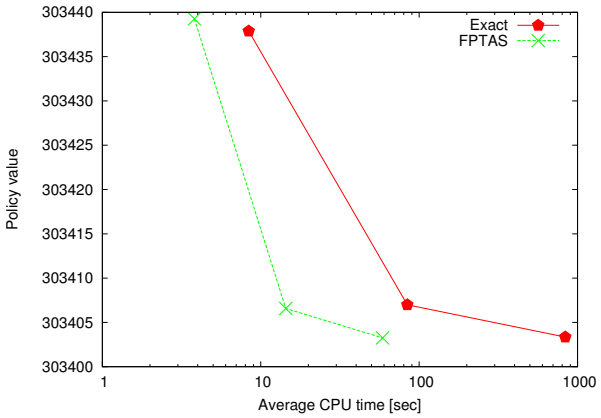


(a) CPU time for instances with linear costs.

(b) CPU time for instances with quadratic costs.



(c) CPU time for instances with cubic costs.



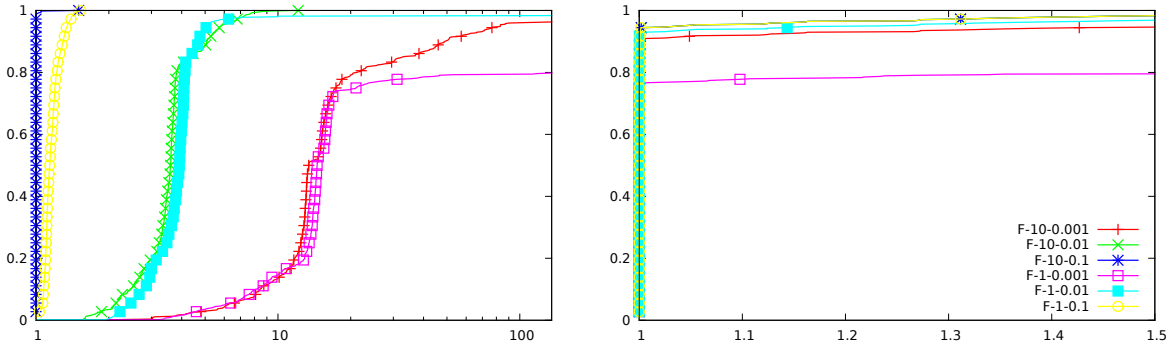
(d) CPU time versus policy value.

Figure 2: Figures (a)-(c) report the running time of different algorithms for all groups of inventory control instances; the x-axis indicats the number of time periods T and the size of the state space M. Figure (d) reports the average CPU time versus the average policy value for 20 inventory control instances with $T = 10, M = 100, d = 2$.

B.2 On the effect of varying Σ

In Appendix B.1 we reported results obtained with the (Σ, Π)-FPTAS with $\Sigma = 10$ and different values of Π. The effect on the running time of varying Π is evident from Fig. 1(a). It is natural to ask whether Σ has the same level of impact. In Fig. 3, we provide performance profiles for the same inventory control problem used in Appendix B.1 for six parametrizations of the FPTAS: all combinations of $\Sigma \in \{1, 10\}$ and $\Pi \in \{1.001, 1.01, 1.1\}$. The label for the algorithms is of the form F-Σ-($\Pi - 1$).

As can be seen from Fig. 3(a), varying Σ is significantly less impactful on the running time than varying Π: decreasing Σ from 10 to 1 typically only changes the running time by a small factor (remember that the x axis of Fig. 3(a) is on a log scale). The quality of the policies does not seem to be affected by Σ. However, an iteresting effect can be observed when $\Pi = 1.001$: roughly 20% of the instances that cannot be solved with $\Sigma = 1$ are solved with $\Sigma = 10$. These are all relatively small instances ($T = 5$ or 10, and $M = 100$) with small total cost: because of the very small Π, finding a sufficiently accurate (Σ, Π)-approximation of the value functions is not possible within the time limit when Σ is also small. This empirical observation is in agreement with our theoretical result that a purely multiplicative approximation (i.e. $\Sigma = 0$) is not possible in general. However, this difficulty is not present on other



(a) CPU time.

(b) Policy value.

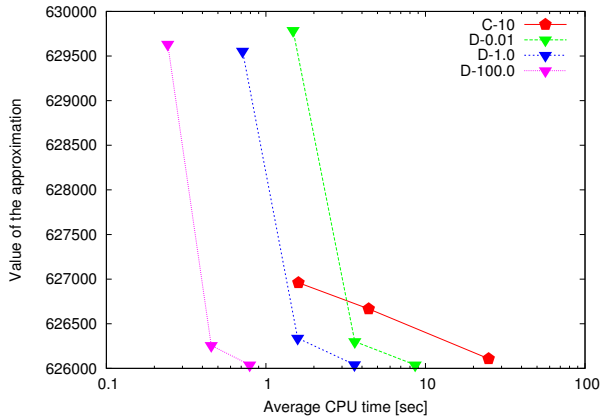
Figure 3: Performance profile of the running time (left) and policy value (right) on the inventory control problem for different parametrizations of the FPTAS. The x-axis of the left graph is a on a log scale.

instances with larger total cost, because in those case the multiplicative part of the approximation is easier to satisfy (due to a larger value of the "starting point" of the approximation routines), and allows longer intervals between breakpoints of the approximation, regardless of Σ.

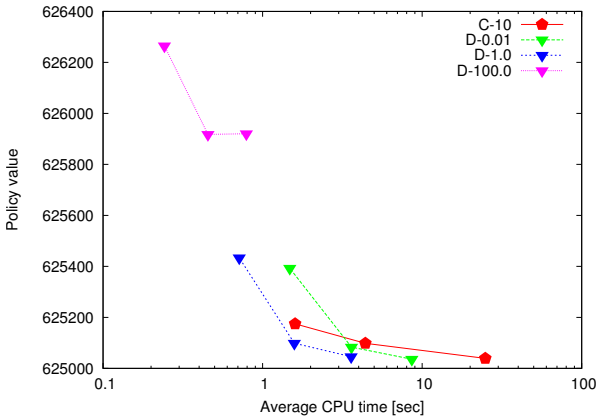
B.3 Direct approximation versus discretization plus approximation

A natural question raised by this paper is whether it is better to approximate a continuous DP directly, which is the approach we follow in this paper, or to discretize the DP and approximate the discretized version. Thm. 1.2 indicates that discretizing forsakes the error bounds, but one would expect that in practice a fine discretization yields accurate solutions. In this section we provide an attempt at answering this question from a numerical point of view.

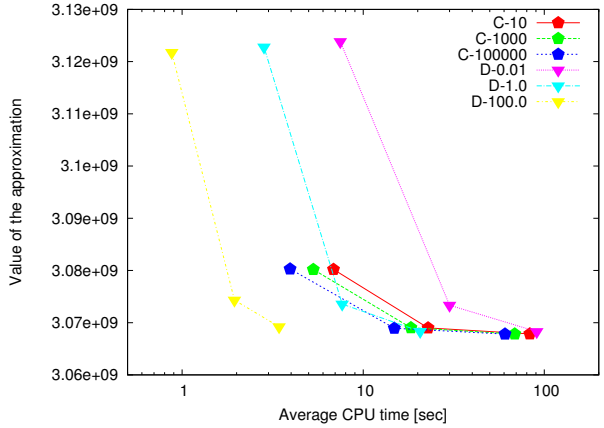
[HNO15] describes an FPTAS for discrete convex DP, using the discrete counterpart of the approximation techniques discussed in this paper, and shows that the FPTAS is effective from a numerical point of view. We compare the FPTAS of [HNO15] with discretization step $\Delta = 10^k$ for $k = -3, \ldots, 3$, to our (Σ, Π)-FPTAS for different values of Σ. In Fig. 4 we report the average objective function value (i.e. approximation of $z^*(I_1)$ in (1)) and policy value versus the corresponding average CPU time over a group of instances of the same size. These graphs are representative of the behavior over other groups of instances. We only report $\Delta = 0.01, 1.0, 100.0$ to avoid cluttering the graphs. Fig. 4 shows that the (Σ, Π)-FPTAS applied to the continuous problem returns better objective function values and policy values for equal value of Π in all cases (in our experiments, the discrete FPTAS with discretization step $\Delta = 0.001$ can sometimes be more accurate, but in that case the solution time of the discretized problem is much larger and sometimes above the 1000 second time limit). Coarse discretization with Δ up to 10 and approximation with the discrete FPTAS of [HNO15] is fast and returns values of good quality, especially on instances with linear costs. With $\Delta > 10$, the discretization error is significant and the solution quality rapidly deteriorates. On instances with quadratic (or larger degree) costs, the function values grow faster and the discretization approach suffers from larger errors, hence approximating the continuous problem directly yields better quality solutions. In particular, the (Σ, Π)-FPTAS with $\Sigma = 10^5$ is always better for equal CPU time: there is only one point in the graph where the discrete FPTAS is slightly below the curve of the (Σ, Π)-FPTAS, but this seems due to the coarse linearization of the curves in the graph. It is also noteworthy that in our experiments, the (Σ, Π)-FPTAS with $\Pi = 1.1$ is significantly more accurate than the discretized FPTAS with the same approximation guarantee $\Pi = 1.1$, regardless of the size of the discretization step. Finally, we remark that the difference in the values of the approximation of $z^*(I_1)$ is hardly noticeable for different values of Δ up to 100 (but there is a significant deterioration for $\Delta = 1000$ in our experiments); however, the value of the policy produced with $\Delta = 100$ is quite close to the approximation guarantee, whereas with finer discretization steps (or with the continuous FPTAS) the value of the policy is much better than the approximation guarantee.



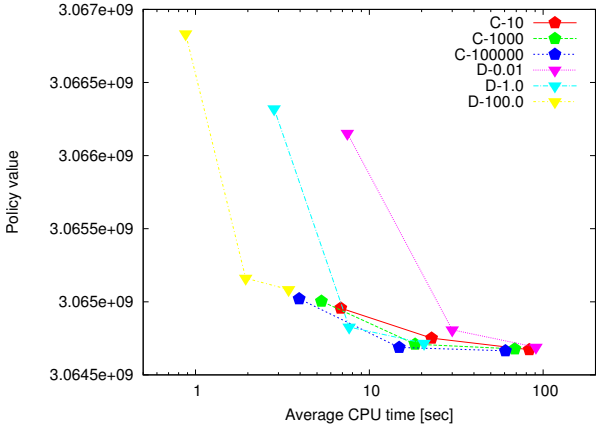
(a) Value of the approximation for instances with linear costs.



(b) Policy value for instances with linear costs.



(c) Value of the approximation for instances with quadratic costs.



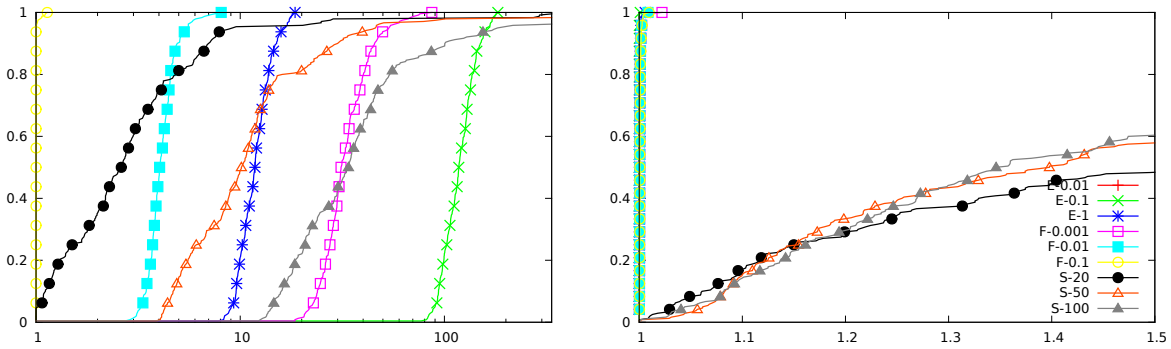
(d) Policy value for instances with quadratic costs.

Figure 4: Average value of the approximation of $z^*(I_1)$ (left) and policy value (right) versus CPU time for 20 inventory control instances with $T = 10, M = 10000$ and $d = 1$ (top) or $d = 2$ (bottom). We test the (Σ, Π)-FPTAS on the continuous problem (label "C-Σ") and the FPTAS of [HNO15] on the discretized problem with discretization step Δ (label "D-Δ"). For each algorithm, we test relative approximation factors $\Pi = 1.1, 1.01, 1.001$, reported in this order from left to right on each curve. In Figures (a) and (b) only $\Sigma = 10$ is reported because larger Σ did not yield significant reduction of the running time.

To summarize, in our experiments discretizing the continuous problem and solving it approximately forsakes the approximation guarantee, but in practice it works well as a heuristic if the discretization step is not too large. However, the performance of the discretization approach decreases when the slope of the cost function increases more rapidly: already for quadratic cost functions, approximating the continuous problem directly yields better solutions for comparable CPU time, while also providing an approximation guarantee, and seems therefore the superior algorithm.

B.4 Experiments with the energy storage problem

As mentioned in Appendix B.1, the results on the energy storage problem confirm our previous conclusions. We report them here for completeness. On this problem, we only test linear functions to be consistent with the literature. The capacity of the battery is set to 1000 MWh, the net balance (energy generated from the renewable energy source minus household demand) expressed in MW is drawn uniformly at random in $[-M, M]$ where we use 100, 200 in our experiments, the cost of buying from the grid is drawn uniformly at random in $[20, \ldots, 100]$ \$/MWh, and the storage cost is fixed at \$2/MWh.



(a) CPU time.

(b) Policy value.

Figure 5: Performance profile of the running time (left) and policy value (right) on the energy storage problem for different algorithms. The x-axis of the left graph is a on a log scale.

These values are based on [Zhou et al., *Managing Wind-based Electricity Generation in the Presence of Storage and Transmission Capacity*, Tepper Working Paper 2011-E36, revised January 2013]. Each time period corresponds to one hour and we perform experiments with $T \in \{10, 20, 24\}$ time periods.

To assess the performance of the algorithms on the energy storage problem, in Fig. 5 we report performance profiles. In this case, the number of instances is 240, because we test all the combinations of $T \in \{10, 20, 24\}$ and $M \in \{100, 200\}$ with two possible sizes of the support of the net energy balance (5, 10), constructing 20 random instances per group. We report results for the same algorithms and parametrizations as in Appendix B.1. Note that E-0.1 does not solve any instance so it does not appear in the graphs. Once again, we can see that the the (Σ, Π)-approximation scheme seems to outperform the other methods. It is significantly faster than EXACT, while finding policies of comparable value; and it is as fast and generally faster than SPAR, while finding much better policies. It should be noted that on this problem, EXACT with a coarse discretization seems to work quite well (athough it is slower than the FPTAS by at least one order of magnitude) and although it is slower than the FPTAS and SPAR, it solves most of the instances. This is due to the relatively small size of the state space, which is a consequence of the fact that the battery has limited capacity, independent of the number of stages. On the other hand, on the inventory control problem reported in Appendix B.1, the size of the state space (i.e. the number of possible inventory levels that must be considered) grows with the number of stages, yielding much larger state spaces for our most difficult instances. APXSETCONV (specifically, line 2 of Alg. 3) sporadically encounters numerical difficulties on some problems with cubic or quadratic costs due to limited precision; when that happens ($\approx 1\%$ of the inventory control instances), the approximation factor is not guaranteed. We did not observe this issue on problems with linear costs, such as the energy storage instances. because the slope of the value function is not too large and the machine precision seems to be sufficient to perform the computations to an acceptable level of accuracy.

It is important to remark that in our current setting the random variables are independent; one of the greatest advantages of SPAR is its capability of scaling well to Markov decision processes with large state space, and in our experiments this is not exploited. Therefore, it is perhaps not too surprising that EXACT with a coarse discretization performs better the SPAR in the absence of a complex underlying Markov decision process.

In conclusion, our computational evaluation indicates that the ideas proposed in the paper have the potential to be very successful in practice on the class of problems for which they apply, greatly outperforming existing approaches.