A Fast 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 an efficient Fully Polynomial-Time Approximation Scheme (FPTAS) for stochastic convex 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 practical performance, and show that it is faster than an exact algorithm even for small problems 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: $I_{t+1} = f(I_t, x_t, D_t), \quad t = 1, \ldots, T$, where:

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

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. Costs are accumulated over all time periods: the total incurred cost is equal to $\sum_{t=1}^{T} g_t(I_t, x_t, D_t) + g_{T+1}(I_{T+1})$. In this expression, $g_{T+1}(I_{T+1})$ is the cost paid if the system ends in state I_{T+1}, and the sequence of states is defined by the system dynamics. The problem is that of choosing a sequence of actions $x_1, \ldots, x_T$ that minimizes the expectation of the total incurred cost. This problem is called a *stochastic dynamic program.* Formally, we want to determine:

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

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. *[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 $I_t \in S_t$, 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 in each time period to minimize the expected procurement and holding/backlogging costs. If procurement and holding/backlogging costs are convex, this problem falls under the convex DP case. 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 can borrow money from the bank at some cost. If the costs are convex (in practical applications, V-shaped piecewise linear are typically used), this problem falls 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 practically 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 the next section we formally 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.

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 $\{\ell, \ell+1, \dots, u\}$ 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 $x \in D \setminus \{D^{\max}\}$, $\sigma_\varphi(D^{\max}) := \sigma_\varphi(\text{prev}(D^{\max}, D))$. 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, 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]. (For a maximization problem, the analogue of a convex DP is a concave DP.) [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] or the Appendix A.1.

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 $t = 1, \dots, T+1$ 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 an integer-valued 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 $g_t(I, x, d) = g^I_t(I, d) + g^x_t(x, d) + u_t(f_t(I, x, d))$, and function f_t can be expressed as $f_t(I, x, d) = a(d)I + b(d)x + c(d)$ where $g^I_t(\cdot, d), g^x_t(\cdot, d), u_t(\cdot)$ are univariate integer-valued convex functions, $a(d) \in \mathbb{Z}, b(d) \in \{-1, 0, 1\}$, and $c(d) \in \mathbb{Z}$.

Let $U_{\mathcal{S}} := \max_{t=1,\dots,T+1} |\mathcal{S}_t|$, $U_{\mathcal{A}} := \max_{t=1,\dots,T} |\mathcal{A}_t|$ and $U_g := \frac{\max_{t=1,\dots,T+1} g^{\max}_t}{\min_{t=1,\dots,T+1} g^{\min}_t}$. Given $\varphi : D \to \mathbb{R}$, let $\sigma^{\max}_\varphi := \max_{x \in D}\{|\sigma_\varphi(x)|\}$ and $\sigma^{\min}_\varphi := \min_{x \in D}\{|\sigma_\varphi(x)| : |\sigma_\varphi(x)| > 0\}$. For $t = 1, \dots, T$, we define $\sigma^{\max}_{g_t} := \max_{x_t \in \mathcal{A}_t, d_t \in D_t} \sigma^{\max}_{g_t(\cdot, x_t, d_t)}$ and $\sigma^{\min}_{g_t} := \min_{x_t \in \mathcal{A}_t, d_t \in D_t} \sigma^{\min}_{g_t(\cdot, x_t, d_t)}$. Let $U_\sigma := \frac{\max_{t=1,\dots,T+1} \sigma^{\max}_{g_t}}{\min_{t=1,\dots,T+1} \sigma^{\min}_{g_t}}$. 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 $t = 1, \dots, T$, 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 also given positive integers $q_{t,1}, \dots, q_{t,n_t}$ such that $P[D_t = d_{t,i}] = \frac{q_{t,i}}{\sum_{j=1}^{n_t} q_{t,j}}$. Note that the framework can be extended to 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 [9], but for simplicity in this paper we keep the more restrictive assumption. 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,
$Q_t := \sum_{j=1}^{n_t} q_{t,j}$: a common denominator of all probabilities at stage t,
$M_t := \prod_{j=t}^{T} Q_j$: a common denominator of all probabilities in all stages from t to T.

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 $\varphi(x) \leq \tilde{\varphi}(x) \leq K\varphi(x)$.*

Algorithm 1 ApxSet(φ, D, K)

```
1: $x \leftarrow D^{\max}$
2: $W \leftarrow \{D^{\min}, D^{\max}\}$
3: **while** $x > D^{\min}$ **do**
4:     $x \leftarrow \min\{\text{prev}(x, D), \min\{y \in D : (y \geq x^*) \wedge (K\varphi(y) \geq \varphi(x))\}\}$
5:     $W \leftarrow W \cup \{x\}$
6: **return** $W$
```

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 particular, [3, 10, 5] define a step function approximation of a function φ over a set W, and its piecewise linear extension. In the convex case, we only deal with the piecewise linear extension without invalidating the proofs of the results from Halman et al. reported in the present paper. 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 Algorithm 1, which computes a K-approximation set of a function φ over the domain D. 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 nondecreasing 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^*.*

Algorithm 2 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)
6:     $W_t \leftarrow \textsc{ApxSet}(\lfloor M_t \bar{z}_t \rfloor, \mathcal{S}_t, K)$
7:     Let $\hat{z}_t$ be the convex extension of $\frac{\lfloor M_t \bar{z}_t \rfloor}{M_t}$ induced by $W_t$
8: return $\hat{z}_1(I_1)$
```

Proposition 2.6. *[10] 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 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

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

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 2. 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 2 runs in $O((t_g + t_f + \log(\frac{T}{\epsilon} \log U_g)) \frac{n^* T^2}{\epsilon} \log U_g \log U_{\mathcal{S}} \log U_{\mathcal{A}})$ 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 Thm. 2.7.

In the framework of [10], monotone functions are approximated by a step functions, and Def. 2.3 guarantees the K-approximation property for this case. However, ApxSet (Algorithm 1) 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 $x \in D$, $\hat{\varphi}(x) \leq K\varphi(x)$.*

Algorithm 3 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 $W$
```

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 3 (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 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 $O((t_g + t_f + \log(\frac{T}{\epsilon} \log \min\{U_\sigma, U_g\}))\frac{n^* T^2}{\epsilon} \log \min\{U_\sigma, U_g\} \log U_{\mathcal{S}} \log U_{\mathcal{A}})$ time, which is polynomial in both $1/\epsilon$ and the (binary) input size.*

4 From theory to practice

In this section we discuss how to implement the FPTAS in order to achieve better practical performance while not deteriorating the worst-case running time of Thm. 3.3.

4.1 Improvements in the computation of K-approximation sets

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

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

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

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

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

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

By Prop. 4.1, we can compute the set of breakpoints B to define a procedure that computes in $O(k - j)$ time 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. ApxSetConvex yields the same asymptotic performance as ApxSetSlope, however its practical performance turned out to be superior for two reasons: it produces smaller approximation sets, and evaluates $\bar{z}$ fewer times (each evaluation of $\bar{z}$ is expensive, see below).

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 $O(t_\varphi \log_K \min\{\frac{\sigma_\varphi^{\max}}{\sigma_\varphi^{\min}}, \frac{\varphi^{\max}}{\varphi^{\min}}\} \log |D|)$.*

4.2 Efficient implementation in floating point arithmetics

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 make sure that the minimization in the definition (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 (1).

Second, we do not approximate g_t when computing (1), i.e. we use $K_1 = 1$ in Prop. 2.6, as is also suggested in the proof of Theorem 9.4 of [10]. In a practical setting, the cost function g_t is available as an $O(1)$ time oracle: setting $K_1 = 1$ avoids several time-consuming steps.

Algorithm 4 ApxSetConvex(φ, D, K).

```
 1: $W \leftarrow \{D^{\min}, D^{\max}\}$
 2: $x \leftarrow D^{\max}$
 3: $E \leftarrow \{D^{\min}, D^{\max}\}$
 4: while $x > D^{\min}$ do
 5:    $\ell \leftarrow D^{\min}, r \leftarrow x, s \leftarrow r - \ell, \text{counter} \leftarrow 0, z \leftarrow x$
 6:    while $r > \text{next}(\ell, D)$ do
 7:       $E \leftarrow E \cup \{\lfloor(\ell + r)/2\rfloor\}$
 8:       $w \leftarrow \min\{y \in E : (y > D^{\min}) \wedge (\gamma_E(y, x) \leq K)\}$
 9:       $\ell \leftarrow \text{prev}(w, E), r \leftarrow w, \text{counter}{+}{+}$
10:       if counter $> \Lambda \log(|D|)$ then
11:          if $\varphi(z) > K\varphi(r)$ then
12:             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:    $W \leftarrow W \cup \{x\}$
17: return $W$
```

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 [11], 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.

We now discuss our implementation in floating point arithmetics. 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 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.

We decided to use floating point arithmetics for the sake of speed. In Appendix B we describe how our implementation guarantees that the final result satisfies the desired approximation guarantee. This involves careful algorithm design so that we can bound the error introduced by the 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 computations. The choice of K_t is based on the actual maximum approximation factor recorded at all stages $\tau > t$. More details are provided in Appendix C. We summarize the FPTAS for convex DPs in Algorithm 5.

Algorithm 5 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)
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 $\hat{z}_1(I_1)$
```

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 a single-core of an Intel Xeon X5687.

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. Full details can be found in Appendix D. For each instance we require 4 parameters: the number of time periods T, the state space size parameter M_d, the size of the support of the random variable N_p, 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 $M_d/2$ and M_d, so that the difficulty of the instance scales with M_d. 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 random instances are labeled $\textsc{Inventory}(T, M_d, N_p, d)$ and $\textsc{Cash}(T, M_d, N_p, d)$ 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 20 random Inventory and Cash instances for each possible combination of the following values: $T \in \{5, 10, 20\}, M_d \in \{100, 1000, 10000\}$, $N_p \in \{5, 10\}$. We applied to each instance the following algorithms: an exact DP algorithm (label Exact), and the FPTAS for $\epsilon \in \{0.001, 0.01, 0.1\}$ 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 and using golden section search, see Sect. 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 tests – see below.

The maximum CPU time for each instance was set to 600 seconds. Results are reported in the Appendix, Figures 2 through 7.

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 roughly a factor 2 even on the smallest instances in our test set: $T = 5, M_d = 100, N_d = 5, d = 1$ (on Inventory, 0.18 seconds for Exact vs. 0.09 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 instances with cost functions of degree 1 the FPTAS with ApxSetConvex and $\epsilon = 1\%$ can be faster than Exact by 2 orders of magnitude, 3 orders of magnitude for $\epsilon = 10\%$ (160 seconds for Exact, 0.7 seconds for ApxSetConvex). 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. 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, except on large instances with linear costs (both Inventory and Cash). For $d = 1$, we verified applying a Wilcoxon rank-sum test at the 95% significance level that ApxSetSlope is faster than ApxSetConvex when $N_p = 5$, but the converse is true when $N_p = 10$. We explain this as follows: on large instances with linear costs and small N_p, the value function at each stage is almost linear, so ApxSetSlope finds small K-approximation sets without the overhead of ApxSetConvex. In all remaining situations, ApxSetConvex yields smaller K-approximation sets, achieving much better running time. The number of breakpoints in the optimal value function increases with N_p, therefore ApxSetConvex performs better as N_p grows. In most practical situations, we expect N_p to be large.

Finally, we analyze the quality of the approximated solutions: see Table 1. The computed solutions are very close to the optimum: even with $\epsilon = 10\%$, the solutions are on average only 1.47% (respectively 1.04%) away from the optimum for ApxSetConvex on Inventory (respectively Cash) problems. ApxSetConvex yields the largest errors because it computes smaller approximation sets. Still, its average recorded approximation factor is roughly 1 order of magnitude smaller than ϵ. The maximum error recorded among all instances and algorithms is 3.12%, much lower than the allowed 10%.

References

1. Bertsekas, D.P.: Dynamic Programming and Optimal Control. Athena Scientific, Belmont, MA (1995)
2. Bellman, R.: Dynamic Programming. Princeton University Press, Princeton, NJ (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, SIAM (2008) 700–709
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) (2009) 674–685
6. Nascimento, J., Powell, W.: Dynamic programming models and algorithms for the mutual fund cash balance problem. Management Science **56**(5) (2010) 801–815
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) (2009) 47–56
8. Murota, K.: Discrete Convex Analysis. SIAM, Philadelphia, PA (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) (2012) 429–446
10. Halman, N., Klabjan, D., Li, C.L., Orlin, J., Simchi-Levi, D.: Fully polynomial time approximation schemes for stochastic dynamic programs. Technical report, Sloan School of Management, Massachusetts Institute of Technology (August 2012)
11. Kiefer, J.: Sequential minimax search for a maximum. Proceedings of the American Mathematical Society **4**(3) (1953) 502–506

		ApxSet			ApxSetSlope			ApxSetConvex		
	ϵ	Mean	σ	Time	Mean	σ	Time	Mean	σ	Time
Inventory	0.1%	0.000	0.000	20.89	0.001	0.001	6.33	0.026	0.028	4.97
	1.0%	0.004	0.002	20.68	0.053	0.023	3.59	0.524	0.239	2.59
	10.0%	0.011	0.008	14.43	0.144	0.077	1.98	1.473	0.709	1.45
Cash	0.1%	0.000	0.000	46.00	0.002	0.005	3.97	0.011	0.007	3.01
	1.0%	0.001	0.003	29.84	0.031	0.037	3.52	0.167	0.076	2.36
	10.0%	0.034	0.035	15.07	0.323	0.360	2.92	1.039	0.852	1.78

Table 1. Sample mean and standard deviation of the quality of the returned solutions, computed as $100(f^{\text{apx}}/f^{*} - 1)$, and geometric mean of CPU times (in seconds). Only instances for which *all* algorithms return a solution before the time limit are considered.

A Definitions and proofs

Definition A.1. *Let X be a contiguous interval, and $Y(x)$ a nonempty contiguous interval for all $x \in X$. The set $X \otimes Y = \bigcup_{x \in X}(\{x\}, Y(x)) \subset \mathbb{Z}^2$ 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 $45°$.*

Proof of Thm. 3.2.

Proof. 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.

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 $x = \max\{w \in W : w \leq y\}$ and $x' = \min\{w \in W : w \geq y\}$. Clearly $\varphi(y) \geq \varphi(x)+\sigma_\varphi(x)(y-x)$. Notice that $\hat{\varphi}(y) = \varphi(x)+\frac{\varphi(x')-\varphi(x)}{x'-x}(y-x)$ and $\frac{\varphi(x')-\varphi(x)}{x'-x} \leq \frac{K(\varphi(x)+\sigma_\varphi(x)(x'-x))-\varphi(x)}{x'-x} = K\sigma_\varphi(x)+\frac{(K-1)\varphi(x)}{x'-x}$ by construction, so that:

$$\begin{aligned}\frac{\hat{\varphi}(y)}{\varphi(y)} &\leq \frac{\varphi(x)+\frac{\varphi(x')-\varphi(x)}{x'-x}(y-x)}{\varphi(x)+\sigma_\varphi(x)(y-x)} \leq \frac{\varphi(x)+K\sigma_\varphi(x)(y-x)+\frac{(K-1)\varphi(x)(y-x)}{x'-x}}{\varphi(x)+\sigma_\varphi(x)(y-x)} \\ &\leq \frac{\varphi(x)+K\sigma_\varphi(x)(y-x)+(K-1)\varphi(x)}{\varphi(x)+\sigma_\varphi(x)(y-x)} = K.\end{aligned}$$

Proof of Prop. 4.1.

Proof. Let $\underline{\varphi}(x) : \sum_{i=1}^{h-1} \mathrm{LB}_i(x)$. An example is shown in Figure 1. Clearly $\underline{\varphi}(x)$ is piecewise linear. Note that by convexity, LB_i underestimates φ over $[x_i, x_{i+1}]$, hence $\underline{\varphi}(x) \leq \varphi(x)\ \forall x$.

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 non differentiability 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

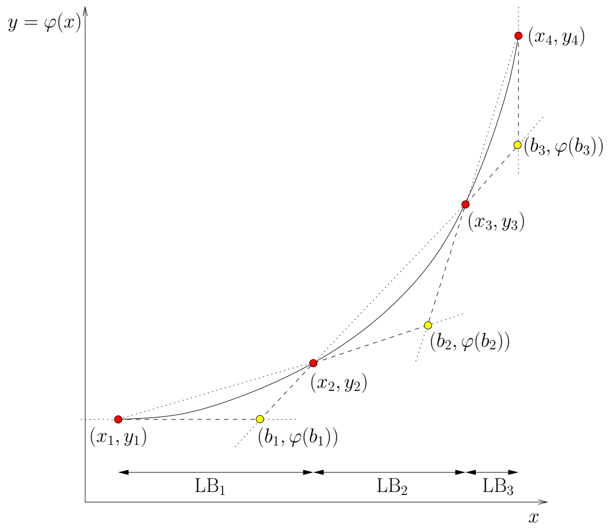

Fig. 1. Example of Prop. 4.1. We assume that the value of φ at x_1, x_2, x_3, x_4 is known. The dashed line represents the function $\underline{\varphi}$ defined as the sum of the LB_i functions. Each LB_i function is the maximum of the dotted lines below φ over the intervals $[x_1, x_2], [x_2, x_3], [x_3, x_4]$.

of $\frac{\textsc{Line}((x_j,y_j),(x_k,y_k),\cdot)}{\underline{\varphi}(\cdot)}$ is one of the points of non differentiability 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 non differentiability are contained in $E \cup B$. Hence,

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

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

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

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 of Sect. B.

Before proving Thm. 4.2, 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. Let ℓ, r and ℓ', r' be the endpoints of the search interval respectively at the beginning and at the end of an inner loop iteration, i.e. at lines 7 and 9. Note that $r' \leq r$, i.e. the right endpoint of the search interval is nonincreasing. We use the notation $\mathrm{prev}^n(x, D), \mathrm{next}^n(x, D)$ to indicate applications of the prev and next operators n times.

Proof of Thm. 4.2.

Proof. ApxSetConvex 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 ApxSetConvex to the left and to the right of the minimum.

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 first show that after $O(\log |D|)$ time 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. We call this an *interval shrinking iteration.* This implies $K^{|W|} < \frac{\sigma_\varphi^{\max}}{\sigma_\varphi^{\min}}$, from which the first part of the time and size complexity result follows.

Note that $r - \ell \geq 2(r' - \ell')$ unless $r' \leq \ell$. It follows that in $\log |D|$ iterations either the binary search converges and a new point w is added to W, or $r' \leq \ell$ at least once and no point is added to W.

Let $z' = \text{prev}(w, D)$ in the former case, $z' = \text{prev}(\ell, D)$ in the latter case. Then $\gamma_{\tilde{E}}(z', x) > K$ at some point during the algorithm for some set $\tilde{E}$ (otherwise line 8 would give $r \leq z'$). Let x_e be the point at which $\gamma_{\tilde{E}}(z', x)$ is maximized. Let $s_1 = \frac{\varphi(x) - \varphi(z')}{x - x'}$ and $s_2 = \frac{\varphi(x_e) - \varphi(z')}{x_e - z'}$. We have $\frac{\varphi(z') + s_1(x_e - z')}{\varphi(z') + s_2(x_e - x')} =$ $\textsc{Line}((z', \varphi(z')), (x, \varphi(x)), x_e)/\varphi(x_e) = \gamma_{\tilde{E}}(z', x) > K$. Because $\varphi(z') \geq 0$, it follows by simple algebraic manipulations that $K < \frac{\varphi(z') + s_1(x_e - z')}{\varphi(z') + s_2(x_e - z')} \leq s_1/s_2$. We observe that $\sigma_\varphi(\text{prev}^2(x_e, \tilde{E})) \leq s_2$ and $s_1 \leq \sigma_\varphi(\text{next}^2(x_e, \tilde{E}))$, by construction of x_e and because x_e maximizes $\gamma_{\tilde{E}}(z', x)$ (this is needed for the second inequality).

Note that $z' \leq \text{prev}^2(x_e, E)$. Suppose not: $\text{next}(z', E) \geq x_e$ would imply $\gamma_E(\text{next}(z', E), x) = 0$, which is a contradiction. It follows that $\sigma_\varphi(z') \leq s_2$.

We now distinguish a few cases, to show that each interval shrinking iteration runs in $O(\log |D|)$ time, adds at most one point to W, and reduces the slope of the function by a factor K.

1. A point w is added to W in the current interval shrinking iteration.
   (a) $z = x$, i.e. z is the last point added to W. Then it is easy to show that $s_1 \leq \sigma_\varphi(x)$. Thus, $K\sigma_\varphi(z') < \sigma_\varphi(z)$.
   (b) $z < x$. No other point $< x$ was added to W, therefore this can be considered part of the previous interval shrinking iteration, which must fall under the cases 2(a) or 2(b) below. Because those cases require at most $2 \log |D|$ iterations of the inner loop and they show $K\sigma_\varphi(z') < \sigma_\varphi(z)$, we are done.
2. No point w is added to W in the current interval shrinking iteration. Only $t < \log |D|$ iterations of the inner loop are performed before $r' \leq \ell$ for the first time.
   (a) If $z \geq \text{next}^2(x_e, E)$, $K\sigma_\varphi(z') < \sigma_\varphi(z)$ and we are done.
   (b) If $z < \text{next}^2(x_e, E)$: observe that $z \geq \text{prev}(x_e, E)$. Suppose not. Then $\gamma_E(z', x) > K$ regardless of points added to E in subsequent iterations (all these points are $< z$). This implies that a point is added to W after at most $\log |D|$ iterations of the inner loop, which is a contradiction. So $\text{prev}(x_e, E) \leq z < \text{next}^2(x_e, E)$. In this case, because $z' \leq \text{prev}^2(x_e, E)$

and $r' \leq \ell$, after an additional $\log|D|$ inner loop iterations we must have $r' \leq \ell$ again (otherwise, a point would be added to W). We can now repeat the argument used to define z' and x_e to find z'', x''_e with $z \geq \text{next}^2(z', E) \geq \text{next}^2(x''_e, E) \geq z''$. This shows $K\sigma_\varphi(z'') < \sigma_\varphi(z)$, and the interval shrinking iteration is complete in $2\log|D|$ iterations of the inner loop.

This concludes the first part of the proof.

Next, we show that after $O(\log|D|)$ time the search interval $[D^{\min}, \ldots, z]$ is reduced to $[D^{\min}, \ldots, z']$ 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 size and time complexity result 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 Prop. 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 Prop. 4.1 and the fact that whenever w is added to W, for all points y in the interval $[w, x]$ we have $\gamma_E(y, x) \leq K$.

We note that without lines 10–13 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 $O(t_\varphi \log_K \frac{\sigma_\varphi^{\max}}{\sigma_\varphi^{\min}} \log|D|)$. 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 winner.

B Floating point arithmetics

In this section we describe how our floating point implementation guarantees that the final result satisfies the desired approximation guarantee. Note that we also provide a version of our software that uses rational arithmetics, more as a proof of concept than as a practical tool.

We make the following assumptions.

1. All the functions involved in the problem (including the linear interpolation between function points) are evaluated at integer points only.
2. All integer numbers appearing in the problem are contained in $[-2^{53}, \ldots, 2^{53}]$, and all integers in this interval can be represented exactly as floating point numbers.
3. Each addition, multiplication or division introduces at most an additional ϵ_m relative error, where ϵ_m is the machine epsilon.
4. $\epsilon_m \ll 1$ so that all terms in the order of $O(\epsilon_m^2)$ can be considered zero.

Assumption 1 is verified by our algorithms. Assumptions 2 and 3 hold for standard double precision floating point arithmetics.

We now proceed to bound the error introduced by the calculations required by the FPTAS. In the remainder of this paper, we will refer to the error bound $\max\{\varphi(x_j), \varphi(x_k)\}/\min\{\varphi(x_j), \varphi(x_k)\}$ over an interval $[x_j, x_k]$ as the *step* bound, which is used by ApxSet. If we look at a typical iteration of Algorithm 2, there are three key operations: the computation of the value function $\bar{z}_t$ at the current stage (see (1)), the computation of a K-approximation set for $\bar{z}_t$, and the evaluation of the convex extension $\hat{z}_t$. We also observe that we are not required to have integer valued functions, hence we can compute $W_t \leftarrow$ ApxSetConvex$(\bar{z}_t, \mathcal{S}_t, K)$ at line 6 of Algorithm 2 (instead of $W_t \leftarrow$ ApxSet$(\lfloor M_t \bar{z}_t \rfloor, \mathcal{S}_t, K)$). At each stage, the relative errors introduced by the floating point operations is on top of the approximation error inherited from the subsequent stage.

Recall the expression of $\bar{z}_t$ in (1). Each value of $\hat{G}(I_t, x_t) + \hat{Z}_{t+1}(I_t, x_t)$ for $x_t \in \mathcal{A}_t(I_t)$ can be expressed as $\sum_{i=1}^{n_t} p_i g_t(I_t, x_t, d_{t,i}) + \sum_{i=1}^{n_t} p_i \hat{z}_{t+1}(f_t(I_t, x_t, d_{t,i}))$. Each term of the sum is the multiplication of two positive numbers, therefore the relative error of each term is bounded by $3\epsilon_m$. (We assume that the p_i's and the values of g_t are rational numbers that have to be rounded when stored as double precision floating point.) We use Kahan's summation algorithm [4] to carry out the sum. This introduces at most an additional $2\epsilon_m$ error because all numbers are positive, independent of the number of additions. Kahan's summation algorithm requires $O(1)$ space and runs in linear time in the number of summands. In total, the relative error in the computation of $\bar{z}_t$ at a point is bounded by $5\epsilon_m$.

The error introduced by computing the linear interpolation of φ between x_1, x_2 can be bounded as follows. Let x be the point at which we want to evalute the linear interpolation. Then we compute:

$$\frac{\varphi(x_1)(x_2 - x) + \varphi(x_2)(x - x_1)}{x_2 - x_1}.$$

By assumption, we can compute the differences exactly because they are integer valued. It follows that we are introducing a relative error bounded by $4\epsilon_m$, on top of the error for the computation of φ.

Finally, we bound the error introduced by the computation of the K-approximation set. Recall that we have to compute the ratio between two values, both obtained through linear interpolation, and compare it to K. Each value is affected by an error of at most $4\epsilon_m$, and the multiplication increases the bound to $9\epsilon_m$.

Putting everything together, to achieve an approximation factor of K with a call to ApxSetConvex(φ, D, K) using floating point arithmetics it is sufficient to replace 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 $K_1 K_2$ approximation of z_t.

[4] W. Kahan. Pracniques: further remarks on reducing truncation errors. *Communications of the ACM*, 8(1), 1965.

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 Prop. 4.1 up and down as suggested in Sect. 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.

Prop. 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:

$$\frac{(y_4(x_2-x_1)(x_3-x_4)+y_2x_1(x_3-x_4))-(y_1(x_2-x_1)(x_3-x_4)+y_1x_1(x_3-x_4))}{(y_2(x_3-x_4)+y_4(x_2-x_1))-(y_1(x_3-x_4)+y_3(x_2-x_1))} \tag{2}$$

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 affected by a relative error of at most $5\epsilon_m$. To bound the error of (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 $\rho := \sqrt{\frac{1}{\epsilon_m}}$. If the condition number of the operands does not satisfy the limit, we abort the computation of γ_E and revert to the step bound, which is numerically safe. If the limit is satisfied, it can be shown that the relative error of (2) is bounded by $11\sqrt{\epsilon_m}$. Then we do the following. Let x_e be the value of (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 step bound.

We remark that the value of ρ affects the practical performance of the algorithm. Our choice $\rho = \sqrt{\frac{1}{\epsilon_m}}$ shows good practical performance and yields easy-to-compute error bound. In our tests, for too small values of ρ we often had to resort to the step bound, resulting in a loss of performance.

To ensure that we do not exceed the approximation factor allowed, we set the rounding mode for the Floating Point Unit (FPU) 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. This can be easily done using the `fesetround` C function under Linux/Unix or the `_controlfp` C function under Windows.

C Selection of approximation factor

We now describe how the selection of the approximation factor at each stage is carried out. If K_t is the approximation factor used at stage $t = 1, \dots, T+1$, 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 $\prod_{t=1}^{T+1} K_t \leq 1 + \epsilon$. [3] suggests using $K_t = 1 + \frac{\epsilon}{2(T+1)}$ at each stage, because of the inequality $(1 + x/n)^n \leq 1 + 2x$ valid for $0 \leq x \leq 1$. 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 a guarantee on the actual approximation factor achieved. Note that $K'_t \leq K_t$. Then for $t = T+1, \dots, 1$ 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:

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

as desired.

D Generation of random instances

D.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_d, and the number of different possible demands in each period N_p. At each time period t, the set $\mathcal{D}_t$ of the N_p demand values is generated as follows: first we draw the maximum demand w_t at stage t uniformly at random in $[\lfloor M_d/2 \rfloor, \dots, M_d]$, then we draw the remaining $N_p - 1$ values in $[1, \dots, w_t]$ (without replacement). This method ensures that none of the instances we generate is too easy and that the difficulty scales with M_d. The probabilities with which demands occur are randomly assigned by generating N_p integers $q_i, i = 1, \dots, N_p$ in $[1, \dots, 10]$, and computing the probability of the k-th value of the demands as $q_k/(\sum_{i=1}^{N_p} q_i)$.

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 $[1, \dots, 20]$ for unit procurement cost, in $[1, \dots, 10]$ for storage costs, in $[1, \dots, 30]$ 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 Inventory(T, M_d, N_p, 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.

D.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_d, and the number of different possible deposit/withdrawal amounts in each time period N_p. At each time period t, the set $\mathcal{D}_t$ of the N_p deposit/withdrawal amount values is generated as follows: first we draw the difference w_t between the maximum and minimum values in $[\lfloor M_d/2 \rfloor, \dots, M_d]$, then we draw the minimum value m_t in $[-\lfloor M_d/2 \rfloor, \dots, -\lfloor M_d/2 \rfloor + M_d - w_t]$, finally we draw a set S_t of $N_p - 2$ values in from $[m_t, \dots, m_t + w_t]$ (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_d. 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 $\{0.5, 1, 1.5, \dots, 4\}$ for the opportunity cost, and in $\{0.5, 1, 1.5, \dots, 8\}$ for borrowing 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 stocks incurs a cost of $b_t x_t$ where x_t is the (positive) transaction amount and b_t is drawn uniformly at random in $\{0, 0.025, 0.05, \dots, 0.125\}$, and each transaction involving selling stocks incurs a cost of $-s_t x_t$ where x_t is the (negative) transaction amount and s_t is drawn uniformly at random in $\{0, 0.025, 0.05, \dots, 0.125\}$. We label the cash-flow management instances $\textsc{Cash}(T, M_d, M_p, d)$ following the usual convention.

E Additional figures

For each group of instances we do not report values for algorithm that fail to return a solution on 10 or more out of 20 instances within the allotted time.

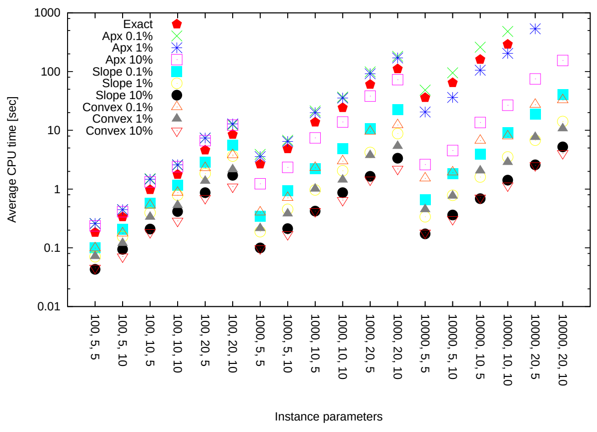

Fig. 2. Average CPU time on Inventory$(T, M_d, N_p, 1)$ instances. On the x-axis we identify the group of instances with the label M_d, T, N_p. The y-axis is on a logarithmic scale. The label of the approximation algorithms indicates the value of ϵ and the subroutine used to compute the K-approximation sets, namely: "Apx" uses ApxSet, "Slope" uses ApxSetSlope, "Convex" uses ApxSetConvex.

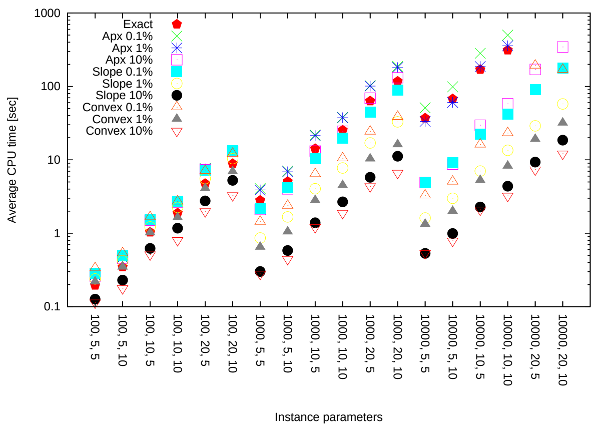

Fig. 3. Average CPU time on Inventory$(T, M_d, N_p, 2)$ instances. On the x-axis we identify the group of instances with the label M_d, T, N_p. The y-axis is on a logarithmic scale. The label of the approximation algorithms indicates the value of ϵ and the subroutine used to compute the K-approximation sets, namely: "Apx" uses ApxSet, "Slope" uses ApxSetSlope, "Convex" uses ApxSetConvex.

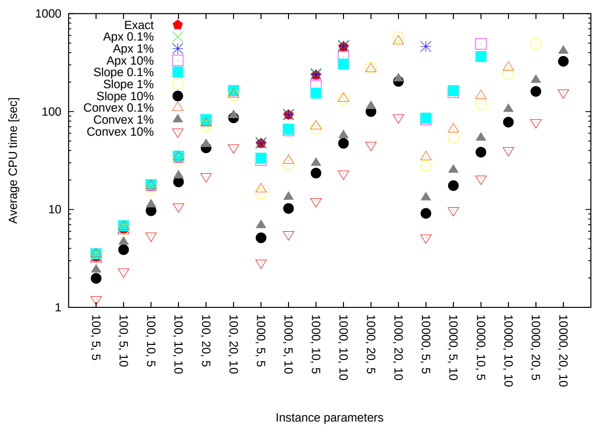

Fig. 4. Average CPU time on Inventory$(T, M_d, N_p, 3)$ instances. On the x-axis we identify the group of instances with the label M_d, T, N_p. The y-axis is on a logarithmic scale. The label of the approximation algorithms indicates the value of ϵ and the subroutine used to compute the K-approximation sets, namely: "Apx" uses ApxSet, "Slope" uses ApxSetSlope, "Convex" uses ApxSetConvex.

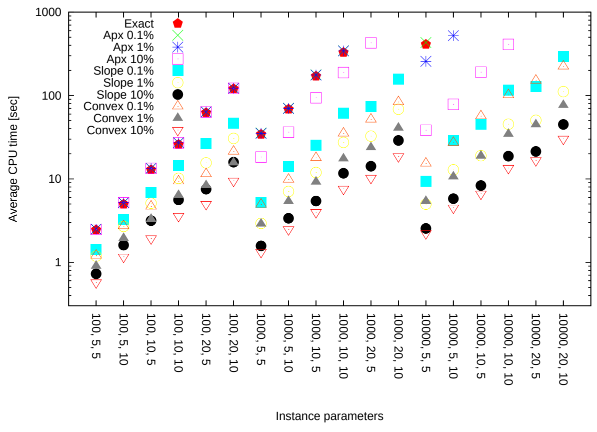

Fig. 5. Average CPU time on Cash$(T, M_d, N_p, 1)$ instances. On the x-axis we identify the group of instances with the label M_d, T, N_p. The y-axis is on a logarithmic scale. The label of the approximation algorithms indicates the value of ϵ and the subroutine used to compute the K-approximation sets, namely: "Apx" uses ApxSet, "Slope" uses ApxSetSlope, "Convex" uses ApxSetConvex.

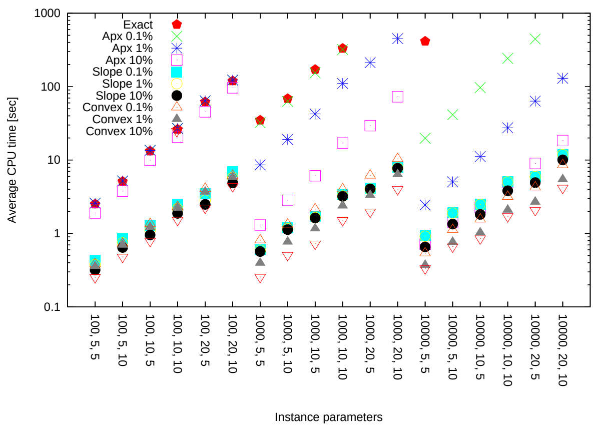

Fig. 6. Average CPU time on CASH$(T, M_d, N_p, 2)$ instances. On the x-axis we identify the group of instances with the label M_d, T, N_p. The y-axis is on a logarithmic scale. The label of the approximation algorithms indicates the value of ϵ and the subroutine used to compute the K-approximation sets, namely: "Apx" uses APXSET, "Slope" uses APXSETSLOPE, "Convex" uses APXSETCONVEX.

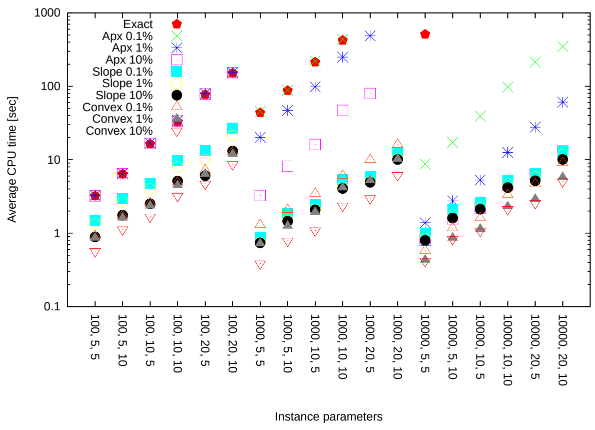

Fig. 7. Average CPU time on CASH$(T, M_d, N_p, 3)$ instances. On the x-axis we identify the group of instances with the label M_d, T, N_p. The y-axis is on a logarithmic scale. The label of the approximation algorithms indicates the value of ϵ and the subroutine used to compute the K-approximation sets, namely: "Apx" uses APXSET, "Slope" uses APXSETSLOPE, "Convex" uses APXSETCONVEX.