A Computationally Efficient FPTAS for Convex Stochastic Dynamic Programs

SIAM J. OPTIM.
Vol. 25, No. 1, pp. 317–350

A COMPUTATIONALLY EFFICIENT FPTAS FOR CONVEX STOCHASTIC DYNAMIC PROGRAMS*

NIR HALMAN†, GIACOMO NANNICINI‡, AND JAMES ORLIN§

Abstract. We propose a computationally efficient fully polynomial-time approximation scheme (FPTAS) to compute an approximation with arbitrary precision of the value function of convex stochastic dynamic programs, using the technique of K-approximation sets and functions introduced by Halman et al. [*Math. Oper. Res.*, 34, (2009), pp. 674–685]. This paper deals with the convex case only, and it has the following contributions. First, we improve on the worst-case running time given by Halman et al. Second, we design and implement an FPTAS with excellent computational performance and show that it is faster than an exact algorithm even for small problem instances and small approximation factors, becoming orders of magnitude faster as the problem size increases. Third, we show that with careful algorithm design, the errors introduced by floating point computations can be bounded, so that we can provide a guarantee on the approximation factor over an exact infinite-precision solution. We provide an extensive computational evaluation based on randomly generated problem instances coming from applications in supply chain management and finance. The running time of the FPTAS is both theoretically and experimentally linear in the size of the uncertainty set.

Key words. dynamic programming, approximation algorithms, inventory control, fully polynomial-time approximation scheme, discrete convexity

AMS subject classifications. 90C27, 90C39, 90C59

DOI. 10.1137/13094774X

1. Introduction.

We consider a discrete-time finite-horizon undiscounted stochastic dynamic program (DP) with finite state and action spaces, as defined in [4]. We now introduce the type of problems addressed in this paper, postponing a rigorous definition of each symbol until section 2. Our model has an underlying discrete-time system that evolves through time with dynamics of the form $I_{t+1} = f(I_t, x_t, D_t)$, $t = 1, \ldots, T$, where

t:	discrete-time index;
$I_t \in \mathcal{S}_t$:	state of the system at time t ($\mathcal{S}_t$ is the *state space* at stage t);
$x_t \in \mathcal{A}_t(I_t)$:	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:	discrete random variable over the sample space $\mathcal{D}_t$;
T:	number of time periods.

*Received by the editors December 5, 2013; accepted for publication (in revised form) October 21, 2014; published electronically January 29, 2015. An extended abstract of this work appeared in *Proceedings of ESA* 2013, Lecture Notes in Comput. Sci. 8125, H. Bodlaender and G. Italiano, eds., Springer, Berlin, 2013, pp. 577–588.
http://www.siam.org/journals/siopt/25-1/94774.html

†Jerusalem School of Business Administration, Hebrew University, Jerusalem, Israel, and Department of Civil and Environmental Engineering, Massachusetts Institute of Technology, Cambridge, MA 02139 (halman@huji.ac.il). Partial support for this author's research was provided by EU FP7/2007–2013 grant 247757 and the Recanati Fund of the Jerusalem School of Business Administration.

‡Engineering Systems and Design, Singapore University of Technology and Design, Singapore (nannicini@sutd.edu.sg). Partial support for this author's research was provided by SUTD grant SRES11012 and IDC grants IDSF1200108 and IDG21300102.

§Sloan School of Management, Massachusetts Institute of Technology, Cambridge, MA 02139 (jorlin@mit.edu). Partial support for this author's research was provided by ONR grant N000141410073.

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. The random variables are assumed independent but not necessarily identically distributed. A relaxation of this assumption to handle the case of finite-horizon Markov decision processes (MDPs) is discussed later in the paper. 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. A discount factor α is sometimes introduced to weigh the "value of money" in the future, yielding the expression $\sum_{t=1}^{T} \alpha^t g_t(I_t, x_t, D_t) + \alpha^{T+1} g_{T+1}(I_{T+1})$ for the total cost. This variant can be taken into account within the framework discussed in this paper by suitably modifying the g_t functions. 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 DP. Formally, we want to determine

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

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 (see [3]). *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 the recursion:*

$$z_t(I_t) = \begin{cases} g_{T+1}(I_{T+1}) & \text{if } t = T+1, \\ \min_{x_t \in \mathcal{A}_t(I_t)} E_{D_t}\left[g_t(I_t, x_t, D_t) + z_{t+1}(f(I_t, x_t, D_t)) \right] & \text{if } t = 1,\ldots,T. \end{cases}$$

Assuming that $|\mathcal{A}_t(I_t)| = |\mathcal{A}|$ and $|\mathcal{S}_t| = |\mathcal{S}|$ for every t and $I_t \in \mathcal{S}_t$, this gives a pseudopolynomial algorithm that runs in time $O(T|\mathcal{A}||\mathcal{S}|)$.

A *fully polynomial-time approximation scheme* (FPTAS) is an approximation algorithm that, for every given $\epsilon > 0$, returns a solution with relative error at most ϵ in polynomial time in the size of the instance and $1/\epsilon$. The work [19] (an earlier version appeared in [18]) gives an FPTAS for three classes of problems that fall into this framework. This FPTAS is not problem-specific but relies solely on structural properties of the DP, and it computes an ϵ-approximation of the optimal value function. The three classes of [19] are convex DP, nondecreasing DP, and nonincreasing DP. In this paper we propose a modification of the FPTAS for the convex DP case that achieves better running time and is practically viable. Several applications of convex DPs are the resource management problems discussed in [31]. Two examples are the following:

1. Stochastic single-item inventory control [17, 20]. We want to find replenishment quantities in each time period for a warehouse managing a single item, to minimize expected procurement and holding/backlogging costs while facing uncertain demand. This is a classic problem in supply chain management.
2. Cash management [14, 32]. we want to manage the cash flow of a mutual fund taking into account uncertainty in the amount of money deposited or withdrawn from the fund in each time period. At the beginning of each stage we can buy or sell stocks, thereby changing the cash balance. At the end of each stage the net value of deposits/withdrawals is revealed. If the

cash balance of the mutual fund is positive, we incur some opportunity cost because the money could have been invested somewhere. If the cash balance is negative, we must borrow money from the bank at some cost.

Assuming convex cost functions, these problems fall under the convex DP case.

Our modification of the FPTAS is designed to improve the theoretical worst-case running time while making the algorithm a computationally useful tool. We show that our algorithm has excellent performance on randomly generated problem instances of the two applications described above: it is faster than an exact algorithm even on small instances where no large numbers are involved and for low approximation factors (0.1%), becoming orders of magnitude faster on larger instances. The algorithm is significantly faster than the original FPTAS of [19]. Furthermore, it compares favorably to a provably convergent approach based on a hybrid value-policy iteration scheme (see, e.g., [34]). We show that the running time of the FPTAS is in practice linear in the support of the random variables involved in the problem and in the number of states of the underlying MDP; thus our method is particularly effective if the DP has a small uncertainty set. The three *curses of dimensionality* of DPs described by [34] are the sizes of the state space, the action space, and the outcome space. Our method deals effectively with the first two for a large class of problems with special structure. While the FPTAS currently has limitations, in particular because multivariate convex functions cannot be efficiently approximated in general [19], we show that it works well under some practically relevant scenarios, and we discuss how these limitations could be overcome in the future. We believe that the proposed method could be successfully used as a subroutine within other approximate DP approaches. This claim is supported by [6], which reports good experimental results using one of the routines proposed in the present paper to implement an approximation algorithm for a stochastic DP with multidimensional state space [7]. Being an FPTAS, our method has the attractive feature of a single input parameter ϵ that provides an a priori guarantee on the relative approximation error and an upper bound on the running time, allowing the user to adjust this trade-off. To the best of our knowledge, this is the first time that a framework for the automatic generation of FPTAS is shown to be a practically as well as a theoretically useful tool. The only computational evaluation with positive results of an FPTAS we could find in the literature is [2], which tackles a specific problem (multiobjective 0–1 knapsack), whereas our algorithm addresses a whole class of problems, without explicit knowledge of problem-specific features. We believe that this paper is a first step in showing that FPTAS do not have to be looked at as a purely theoretical tool. Another novelty of our approach is that the algorithm design allows bounding the errors introduced by floating point computations, so that we can guarantee the specified approximation factor with respect to the optimal infinite-precision solution under reasonable assumptions. The implementation described in this paper is open-source and available from the second author.

The rest of this paper is organized as follows. In section 2, we formally introduce our notation and the original FPTAS of [19], discussing its applicability and limitations. In section 3, we improve the worst-case running time of the algorithm. In section 4, we discuss our implementation of the FPTAS. In section 5, we analyze the applicability of the proposed method on problems with vector (as opposed to scalar) states and actions. Section 6 contains an experimental evaluation of the proposed method, while section 7 concludes the paper with final remarks and future research perspectives. In the rest of this section we provide a review of existing literature.

1.1. Literature review. Stochastic dynamic programming is a natural paradigm to model and solve an enormous number of real-world problems, and therefore literature on the topic is abundant. Comprehensive references are [4, 34], which also point to many fundamental works in this area.

While there are many possible approximate solution approaches that perform well in practice, a common shortcoming is a lack of bounds on the approximation error or on the running time. There are of course notable exceptions, typically based on computing value function approximations as a linear combination of a suitable set of basis functions. The work [10] is a seminal paper that proposes an approximate linear programming (LP) approach with error bounds on the value function approximation; see also follow-up work and extensions in the same spirit such as [11, 12]. These papers discuss infinite-horizon discounted DPs and provide a bound on the approximation error as compared to the best value function approximation that can be obtained given the choice of basis functions.

A fundamentally different but commonly used methodology is the least squares Monte Carlo approach [5, 26, 39] stemming from applications in finance, in particular pricing of American options. This problem is naturally cast as a finite-horizon discounted DP because real-world financial contracts almost invariably prescribe finite expiration times. The derivation of error bounds when using least squares Monte Carlo approaches has been investigated in [13, 30]. These error bounds are similar in nature to the ones for approximate LP approaches, that is, they provide guarantees on the error with respect to some best value function approximation given the basis functions.

A third, related stream of work is that on the successive projective approximation routine (SPAR) that exploits concavity (for a maximization problem) of the optimal value function [36]. This approach builds a piecewise linear approximation of the value function that maintains concavity, and it is provably convergent for certain classes of finite-horizon problems with special structure [32, 33]. The concavity property exploited by SPAR is strictly related to the convexity properties required in the framework presented in this paper, and in fact the resource management problems of [31] to which SPAR is applied also fit in our framework. While SPAR converges to the optimal value function, it does not offer error bounds in terms of the number of sample paths of the simulations or in connection to the running time.

The approaches discussed above are more general than the FPTAS discussed in this paper, but there are fundamental differences in the type of error bounds provided. Indeed, the literature on approximate LP and least squares Monte Carlo is concerned with finding a good approximation using a set of prespecified basis functions. If the basis functions do not capture the structure of the optimal value function, then the resulting approximation can be poor. The burden of choosing appropriate basis functions rests on the user. Moreover, the approximation guarantees are only given a posteriori. The FPTAS discussed in this paper requires DPs with a certain structure, but it does not require a choice of basis functions, and it provides an a priori approximation guarantee with respect to the optimal solution of the problem. For this reason, the interpretation of the error bound is more intuitive, and the FPTAS requires a single input parameter to determine the approximation guarantee.

2. Preliminaries. In this section we formally introduce our notation and give the necessary definitions, discussing the type of problems that can be handled by the algorithm we propose. We then review the FPTAS as presented in [19], which constitutes the starting point for our method. We remark that this paper deals with

convex functions over finite sets, which are analyzed differently from convex functions over continuous domains [29].

2.1. Definitions. Let $\mathbb{N}$, $\mathbb{Z}$, $\mathbb{Q}$, $\mathbb{R}$ be the sets of nonnegative integers, integers, rational numbers, and real numbers, respectively. For $\ell, u \in \mathbb{Z}$, we call any set of the form $\{\ell, \ell+1, \dots, u\}$ a *contiguous* interval. We denote a contiguous interval by $[\ell, \dots, u]$, whereas $[\ell, u]$ denotes a real interval. Given $D \subset \mathbb{R}$ and $\varphi : D \to \mathbb{R}$ such that φ is not identically zero, we denote $D^{\min} := \min\{x \in D\}$, $D^{\max} := \max\{x \in D\}$, $\varphi^{\min} := \min_{x \in D}\{|\varphi(x)| : \varphi(x) \neq 0\}$, and $\varphi^{\max} := \max_{x \in D}\{|\varphi(x)|\}$. Given a finite set $D \subset \mathbb{R}$ and $x \in [D^{\min}, D^{\max}]$, for $x < D^{\max}$ let $\text{next}(x, D) := \min\{y \in D : y > x\}$, and for $x > D^{\min}$ let $\text{prev}(x, D) := \max\{y \in D : y < x\}$. Given a function defined over a finite set $\varphi : D \to \mathbb{R}$, we define $\sigma_\varphi(x) := \frac{\varphi(\text{next}(x,D)) - \varphi(x)}{\text{next}(x,D) - x}$ as the slope of φ at x for any $x \in D \setminus \{D^{\max}\}$, $\sigma_\varphi(D^{\max}) := \sigma_\varphi(\text{prev}(D^{\max}, D))$. Given a set X and a set $Y(x) \quad \forall x \in X$, define $X \otimes Y := \bigcup_{x \in X}(\{x\}, Y(x))$. Let $\mathcal{S}_{T+1}$ and $\mathcal{S}_t$ be contiguous intervals for $t = 1, \dots, T$. For $t = 1, \dots, T$ and $I_t \in \mathcal{S}_t$, let $\mathcal{A}_t$ and $\mathcal{A}_t(I_t) \subseteq \mathcal{A}_t$ be contiguous intervals. For $t = 1, \dots, T$ let $\mathcal{D}_t \subset \mathbb{Q}$ be a finite set, and let $g_t : \mathcal{S}_t \otimes \mathcal{A}_t \times \mathcal{D}_t \to \mathbb{Q}^+$ and $f_t : \mathcal{S}_t \otimes \mathcal{A}_t \times \mathcal{D}_t \to \mathcal{S}_{t+1}$. Finally, let $g_{T+1} : \mathcal{S}_{T+1} \to \mathbb{Q}^+$.

In this paper we deal with a class of problems labeled "convex DP" for which an FPTAS is given by [19]; [19] additionally defines two classes of monotone DPs, but in this paper we address the convex DP case only. The definition of a convex DP requires the notion of an integrally convex set; see [29].

DEFINITION 2.1. *Let X be a contiguous interval and $Y(x)$ a nonempty contiguous interval $\forall x \in X$. The set $X \otimes Y \subset \mathbb{Z}^2$ is* integrally convex *if there exists a polyhedron C_{XY} such that $X \otimes Y = C_{XY} \cap \mathbb{Z}^2$, and the slope of the edges of C_{XY} is an integer multiple of $45°$.*

DEFINITION 2.2 (see [19]). *A DP is a* convex DP *if the terminal state space $\mathcal{S}_{T+1}$ is a contiguous interval. For all $t = 1, \dots, T+1$ and $I_t \in \mathcal{S}_t$, the state space $\mathcal{S}_t$ and the action space $\mathcal{A}_t(I_t)$ are contiguous intervals. g_{T+1} is a nonnegative convex function over $\mathcal{S}_{T+1}$. For every $t = 1, \dots, T$, the set $\mathcal{S}_t \otimes \mathcal{A}_t$ is integrally convex, function g_t can be expressed as $g_t(I, x, d) = g_t^I(I, d) + g_t^x(x, d) + u_t(f_t(I, x, d))$, and function f_t can be expressed as $f_t(I, x, d) = a(d)I + b(d)x + c(d)$, where $g_t^I(\cdot, d), g_t^x(\cdot, d), u_t(\cdot)$ are univariate nonnegative convex functions, $a(d) \in \mathbb{Z}, b(d) \in \{-1, 0, 1\}$, and $c(d) \in \mathbb{Z}$.*

Let $U_{\mathcal{S}} := \max_{t=1,\dots,T+1} |\mathcal{S}_t|$, $U_{\mathcal{A}} := \max_{t=1,\dots,T, I_t \in \mathcal{S}_t} |\mathcal{A}_t(I_t)|$, and $U_g := \frac{\max_{t=1,\dots,T+1} g_t^{\max}}{\min_{t=1,\dots,T+1} g_t^{\min}}$. Given $\varphi : D \to \mathbb{R}$, let $\sigma_\varphi^{\max} := \max_{x \in D}\{|\sigma_\varphi(x)|\}$ and $\sigma_\varphi^{\min} := \min_{x \in D}\{|\sigma_\varphi(x)| : |\sigma_\varphi(x)| > 0\}$. For $t = 1, \dots, T$, we define $\sigma_{g_t}^{\max} := \max_{x_t \in \mathcal{A}_t, d_t \in D_t} \sigma^{\max}_{g_t(\cdot, x_t, d_t)}$, and $\sigma_{g_t}^{\min} := \min_{x_t \in \mathcal{A}_t, d_t \in D_t} \sigma^{\min}_{g_t(\cdot, x_t, d_t)}$. Let $U_\sigma := \frac{\max_{t=1,\dots,T+1} \sigma_{g_t}^{\max}}{\min_{t=1,\dots,T+1} \sigma_{g_t}^{\min}}$. We require that $\log U_{\mathcal{S}}$, $\log U_{\mathcal{A}}$, and $\log U_g$ are polynomially bounded in the input size. This implies that $\log U_\sigma$ is polynomially bounded.

Under these conditions, it is shown by [19] that a convex DP admits an FPTAS, using a framework that we review later in this section. The input data of a DP problem consists of the number of time periods T, the initial state I_1, an oracle that evaluates g_{T+1}, oracles that evaluate the functions g_t and f_t for each time period $t = 1, \dots, T$, and the discrete random variable D_t. For each D_t we are given n_t, the number of different values it admits with positive probability, and its support $\mathcal{D}_t = \{d_{t,1}, \dots, d_{t,n_t}\}$, where $d_{t,i} < d_{t,j}$ for $i < j$. Moreover, we are given positive integers $q_{t,1}, \dots, q_{t,n_t}$ such that $P[D_t = d_{t,i}] = \frac{q_{t,i}}{\sum_{j=1}^{n_t} q_{t,j}}$. For every $t = 1, \dots, T$ and $i = 1, \dots, n_t$, we define the following values:

$p_{t,i} := P[D_t = d_{t,i}]$: probability that D_t takes value $d_{t,i}$;
$n^* := \max_t n_t$: maximum number of different values that D_t can take.

For any function $\varphi : D \to \mathbb{R}$, t_φ denotes an upper bound on the time needed to evaluate φ.

2.2. Limitations of convex DPs and extensions. We now discuss the restrictions imposed by Definition 2.2.

The condition $b(d) \in \{-1, 0, 1\}$ imposes a specific structure on the stochastic DPs we can model, namely, one where state transitions do not depend on the action, or the action allows reaching a set of contiguous states. The work [19] shows that a convex DP where $b(d) \notin \{-1, 0, 1\}$ in Definition 2.2 does not necessarily admit an FPTAS unless P = NP.

The condition that $\mathcal{S}_t$ and $\mathcal{A}_t(I_t)$ are contiguous intervals $\forall t$ allows us to model only stochastic DPs where the state and action spaces are unidimensional. We can relax this restriction in some special cases that are discussed in section 5; see also the discussion in [19, sect. 10.3]. One such case with practical importance is that of an MDP where the one-step transition matrix is given explicitly. We study this case experimentally in section 6.6.

The condition that the random variables D_t are independent cannot be relaxed in general unless P = NP; see [19, sect. 10.3]. However, some types of dependency such as MDPs can be handled. In this paper we mostly deal with independent random variables until section 6.6.

The limitation of explicitly listing the support of the random variables as a problem input is quite severe in practice and does not allow us to effectively handle problems with a very rich uncertainty model, e.g., problems for which computing expectations is impractical. The work [22] proposes an approach to deal with the more general case where we only have an oracle for the cumulative distribution function of D_t instead of explicit knowledge of the probability distribution function. This is not straightforward with the modifications to the framework proposed in the present paper, hence we keep the more restrictive assumption for now.

We note that the framework of this paper can be applied heuristically in a more general setting. We provide two notable examples. If the state and action spaces are continuous intervals, we can compute an approximation of the value function with a relatively straightforward adaptation of the routines. If the cost function is not convex, one can still adapt the algorithm to treat it as if it were convex. Doing so forsakes any guarantee on the approximation factor or the running time. In this paper, we discuss only the cases for which such guarantees hold.

2.3. The algorithm. The basic idea underlying the FPTAS of Halman et al. [19] is to approximate the functions involved in the DP by keeping only a logarithmic number of points in their domain. We then use a step function or linear interpolation to determine the function value at points that have been eliminated from the domain.

DEFINITION 2.3 (see [19]). *Let $K \geq 1$ and let $\varphi : D \to \mathbb{R}^+$, where $D \subset \mathbb{R}$ is a finite set. We say that $\tilde{\varphi} : D \to \mathbb{R}^+$ is a* K-approximation function *of φ (or more briefly, a K-approximation of φ) if $\forall x \in D$ we have $\varphi(x) \leq \tilde{\varphi}(x) \leq K\varphi(x)$.*

DEFINITION 2.4 (see [19]). *Let $K \geq 1$ and let $\varphi : D \to \mathbb{R}^+$, where $D \subset \mathbb{R}$ is a finite set, be a unimodal function. We say that $W \subseteq D$ is a* K-approximation set of *φ if the following three properties are satisfied:* (i) $D^{\min}, D^{\max} \in W$. (ii) *For every $x \in W \setminus \{D^{\max}\}$, either $next(x, W) = next(x, D)$ or $\max\{\varphi(x), \varphi(next(x, W))\} \leq K \min\{\varphi(x), \varphi(next(x, W))\}$.* (iii) *For every $x \in D \setminus W$, we have $\varphi(x) \leq \max\{\varphi(prev(x, W)), \varphi(next(x, W))\} \leq K\varphi(x)$.*

An algorithm to construct K-approximations of functions with special structure (namely, convex or monotone) in polylogarithmic time was first introduced in [20]. In

ALGORITHM 1. APXSET(φ, D, K).

$x \leftarrow D^{\max}$
$W \leftarrow \{D^{\min}, D^{\max}\}$
while $x > D^{\min}$ **do**
  $x \leftarrow \min\{\text{prev}(x, D), \min\{y \in D : K\varphi(y) \geq \varphi(x)\}\}$
  $W \leftarrow W \cup \{x\}$
return W

this paper we only deal with the convex case; therefore when presenting results from [19, 20] we try to avoid the complications of the two monotone cases. In the rest of this paper it is assumed that the conditions of Definition 2.2 are met.

DEFINITION 2.5 (see [19]). *Let $\varphi : D \to \mathbb{R}$. For all $E \subseteq D$, the* convex extension of φ induced by E *is the function $\hat{\varphi}$ defined as the lower envelope of the convex hull of $\{(x, \varphi(x)) : x \in E\}$.*

The main building block of the FPTAS is the routine APXSET given in Algorithm 1, which computes a K-approximation set of a function φ over the domain D. The idea is to only keep points in D such that the function value "jumps" by less than a factor K between adjacent points. For brevity, in the rest of this paper the algorithms to compute K-approximation sets are presented for convex nondecreasing functions. They can all be extended to general convex functions by applying the algorithm to the right and to the left of the minimum, which can be found in $O(\log |D|)$ time by binary search. Hence, theorems are presented for the general case.

THEOREM 2.6 (see [19]). *Let $\varphi : D \to \mathbb{R}^+$ be a convex function over a finite domain of real numbers. Then for every $K > 1$ the following holds:* (i) APXSET *computes a K-approximation set W of cardinality $O(1 + \log_K \frac{\varphi^{\max}}{\varphi^{\min}})$ in $O(t_\varphi(1 + \log_K \frac{\varphi^{\max}}{\varphi^{\min}}) \log |D|)$ time.* (ii) *The convex extension $\hat{\varphi}$ of φ induced by W is a convex K-approximation of φ whose value at any point in D can be determined in $O(\log |W|)$ time for any $x \in [D^{\min}, D^{\max}]$ if W is stored in a sorted array $(x, \varphi(x)), x \in W$.*

PROPOSITION 2.7 (see [19]). *Let $1 \leq K_1 \leq K_2, 1 \leq t \leq T, I_t \in \mathcal{S}_t$, be fixed. Let $\hat{g}_t(I_t, \cdot, d_{t,i})$ be a convex K_1-approximation of $g_t(I_t, \cdot, d_{t,i})$ for every $i = 1, \ldots, n_t$. Let $\hat{z}_{t+1}$ be a convex K_2-approximation of z_{t+1}, and define $\hat{G}_t(I_t, \cdot) := E_{D_t}[\hat{g}_t(I_t, \cdot, D_t)]$, $\hat{Z}_{t+1}(I_t, \cdot) := E_{D_t}[\hat{z}_{t+1}(f_t(I_t, \cdot, D_t))]$. Then*

$$\bar{z}_t(I_t) := \min_{x_t \in \mathcal{A}(I_t)} \left\{ \hat{G}_t(I_t, x_t) + \hat{Z}_{t+1}(I_t, x_t) \right\} \tag{2.1}$$

is a K_2-approximation of z_t that can be computed in $O(\log(|A_t(I_t)|) n_t (t_{\hat{g}} + t_{\hat{z}_t} + t_{f_t}))$ time for each value of I_t.

We can now describe the FPTAS for convex DPs given by [19]. The algorithm is given in Algorithm 2. It is shown in [19] that $z_t, \bar{z}_t$ are convex for every t.

THEOREM 2.8 (see [19]). *Given $0 < \epsilon < 1$, for every initial state I_1, $\hat{z}_1(I_1)$ is a $(1 + \epsilon)$-approximation of the optimal cost $z^*(I_1)$. Moreover, Algorithm 2 runs in $O((t_g + t_f + \log(\frac{T}{\epsilon} \log(TU_g))) \frac{n^* T^2}{\epsilon} \log(TU_g) \log U_{\mathcal{S}} \log U_{\mathcal{A}})$ time, which is polynomial in $1/\epsilon$ and the input size.*

3. Improved running time. In this section we show that for the convex DP case, we can improve the running time given in Theorem 2.8.

In the framework of [19], monotone functions are approximated by a step function, and Definition 2.4 guarantees the K-approximation property for this case. However, APXSET greatly overestimates the error committed by the convex extension induced

ALGORITHM 2. ORIGINAL FPTAS FOR CONVEX DP.

$K \leftarrow 1 + \frac{\epsilon}{2(T+1)}$
$W_{T+1} \leftarrow \text{APXSET}(g_{T+1}, \mathcal{S}_{T+1}, K)$
Let $\hat{z}_{T+1}$ be the convex extension of g_{T+1} induced by W_{T+1}
for $t = T, \ldots, 1$ **do**
  Define $\bar{z}_t$ as in (2.1) with $\hat{g}_t$ set equal to g_t
  $W_t \leftarrow \text{APXSET}(\bar{z}_t, \mathcal{S}_t, K)$
  Let $\hat{z}_t$ be the convex extension of $\bar{z}_t$ induced by W_t
return $\hat{z}_1(I_1)$

ALGORITHM 3. APXSETSLOPE(φ, D, K).

$x \leftarrow D^{\min}$
$W \leftarrow \{D^{\min}\}$
while $x < D^{\max}$ **do**
  $x \leftarrow \max\{\text{next}(x, D), \max\{y \in D : \varphi(y) \leq K(\varphi(x) + \sigma_\varphi(x)(y - x)\}\}$
  $W \leftarrow W \cup \{x\}$
return W

by W. For the convex DP case we propose the simpler Definition 3.1 of the K-approximation set, which preserves correctness of the FPTAS and the analysis carried out in [19].

DEFINITION 3.1. *Let $K \geq 1$ and let $\varphi : D \to \mathbb{R}^+$, where $D \subset \mathbb{R}$ is a finite set, be a convex function. Let $W \subseteq D$ and let $\hat{\varphi}$ be the convex extension of φ induced by W. We say that W is a* K-approximation set of φ *if* (i) $D^{\min}, D^{\max} \in W$; (ii) *for every* $x \in D$, $\hat{\varphi}(x) \leq K\varphi(x)$.

Note that a K-approximation set according to the new definition is not necessarily such under the original Definition 2.4 as given in [19]. For example, $D = \{0, 1, 2\}, \varphi(0) = 0, \varphi(1) = 1, \varphi(2) = 2K$; the only K-approximation set according to the original definition is D itself, whereas $\{0, 2\}$ is a (smaller) K-approximation set in the sense of Definition 3.1. An algorithm to compute a K-approximation set in the sense of Definition 3.1 is given in Algorithm 3 (for nondecreasing functions).

THEOREM 3.2. *Let $\varphi : D \to \mathbb{Q}^+$ be a convex function over a finite domain of integers. Then for every $K > 1$,* APXSETSLOPE(φ, D, K) *computes a K-approximation set W of size $O(\log_K \min\{\frac{\sigma_\varphi^{\max}}{\sigma_\varphi^{\min}}, \frac{\varphi^{\max}}{\varphi^{\min}}\})$ in $O(t_\varphi \log_K \min\{\frac{\sigma_\varphi^{\max}}{\sigma_\varphi^{\min}}, \frac{\varphi^{\max}}{\varphi^{\min}}\} \log |D|)$ time.*

The proof of Theorem 3.2 is given in Appendix C. As a consequence of this theorem, we can approximate linear or V-shaped piecewise linear functions very efficiently with an approximation set of constant size, because in this case $\sigma_\varphi^{\max}/\sigma_\varphi^{\min} = 1$. Note that in general if the cost functions grow as a polynomial of degree d, their slopes grow as a polynomial of degree $d - 1$; therefore an approximation algorithm based on slopes can yield a significant practical advantage. We can improve on Theorem 2.8 by replacing each call to APXSET with a call to APXSETSLOPE in Algorithm 2.

THEOREM 3.3. *Given $0 < \epsilon < 1$, for every initial state I_1, $\hat{z}_1(I_1)$ is a $(1 + \epsilon)$-approximation of the optimal cost $z^*(I_1)$. Moreover, Algorithm 2 runs in*

$$O\left(\left(t_g + t_f + \log\left(\frac{T}{\epsilon}\log(\min\{U_\sigma, TU_g\})\right)\right)\frac{n^* T^2}{\epsilon}\log(\min\{U_\sigma, TU_g\})\log U_{\mathcal{S}} \log U_{\mathcal{A}}\right)$$

time, which is polynomial in both $1/\epsilon$ and the (binary) input size.

4. From theory to practice. In this section we introduce an algorithm that builds on the idea of section 3 to compute smaller K-approximation sets than ApxSetSlope in practice, although we do not improve over Theorem 3.2 from a theoretical standpoint, and the analysis is more complex. Then, we discuss how to implement the FPTAS efficiently and how to handle floating point computations to guarantee the desired approximation factor despite rounding errors.

4.1. An improved routine for K-approximation sets. Given two points $(x,y),(x',y') \in \mathbb{R}^2$ we denote by $\textsc{Line}((x,y),(x',y'),\cdot) : \mathbb{R} \to \mathbb{R}$ the straight line through them. We first discuss how to exploit convexity of φ to compute a bound on the approximation error $\frac{\textsc{Line}((x,\varphi(x)),(x',\varphi(x')),w)}{\varphi(w)}$ $\forall w \in [x,\dots,x']$.

Proposition 4.1. *Let $\varphi : [\ell,u] \to \mathbb{R}^+$ be a nondecreasing convex function. Let $h \geq 3$, $E = \{x_i : x_i \in [\ell,u], i = 1,\dots,h\}$ with $\ell = x_1 < x_2 < \cdots < x_{h-1} < x_h = u$, let $y_i := \varphi(x_i) \forall i$, $(x_0,y_0) := (x_1 - 1, y_1)$ and $(x_{h+1},y_{h+1}) := (x_h + 1, 2y_h - f(x_h - 1))$. For all $i = 1,\dots,h-1$, define $LB_i(x)$ as*

$$LB_i(x) := \begin{cases} \max \begin{Bmatrix} \textsc{Line}((x_{i-1},y_{i-1}),(x_i,y_i),x), \\ \textsc{Line}((x_{i+1},y_{i+1}),(x_{i+2},y_{i+2}),x) \end{Bmatrix} & \text{if } x \in [x_i,x_{i+1}], \\ 0 & \text{otherwise.} \end{cases}$$

Define $\underline{\varphi}(x) := \sum_{i=1}^{h-1} LB_i(x)$. Observe that $LB_i(x)$ is the maximum of two linear functions, so it has at most one breakpoint over the interval (x_i,x_{i+1}). Let B be the set of these breakpoints. For $1 \leq j < k \leq h$ let

$$\gamma_E(x_j,x_k) := \max_{x_e \in [x_j,x_k] \cap (E \cup B)} \left\{ \frac{\textsc{Line}((x_j,y_j),(x_k,y_k),x_e)}{\underline{\varphi}(x_e)} \right\}. \tag{4.1}$$

Then

$$\left| \frac{\textsc{Line}((x_j,y_j),(x_k,y_k),w)}{\varphi(w)} \right| \leq \gamma_E(x_j,x_k) \leq \frac{y_k}{y_j} \ \forall w \in [x_j,x_k].$$

It is easy to show that if φ is defined only on integer points, then if $x_e \in B$ and $x_e \notin \mathbb{Z}$, we can replace x_e with the two points $\lfloor x_e \rfloor, \lceil x_e \rceil$ in B. This gives slightly tighter error bounds and, more importantly, it allows us to compute $\underline{\varphi}$ only at integer points, which is one of the assumptions in section 4.3.

The set B of Proposition 4.1 allows the computation of a bound $\gamma_E(x_j,x_k)$ on the error committed by approximating φ with a linear function between x_j, x_k. We use this bound in ApxSetConvex; see Algorithm 4 (for nondecreasing functions). In the description of ApxSetConvex, $\Lambda > 1$ is a given constant. We used $\Lambda = 2$ in our experiments.

The idea of the algorithm is to construct a good underestimator $\underline{\varphi}$ of φ using all points in the domain where the function value is known, following Proposition 4.1. This underestimator improves at every iteration and is used to build an approximation set with few points. One important difference of ApxSetConvex with respect to ApxSetSlope is the following: ApxSetSlope looks for the next point to be added to the approximation set using binary search, therefore the search interval is iteratively halved until it collapses to a single point; ApxSetConvex employs a modified binary search that may discard the search interval without adding points to the approximation set, if it can show—thanks to the progressively improving underestimator $\underline{\varphi}$—that no approximation point is necessary in that interval.

The running time of ApxSetConvex can be a factor $\log^2 \log |D|$ slower than ApxSetSlope, but its practical performance is superior for two reasons: it produces

ALGORITHM 4. APXSETCONVEX(φ, D, K).

1: $W \leftarrow \{D^{\min}, D^{\max}\}$
2: $x \leftarrow D^{\max}$
3: $E \leftarrow \{D^{\min}, \lfloor (D^{\min} + D^{\max})/2 \rfloor, D^{\max}\}$
4: **while** $x > D^{\min}$ **do**
5: $\quad \ell \leftarrow D^{\min}, r \leftarrow x, \text{counter} \leftarrow 0, z \leftarrow x$
6: $\quad$ **while** $r > \text{next}(\ell, D)$ **do**
7: $\quad\quad w \leftarrow \min\{y \in E : (y > D^{\min}) \text{ and } (\gamma_E(y, x) \leq K)\}$
8: $\quad\quad \ell \leftarrow \text{prev}(w, E), r \leftarrow w, \text{counter++}$
9: $\quad\quad E \leftarrow E \cup \{\lfloor (\ell + r)/2 \rfloor\} \cap [D^{\min}, \max\{\text{next}(r, E), \arg\gamma_E(r, x)\}]$
10: $\quad\quad$ **if** $\text{counter} > \Lambda \log(|D|)$ **then**
11: $\quad\quad\quad$ **if** $\varphi(z) > K\varphi(r)$ **then**
12: $\quad\quad\quad\quad \text{counter} \leftarrow 0, z \leftarrow r$
13: $\quad\quad\quad$ **else**
14: $\quad\quad\quad\quad \ell \leftarrow \text{prev}(r, D)$
15: $\quad x \leftarrow \min\{\text{prev}(x, D), r\}$
16: $\quad W \leftarrow W \cup \{x\}, E \leftarrow E \cap [D^{\min}, x]$
17: **return** W

smaller approximation sets and it evaluates $\bar{z}$ fewer times (each evaluation of $\bar{z}$ is expensive; see below). We remark that we experimented with applying Proposition 4.1 in conjunction with APXSETSLOPE, but this did not improve the algorithm's performance; therefore we omit the discussion for space reasons.

THEOREM 4.2. *Let $\varphi : D \to \mathbb{Q}^+$ be a convex function defined over a set of integers. Then* APXSETCONVEX(φ, D, K) *computes a K-approximation set of φ of size $O(\log_K \min\{\frac{\sigma_\varphi^{\max}}{\sigma_\varphi^{\min}}, \frac{\varphi^{\max}}{\varphi^{\min}}\})$ in time $O(t_\varphi \log_K \min\{\frac{\sigma_\varphi^{\max}}{\sigma_\varphi^{\min}}, \frac{\varphi^{\max}}{\varphi^{\min}}\} \log|D| \log^2 \log|D|)$.*

For space reasons we omit a full proof of Theorem 4.2, which is given in Appendix C. We provide a sketch of the main ideas. First, we show that we can safely ignore all points to the right of $\max\{\text{next}(r, E), \arg\gamma_E(r, x)\}$ on line 9. Then, we show that APXSETCONVEX computes an approximation set of size $O(\log_K \min\{\frac{\sigma_\varphi^{\max}}{\sigma_\varphi^{\min}}, \frac{\varphi^{\max}}{\varphi^{\min}}\})$, and line 7 is executed $O(\log|D|)$ times for each point added to the approximation set. The crucial part is bounding the running time of the loop that finds the subsequent point to be added to the approximation set. For this, we first show that $|E| = O(\log|D|)$ throughout the algorithm and that line 7 can be executed in $O(\log^2 \log|D|)$ time by performing binary search separately on the sets E and B of Proposition 4.1. This running time relies on skip lists [37] that allow for $O(\log n)$ insertion, deletion and search on a list with n elements. The worst-case running times for operations on skip lists are "with high probability" in the original paper [37], but [28] describes an implementation that achieves the same bounds in deterministic time.

4.2. Efficient implementation. We see from Algorithm 2 that at each iteration of the main loop we have to construct an approximation $\hat{z}_t$ of the value function $\bar{z}_t$ at the current stage t. By (2.1), each evaluation of $\bar{z}_t$ requires the minimization of a convex function. Hence, evaluating $\bar{z}_t$ is expensive. We briefly discuss our approach to perform the computation of a K-approximation set of $\bar{z}$ efficiently, which is crucial because such computation is carried out T times in the FPTAS.

First, we make sure that the minimization in the definition (2.1) of $\bar{z}_t$ is performed at most once for each state I_t. We use a dictionary to store function values of $\bar{z}_t$ at

all previously evaluated points I_t. Whenever we require an evaluation of $\bar{z}_t$ at I_t, we first look up if I_t is present in the dictionary ($O(1)$ time on average), and if so, we avoid performing the minimization in (2.1).

Second, we do not approximate g_t when computing (2.1), i.e., we use $K_1 = 1$ in Proposition 2.7, as also suggested in the proof of Theorem 9.3 of [19]. g_t is typically available as an $O(1)$ time oracle: setting $K_1 = 1$ avoids several time-consuming steps.

Third, whenever we compute the minimum of a convex function such that each function evaluation requires more than $O(1)$ time, we use golden section search [25], which saves $\approx 30\%$ function evaluations compared to binary search. In particular, this is done at each evaluation of $\bar{z}_t$ in Algorithm 2, and each application of ApxSet routines, which require finding the minimum of a function. Note that our implementation of an exact DP approach also uses golden section search to evaluate the optimal value function at each stage depending on the subsequent stage. This exploits convexity over the action space.

Fourth, in our Python implemention we use standard lists instead of skip lists to store E and B in ApxSetConvex as discussed in section 4.1. Note that Python lists are essentially dynamically allocated arrays with $O(1)$ access time and $O(n)$ insertion, rather than linked lists. In our tests, E and B typically contain less than 20 elements, and we found Python lists to be more efficient in this scenario. In this case the $O(\log^2 \log |D|)$ factor in the running time of Theorem 4.2 increases to $O(\log |D|)$.

4.3. Floating point arithmetics. Our implementation uses floating point arithmetics for the sake of speed. Most modern platforms provide both floating point (fixed precision) arithmetics and rational (arbitrary precision) arithmetics. The latter is considerably slower because it is not implemented in hardware, but it does not incur into numerical errors and can handle arbitrarily large numbers. Only by using arbitrary precision can one guarantee to find the optimal solution of a problem instance: fixed precision computations suffer from accumulated errors at every arithmetic operation, and the errors may compound. Approaches to deal with this type of error are typically not discussed in the dynamic programming literature. However, they have been studied in other branches of optimization; see, for example, [1, 8, 9, 40].

In this section we describe how our floating point implementation guarantees that the final result satisfies the desired approximation guarantee. We make the following assumptions: (i) All the functions involved in the problem (including the linear interpolation between function points) are evaluated at integer points only. (ii) All integer numbers appearing in the problem are contained in $[-2^{53}, \dots, 2^{53}]$, and all integers in this interval can be represented exactly as floating point numbers. (iii) Each addition, multiplication, or division introduces at most an additional ϵ_m relative error, where ϵ_m is the machine epsilon. (iv) $\epsilon_m \ll 1$ so that any term that is order of ϵ_m^2 can be considered zero.

Assumption (i) is verified by our algorithms. Assumptions (ii) and (iii) hold for standard double precision floating point arithmetics [23]. Note that large rounding errors in this context are mostly due to subtraction, which may produce catastrophic cancellation if carelessly performed.

We now proceed to bound the error introduced by the calculations required by the FPTAS. An iteration of Algorithm 2 comprises three key operations: the computation of the value function $\bar{z}_t$ at the current stage (see (2.1)), the computation of a K-approximation set for $\bar{z}_t$, and the evaluation of the convex extension $\hat{z}_t$. At each stage, the relative errors introduced by the floating point operations is on top of the approximation error inherited from the subsequent stage.

By employing Kahan's summation algorithm [24], one can show that computing $\bar{z}_t$ (2.1) at a point introduces at most $5\epsilon_m$ relative error. The error introduced by computing the linear interpolation of φ between x_1, x_2 at point x is bounded by $4\epsilon_m$. This is because the differences occurring in the formula $\frac{\varphi(x_1)(x_2-x)+\varphi(x_2)(x-x_1)}{x_2-x_1}$ are among integer values, hence they can be computed exactly. Finally, the error introduced by the computation of the K-approximation set is bounded by $9\epsilon_m$ because it requires the multiplication of two values obtained by linear interpolation.

Putting everything together, when using floating point arithmetic we can achieve an approximation factor of K with a call to ApxSetConvex(φ, D, K) by replacing K with $K/(1 + 18\epsilon_m)$ on line 8 of ApxSetConvex. With this modification, if $\hat{z}_{t+1}$ is a K_1 approximation of z_{t+1} and K_2 is the approximation factor used by ApxSetConvex at stage t, then $\hat{z}_t$ is a guaranteed K_1K_2 approximation of z_t.

Another observation is in order. We assumed that the functions and the linear interpolations are always evaluated at integer points. To verify this assumption, we round the points computed following Proposition 4.1 up and down as suggested in section 4.1. However, because the computations are subject to error, if the x-axis coordinate of the point to be rounded is very close to an integer, it may not be clear how to round the point up and down. Note that care has to be taken in this respect because otherwise the computation of the error bounds may be affected by mistakes.

Proposition 4.1 requires the computation of the intersection between two lines, each of which is defined by two points, say, $(x_1, y_1), (x_2, y_2)$ and $(x_3, y_3), (x_4, y_4)$. The x-axis coordinate of the intersection is given by

$$\frac{(y_4(x_2 - x_1)(x_3 - x_4) + y_2x_1(x_3 - x_4)) - (y_1(x_2 - x_1)(x_3 - x_4) + y_1x_1(x_3 - x_4))}{(y_2(x_3 - x_4) + y_4(x_2 - x_1)) - (y_1(x_3 - x_4) + y_3(x_2 - x_1))}. \tag{4.2}$$

Because of the subtraction at the numerator and denominator, we cannot bound the relative error introduced by this computation independently of the operands. We can rewrite the expression above as $\frac{a-b}{c-d}$, where each number is positive and affected by a relative error of at most $5\epsilon_m$. To bound the error of (4.2), we impose an upper limit ρ on the condition number of the sums $a - b$ and $c - d$. (Recall that the condition number of a sum $x + y$ is $\frac{|x|+|y|}{|x+y|}$.) After preliminary computational testing, we chose $\rho := \sqrt{1/\epsilon_m}$. If the condition number of the operands does not satisfy the limit, we abort the computation of γ_E and compute a more conservative error bound with smaller numerical error: $y_3/\text{Line}((x_1, y_1), (x_2, y_2), x_3 - 1)$ for a nondecreasing function. If the limit is satisfied, it can be shown that the relative error of (4.2) is bounded by $11\sqrt{\epsilon_m}$. Then we do the following. Let x_e be the value of (4.2), and let $I = [x_e/(1 + 11\sqrt{\epsilon_m}), x_e(1 + 11\sqrt{\epsilon_m})]$. If $x_e \in \mathbb{Z}$, that is the only point we need to check. If $I \cap \mathbb{Z} = \emptyset$, we can safely round x_e up and down and we proceed as usual. If $I \cap \mathbb{Z} = \lfloor x_e \rfloor$, the actual intersection point could be $< \lfloor x_e \rfloor$, hence we check the error at $\lfloor x_e \rfloor - 1, \lfloor x_e \rfloor, \lceil x_e \rceil$. This guarantees correctness of the error bounds given by γ_E. Similarly, if $I \cap \mathbb{Z} = \lceil x_e \rceil$ we check the error at $\lfloor x_e \rfloor, \lceil x_e \rceil, \lceil x_e \rceil + 1$. Finally, if $|I \cap \mathbb{Z}| > 1$ we again resort to the conservative bound.

To ensure that we do not exceed the approximation factor allowed, we set the rounding mode for the floating point unit to the appropriate direction for all key calculations. In particular, we round up (i.e., we overestimate) the errors committed by the algorithm, and we round down (i.e., we underestimate) the maximum allowed approximation factor at each iteration. The computational overhead for these safe computations as opposed to potentially unsafe ones is negligible.

ALGORITHM 5. IMPLEMENTATION OF THE FPTAS FOR CONVEX DP.

1: $K \leftarrow (1+\epsilon)^{\frac{1}{T+1}}$
2: $W_{T+1} \leftarrow \text{APXSETCONVEX}(g_{T+1}, \mathcal{S}_{T+1}, K)$
3: Let K'_{T+1} be the maximum value of γ_E recorded by APXSETCONVEX
4: Let $\hat{z}_{T+1}$ be the convex extension of g_{T+1} induced by W_{T+1}
5: **for** $t = T, \ldots, 1$ **do**
6: Define $\bar{z}_t$ as in (2.1) with $\hat{g}_t$ set equal to g_t
7: $K_t \leftarrow \frac{K^{T+2-t}}{\prod_{j=t+1}^{T+1} K'_j}$.
8: $W_t \leftarrow \text{APXSETCONVEX}(\bar{z}_t, \mathcal{S}_t, K_t)$
9: Let K'_t be the maximum value of γ_E recorded by APXSETCONVEX
10: Let $\hat{z}_t$ be the convex extension of $\bar{z}_t$ induced by W_t
11: **return** $\hat{z}_1(I_1)$

4.4. Adaptive selection strategy of the approximation factor. At each stage of the DP, we adaptively compute an approximation factor K_t that guarantees the approximation factor $(1+\epsilon)$ for the value function at stage 1 taking into account the errors in the floating point computations.

We now describe the selection of the approximation factor at each stage in more detail. If K_t is the approximation factor used at stage $t = 1, \ldots, T+1$, then the FPTAS returns a solution within $\prod_{t=1}^{T+1} K_t$ of optimality (recall that the error introduced by the floating point computations is taken into account by replacing K with $K/(1+18\epsilon_m)$ in APXSETCONVEX). Hence we want $\prod_{t=1}^{T+1} K_t \le 1+\epsilon$. The work [19] suggests using $K_t = 1 + \frac{\epsilon}{2(T+1)}$ at each stage, because of the inequality $(1+x/n)^n \le 1+2x$ valid for $0 \le x \le 1$. However, this bound is very loose for small n.

We used the following strategy. Let $\alpha = (1+\epsilon)^{\frac{1}{T+1}}$. At each stage, we keep track of the maximum approximation error K'_t recorded during the application of APXSETCONVEX. This corresponds to the largest recorded value of γ_E and is an a posteriori guarantee on the actual approximation factor achieved. When APXSET or APXSETSLOPE is employed to compute approximation sets, we use the largest recorded ratio of $\varphi(y)/\varphi(x)$ and $\varphi(y)/(\varphi(x)+\sigma_\varphi(x)(y-x))$, respectively, on line 4 of the two algorithms as K'_t. Note that $K'_t \le K_t$. Then for $t = T+1, \ldots, 1$ we set $K_t := \frac{\alpha^{T+2-t}}{\prod_{j=t+1}^{T+1} K'_j}$. The total approximation factor achieved at stage 1 is $\prod_{t=1}^{T+1} K'_t$, and we have

$$K_1 = \frac{\alpha^{T+1}}{\prod_{j=2}^{T+1} K'_j}, \quad \text{therefore} \quad \prod_{t=1}^{T+1} K'_t = K'_1 \prod_{j=2}^{T+1} K'_j = K'_1 \frac{\alpha^{T+1}}{K_1} \le \alpha^{T+1} = 1+\epsilon,$$

as desired. We summarize the FPTAS for convex DPs in Algorithm 5.

5. Handling multidimensional state or action spaces. The methodology discussed in this paper relies heavily on the properties of discrete convex functions. For an extensive treatment of this topic, we refer to [29]. A fairly broad class of discrete convex functions is that of Miller; see [27]. It is known [19, Thm. 10.10] that a multivariate discrete convex function in the sense of Miller does not necessarily admit a succinct representation, i.e., it may not be possible to compute a K-approximation that requires storing a polylogarithmic number of points in the domain of the function. Thus, we cannot approximate a multivariate optimal value function, even if it is Miller discrete convex. Another well-known class of discrete convex functions is that of $L^\natural$-convex functions [29], which is a natural candidate to study the applicability of our

method in a multivariate setting (see [7, Lem. 7, 8]). Unfortunately, we are not aware of any algorithm to compute K-approximation sets for multivariate $L^\natural$-convex functions "one dimension at a time"; hence it is possible that the FPTAS does not extend to this setting.

Despite these negative results, our approach can be extended to multivariate state and action in some special cases. We discuss some of them below.

An easy case is that in which the problem is separable: $\mathcal{S}_t \subset \mathbb{Z}^k, \mathcal{A}_t \subset \mathbb{Z}^k \quad \forall t$, the transition function $f_t(I, x, d) : \mathcal{S}_t \otimes \mathcal{A}_t \times \mathcal{D}_t \to \mathcal{S}_{t+1}$ can be expressed as $f_t(I, x, d) = (f_t^1(I_1, x_1, d), \ldots, f_t^k(I_k, x_k, d))$, and $g_t(I, x, d) = \sum_{i=1}^k g_t^i(I_i, x_i, d)$. In this case, the min and the expectation in (2.1) can be split into k separate problems, allowing the solution of the problem as k unidimensional DPs. Thus, the FPTAS can be applied provided the conditions of Definition 2.2 are satisfied by each g_t^i, f_t^i. This corresponds to the case in which the functions g_t are convex separable [29, (6.31)].

A more practically relevant case occurs when additional components of the state vector are bounded by U_B, where U_B is polynomial in the input size. Then, the problem admits an FPTAS by applying the proposed approximation scheme separately for each possible value of the additional state vector components. One typical example is that of an MDP (see, e.g., [34, Chap. 3]), where the one-step transition matrix is given explicitly. We provide a computational assessment of this case in section 6.6. If the transition probabilities are given implicitly, e.g., as an oracle, then this approach is no longer polynomial-time in general [19, sect. 10.3].

A further interesting case to which our method applies is given by the setting of [33], where the state is a scalar but the action is a vector, possibly very large, and the optimal action at a given state can be computed by solving an LP. A rigorous treatment of this case is outside the scope of this paper. Below, we skip all proofs but sketch the main ideas to show that the method discussed in this paper has potential beyond the scope of this work. Assume that the action space $\mathcal{A}_t(I_t)$ is given by $\{x_t : A_t x_t = b_t(I_t), x_t \le u_t(I_t), x_t \ge 0\}$, where A_t is a matrix, u_t is a vector of upper bounds, and b_t is a vector of right-hand-side values that are linear in the current state I_t. Assume further that the state transition can be expressed as $I_{t+1} = I_t + a^S x_t + b^S(D_t)$, $\binom{A_t}{a^S}$ is totally unimodular (TU), $u_t(I_t), b_t(I_t)$ are integer for all I_t, and the one-period cost function is linear in x_t and I_{t+1}, i.e., $g_t(I_t, x_t, D_t) = c^x x_t + c^I I_{t+1} + c(D_t)$. These assumptions are verified if the optimal action and state transition can be computed solving a network flow problem, which is the case in some resource management problems with a central storage (represented by the unidimensional state) employed to support satellite activities. Some examples are given in [33, 38]. Under these assumptions, a K-approximation of the optimal cost-to-go at stage t and state I_t can be computed by solving the following optimization problem:

$$
\left.
\begin{array}{lrcl}
\min & c^x x_t + \sum_{i=1}^{n_t} p_{t,i} \left(y_i + c^I I_{t+1,i} + c(d_{t,i}) \right) & & \\
 & A_t x_t & = & b_t(I_t) \\
\forall i = 1, \ldots, n_t & y_i - \hat{z}_{t+1}(I_{t+1,i}) & \ge & 0 \\
\forall i = 1, \ldots, n_t & I_{t+1,i} - a^S x_t & = & I_t + b^S(d_{t,i}) \\
 & x_t & \le & u_t(I_t) \\
 & x_t & \ge & 0
\end{array}
\right\}, \tag{5.1}
$$

where the decision variables are $x_t, I_{t+1,i}, y_i$, and $\hat{z}_{t+1}$ is a (piecewise linear) K-approximation of the value function at stage $t+1$. Notice that $I_{t+1,i}$ represents the next state if the realization of D_t is $d_{t,i}$. Because $\hat{z}_{t+1}$ is piecewise linear, (5.1) is an LP that can be shown to have polynomial size. Under suitable assumptions we

can apply the FPTAS by solving an LP of the form (5.1) instead of (2.1) at every iteration. Hence, in this case we can deal with a potentially very large action space in polynomial time. From a practical standpoint, this approach significantly reduces the number of LPs that have to be solved as compared to an exact approach. A very similar idea can be applied heuristically to the case where the constraint matrix of the LP is not TU, and the state space is therefore a continuous interval; it is an open research question whether we can construct an FPTAS under this setting.

6. Computational experiments. We implemented the FPTAS in Python 3.3. Better performance could be obtained using a faster language such as C. However, in this context we are interested in comparing different approaches, and because all the tested algorithms are implemented in Python, the comparison is fair. The tests were run on Linux on an Intel Xeon E5-4620 at 2.20 GHz (HyperThreading and TurboBoost disabled).

6.1. Generation of random instances. We now give an overview of how the problem instances used in the computational testing phase are generated. More details are provided in Appendix A. We consider two types of problems: stochastic single-item inventory control problems and cash management problems, as described in section 1. In the future we plan to experiment on real-world data. At this stage, we study random instances with "reasonable" numbers. The biggest advantages of testing on random instance are the generation of as many instances as necessary to reduce the sample variance in the results, and the possibility of constructing instances ad hoc in order to find which characteristics affect performance. This is especially important when assessing a new methodology.

We consider two types of problems: stochastic single-item inventory control problems and cash management problems. For each instance we require four parameters: the number of time periods T, the state space size parameter M, the size of the support of the random variable N, and the degree of the cost functions d. The state space size parameter determines the maximum demand in each period for single-item inventory control instances and the maximum difference between cash deposit and cash withdrawal for cash management instances. The actual values of these quantities for each instance are determined randomly, but we ensure that they are between $M/2$ and M, so that the difficulty of the instance scales with M. Each instance requires the generation of some costs: procurement, storage, and backlogging costs for inventory control instances; opportunity, borrowing, and transaction costs for cash management instances. We use polynomial functions $c_t x^d$ to determine these costs, where c_t is a coefficient that is drawn uniformly at random in a suitable set of values for each stage, and $d \in \{1, 2, 3\}$. The difficulty of the instances increases with d, as the numbers involved and U_σ grow. The probabilities for the random variables are uniformly drawn at random at each stage from a suitable set. The transaction costs for cash management instances are fixed at 0.01 per dollar.

The random instances are labeled INVENTORY(T, M, N, d) and CASH(T, M, N, d) to indicate the values of the parameters.

6.2. Overview of the computational evaluation. We generated 20 random INVENTORY and CASH instances for each possible combination of the following values: $T \in \{5, 10, 20, 50\}, M \in \{100, 1000, 10000\}, N \in \{1, 2, 5, 10, 20, 50\}, d \in \{1, 2, 3\}$. This yielded a total of 8640 problem instances. We applied to each of them the following algorithms: an exact DP algorithm (label EXACT), and the FPTAS for $\epsilon \in \{0.001, 0.01, 0.1\}$ as described in Algorithm 5 using one of the subroutines

TABLE 1

Relationship between the value of N and average CPU time. For each algorithm and problem class, we report the minimum recorded Pearson correlation coefficient (denoted ρ) and the 95% sample quantile of the maximum relative error of a linear regression (denoted $MRE_{0.95}$).

Problem	EXACT		APXSET		APXSETSLOPE		APXSETCONVEX	
	ρ	$\text{MRE}_{0.95}$	ρ	$\text{MRE}_{0.95}$	ρ	$\text{MRE}_{0.95}$	ρ	$\text{MRE}_{0.95}$
INVENTORY	0.9996	0.11%	0.9988	0.41%	0.9985	7.57%	0.9981	5.90%
CASH	0.9944	4.32%	0.9913	0.41%	0.9927	5.42%	0.9971	2.42%

APXSET, APXSETSLOPE, or APXSETCONVEX. This allows us to verify whether the modifications proposed in this paper are beneficial. We remark that our implementation of EXACT runs in $O(T|\mathcal{S}|\log|\mathcal{A}|)$ time exploiting convexity and using golden section search; see section 4.2.

For each group of instances generated with the same parameters, we look at the sample mean of the running time for each algorithm. Because the sample standard deviation is typically low, comparing average values is meaningful. When the sample means are very close, we rely on additional statistical tests.

The maximum CPU time for each instance was set to 1000 seconds. In Appendix B we report the number of solved problem instances for each algorithm and approximation factor. In the following, an instance is considered "solved" by an algorithm if the algorithm is able to compute (an approximation of) the initial-stage value function before the time limit.

6.3. Support of the random variables and running time. In this section we analyze the relationship between the running time of the algorithms and the size of the support of the random variables N. The pseudocode of the algorithm suggests that the running time should be linear in N. Here we confirm this conjecture, which simplifies the analysis in the rest of this paper.

For every combination of T, M, d, ϵ and solution algorithm (EXACT, APXSET, APXSETSLOPE, APXSETCONVEX), we compute the Pearson correlation coefficient between the six tested values of N and the corresponding average CPU time. We perform this computation only if the instances are solved before the time limit for at least four values of N; otherwise the corresponding combination of T, M, d, ϵ and solution algorithm is ignored.

The results show a striking linear relationship between N and the running time. In Table 1 we report, for each algorithm and problem class, the minimum recorded correlation coefficient. The results strongly support our conjecture.

It could be argued that the correlation coefficient is misleading because it implicitly assumes homoscedasticity of the random variables representing the average CPU time for a group of instance. As a safety check, for every combination of T, M, d, ϵ and solution algorithm we perform linear regression between average CPU time and N, minimizing the sum of the squared *relative* errors. This corresponds to a maximum likelihood approach if we assume that the variances of the average CPU times are proportional to the square of the mean. As can be seen in Table 1, the 95% sample quantile of the maximum relative error is typically very small. (We report the 95% sample quantile to eliminate some outliers due to instances solved in fractions of a second.) Visual inspection of the linear regression plots confirms the almost perfect positive correlation across all algorithms.

We remark that while a linear dependency of the running time with T could also be expected, that is not the case because of the way we generated the instances: the size of the state space at stage t is proportional to t, hence larger T yields larger state

spaces. If the size of the state space were constant at each stage, we expect that CPU times would exhibit linear dependency on T.

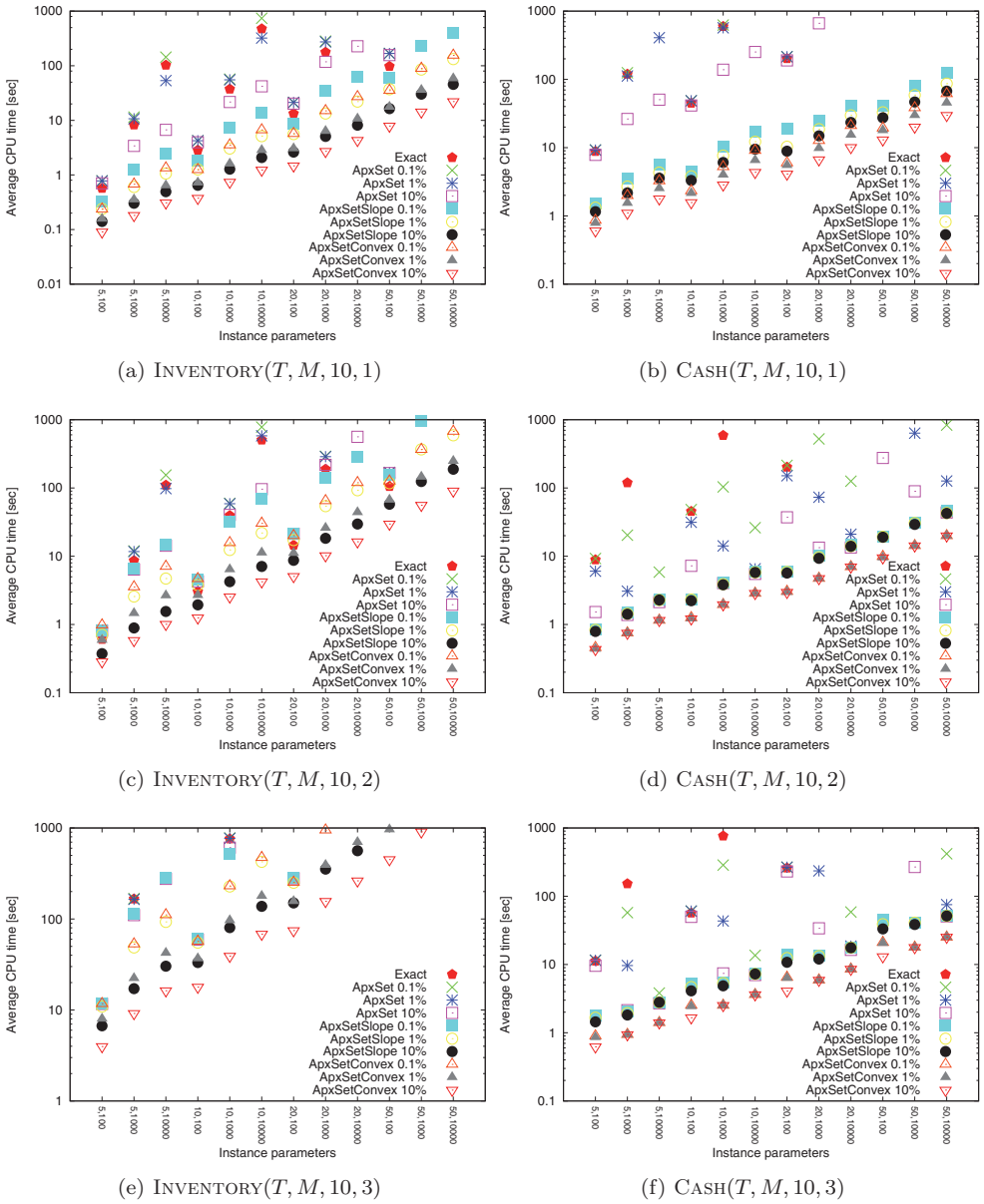
6.4. Analysis of solution times. We now report average CPU times over our test set to compare the different algorithms. Following the discussion of section 6.3, we only report results for $N = 10$: we verified that different values of N do not change the relative ranking of the algorithms, and by linearity of the running times with respect to N, the ranking should hold for all values of N in the tested range. Our results suggest that it should hold even beyond the tested range. Results are reported in Figure 1. For each group of instances we do not report values for algorithms that fail to return a solution on 10 or more out of 20 instances within the allotted time.

It should be noted that the relative standard deviations of the CPU times are typically small: the 95% quantile of the relative standard deviations of the CPU times is 0.19, while the 75% quantile is 0.07. Therefore, comparing the relative performance of the algorithms based on average CPU times is meaningful. In the following, whenever we claim that "an algorithm A is faster than an algorithm B on a group of instances X," it means that in addition to a comparison of the average CPU times, the statement is confirmed by a Wilcoxon signed rank test at the 95% significance level where each observation is represented by the CPU time on a single instance of group X.

We summarize the results. The FPTAS with ApxSet is typically slower than Exact except for some very large instances, whereas ApxSetSlope and ApxSetConvex are always faster than Exact. Surprisingly, they are faster than Exact by more than a factor of 2 even on the smallest instances in our test set: $T = 5, M = 100, N = 10, d = 1$ (on Inventory, 0.57 seconds for Exact versus 0.23 seconds for ApxSetConvex with $\epsilon = 0.1\%$; on Cash, 8.81 seconds for Exact versus 0.86 for ApxSetConvex with $\epsilon = 0.1\%$). The graphs show that ApxSetSlope and ApxSetConvex are consistently faster than an exact solution approach across all instance sizes: the CPU time savings are small but significant for problems solved in ≈ 1 second and become increasingly larger as the difficulty of the instances grows. It is clear that ApxSetSlope and ApxSetConvex scale much better than Exact and ApxSet with the problem size. We record CPU time savings of at least three orders of magnitude, and the graphs show that the savings would become larger on larger problems. ApxSetSlope and ApxSetConvex solve many problems that Exact and ApxSet are not able to solve before the time limit; see also Appendix B. The pictures show that ApxSetConvex is always faster than ApxSetSlope for fixed ϵ, and it is often faster even while using a smaller ϵ.

In Figure 2 we report average CPU times for increasing value of N for instances Inventory$(20, 1000, N, d)$ with $N \in \{1, 2, 5, 10, 20, 50\}, d \in \{1, 2\}$. The graph suggests that the ranking of the three versions of the FPTAS should not change for larger values of N, i.e., we expect ApxSetConvex to be faster than ApxSetSlope, which is in turn faster than ApxSet. The trend shown in Figure 2 indicates that we can expect ApxSetConvex with $\epsilon = 1\%$ to become faster than ApxSetSlope with $\epsilon = 10\%$ for large N. The same conclusion can be drawn from other groups of instances.

We note in Figures 1(d) and 1(f) that the average CPU time taken by ApxSetSlope and ApxSetConvex is essentially the same on Cash problems with $d = 2, 3$ regardless of the value of ϵ. This can be explained because when $d = 2, 3$, the cost at each stage is dominated by the opportunity and borrowing costs, while transaction costs (that do not scale with d) can almost be neglected. For this reason, the difference in the cost-to-go between adjacent states (i.e., resource levels) is very small: the action allows increase or reduction of the resource level paying a very small



(a) Inventory$(T, M, 10, 1)$ (b) Cash$(T, M, 10, 1)$

(c) Inventory$(T, M, 10, 2)$ (d) Cash$(T, M, 10, 2)$

(e) Inventory$(T, M, 10, 3)$ (f) Cash$(T, M, 10, 3)$

Fig. 1. *Average CPU time on* Inventory *and* Cash. *On the x-axis we identify the group of instances with the label T, M. The y-axis is on a logarithmic scale.*

transaction cost. It follows that the shape of the optimal value function is a convex function that has a minimum with a large value and grows slowly when moving away from the minimum. This is the ideal situation for our approximation algorithm: the value functions at each stage can be approximated with very few linear pieces regardless of ϵ, as the relative errors are very small. We verified that if transaction costs scale with d, then Cash instances for $d = 2, 3$ become more difficult for our algorithms. We kept transaction costs independent of d to showcase this unusual characteristic of the

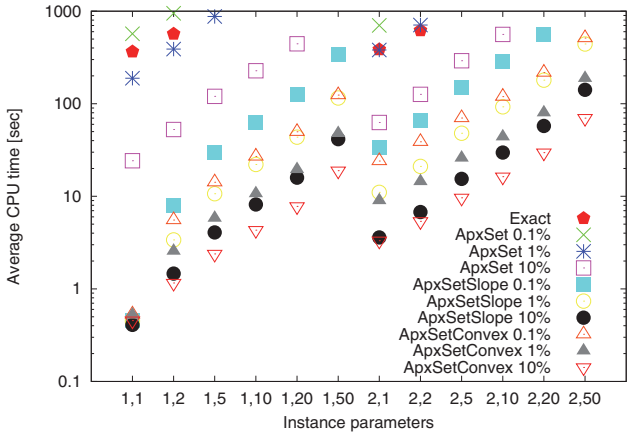


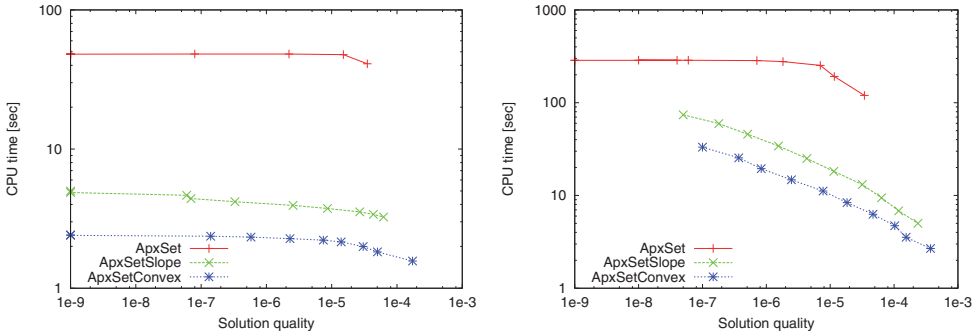
FIG. 2. *Average CPU time on* IVENTORY$(20, 1000, N, d)$ *instances. On the x-axis we identify the group of instances with the label* d, N*. The y-axis is on a logarithmic scale.*

algorithms we propose: while typically the difficulty of an instance increases with the size of the numbers involved (this happens for EXACT and APXSET), it can happen that larger numbers facilitate the task of finding a good piecewise linear approximation of the value function, therefore reducing the running time of the FPTAS.

To summarize, our results show that APXSETCONVEX is consistently faster than APXSETSLOPE, and they are both significantly faster than APXSET and EXACT. EXACT is often faster than APXSET except on large instances with large approximation factors. The proposed algorithms scale much better than an exact algorithm and the CPU time savings from using the faster version of the FPTAS can be very significant: more than three orders of magnitude. The success of APXSETCONVEX compared to APXSETSLOPE and APXSET is due to its ability to find good piecewise linear approximations of the value function using considerably fewer pieces; see Appendix B.2.

6.5. Solution quality. Computational results in section 6.4 show that APXSETCONVEX is the fastest algorithm for equal ϵ. Because ϵ is only an upper bound to the approximation factor, it is still possible that APXSETCONVEX is not the fastest algorithm for equal solution quality. In this section we study whether this is the case by analyzing the trade-offs between solution quality and speed offered by the three versions of the FPTAS.

To do so, we fix $N = 10$ and analyze a number of instances for which we know the optimal solution. Then, we run the three versions of the FPTAS with 10 values of ϵ, equally spaced on a logarithmic scale between 10^{-4} and 10^{-1}. Finally, we compute the exact value of the approximate *policies*, i.e., sequence of actions, returned by the algorithm (as opposed to the approximate value of the policy returned by the FPTAS), and compare performance to the optimal policy. We plot the relative distance to the optimal policy and the corresponding running time on a graph, obtaining one curve for each version of the FPTAS representing the trade-off between solution quality and speed. This experiment is carried out for all instances obtained combining $T \in \{5, 10, 20\}, M \in \{100, 1000, 10000\}, d \in \{1, 2, 3\}$ for which EXACT finds the optimum before the 1000 second time limit.



(a) CASH(10, 100, 10, 1) (b) INVENTORY(20, 1000, 10, 1)

FIG. 3. *Average solution quality (computed as $f^{apx}/f^* - 1$, where f^{apx} is the cost of the approximate policy, and f^* the cost of the optimal policy) versus average CPU time. On each curve, the leftmost point represents the quality of the solution returned for $\epsilon = 10^{-4}$, and the rightmost point for $\epsilon = 10^{-1}$. If such quality is $\leq 10^{-9}$, e.g., if it is 0, it is reported as 10^{-9} on the graph.*

The graphs for CASH with $T = 10, M = 100, d = 1$ and for INVENTORY with $T = 20, M = 1000, d = 1$ are reported in Figure 3. They show very clearly that APXSETCONVEX is faster than APXSETSLOPE for equal solution quality, and both algorithms are much faster than APXSET. The graphs for the vast majority of the instances yield similar conclusions and do not contribute further insight.

Overall, we conclude that both APXSETSLOPE and APXSETCONVEX yield faster CPU times for equal solution quality than APXSET, and in the vast majority of cases APXSETCONVEX gives a better trade-off than APXSETSLOPE, except on some small instances with $d = 2$ where they are comparable. However, APXSETCONVEX gives a better approximation factor guarantee for equal CPU time and seems therefore the best algorithm.

Finally, we analyze the a posteriori approximation guarantees (APAGs). Remember that when applying the FPTAS we can compute an APAG, which is an upper bound on the relative error of the returned policy. In our experiments, APXSET and APXSETSLOPE yield an APAG that can be considerably smaller than the input value of ϵ. For all algorithms, the policies computed by the FPTAS are in relative terms at least one order of magnitude better than the returned APAG. In particular APXSETCONVEX seems to find much better policies than the computed APAG. While APXSETCONVEX has larger APAG than the other algorithms, our analysis shows that it typically finds policies of comparable quality to the other algorithms, in less CPU time. More details can be found in Appendix B.3.

6.6. Markov decision processes. The problems analyzed so far had independent random variables at each stage. We now study an MDP, where the random variable at each stage depends on the state of the DP at the previous stage. We assume that the one-step transition matrix is given explicitly; see section 5.

We consider a stochastic single-item inventory control problem with a two-dimensional state variable $I_t = (R_t, L_t)$, where R_t indicates the resource level, and L_t indicates the demand level. L_t can be thought of as a discrete demand level such as "very low," "low," and so on. The support and the probabilities of the discrete random variable D_t are fully determined given L_{t-1}. In this case, the FPTAS proceeds by computing an approximation of the value function for each value of L_t. Let $\mathcal{L}_t$

be the set of all possible values for L_t. The worst-case running time of the FPTAS increases by a factor $\max_t |\mathcal{L}_t|$, which is still polynomial in the input size [19]. We show that the running time in practice also increases linearly with $\mathcal{L}_t$.

We remark that the implementation of the FPTAS with a two-dimensional state variable is inherently slower than the unidimensional version, by a factor ≈ 2.13 in our experiments. This factor was computed testing the implementation on unidimensional problems cast as two-dimensional. We should note that a more finely tuned implementation in C could probably reduce this slowdown factor.

Let the number of distinct values that L_t can take at each stage $t = 1, \ldots, T$ be a constant L. We generate instances of the form INVENTORY(T, M, N, L, d) where T, M, N, d have the same meaning as before, and $L \leq N$ indicates the number of possible demand levels. At each stage $t = 1, \ldots, T$, the support of the random variable D_t does not change, but the associated probabilities depend on L_{t-1} and are generated using the same random scheme as in section 6.1. As a result, the difficulty of the instances across different levels of L_t should on average be the same.

We test instances with $T \in \{5, 10, 20\}$, $M \in \{100, 1000, 10000\}$, $N \in \{5, 10, 20, 50\}$, $d \in \{1\}$, $L \in \{1, 5, 10, 20, 50\}$ whenever $L \leq N$. At state (R_t, L_t), if the realization of D_t is the value $d_{t,i}$ with $(k-1)N/L \leq i \leq kN/L$, we transition to a state (R_{t+1}, k) at the subsequent stage. We then compute, for each value of T, M, N, the correlation coefficient between L and the average CPU time for the corresponding group of instances. The minimum correlation coefficient recorded for APXSETCONVEX with $\epsilon \in \{0.001, 0.01, 0.1\}$ is 0.9992. Similarly to section 6.3, we also fit a linear regression model that minimizes the sum of the squared relative errors, and we find that the maximum such relative error is only 4.5%. We conclude that the running time grows linearly with the value of L: if we know the average CPU time s required by our FPTAS to solve some instances with $L = 1$ (i.e., there is only one possible demand level), then Ls is a very good estimation of the running time for larger L.

6.7. Comparison with a hybrid value-policy iteration scheme. It is natural to ask how the performance of the proposed FPTAS compares to existing algorithms. The literature on ADP is abundant and proposes many approximation schemes, as surveyed in section 1.1. In this paper we analyze finite-horizon undiscounted problems, and we are not aware of any approach that offers approximation guarantees with respect to the optimal solution with a bound on the running time as our FPTAS. However, convergent approximation schemes have been proposed. One such scheme is the SPAR algorithm; see [32, 34]. Note that while the Mutual-SPAR algorithm used in [32] applies to the problems discussed in this paper, the context for optimal performance of the algorithm is different: in particular, Mutual-SPAR assumes a bounded and relatively small-sized state space in the resource level (in [32], the resource level is bounded by 600) but effectively approximates along other dimensions of the state variable (associated with the state of the stochastic processes). In the instances tested in this paper, the resource level is very large: up to $\approx 5 \times 10^5$ for the largest problems. Ours is clearly a much different setting than [32]. Moreover, despite our best efforts we could not replicate the dataset used in [32] by following the procedure described therein and in [31]. We considered another method [35] to obviate the difficulty of SPAR in dealing with a large state space in the resource level. The method is called the concave, adaptive value estimation (CAVE) algorithm [15, 16]. The approach developed in this section is related to CAVE and to [33], as we discuss below.

ALGORITHM 6. VPITER(S, τ): VALUE-POLICY ITERATION.

Initialize π to a base stock policy with base stock $E[D_t]$ at stage $t, t = 1, \ldots, T$.
$k \leftarrow 1$.
while time limit τ not hit **do**
  while policy π improves **do**
    for $t = T, \ldots, 1$ **do**
      Sample the value function at stage t at $kS(t)$ equally spaced points using the available approximation V_{t+1} and the current policy π
      Update the value function approximation V_t
    Update base stock levels in π using approximations V_t for $t = 1, \ldots, T$
  $k \leftarrow 2k$

We compare the FPTAS to a hybrid value-policy iteration [34, Ch. 3.6] algorithm developed ad hoc for the INVENTORY instances. This algorithm is similar in spirit to the policy improvement scheme discussed in [4, Chap. 6.4]. It works as follows. We start with a base policy π and construct an approximation of the (suboptimal) value function at each stage using this policy, working our way backward from the terminal stage. Then we iteratively improve the policy π, and if the policy can be improved no further, we improve the approximation of the value functions. The value function approximation is a convex piecewise linear function, obtained as the piecewise linear extension of a given number n of equally spaced sample points within the state space. To improve the approximation, we increase n. When $n \geq |\mathcal{S}_t|$, we are computing the true value function. The algorithm, which we call VPITER, is described in Algorithm 6.

We now discuss how VPITER relates to CAVE. CAVE keeps a piecewise linear concave approximation of the value function (for a maximization problem) and updates its slope over an interval of size δ, where δ starts large and decreases over time. [34, Chap. 13.3] suggests halving δ whenever the objective function stops improving. The slope update is performed by gathering samples of the value function and using a smoothing approach. Note that CAVE does not enjoy a general convergence proof and according to the authors requires fine-tuning to be successful [34, Chap. 13.3], but a closely related algorithm is shown to be convergent on problems with scalar state and linear costs that satisfy some additional technical conditions [33], which are similar to the problems discussed in the present paper. VPITER keeps a piecewise linear convex value function approximation by gathering samples of the value function, and we halve the distance between breakpoints whenever the objective function stops improving. There are three main differences: first, CAVE follows sample realizations of the random variables, whereas VPITER evaluates the value function at equally spaced states; second, CAVE estimates expected values, whereas VPITER computes them exactly because of the assumption of small uncertainty set; third, VPITER enforces a specific policy structure known to be optimal. For these reasons, VPITER can be seen as an adaptation of CAVE to our specific problem structure.

We look for policies within the set of all base stock (i.e., "order up to") policies that are optimal for inventory control problems with V-shaped piecewise linear costs; see, e.g., [17]. For INVENTORY instances with $d = 1$ it can be shown that the proposed scheme converges to the optimal policy. For instances with $d > 1$, the scheme may fail to converge as the optimal policy may not be a base stock policy. Iterations of this hybrid value-policy iteration scheme are initially very fast, because finding improved base-stock policies or value function approximations is computationally inexpensive

TABLE 2
Solution quality of APXSETCONVEX *and* VPITER *for equal CPU time.*

	APXSETCONVEX		VPITER		# instances	
ϵ	$\bar{q}$	σ_q	$\bar{q}$	σ_q	Better	Total
0.001	0.000%	0.001	0.144%	0.235	3	936
0.01	0.005%	0.010	0.407%	0.547	13	926
0.1	0.064%	0.122	1.555%	1.963	34	852

when the number of sampling points is low. We are interested in analyzing whether this simple but provably convergent scheme finds better policies than our FPTAS, given equal CPU time. Comparing the FPTAS to VPITER allows us to establish the relative performance of the FPTAS not only to an exact solution method, but also to another approximation scheme based on popular techniques that exploits knowledge of the structure of the optimal policy. Note that the FPTAS can in general find policies with different structure, as it uses a lookup table representation.

To compare VPITER with APXSETCONVEX, on each instance we set the time limit for VPITER to 1% more than the time taken by APXSETCONVEX on the same instance. Note that VPITER requires a function S to determine the initial number of samples. We test three functions: the constant functions $S(t) = 3, S(t) = 100$ for all t, and $S(t) = \log|\mathcal{S}_t|$, obtaining three versions of the algorithm. In our analysis below, APXSETCONVEX is always compared to the best version of VPITER for each instance, i.e., the version that returns the policy with the lowest cost within the time limit, therefore giving an advantage to VPITER.

In Table 2 we report the average solution quality $\bar{q}$, computed as in Figure 3, of APXSETCONVEX for three values of ϵ, and of VPITER with time limit set as described above. We also report the number of instances where VPITER returns a better policy than APXSETCONVEX, and the total number of instances used for these statistics (instances where EXACT does not terminate before 1000 seconds are excluded). Table 2 only takes into account INVENTORY instances with $d = 1$.

The experiments show that APXSETCONVEX consistently returns a solution of better quality than VPITER. In particular there are extremely few instances in which VPITER finds a better policy than APXSETCONVEX, and on average APXSETCONVEX is better by 0.65%. For the sake of completeness, we report that for INVENTORY instances with $d = 2$ (resp., $d = 3$) VPITER returns solutions that are $\approx 7.4\%$ (resp., 19.2%) worse than APXSETCONVEX on average. We remark that as discussed above VPITER may not converge to an optimal policy on these instances.

To summarize, the FPTAS we propose compares very favorably to a provably convergent value-policy iteration algorithm. Not only does it find solutions of better quality in the same CPU time, but it does so while allowing the user to specify an approximation factor guarantee.

7. Final remarks. This paper presents an extension of the method of K-approximation sets and functions aimed at being a practically viable alternative or complement to existing approximate dynamic programming approaches. Our contribution comprises a set of new algorithms and their analysis to improve on the theoretical guarantees provided by the framework, and the description of a computer implementation that achieves its goal of being competitive in practice. Our implementation is open-source and requires the user to input a single algorithmic parameter to control the trade-off between running time and approximation guarantee.

The proposed methodology has limitations: the necessity of explicitly listing the support of the random variables as input, and the assumption that state and action

spaces are unidimensional. Our future research plans include extensions of the framework to partially relax these restrictions. Despite the restrictions, the proposed FPTAS has the merit of handling structured problems with large state and action spaces efficiently, and we believe that it is a first step in the direction of showing that with some ingenuity FPTASes can evolve from a powerful theoretical tool to additionally being a practically useful instrument.

Appendix A. Generation of random instances. In this appendix we give a detailed description of how the random instances discussed in the computational experiments are generated.

A.1. Single-item inventory control. 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 number of different possible demands in each period N. At each time period t, the set $\mathcal{D}_t$ of the N demand values is generated as follows: first we draw the maximum demand w_t at stage t uniformly at random in $[\lfloor M/2 \rfloor, \ldots, M]$, then we draw the remaining $N-1$ values in $[1, \ldots, w_t]$ (without replacement). 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 kth 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 ($f(x) = cx^3/1000$). We label the inventory control instances $\textsc{Inventory}(T, M, N, d)$, where d is the degree of the polynomial describing the cost functions, i.e., 1, 2, or 3. We note that we also experimented with piecewise linear cost functions. However, the practical performance of the algorithms on piecewise linear functions turned out to be very similar to the cost functions with $d = 2$, hence we omit results for space reasons.

A.2. Cash management. An instance of the cash flow management problem is defined by the number of time periods T, the maximum deposit/withdrawal in each period M, and the number of different possible deposit/withdrawal amounts in each time period N. At each time period t, the set $\mathcal{D}_t$ of the N deposit/withdrawal amount values is generated as follows: first we draw the difference w_t between the maximum and minimum values in $[\lfloor M/2 \rfloor, \ldots, M]$, then we draw the minimum value m_t in $[-\lfloor M/2 \rfloor, \ldots, -\lfloor M/2 \rfloor + M - w_t]$, finally we draw a set S_t of $N-2$ values in from $[m_t, \ldots, m_t + w_t]$ (without replacement) and define $\mathcal{D}_t := \{m_t, m_t + w_t\} \cup S_t$. Similar to the inventory control instances, this method ensures that none of the instances we generate is too easy and that the difficulty scales with M. Probabilities are randomly assigned to each deposit/withdrawal value using the method described for inventory control instances.

We must also define two cost functions: the opportunity cost for not investing money if the cash balance is positive (we can assume that this corresponds to the average return rate of invested money) and borrowing costs from the bank in case the cash balance is negative. We use a similar approach to the inventory control problem. We select a coefficient c uniformly at random in $\{0.25, 0.5, \ldots, 5\}$ for the opportunity cost and in $\{2.5, 2.75, \ldots, 7.5\}$ for borrowing costs. We remark that in our tests the difficulty of the instances did not seem to be affected by imposing that borrowing

costs are larger than opportunity costs. We use the same three types of cost functions as for inventory control problems. We also allow for nonzero per-dollar buying and selling costs: each transaction involving buying or selling stocks incurs a cost of $0.01x_t$, where x_t is the transaction amount. We label the cash flow management instances CASH(T, M, N, d) following the usual convention.

Appendix B. Additional computational experiments.

B.1. Number of solved instances. In Tables 3 and 4 we report the number of solved problem instances for each algorithm and approximation factor. An instance is considered "solved" by an algorithm if the algorithm is able to compute (an approximation of) the initial-stage value function before the time limit. Results are aggregated over the six possible values of N.

B.2. Example of value function. As mentioned in the main paper, the success of APXSETCONVEX compared to APXSETSLOPE and APXSET is due to its ability to find good piecewise linear approximations of the value function using considerably fewer pieces. As an example, in Figure 4 we show part of the optimal value function approximation obtained by the three approximation routines on an instance of class INVENTORY$(10, 1000, 10, 1)$. While APXSET obtains a slightly better approximation overall (lower objective function values), the three approximations perform equally in the region close to the optimum, and APXSETCONVEX uses fewer breakpoints.

TABLE 3
Total number of solved instances for INVENTORY *problems.*

			EXACT	APXSET			APXSETSLOPE			APXSETCONVEX		
d	T	M		.001	.01	.1	.001	.01	0.1	.001	.01	.1
1	5	100	120	120	120	120	120	120	120	120	120	120
		1000	120	120	120	120	120	120	120	120	120	120
		10000	120	120	120	120	120	120	120	120	120	120
	10	100	120	120	120	120	120	120	120	120	120	120
		1000	120	120	120	120	120	120	120	120	120	120
		10000	99	80	100	120	120	120	120	120	120	120
	20	100	120	120	120	120	120	120	120	120	120	120
		1000	120	100	100	120	120	120	120	120	120	120
		10000	41	37	60	100	120	120	120	120	120	120
	50	100	120	120	120	120	120	120	120	120	120	120
		1000	60	40	40	62	100	120	120	120	120	120
		10000	0	0	0	40	100	120	120	120	120	120
2	5	100	120	120	120	120	120	120	120	120	120	120
		1000	120	120	120	120	120	120	120	120	120	120
		10000	120	120	120	120	120	120	120	120	120	120
	10	100	120	120	120	120	120	120	120	120	120	120
		1000	120	120	120	120	120	120	120	120	120	120
		10000	92	80	81	120	120	120	120	120	120	120
	20	100	120	120	120	120	120	120	120	120	120	120
		1000	119	100	100	100	120	120	120	120	120	120
		10000	40	21	40	80	100	120	120	120	120	120
	50	100	120	120	120	120	120	120	120	120	120	120
		1000	60	40	40	60	80	100	120	100	120	120
		10000	0	0	0	20	60	80	120	80	100	120
3	5	100	120	120	120	120	120	120	120	120	120	120
		1000	120	120	120	120	120	120	120	120	120	120
		10000	50	50	60	100	100	120	120	120	120	120
	10	100	120	120	120	120	120	120	120	120	120	120
		1000	80	80	80	80	81	100	120	100	120	120
		10000	7	6	20	60	60	100	120	100	120	120
	20	100	100	100	100	100	100	100	120	100	120	120
		1000	40	40	40	40	40	67	100	80	100	120
		10000	0	0	0	20	26	60	80	60	80	100
	50	100	54	43	43	44	42	60	66	60	80	100
		1000	0	0	0	0	0	20	40	20	40	80
		10000	0	0	0	0	0	9	40	0	40	60

B.3. A posteriori error analysis. To give a better sense of how the approximation factor guarantee relates to the quality of the approximate solutions returned by the algorithms, in Table 5 we compare the two quantities over our test set, for all problems for which we know the optimal solution. In particular we show the average approximation factor guarantee $\bar{\epsilon}_g$, which is an a posteriori upper bound on the approximation error, computed by measuring the maximum approximation errors at each stage of the FPTAS; the average ratio $\bar{r}$ between the relative distance of the approximate policy from the optimum policy and $\bar{\epsilon}_g$; and the sample standard deviation σ_r of these ratios. Note that by definition, $\bar{r}$ is a measure of the average improvement of the approximate policy over the a posteriori approximation factor guarantee, e.g., $\bar{r} = 0.1$ means that on average the relative distance of the approximate policy from the optimum policy is a factor 10 better than the computed approximation factor guarantee.

Table 5 shows that the error bounds used by ApxSet and ApxSetSlope are not as tight as the ones used by ApxSetConvex; therefore the first two algorithms yield an a posteriori approximation factor that can be considerably smaller than the input value of ϵ. We also note that $\bar{r}$ is typically smaller than 0.1, so the policies computed by the FPTAS are in relative terms at least one order of magnitude better than what could be expected from the approximation guarantee. In particular, ApxSetConvex seems to find much better policies than the computed approximation factor. While

Table 4
Total number of solved instances for Cash *problems.*

			Exact	ApxSet			ApxSetSlope			ApxSetConvex		
d	T	M		.001	.01	.1	.001	.01	0.1	.001	.01	.1
1	5	100	120	120	120	120	120	120	120	120	120	120
		1000	120	120	120	120	120	120	120	120	120	120
		10000	60	60	99	120	120	120	120	120	120	120
	10	100	120	120	120	120	120	120	120	120	120	120
		1000	80	80	81	120	120	120	120	120	120	120
		10000	20	20	56	101	120	120	120	120	120	120
	20	100	104	102	102	116	119	120	120	120	120	120
		1000	40	40	40	80	120	120	120	120	120	120
		10000	20	20	20	60	120	120	120	120	120	120
	50	100	60	60	60	60	120	120	120	120	120	120
		1000	20	20	20	38	120	120	120	120	120	120
		10000	20	20	20	20	103	120	120	120	120	120
2	5	100	120	120	120	120	120	120	120	120	120	120
		1000	120	120	120	120	120	120	120	120	120	120
		10000	60	120	120	120	120	120	120	120	120	120
	10	100	120	120	120	120	120	120	120	120	120	120
		1000	80	119	120	120	120	120	120	120	120	120
		10000	20	120	120	120	120	120	120	120	120	120
	20	100	104	101	120	120	120	120	120	120	120	120
		1000	40	84	120	120	120	120	120	120	120	120
		10000	20	116	120	120	120	120	120	120	120	120
	50	100	60	60	60	100	120	120	120	120	120	120
		1000	20	40	82	120	120	120	120	120	120	120
		10000	20	72	119	120	120	120	120	120	120	120
3	5	100	120	120	120	120	120	120	120	120	120	120
		1000	120	120	120	120	120	120	120	120	120	120
		10000	55	120	120	120	120	120	120	120	120	120
	10	100	120	120	120	120	120	120	120	120	120	120
		1000	80	100	120	120	120	120	120	120	120	120
		10000	20	120	120	120	120	120	120	120	120	120
	20	100	100	100	100	100	120	120	120	120	120	120
		1000	40	60	101	120	120	120	120	120	120	120
		10000	20	119	120	120	120	120	120	120	120	120
	50	100	60	46	45	60	120	120	120	120	120	120
		1000	20	21	59	100	120	120	120	120	120	120
		10000	20	92	120	120	120	120	120	120	120	120

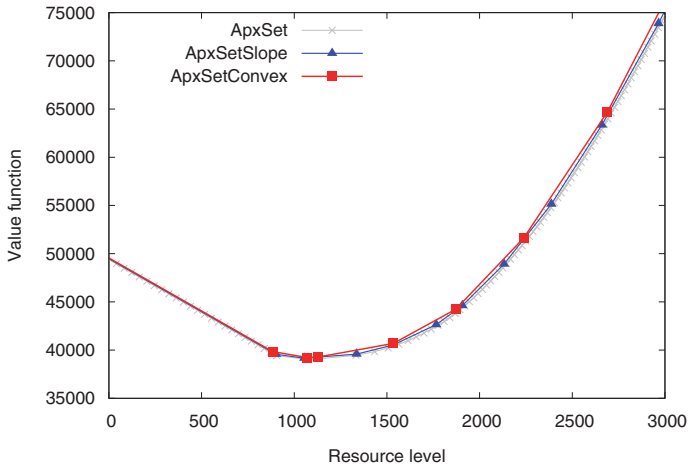


FIG. 4. *Value function approximations for an instance of class* INVENTORY$(10, 1000, 10, 1)$*, using* $\epsilon = 10\%$ *and* APXSET*,* APXSETSLOPE*, or* APXSETCONVEX*. The points represent the elements of the corresponding* K*-approximation sets.*

TABLE 5

Comparison of approximation factor guarantee and quality of the approximate policy returned by the algorithms.

Instance	ϵ	APXSET			APXSETSLOPE			APXSETCONVEX		
		$\bar{\epsilon}_g$	$\bar{r}$	σ_r	$\bar{\epsilon}_g$	$\bar{r}$	σ_r	$\bar{\epsilon}_g$	$\bar{r}$	σ_r
INVENTORY	0.0010	0.00000	0.07934	0.21654	0.00004	0.06004	0.11212	0.00011	0.05762	0.10421
	0.0100	0.00001	0.12960	0.24629	0.00050	0.08170	0.12367	0.00142	0.06824	0.10710
	0.1000	0.00031	0.13040	0.22881	0.00507	0.08156	0.11696	0.01473	0.06286	0.09052
CASHMANAG	0.0010	0.00000	0.09524	0.29710	0.00001	0.04278	0.16889	0.00007	0.00699	0.02002
	0.0100	0.00001	0.07170	0.17359	0.00014	0.02732	0.04574	0.00082	0.00812	0.02060
	0.1000	0.00029	0.03421	0.06189	0.00127	0.01614	0.02210	0.00702	0.00696	0.01810

APXSETCONVEX has larger $\bar{\epsilon}_g$ than the other algorithms, our analysis above and the smaller value of $\bar{r}$ show that it typically finds policies of comparable quality to the other algorithms, in less CPU time.

Appendix C. Proofs.

Proof of Theorem 3.2. APXSETSLOPE is presented for a convex nondecreasing function. It is enough to prove the theorem in this case, and the result for a general convex function follows by applying APXSETSLOPE to the left and to the right of the minimum.

Let y be the point computed at line 4 of APXSETSLOPE, and assume $y < D^{\max}$. Clearly $\varphi(y+1) > K\varphi(x)$; therefore line 4 will be executed at most $O(\log_K \frac{\varphi^{\max}}{\varphi^{\min}})$ times. By convexity, we have

$$\sigma_\varphi(y+1) > \frac{\varphi(y+1)-\varphi(x)}{y+1-x} > \frac{K(\varphi(x)+\sigma_\varphi(x)(y+1-x))-\varphi(x)}{y+1-x} > K\sigma_\varphi(x).$$

It follows that line 4 will be executed at most $O(\log_K \frac{\sigma_\varphi^{\max}}{\sigma_\varphi^{\min}})$ times. Furthermore, because $\varphi(y) - K(\varphi(x)+\sigma_\varphi(x)(y-x))$ is a monotonically nondecreasing function, its zeroes can be found in $O(\log |D|)$ time by binary search.

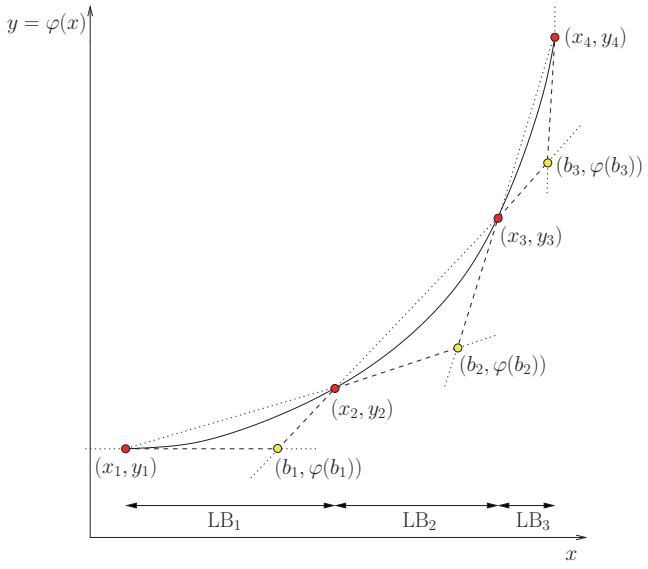


FIG. 5. *Example of Proposition* 4.1. *We assume that the value of* φ *at* x_1, x_2, x_3, x_4 *is known. The dashed line represents the function* $\underline{\varphi}$ *defined as the sum of the* LB_i *functions. Each* LB_i *function is the maximum of the dotted lines below* φ *over the intervals* $[x_1, x_2], [x_2, x_3], [x_3, x_4]$.

It remains to show that W is a K-approximation set. Let $\hat{\varphi}$ be the convex extension of φ induced by W. Take any point y in $D \setminus W$, and let $x := \max\{w \in W : w \le y\}$ and $x' := \min\{w \in W : w \ge y\}$. Clearly $\varphi(y) \ge \varphi(x) + \sigma_\varphi(x)(y - x)$. Notice that $\hat{\varphi}(y) = \varphi(x) + \frac{\varphi(x') - \varphi(x)}{x' - x}(y - x)$ and $\frac{\varphi(x') - \varphi(x)}{x' - x} \le \frac{K(\varphi(x) + \sigma_\varphi(x)(x' - x)) - \varphi(x)}{x' - x} = K\sigma_\varphi(x) + \frac{(K-1)\varphi(x)}{x' - x}$ by construction, so that

$$\frac{\hat{\varphi}(y)}{\varphi(y)} \le \frac{\varphi(x) + \frac{\varphi(x') - \varphi(x)}{x' - x}(y - x)}{\varphi(x) + \sigma_\varphi(x)(y - x)} \le \frac{\varphi(x) + K\sigma_\varphi(x)(y - x) + \frac{(K-1)\varphi(x)(y-x)}{x' - x}}{\varphi(x) + \sigma_\varphi(x)(y - x)}$$
$$\le \frac{\varphi(x) + K\sigma_\varphi(x)(y - x) + (K - 1)\varphi(x)}{\varphi(x) + \sigma_\varphi(x)(y - x)} = K. \qquad \square$$

Proof of Proposition. 4.1. An example to clarify the definitions is shown in Figure 5. Clearly $\underline{\varphi}(x)$ is piecewise linear. Note that by convexity, LB_i underestimates φ over $[x_i, x_{i+1}]$, hence $\underline{\varphi}(x) \le \varphi(x)\ \forall x$.

Observe that $\frac{\textsc{Line}((x_j, y_j), (x_k, y_k), \cdot)}{\underline{\varphi}(\cdot)}$ can be defined piecewise, where the pieces are separated by the points of nondifferentiability of $\underline{\varphi}(\cdot)$. Within each piece, $\frac{\textsc{Line}((x_j, y_j), (x_k, y_k), \cdot)}{\underline{\varphi}(\cdot)}$ is defined as the ratio of two linear functions; therefore there is no stationary point in the interior of each piece. It follows that the maximum of $\frac{\textsc{Line}((x_j, y_j), (x_k, y_k), \cdot)}{\underline{\varphi}(\cdot)}$ is one of the points of nondifferentiability of $\underline{\varphi}(\cdot)$, except (x_j, y_j) and (x_k, y_k), where two local minima are located by construction. All points of nondifferentiability are contained in $x_e \in [x_j, x_k] \cap (E \cup B)$. Hence,

$$\gamma_E(x_j, x_k) := \max_{x_e \in x_e \in [x_j, x_k] \cap (E \cup B)} \left\{ \frac{\textsc{Line}((x_j, y_j), (x_k, y_k), x_e)}{\underline{\varphi}(x_e)} \right\}$$
$$\ge \left| \frac{\textsc{Line}((x_j, y_j), (x_k, y_k), w)}{\varphi(w)} \right| \quad \forall w \in [x_j, x_k].$$

Finally, note that we can trivially bound the numerator and denominator of $\gamma_E(x_j, x_k)$ to obtain

$$\gamma_E(x_j, x_k) = \max_{x_e \in E x_e \in [x_j, x_k] \cap (E \cup B)} \left\{ \frac{\textsc{Line}((x_j, y_j), (x_k, y_k), x_e)}{\varphi(x_e)} \right\} \leq \frac{\varphi(x_k)}{\varphi(x_j)}. \qquad \square$$

The proof of Theorem 4.2 requires a few lemmas first. In Lemma C.1 we show that we can safely ignore all points to the right of $\max\{\text{next}(r, E), \arg \gamma_E(r, x)\}$ on line 9 of the algorithm. Then, we bound the size of the approximation sets computed by ApxSetConvex in Lemma C.2. We proceed to bound the time required for the crucial operations in Lemmas C.3 and C.4. After this step, we can provide a full proof of Theorem 4.2.

Lemma C.1. *Let $\varphi : D \to \mathbb{Q}^+$ be a convex function defined over a set of integers. Let E, r and x be defined as in Algorithm 4. Let $v \leq r$ and $x_e := \arg\max_{x_e \in [r,x] \cap (E \cup B)} \{\frac{\textsc{Line}((r,\varphi(r)),(x,\varphi(x)),x_e)}{\underline{\varphi}(x_e)}\}$, i.e., the point that yields the largest approximation error in the definition of $\gamma_E(r, x)$. Let $E' := E \cap [D^{\min}, \max\{next(r, E), x_e\}]$. Then $\gamma_E(v, x) = \gamma_{E'}(v, x)$.*

Proof of Lemma C.1. We show that for every $x'_e \in E, x'_e > x_e, x'_e > r$,

$$\frac{\textsc{Line}((v,\varphi(v)),(x,\varphi(x)),x'_e)}{\underline{\varphi}(x'_e)} \leq \frac{\textsc{Line}((v,\varphi(v)),(x,\varphi(x)),x_e)}{\underline{\varphi}(x_e)}.$$

By definition of x_e,

$$\frac{\textsc{Line}((r,\varphi(r)),(x,\varphi(x)),x'_e)}{\underline{\varphi}(x'_e)} \leq \frac{\textsc{Line}((r,\varphi(r)),(x,\varphi(x)),x_e)}{\underline{\varphi}(x_e)}.$$

Therefore,

$$\begin{aligned}
&\frac{\textsc{Line}((v,\varphi(v)),(x,\varphi(x)),x'_e)}{\underline{\varphi}(x'_e)} \\
&= \frac{\textsc{Line}((r,\varphi(r)),(x,\varphi(x)),x'_e)}{\underline{\varphi}(x'_e)} + \frac{\textsc{Line}((v,\varphi(v)),(x,\varphi(x)),x'_e) - \textsc{Line}((r,\varphi(r)),(x,\varphi(x)),x'_e)}{\underline{\varphi}(x'_e)} \\
&\leq \frac{\textsc{Line}((r,\varphi(r)),(x,\varphi(x)),x_e)}{\underline{\varphi}(x_e)} + \frac{\textsc{Line}((v,\varphi(v)),(x,\varphi(x)),x'_e) - \textsc{Line}((r,\varphi(r)),(x,\varphi(x)),x'_e)}{\underline{\varphi}(x'_e)} \\
&\leq \frac{\textsc{Line}((r,\varphi(r)),(x,\varphi(x)),x_e)}{\underline{\varphi}(x_e)} + \frac{\textsc{Line}((v,\varphi(v)),(x,\varphi(x)),x_e) - \textsc{Line}((r,\varphi(r)),(x,\varphi(x)),x_e)}{\underline{\varphi}(x_e)} \\
&= \frac{\textsc{Line}((v,\varphi(v)),(x,\varphi(x)),x_e)}{\underline{\varphi}(x_e)},
\end{aligned}$$

where the last inequality follows by $\underline{\varphi}(x'_e) \geq \underline{\varphi}(x_e)$, the fact that the slope of $\textsc{Line}((v,\varphi(v)),(x,\varphi(x)),\cdot)$ is smaller than the slope of $\textsc{Line}((r,\varphi(r)),(x,\varphi(x)),\cdot)$ by convexity, and $v \leq r$.

It follows that $\gamma_E(v, x) = \gamma_{E'}(v, x)$. $\square$

Lemma C.2. *Let $\varphi : D \to \mathbb{Q}^+$ be a convex function defined over a set of integers. Then* ApxSetConvex(φ, D, K) *computes a K-approximation set of φ of size $O(\log_K \min\{\frac{\sigma_\varphi^{\max}}{\sigma_\varphi^{\min}}, \frac{\varphi^{\max}}{\varphi^{\min}}\})$ and executes the loop at lines 6–14 $O(\log_K \min\{\frac{\sigma_\varphi^{\max}}{\sigma_\varphi^{\min}}, \frac{\varphi^{\max}}{\varphi^{\min}}\} \log |D|)$ times.*

Proof of Lemma C.2. ApxSetConvex is presented for a convex nondecreasing function. It is enough to prove the lemma in this case, and the result for a general convex function follows by applying ApxSetConvex to the left and to the right of the minimum.

We introduce the necessary definitions. We call *outer* loop the loop at line 4 of ApxSetConvex and *inner* loop the loop at line 6. At any iteration of the outer loop, let x be the last point added to W. We keep the notation ℓ, r for the value of the respective variables at the beginning of an inner loop iteration, i.e., at the beginning of line 7. We denote by ℓ', r' the value of the variables ℓ, r at the end of an inner loop iteration, i.e., at the end of line 8 (this is the only line of the inner loop where ℓ, r are modified). We use the notation $\text{prev}^n(x, D), \text{next}^n(x, D)$ to indicate the application of the previous and next operators n times.

Note that $r' \leq r$. This is because γ_E does not increase as points are added to E; hence if line 7 returns point w at some iteration of the inner loop, it will only return points $\leq w$ at subsequent iterations. Furthermore, $r - \ell \geq 2(r' - \ell')$ unless $r' \leq \ell$. It follows that in $\log |D|$ iterations either the inner loop terminates and a new point w is added to W or $r' \leq \ell$ at least once and no point is added to W.

The algorithm searches for the next point to be added to W in the interval $[D^{\min}, \dots, z]$, where initially $z = D^{\max}$. We show that after $O(\log |D|)$ iterations of the inner loop this interval is reduced to $[D^{\min}, \dots, z']$, where $K\sigma_\varphi(z') < \sigma_\varphi(z)$, and at most one point was added to W. This implies $K^{|W|} < \frac{\sigma_\varphi^{\max}}{\sigma_\varphi^{\min}}$, from which the first part of the lemma follows.

(a) We first analyze the case where a point w is added to W in the first $4 \log |D|$ inner loop iterations of an outer loop iteration. In this case, $z = x$. Let $z' = \text{prev}(w, D)$. Then $\gamma_{\tilde{E}}(z', x) > K$ at some point during the algorithm for some set $\tilde{E}$ (otherwise lines 7–8 would give $r \leq z'$). Let x_e be the point that yields the maximum for $\gamma_{\tilde{E}}(z', x)$. Let $s_1 = \frac{\varphi(x) - \varphi(z')}{x - z'}$ and $s_2 = \frac{\underline{\varphi}(x_e) - \varphi(z')}{x_e - z'}$. We have $\frac{\varphi(z') + s_1(x_e - z')}{\varphi(z') + s_2(x_e - z')} = \text{Line}((z', \varphi(z')), (x, \varphi(x)), x_e)/\underline{\varphi}(x_e) = \gamma_{\tilde{E}}(z', x) > K$. Because $\varphi(z') \geq 0$, it follows by simple algebraic manipulations that $s_1/s_2 \geq \frac{\varphi(z') + s_1(x_e - z')}{\varphi(z') + s_2(x_e - z')} > K$. Furthermore, it is easy to show $\sigma_\varphi(z') \leq s_2$ and $s_1 \leq \sigma_\varphi(z)$. Therefore the search is restricted from $[D^{\min}, \dots, z = x]$ to $[D^{\min}, \dots, z']$ with $K\sigma_\varphi(z') < \sigma_\varphi(z)$.

(b) Then we analyze the case where we perform $4 \log |D|$ successive iterations of the inner loop without adding any point to W. Because no point is added to W, we must have $r' \leq \ell$ at least four times. Lines 7–8 imply that there exist four points z^1, z^2, z^3, z^4 such that $\gamma_{E^i}(z^i, x) > K$, $i = 1, 2, 3, 4$, for some sets E^i. Assume without loss of generality that $z^i < z^{i+1}$, and let x_e^i be the points that yield the maximum in $\gamma_{E^i}(z^i, x)$. Note that we are not assuming $z = x$. Define $s_1 = \frac{\varphi(x) - \varphi(z^1)}{x - z^1}$ and $s_2 = \frac{\underline{\varphi}(x_e^2) - \varphi(z^1)}{x_e^2 - z^1}$. We can repeat the argument of case (a) to show $Ks_2 < s_1$. We have $z^1 < z^2 < x_e^2$; therefore by construction $\sigma_\varphi(z^1) \leq s_2$. We want to show $s_1 \leq \sigma_\varphi(z)$. Notice that because x_e^2 maximizes $\gamma_{E^2}(z^2, x)$ and by construction of x_e^2, we must have $s_1 \leq \frac{\varphi(\text{next}^2(x_e^2, E^2)) - \varphi(\text{next}(x_e^2, E^2))}{\text{next}^2(x_e^2, E^2) - \text{next}(x_e^2, E^2)}$. Therefore, $s_1 \leq \sigma_\varphi(\text{next}^2(x_e^2, E^2))$. If we can show $z \geq \text{next}^2(x_e^2, E^2)$, the desired result follows by convexity.

Observe that $x_e^1 \leq x_e^2 \leq x_e^3 \leq x_e^4$ because of line 9 (Lemma C.1). Also, $z^i \geq \text{prev}(x_e^i, E^i)$ for $i = 2, 3, 4$. Suppose not. Then $\gamma_{E^i}(z^i, x) > K$ regardless of points added to E^i in subsequent iterations (all these points are $< z^i$). This implies that a point is added to W within the subsequent $\log |D|$ iterations of

the inner loop, which is a contradiction. Thus, $\text{prev}(x_e^2, E^2) \le z^2 < z^3 < z^4 < z$, so $\text{next}^2(x_e^2, E^2) \le z$.

This shows that after $4\log|D|$ successive iterations of the inner loop such that no point is added to W, the search is restricted from $[D^{\min}, \dots, z]$ to $[D^{\min}, \dots, z' = z^1]$ with $K\sigma_\varphi(z') < \sigma_\varphi(z)$.

Combining cases (a) and (b) together shows that a reduction of the search interval with a factor K reduction of the slope of φ takes $O(\log|D|)$ iterations of the inner loop. This concludes the first part of the proof.

Next, we show that after $O(\log|D|)$ iterations of the inner loop $[D^{\min}, \dots, z]$ is reduced to $[D^{\min}, \dots, z']$, where $K\varphi(z') < \varphi(z)$, and at most one point was added to W. This implies $K^{|W|} < \frac{\varphi^{\max}}{\varphi^{\min}}$, from which the second part of the lemma follows.

Observe that in at most $\Lambda\log|D|$ iterations of the inner loop, one of the following happens: either a point w is added to W, or no point is added to W and line 12 ($z \leftarrow r$) is executed. In the first case, let w be the point added to W and let $z' = \text{prev}(w, D)$. Then $\gamma_E(z', x) > K$ at some point during the algorithm, and by Proposition 4.1 $K\varphi(z') < \varphi(z)$. In the second case, let $z' = r$; line 11 ensures $K\varphi(z') < \varphi(z)$. This concludes the second part of the proof.

It remains to show that W is a K-approximation set. This follows from Proposition 4.1, Lemma C.1, and the fact that whenever w is added to W, for all points y in the interval $[w, x]$ we have $\gamma_E(y, x) \le K$. ∎

LEMMA C.3. *Let $\varphi : D \to \mathbb{Q}^+$ be a convex function defined over a set of integers, and $K > 0$. Then $|E|$ is bounded above by $2\log|D|$ during the execution of* ApxSetConvex(φ, D, K).

Proof of Lemma C.3. Observe that points added to E are of the form $\lfloor(\ell + r)/2\rfloor$; therefore at any given time points in E inside the interval $[D^{\min}, r]$ follow a geometric progression, and there are at most $\log|D|$ of them. It remains to show that the number of points in the interval $[\text{next}(r, E), D^{\max}]$ does not exceed $\log|D|$.

First, notice that we are only interested in the interval $[\text{next}(r, E), x]$, because points $> x$ are eliminated from E on line 16. Moreover, E contains only point $\le \max\{\text{next}(r, E), \arg\gamma_E(r, x)\}$ because of line 9. If $x_e = \arg\gamma_E(r, x)\} \le r$, we are done. Otherwise, we observe that, as discussed in the proof of Lemma C.2, subcase (b), if $x_e > \text{next}(r, E)$ only $\log|D|$ iterations of the inner loop are executed (and $\log|D|$ points are added to E) before the loop exits and line 16 is executed, eliminating points in the interval $[x = r, D^{\max}]$. Notice that when the condition $x_e > \text{next}(r, E)$ is first satisfied at a given iteration of the outer loop, E contains no more than $\log|D|$ points as shown above (some of these points could be in the interval $[r, x]$). At most $\log|D|$ points are added to E, hence $|E| \le 2\log|D|$ throughout the algorithm. ∎

LEMMA C.4. *The value of $\gamma_E(x_j, x_k)$ as defined in Proposition.* 4.1 *can be computed in $O(\log|E|)$ time by binary search if E and B are stored in a sorted array.*

Proof of Lemma C.4. By definition, we have

$$\gamma_E(x_j, x_k) := \max_{x_e \in [x_j, x_k] \cap (E \cup B)} \left\{ \frac{\text{LINE}((x_j, y_j), (x_k, y_k), x_e)}{\underline{\varphi}(x_e)} \right\}.$$

Clearly this computation can be split into

$$\gamma'_E(x_j, x_k) := \max_{x_e \in [x_j, x_k] \cap E} \left\{ \frac{\text{LINE}((x_j, y_j), (x_k, y_k), x_e)}{\underline{\varphi}(x_e)} \right\},$$

$$\gamma''_E(x_j, x_k) := \max_{x_e \in [x_j, x_k] \cap B} \left\{ \frac{\text{LINE}((x_j, y_j), (x_k, y_k), x_e)}{\underline{\varphi}(x_e)} \right\},$$

taking the maximum of the two values. We show that the two functions can be maximized by binary search over, respectively, E and B. Define

$$\vartheta(x_j, x_k, x_e) := \frac{\textsc{Line}((x_j, \varphi(x_j)), (x_k, \varphi(x_k)), x_e)}{\underline{\varphi}(x_e)}.$$

First part. We have

$$\gamma'_E(x_j, x_k) := \max_{x_e \in [x_j, x_k] \cap E} \vartheta(x_j, x_k, x_e).$$

Recall that by construction $\vartheta(x_j, x_k, x_e) \geq 1$ for $x_e \in [x_j, x_k] \cap E$, and its value is 1 at the endpoints of the interval. The numerator of ϑ has constant slope, whereas the slope of the denominator is monotonically nondecreasing because $\underline{\varphi}(x_e) = \varphi(x_e)\ \forall x_e \in E$ and φ is convex. It follows that ϑ is unimodal over E. Furthermore, the slope of ϑ is constant only if the slopes of the numerator and denominator are equal, which can happen only on a (possibly empty) connected subset of E by convexity of φ. We conclude that we can partition E into three, not all empty connected subsets such that ϑ is strictly increasing on the first one, constant on the second one, and strictly decreasing on the third one, and the maximum can be found by binary search.

Second part. Recall that $|B| \leq |E|$. We have

$$\gamma''_E(x_j, x_k) := \max_{x_e \in [x_j, x_k] \cap B} \vartheta(x_j, x_k, x_e).$$

We show that the convex extension of the denominator of ϑ over B is a convex function, so that we can apply the same argument as for γ'_E. Let $x_1, x_4, x_7 \in B$ be any three points with the property $x_1 < x_4 < x_7$, and define $x_{i-1} := \text{prev}(x_i, E), x_{i+1} := \text{next}(x_i, E)$ for $i = 1, 4, 7$. We show that

$$\underline{\varphi}(x_7) \geq \textsc{Line}((x_1, \underline{\varphi}(x_1)), (x_4, \underline{\varphi}(x_4)), x_7),$$

which implies the desired result. By definition of $\underline{\varphi}$ over the breakpoints B and by convexity of φ,

$$\underline{\varphi}(x_7) \geq \textsc{Line}((x_3, \underline{\varphi}(x_3)), (x_6, \underline{\varphi}(x_6)), x_7) \geq \textsc{Line}((x_1, \underline{\varphi}(x_1)), (x_4, \underline{\varphi}(x_4)), x_7).$$

This concludes the proof. ∎

Proof of Theorem 4.2. The approximation set size and K-approximation properties of ApxSetConvex follow from Lemma C.2.

By Lemma C.2, ApxSetConvex executes the loop at lines 6–14 $O(\log_K \min\{\frac{\sigma_\varphi^{\max}}{\sigma_\varphi^{\min}}, \frac{\varphi^{\max}}{\varphi^{\min}}\} \log |D|)$ times. We must show that this loop can be executed in $O(\log^2 \log |D|)$ time.

Store the sets E and B of Proposition 4.1 separately in two skip lists sorted by ascending x-coordinate of the points $(x, \varphi(x))$. E has size $O(\log |D|)$ by Lemma C.3; therefore B has size $O(\log |D|)$. Thus, we can insert points in $O(\log \log |D|)$ time, and because we always delete sets of adjacent points at the end of the list, every deletion operation takes $O(\log \log |D|)$ time. Note that whenever a new point y is added to E, all points in B except at most four are left unchanged. The points that have to be recomputed are the two smallest points greater than y and the two largest points smaller than y. This operation can be done in $O(\log \log |D|)$ time by searching for the largest point in B smaller than $\text{prev}(y, E)$.

It remains to show that line 7 can be executed in $O(\log^2 \log |D|)$ time. Lemma C.4 shows that we can find the point x_e that maximizes the approximation error bound (i.e. γ_E) by binary search over the two skip lists containing E and B separately. Each search can be carried out in $O(\log \log |D|)$ time using the skip list structure. Thus, γ_E can be evaluated in $O(\log \log |D|)$ time. Furthermore, $\gamma_E(\cdot, \cdot)$ is monotonically decreasing in its first argument; therefore $\min\{y \in E : (y > D^{\min}) \text{ and } (\gamma_E(y, x) \leq K)\}$ can be computed by binary search over E, evaluating γ_E $O(\log \log |D|)$ times. This concludes the proof. □

We note that without lines 10–13 of ApxSetConvex we could only prove a slightly weaker result, namely, ApxSetConvex computes a K-approximation set of φ of size $O(\log_K \min\{\frac{\sigma_\varphi^{\max}}{\sigma_\varphi^{\min}}, \frac{\varphi^{\max}}{\varphi^{\min}}\})$ in time $O(t_\varphi \log_K \frac{\sigma_\varphi^{\max}}{\sigma_\varphi^{\min}} \log |D| \log^2 \log |D|)$. In practice, lines 10–13 have little effect except in degenerate situations, but they guarantee a better worst-case performance. In our computational tests, there is only a small fluctuation in the size of the approximation sets and in the CPU times if lines 10–13 are removed, with neither version of the algorithm appearing as the winner.

Acknowledgment. We are grateful to the referees and associate editor for their comments that helped improved the paper.

REFERENCES

[1] D. L. Applegate, W. Cook, S. Dash, and D. G. Espinoza, *Exact solutions to linear programming problems*, Oper. Res. Lett., 35 (2007), pp. 693–699.
[2] C. Bazgan, H. Hugot, and D. Vanderpooten, *Implementing an efficient FPTAS for the 0–1 multiobjective knapsack problem*, European J. Oper. Res., 198 (2009), pp. 47–56.
[3] R. Bellman, *Dynamic Programming*, Princeton University Press, Princeton, NJ, 1957.
[4] D. P. Bertsekas, *Dynamic Programming and Optimal Control*, Athena Scientific, Belmont, MA, 1995.
[5] J. F. Carriere, *Valuation of the early-exercise price for options using simulations and nonparametric regression*, Insurance Math. Econom., 19 (1996), pp. 19–30.
[6] W. Chen, *Private communication*, 2013.
[7] W. Chen, M. Dawande, and G. Janakiraman, *Fixed-dimensional stochastic dynamic programs: An approximation scheme and an inventory application*, Oper. Res., 62 (2014), pp. 81–103.
[8] W. Cook, T. Koch, D. E. Steffy, and K. Wolter, *An exact rational mixed-integer programming solver*, in Proceedings of IPCO, O. Günlük and G. J. Woeginger, eds., Lecture Notes in Comput. Sci. 6655, Springer-Verlag, Berlin, 2011, pp. 104–116.
[9] G. Cornuéjols, F. Margot, and G. Nannicini, *On the safety of Gomory cut generators*, Math. Program. Comput., 5 (2013), pp. 345–395.
[10] D. P. de Farias and B. Van Roy, *The linear programming approach to approximate dynamic programming*, Oper. Res., 51 (2003), pp. 850–865.
[11] D. P. de Farias and B. Van Roy, *A cost-shaping Linear Program for average-cost approximate dynamic programming with performance guarantees*, Math. Oper. Res., 31 (2006), pp. 597–620.
[12] V. V. Desai, V. F. Farias, and C. C. Moallemi, *Approximate dynamic programming via a smoothed linear program*, Oper. Res., 60 (2012), pp. 655–674.
[13] V. V. Desai, V. F. Farias, and C. C. Moallemi, *Pathwise optimization for optimal stopping problems*, Management Sci., 58 (2012), pp. 2292–2308.
[14] S. E. Dreyfus and A. M. Law, *The Art and Theory of Dynamic Programming*, Academic Press, New York, 1977.
[15] G. A. Godfrey and W. B. Powell, *An Adaptive Approximation Method for Stochastic, Dynamic Programs, with Applications to Inventory and Distribution Problems*, Technical report SOR-97-10, Department of Civil Engineering and Operations Research, Princeton University, Princeton, NJ, 1998.
[16] G. A. Godfrey and W. B. Powell, *An adaptive, distribution-free algorithm for the newsvendor problem with censored demands, with applications to inventory and distribution*, Management Sci., 47 (2001), pp. 1101–1112.

[17] S. C. Graves, A. H. George R. Kan, and P. H. Zipkin, eds., *Logistics of Production and Inventory*, Handbooks Oper. Res. Management Sci. 4, North-Holland, Amsterdam, 1993.
[18] N. Halman, D. Klabjan, C.-L. Li, J. Orlin, and D. Simchi-Levi, *Fully polynomial time approximation schemes for stochastic dynamic programs*, in Proceedings of SODA, S.-H. Teng, ed., SIAM, Philadelphia, 2008, pp. 700–709.
[19] N. Halman, D. Klabjan, C.-L. Li, J. Orlin, and D. Simchi-Levi, *Fully polynomial time approximation schemes for stochastic dynamic programs*, SIAM J. Discrete Math., 28 (2014), pp. 1725–1796.
[20] N. Halman, D. Klabjan, M. Mostagir, J. Orlin, and D. Simchi-Levi, *A fully polynomial time approximation scheme for single-item inventory control with discrete demand*, Math. Oper. Res., 34 (2009), pp. 674–685.
[21] N. Halman, G. Nannicini, and J. Orlin, *A computationally efficient FPTAS for convex stochastic dynamic programs*, in Proceedings of ESA, H. Bodlaender and G. Italiano, eds., Lecture Notes in Comput. Sci. 8125, Springer, Berlin, 2013, pp. 577–588.
[22] N. Halman, J. B. Orlin, and D. Simchi-Levi, *Approximating the nonlinear newsvendor and single-item stochastic lot-sizing problems when data is given by an oracle*, Oper. Res., 60 (2012), pp. 429–446.
[23] *IEEE Standard for Floating-Point Arithmetic*, 754-2008, IEEE, New York, 2008, pp. 1–70.
[24] W. Kahan, *Pracniques: Further remarks on reducing truncation errors*, Comm. ACM, 8 (1965).
[25] J. Kiefer, *Sequential minimax search for a maximum*, Proc. Amer. Math. Soc., 4 (1953), pp. 502–506.
[26] F. A. Longstaff and E. S. Schwartz, *Valuing American options by simulation: A simple least-squares approach*, Rev. Financial Studi., 14 (2001), pp. 113–147.
[27] B. L. Miller, *On minimizing nonseparable functions defined on the integers with an inventory application*, SIAM J. Appl. Math., 21 (1971), pp. 166–185.
[28] J. I. Munro, T. Papadakis, and R. Sedgewick, *Deterministic skip lists*, in Proceedings of SODA, SIAM, Philadelphia, 1992, pp. 367–375.
[29] K. Murota, *Discrete Convex Analysis*, SIAM, Philadelphia, 2003.
[30] S. Nadarajah, F. Margot, and N. Secomandi, *Improved Least Squares Monte Carlo for Term Structure Option Valuation with Energy Applications*, Technical report 2012-E54, Tepper School of Business, Carnegie Mellon University, Pittsburgh, PA, 2013.
[31] J. M. Nascimento, *Approximate Dynamic Programming for Complex Storage Problems*, Ph.D. thesis, Princeton University, Princeton, NJ, 2008.
[32] J. M. Nascimento and W. B. Powell, *Dynamic programming models and algorithms for the mutual fund cash balance problem*, Management Sci., 56 (2010), pp. 801–815.
[33] J. M. Nascimento and W. B. Powell, *An optimal approximate dynamic programming algorithm for concave, scalar storage problems with vector-valued controls*, IEEE Trans. Automat. Control, 58 (2013), pp. 2995–3010.
[34] W. B. Powell, *Approximate Dynamic Programming: Solving the Curses of Dimensionality,* 2nd ed., Wiley, New York, 2011.
[35] W. B. Powell, *Private communication*, 2013.
[36] W. B. Powell, A. Ruszczyński, and H. Topaloglu, *Learning algorithms for separable approximations of discrete stochastic optimization problems*, Math. Oper. Res., 29 (2004), pp. 814–836.
[37] W. Pugh, *Skip lists: A probabilistic alternative to balanced trees*, Comm. ACM, 33 (1990), pp. 668–676.
[38] D. Salas and W. B. Powell, *Benchmarking a Scalable Approximation Dynamic Programming Algorithm for Stochastic Control of Multidimensional Energy Storage Problems*, Technical report, Princeton Unversity, Princeton, NJ, 2013.
[39] J. N. Tsitsiklis and B. V. Roy, *Regression methods for pricing complex American-style options*, IEEE Trans. Neural Networks, 12 (2001), pp. 694–703.
[40] A. Wächter and L. T. Biegler, *On the implementation of a primal-dual interior point filter line search algorithm for large-scale nonlinear programming*, Math. Program., 106 (2006), pp. 25–57.