A Computationally Efficient FPTAS for Convex Stochastic Dynamic Programs

Nir Halman[1], Giacomo Nannicini[2,⋆], and James Orlin[3]

[1] Jerusalem School of Business Administration, The Hebrew University, Jerusalem, Israel, and Dept. of Civil and Environmental Engineering, MIT, Cambridge, MA
halman@mit.edu
[2] Singapore University of Technology and Design, Singapore
nannicini@sutd.edu.sg
[3] Sloan School of Management, MIT, Cambridge, MA
jorlin@mit.edu

Abstract. We propose a computationally efficient Fully Polynomial-Time Approximation Scheme (FPTAS) for convex stochastic dynamic programs using the technique of K-approximation sets and functions introduced by Halman et al. 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 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. Our computational evaluation is based on randomly generated problem instances coming from applications in supply chain management and finance.

1 Introduction

We consider a finite horizon stochastic dynamic program (DP), as defined in [1]. Our model has an underlying discrete time dynamic system, and a cost function that is additive over time. We now introduce the type of problems addressed in this paper. We postpone a rigorous definition of each symbol until Sect. 2, without compromising clarity. The system dynamics are of the form: $I_{t+1} = f(I_t, x_t, D_t), \quad t = 1, \dots, T$, where:

$$\begin{aligned}
t &: \text{discrete time index,}\\
I_t \in \mathcal{S}_t &: \text{state of the system at time } t\\
&\ (\mathcal{S}_t \text{ is the \textit{state space} at stage } t),\\
x_t \in \mathcal{A}_t(I_t) &: \text{action or decision to be selected at time } t\\
&\ (\mathcal{A}_t(I_t) \text{ is the \textit{action space} at stage } t \text{ and state } I_t),\\
D_t &: \text{discrete random variable over the sample space } \mathcal{D}_t,\\
T &: \text{number of time periods.}
\end{aligned}$$

⋆ Corresponding author.

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. The total incurred cost is $\sum_{t=1}^{T} g_t(I_t, x_t, D_t) + g_{T+1}(I_{T+1})$, where g_{T+1} is the terminal cost function. The problem is that of choosing a sequence of actions $x_1, \ldots, x_T$ that minimizes the expectation of the total incurred cost. This problem is called a *stochastic dynamic program*. Formally, we want to determine:

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

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.

Theorem 1.1. *[2] 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 $O(T|\mathcal{A}||\mathcal{S}|)$.

Halman et al. [3] give 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. The three classes of [3] are: convex DP, nondecreasing DP, nonincreasing DP. In this paper we propose a modification of the FPTAS for the convex DP case that achieves better running time. Several applications of convex DPs are discussed in [4]. Two examples are:

1. **Stochastic single-item inventory control** [5]: we want to find replenishment quantities for a warehouse in each time period to minimize the expected procurement and holding/backlogging costs. This is a classic problem in supply chain management.
2. **Cash Management** [6]: we want to manage the cash flow of a mutual fund. At the beginning of each time period we can buy or sell stocks, thereby changing the cash balance. At the end of each time period, 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, unlike the original FPTAS of [3], 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. To the best of our knowledge, this is the first time that a framework for the automatic generation of FPTASes is shown to be a practically as well as theoretically useful tool. The only computational evaluation with positive results of an FPTAS we could find in the literature is [7], 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 FPTASes 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 rest of this paper is organized as follows. In Sect. 2 we introduce our notation and the original FPTAS of [3]. In Sect. 3, we improve the worst-case running time of the algorithm. In Sect. 4, we discuss our implementation of the FPTAS. Sect. 5 contains an experimental evaluation of the proposed method.

An important difference between our approach and traditional Approximate DP (ADP) methods (see e.g. [1]) is that we provide an approximation guarantee. For this reason, a fair comparison with ADP is difficult and is not undertaken in this work, but a brief discussion is presented in Sect. 5. Note that at this stage, our comparison with ADP methods is only preliminary.

2 Preliminaries and Algorithm Description

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 $\{\ell, \ell+1, \ldots, u\}$ a *contiguous* interval. We denote a contiguous interval by $[\ell, \ldots, u]$, 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 $x \in D \setminus \{D^{\max}\}$, $\sigma_\varphi(D^{\max}) := \sigma_\varphi(\text{prev}(D^{\max}, D))$. Let $\mathcal{S}_{T+1}$ and $\mathcal{S}_t$ be contiguous intervals for $t = 1, \ldots, T$. For $t = 1, \ldots, T$ 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 $t = 1, \ldots, T$ let $\mathcal{D}_t \subset \mathbb{Q}$ be a finite set, let $g_t : \mathcal{S}_t \times \mathcal{A}_t \times \mathcal{D}_t \to \mathbb{N}$ and $f_t : \mathcal{S}_t \times \mathcal{A}_t \times \mathcal{D}_t \to \mathcal{S}_{t+1}$. Finally, let $g_{T+1} : \mathcal{S}_{T+1} \to \mathbb{N}$.

In this paper we deal with a class of problems labeled "convex DP" for which an FPTAS is given by [3]. [3] 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 [8].

Definition 2.1. *[3] A DP is a* Convex DP *if: The terminal state space $\mathcal{S}_{T+1}$ is a contiguous interval. For all $t = 1, \ldots, T+1$ 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 an integer-valued convex function over $\mathcal{S}_{T+1}$. For every $t = 1, \ldots, T$, 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 integer-valued convex functions, $a(d) \in \mathbb{Z}, b(d) \in \{-1, 0, 1\}$, and $c(d) \in \mathbb{Z}$.

Let $U_{\mathcal{S}} := \max_{t=1,\ldots,T+1} |\mathcal{S}_t|$, $U_{\mathcal{A}} := \max_{t=1,\ldots,T} |\mathcal{A}_t|$ and $U_g := \frac{\max_{t=1,\ldots,T+1} g_t^{\max}}{\min_{t=1,\ldots,T+1} g_t^{\min}}$. 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 $t = 1, \ldots, T$, we define $\sigma_{g_t}^{\max} := \max_{x_t \in \mathcal{A}_t, d_t \in D_t} \sigma_{g_t(\cdot, x_t, d_t)}^{\max}$ and $\sigma_{g_t}^{\min} := \min_{x_t \in \mathcal{A}_t, d_t \in D_t} \sigma_{g_t(\cdot, x_t, d_t)}^{\min}$. Let $U_\sigma := \frac{\max_{t=1,\ldots,T+1} \sigma_{g_t}^{\max}}{\min_{t=1,\ldots,T+1} \sigma_{g_t}^{\min}}$. 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 in [3] that a Convex DP admits an FPTAS, using a framework that we review later in this section. Even though these definitions may look burdensome, the conditions cannot be relaxed. In particular, [3] shows that a Convex DP where $b(d) \notin \{-1, 0, 1\}$ in Def. 2.1 does not admit an FPTAS unless P = NP.

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 $t = 1, \ldots, T$, 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 $\mathcal{D}_t = \{d_{t,1}, \ldots, d_{t,n_t}\}$, where $d_{t,i} < d_{t,j}$ for $i < j$. Moreover, we are given positive integers $q_{t,1}, \ldots, q_{t,n_t}$ such that $P[D_t = d_{t,i}] = \frac{q_{t,i}}{\sum_{j=1}^{n_t} q_{t,j}}$ (see [9] for an extension). For every $t = 1, \ldots, T$ and $i = 1, \ldots, n_t$, 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.

The basic idea underlying the FPTAS of Halman et al. 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 2.2. *[10] 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 for all $x \in D$ we have $\varphi(x) \leq \tilde{\varphi}(x) \leq K\varphi(x)$.*

Definition 2.3. *[10] 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 [5]. In this paper we only deal with the convex case, therefore when presenting results from [3,10] we try to avoid the complications of the two monotone cases. In the rest of this paper it is assumed that the conditions of Def. 2.1 are met.

Definition 2.4. *[10] Let $\varphi : D \to \mathbb{R}$. $\forall E \subseteq D$, 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 (see [10]), 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 2.5. *[10] Let $\varphi : D \to \mathbb{R}^+$ be a convex function over a finite domain of real numbers, and let $x^* = \arg\min_{x \in D}\{\varphi(x)\}$. 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 $(x, \varphi(x)), x \in W$. (iii) $\hat{\varphi}$ is minimized at x^*.*

Proposition 2.6. *[10] Let $1 \le K_1 \le K_2, 1 \le t \le T, I_t \in \mathcal{S}_t$ 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 $i = 1, \dots, n_t$. Let $\hat{z}_{t+1}$ be a convex K_2-approximation of z_{t+1}, and let:*

$$\hat{G}_t(I_t, \cdot) := E_{D_t}\left[\hat{g}_t(I_t, \cdot, D_t)\right], \quad \hat{Z}_{t+1}(I_t, \cdot) := E_{D_t}\left[\hat{z}_{t+1}(f_t(I_t, \cdot, D_t))\right].$$

Then

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

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

We now have all the necessary ingredients to describe the FPTAS for convex DPs given in [10]. The algorithm is given in Algorithm 1. It is shown in [10] that $\bar{z}_t, \hat{z}_t$ are convex for every t.

Theorem 2.7. *[10] 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 1 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.*

Algorithm 1. FPTAS for convex DP.

```
1: $K \leftarrow 1 + \frac{\epsilon}{2(T+1)}$
2: $W_{T+1} \leftarrow \textsc{ApxSet}(g_{T+1}, \mathcal{S}_{T+1}, K)$
3: Let $\hat{z}_{T+1}$ be the convex extension of $g_{T+1}$ induced by $W_{T+1}$
4: for $t = T, \ldots, 1$ do
5:     Define $\bar{z}_t$ as in (1) with $\hat{g}_t$ set equal to $g_t$
6:     $W_t \leftarrow \textsc{ApxSet}(\bar{z}_t, \mathcal{S}_t, K)$
7:     Let $\hat{z}_t$ be the convex extension of $\bar{z}_t$ induced by $W_t$
8: return $\hat{z}_1(I_1)$
```

Algorithm 2. ApxSetSlope(φ, D, K)

```
1: $x \leftarrow D^{\min}$
2: $W \leftarrow \{D^{\min}\}$
3: while $x < D^{\max}$ do
4:     $x \leftarrow \max\{\text{next}(x, D), \max\{y \in D : (y \leq D^{\max}) \wedge (\varphi(y) \leq K(\varphi(x) + \sigma_\varphi(x)(y - x))\}\}$
5:     $W \leftarrow W \cup \{x\}$
6: return $W$
```

3 Improved Running Time

In this section we show that for the convex DP case, we can improve the running time given in Thm. 2.7.

In the framework of [10], monotone functions are approximated by a step function, and Def. 2.3 guarantees the K-approximation property for this case. However, ApxSet greatly overestimates the error committed by the convex extension induced by W. For the convex DP case we propose the simpler Def. 3.1 of K-approximation set, that preserves correctness of the FPTAS and the analysis carried out in [10].

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 Def. 2.3 as given in [10]. E.g.: $D = \{0, 1, 2\}, \varphi(0) = 0, \varphi(1) = 1, \varphi(2) = 2K$; the only K-approximation set according to the original definition is D itself, whereas $\{0, 2\}$ is also a K-approximation set in the sense of Def. 3.1. An algorithm to compute a K-approximation set in the sense of Def. 3.1 is given in Algorithm 2 (for nondecreasing functions).

Theorem 3.2. *Let $\varphi : D \to \mathbb{N}^+$ 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.*

We can improve on Thm. 2.7 by replacing each call to ApxSet with a call to ApxSetSlope in Algorithm 1.

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 1 runs in $O((t_g + t_f + \log(\frac{T}{\epsilon} \log \min\{U_\sigma, TU_g\}))\frac{n^* T^2}{\epsilon} \log \min\{U_\sigma, TU_g\} \log U_{\mathcal{S}} \log U_{\mathcal{A}})$ 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 computes smaller K-approximation sets than ApxSetSlope in practice, although we do not improve over Thm. 3.2 from a theoretical standpoint, and the analysis is more complex. 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 $\frac{\textsc{Line}((x,\varphi(x)),(x',\varphi(x')),w)}{\varphi(w)}$ $\forall w \in [x, \ldots, x']$.

Proposition 4.1. *Let $\varphi : [\ell, u] \to \mathbb{R}^+$ be a nondecreasing convex function. Let $h \geq 3$, $E = \{x_i : x_i \in [\ell, u], i = 1, \ldots, h\}$ 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 $i = 1, \ldots, h-1$, define $LB_i(x)$ as:*

$$LB_i(x) := \max\{\textsc{Line}((x_{i-1}, y_{i-1}), (x_i, y_i), x), \textsc{Line}((x_{i+1}, y_{i+1}), (x_{i+2}, y_{i+2}), x)\}$$

for $x \in [x_i, x_{i+1}]$, and $LB_i(x) := 0$ otherwise. 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

$$\gamma_E(x_j, x_k) := \max_{x_e \in [x_j, x_k] \cap (E \cup B)} \left\{ \frac{\textsc{Line}((x_j, y_j), (x_k, y_k), x_e)}{\underline{\varphi}(x_e)} \right\}. \quad (2)$$

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

The set B of Prop. 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 3 (for nondecreasing functions). In the description of ApxSetConvex, $\Lambda > 1$ is a given constant. We used $\Lambda = 2$ in our experiments. The running time of ApxSetConvex can be a factor $\log |D|$ slower than ApxSetSlope, but its practical performance is superior for two reasons: it produces smaller approximation sets, and evaluates $\bar{z}$ fewer times (each evaluation of $\bar{z}$ is expensive, see below). We experimented with applying Prop. 4.1 in conjunction with ApxSetSlope, but this did not improve the algorithm's performance, therefore we omit the discussion.

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

Theorem 4.2. *Let $\varphi : D \to \mathbb{N}^+$ 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 $O(t_\varphi \log_K \min\{\frac{\sigma_\varphi^{\max}}{\sigma_\varphi^{\min}}, \frac{\varphi^{\max}}{\varphi^{\min}}\} \log^2 |D|)$.*

We now briefly discuss our implementation of the FPTAS, omitting details for space reasons. By (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 use a dictionary to store function values of $\bar{z}_t$ to make sure that the minimization in (1) is performed at most once for each state I_t. Second, we use golden section search [11] instead of binary search to minimize convex functions, in all cases where a function evaluation requires more than $O(1)$ time.

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 not implemented in hardware and is considerably slower, but does not incur into numerical errors and can handle arbitrarily large numbers. In particular, only by using arbitrary precision one can guarantee to find the optimal solution of a problem instance. Our implementation guarantees that the final result satisfies the desired approximation guarantee, by bounding the error introduced by the floating point computations, and rounding key calculations appropriately.

Finally, at each stage of the dynamic program, 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

Algorithm 4. Implementation of the FPTAS for convex DP.

```
 1: $K \leftarrow (1+\epsilon)^{\frac{1}{T+1}}$
 2: $W_{T+1} \leftarrow \textsc{ApxSetConvex}(g_{T+1}, \mathcal{S}_{T+1}, K)$
 3: Let $K'_{T+1}$ be the maximum value of $\gamma_E$ recorded by $\textsc{ApxSetConvex}$
 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 (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:     $W_t \leftarrow \textsc{ApxSetConvex}(\bar{z}_t, \mathcal{S}_t, K_t)$
 9:     Let $K'_t$ be the maximum value of $\gamma_E$ recorded by $\textsc{ApxSetConvex}$
10:     Let $\hat{z}_t$ be the convex extension of $\bar{z}_t$ induced by $W_t$
11: return $\hat{z}_1(I_1)$
```

computations. The choice of K_t is based on the actual maximum approximation factor recorded at all stages $\tau > t$. We summarize the FPTAS for convex DPs in Algorithm 4.

5 Computational Experiments

We implemented the FPTAS in Python 3.2. 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@2.20 Ghz (HyperThreading and TurboBoost disabled).

5.1 Generation of Random Instances

We now give an overview of how the problem instances used in the computational testing phase are generated. We consider two types of problems: stochastic single-item inventory control problems, and cash management problems. For each instance we require 4 parameters: the number of time periods T, the state space size parameter M, the size of the support of the random variable N, 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 beetween $M/2$ 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

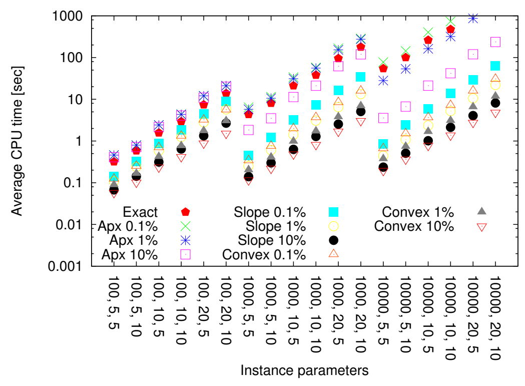


Fig. 1. Average CPU time on INVENTORY$(T, M, N, 1)$ instances. On the x-axis we identify the group of instances with the label M, T, N. The y-axis is on a logarithmic scale. The label of the approximation algorithms indicates the value of ϵ and the subroutine used to compute the K-approximation sets, namely: "Apx" uses APXSET, "Slope" uses APXSETSLOPE, "Convex" uses APXSETCONVEX.

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 random instances are labeled INVENTORY(T, M, N, d) and CASH(T, M, N, d) to indicate the values of the parameters. In the future we plan to experiment on real-world data. At this stage, we limit ourselves to random instances with "reasonable" numbers.

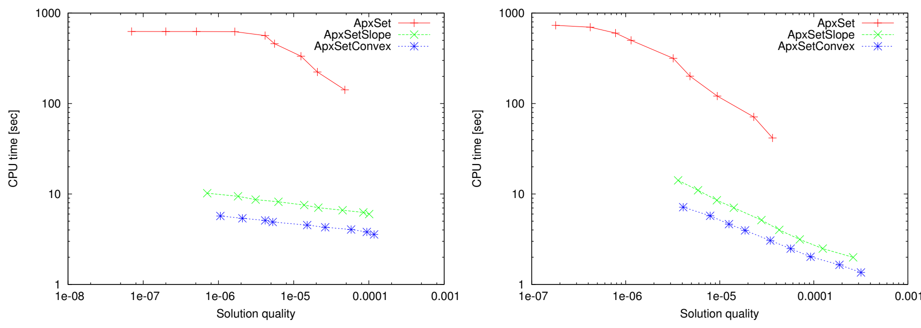
5.2 Analysis of the Results

We generated 50 random INVENTORY and CASH instances for each possible combination of the following values: $T \in \{5, 10, 20\}$, $M \in \{100, 1000, 10000\}$, $N \in \{5, 10\}$. We applied to each instance the following algorithms: an exact DP algorithm (label EXACT), and the FPTAS for $\epsilon \in \{0.001, 0.01, 0.1\}$ as described in Alg. 4 using one of the subroutines APXSET, APXSETSLOPE, APXSETCONVEX. This allows us to verify whether or not 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.

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 tests – see below.

The maximum CPU time for each instance was set to 1000 seconds. Part of the results are reported in Fig. 1, the rest are omitted for space reasons.

We summarize the results. The FPTAS with APXSET is typically slower than EXACT, whereas APXSETSLOPE and APXSETCONVEX are always faster than EXACT. Surprisingly, they are faster than EXACT by more than a factor 2 even on the smallest instances in our test set: $T = 5, M = 100, N = 5, d = 1$ (on INVENTORY, 0.32 seconds for EXACT vs. 0.12 seconds for APXSETCONVEX with $\epsilon = 0.001$). As the size of the instances and the numbers involved grows, the difference increases in favor of the FPTAS. For INVENTORY instances with $d = 1$ the FPTAS with APXSETCONVEX and $\epsilon = 1\%$ can be faster than EXACT by more than 2 orders of magnitude (482.8 seconds for EXACT, 3.0 seconds for



(a) Cash(10, 1000, 10, 1) (b) Inventory(10, 10000, 10, 1)

Fig. 2. Average solution quality (computed as $f^{\text{apx}}/f^{*} - 1$, where f^{apx} is the cost of the approximated policy, and f^{*} the cost of the optimal policy) vs. average CPU time.

ApxSetConvex with $\epsilon = 1\%$). For the largest problems in our test set, Exact does not return a solution before the time limit whereas ApxSetSlope ($\epsilon = 0.1$) and ApxSetConvex ($\epsilon \in \{0.01, 0.1\}$) terminate on *all* instances. The results suggest that the improvements of the FPTAS can be over 3 orders of magnitude and increase with the problem size. The Cash problem seems to heavily favor the FPTAS over Exact, with consistent speed-ups of two (resp. three) orders of magnitude on medium (resp. large) instances. On the largest Cash instance that can be solved by Exact with $d = 1$ ($T = 5, M = 10000, N = 5$), ApxSetConvex takes 0.9 seconds, compared to 717.6 for Exact. The difference increases in favor of the FPTAS for larger d. This may be due to our random instance generator, and has to be investigated further.

ApxSetConvex is faster than ApxSetSlope despite its overhead per iteration on all groups of instances except two (out of 108). This was verified with a Wilcoxon rank-sum test at the 95% significance level. For two groups of Inventory instances with $d = 2$, ApxSetConvex and ApxSetSlope yield comparable CPU time. For $d = 1, 3$ ApxSetConvex is clearly the fastest algorithm, whereas for Inventory and $d = 2$ ApxSetSlope is occasionally comparable with ApxSetConvex, see below the analysis on solution quality. At this stage we do not have a clear explanation for this behavior.

We analyze the quality of the approximated solutions using the three possible ApxSet routines. We run the FPTAS with 9 different values of K (equally spaced on a log scale between 10^{-3} and 10^{-1}) on Inventory and Cash instances with $T = 10$, $N = 10$, and varying M, d. We then compare on a graph the average cost of the policy (i.e. the sequence of actions) obtained with the different ApxSet routines, and the respective average running times. This can only be done for instances on which Exact finds a solution. The graphs for Cash with $M = 1000, d = 1$ and for Inventory with $M = 10000, d = 1$ are reported in Fig. 2. They show very clearly that ApxSetConvex is faster than ApxSetSlope for equal solution quality, and both algorithms are much faster than ApxSet. Graphs for other instances with $d = 1, 3$ yield similar conclusions and do not contribute further insight. On Inventory problems with $d = 2$, in some cases the performance of ApxSetSlope is comparable to ApxSetConvex, as stated above. However, ApxSetConvex gives a better approximation factor guarantee for equal CPU time, and seems therefore the best algorithm.

We conclude with a short comparison with the ADP approach of [6] for Cash problems. The instances and approach discussed in [6] are significantly different from ours, hence we cannot provide a fair comparison. However, to give a rough idea, we observe that [6] reports speed-ups of three orders of magnitude over an exact DP approach with an average approximation factor of 0.28% and no approximation guarantee, whereas our FPTAS achieves similar speed-ups on large instances guaranteeing an approximation factor of 0.1% and achieving an actual approximation factor $< 0.001\%$ in most cases.

Acknowledgements. Partial support was provided by: EU FP7/2007-2013 grant 247757 and Recanati Fund of Jerusalem School of Business Administration (N. Halman), SUTD grant SRES11012 and IDC grants IDSF1200108, IDG21300102 (G. Nannicini), ONR grant N000141110056 (J. Orlin).

References

1. Bertsekas, D.P.: Dynamic Programming and Optimal Control. Athena Scientific, Belmont (1995)
2. Bellman, R.: Dynamic Programming. Princeton University Press, Princeton (1957)
3. Halman, N., Klabjan, D., Li, C.L., Orlin, J., Simchi-Levi, D.: Fully polynomial time approximation schemes for stochastic dynamic programs. In: Teng, S.H. (ed.) SODA, pp. 700–709. SIAM (2008)
4. Nascimento, J.M.: Approximate dynamic programming for complex storage problems. PhD thesis, Princeton University (2008)
5. Halman, N., Klabjan, D., Mostagir, M., Orlin, J., Simchi-Levi, D.: A fully polynomial time approximation scheme for single-item inventory control with discrete demand. Mathematics of Operations Research 34(3), 674–685 (2009)
6. Nascimento, J., Powell, W.: Dynamic programming models and algorithms for the mutual fund cash balance problem. Management Science 56(5), 801–815 (2010)
7. Bazgan, C., Hugot, H., Vanderpooten, D.: Implementing an efficient FPTAS for the 0-1 multiobjective knapsack problem. European Journal of Operations Research 198(1), 47–56 (2009)
8. Murota, K.: Discrete Convex Analysis. SIAM, Philadelphia (2003)
9. Halman, N., Orlin, J.B., Simchi-Levi, D.: Approximating the nonlinear newsvendor and single-item stochastic lot-sizing problems when data is given by an oracle. Operations Research 60(2), 429–446 (2012)
10. Halman, N., Klabjan, D., Li, C.L., Orlin, J., Simchi-Levi, D.: Fully polynomial time approximation schemes for stochastic dynamic programs. Technical Report 3918, Optimization Online (June 2013)
11. Kiefer, J.: Sequential minimax search for a maximum. Proceedings of the American Mathematical Society 4(3), 502–506 (1953)