Artificial Intelligence 329 (2024) 104086

Contents lists available at ScienceDirect

Artificial Intelligence

journal homepage: www.elsevier.com/locate/artint

Efficient optimal Kolmogorov approximation of random variables

Liat Cohen [a,b], Tal Grinshpoun [c,*], Gera Weiss [d]

[a] *Department of Computer Science, Ariel University, Ariel, Israel*
[b] *Department of Mathematics and Computer Science, University of Basel, Basel, Switzerland*
[c] *Department of Industrial Engineering and Management, Ariel University, Ariel, Israel*
[d] *Department of Computer Science, Ben-Gurion University of the Negev, Beer-Sheva, Israel*

A R T I C L E   I N F O

Keywords:
Discrete random variables
Kolmogorov approximation
One-sided approximation
Support size reduction
Data compression
Statistical distance measures
Task scheduling
Deadline constraints

A B S T R A C T

Discrete random variables are essential ingredients in various artificial intelligence problems. These include the estimation of the probability of missing the deadline in a series-parallel schedule and the assignment of suppliers to tasks in a project in a manner that maximizes the probability of meeting the overall project deadline. The solving of such problems involves repetitive operations, such as summation, over random variables. However, these computations are NP-hard. Therefore, we explore techniques and methods for approximating random variables with a given support size and minimal Kolmogorov distance. We examine both the general problem of approximating a random variable and a one-sided version in which over-approximation is allowed but not under-approximation. We propose several algorithms and evaluate their performance through computational complexity analysis and empirical evaluation. All the presented algorithms are optimal in the sense that given an input random variable and a requested support size, they return a new approximated random variable with the requested support size and minimal Kolmogorov distance from the input random variable. Our approximation algorithms offer useful estimations of probabilities in situations where exact computations are not feasible due to NP-hardness complexity.

1. Introduction

Discrete random variables are essential ingredients in many stochastic problem domains. Such variables commonly represent uncertainty in the form of probability distributions. However, the arithmetic of discrete random variables is more elaborate than standard arithmetic due to the potential growth of the set of possible values of the random variable, which is known as the *support* of the random variable. In fact, even the problem of summing a set of random variables is NP-hard, as proved in [1,2], and leads to an exponential growth of the support size. Consequently, various approaches for approximation of probability distributions are studied in the literature [3–7]. Those approaches vary in the types of random variables considered, how they are represented, and in the criteria used to evaluate the quality of the approximations.

The focus of this paper is on approximating random variables using the following approach: Given a random variable X, our aim is to compute a new random variable X' with a smaller range of possible values, denoted as $|X'| < |X|$. The main objective is to find X' in such a way that if we replace X with X' in the subsequent calculations, the resulting outcome closely resembles what we would have obtained using X. To clarify, the support of X, denoted $\mathrm{support}(X)$, represents the set containing all possible

* Corresponding author.
E-mail address: talgr@ariel.ac.il (T. Grinshpoun).

https://doi.org/10.1016/j.artint.2024.104086
Received 27 December 2022; Received in revised form 11 December 2023; Accepted 25 January 2024

Available online 1 February 2024

realizations of random variable X that have a non-zero probability of occurrence; $|X|$ represents the cardinality of that set, i.e., the size of the support. The support size is crucial in assessing the complexity of a variable, as it is proportional to the size of the table typically used as a data structure to represent discrete random variables.

The concept introduced in the preceding paragraph shares a resemblance to the prevalent practice of rounding intermediate numerical results during lengthy computations. This practice becomes essential when working with irrational numbers, such as $\sqrt{2} \approx 1.414$, and is also commonly employed to optimize time and memory usage, as seen in financial computations, where $24.456 \approx 24.46$ after rounding.

The underlying similarity between rounding and shrinking the support of a random variable lies in their capacity to reduce the memory required to represent the object at hand. However, this reduction comes at the cost of introducing some level of inaccuracy to the computation. For variables with infinite support and irrational numbers, rounding becomes a necessary strategy to manage the representation efficiently. Additionally, when dealing with growing support, applying rounding techniques can save valuable computation time. By making these trade-offs, we strike a balance between memory conservation and computational efficiency.

While in the context of real numbers the notion of approximation, i.e., the distance (or rounding error) between two numbers, is clear, it is not so when dealing with random variables. There are various different distances known in the literature, such as the Kolmogorov distance [8] and the Wasserstein distance (Earth mover's distance) [9]. The choice of a specific metric to use depends on the application at hand. In this work, we use the Kolmogorov distance which is commonly used for comparing random variables in statistical practice and literature. As discussed later in this section, our work is motivated by the problem of estimating the probability of meeting deadlines, for which the Kolmogorov distance is very suitable.

Given two random variables X and X' whose cumulative distribution functions (CDF) are F_X and $F_{X'}$, respectively, the Kolmogorov distance between X and X' is $d_K(X, X') = \sup_t |F_X(t) - F_{X'}(t)|$ (see, e.g., [10]). The variable X' is considered a good approximation of X if $d_K(X, X')$ is small. This distance is the basis for the often-used Kolmogorov-Smirnoff test for comparing a sample to a distribution or two samples to each other.

In this paper, we consider two types of approximation. First, the traditional Kolmogorov distance, which we term as the *two-sided* Kolmogorov distance. The term "two sides" corresponds to the fact that for each t, the CDF value of the approximated random variable X' might be either smaller or bigger than that of the original random variable X, i.e., $F_X(t) \geq F_{X'}(t)$ or $F_X(t) \leq F_{X'}(t)$, where F_X is the CDF of X. Second, we also consider a distance in which for each t only $F_X(t) \leq F_{X'}(t)$ is allowed. We term this as the *one-sided* Kolmogorov distance. In what follows, we explain how the one-sided approximation comes in handy in applications.

Various stochastic problems in artificial intelligence, and more specifically, in the decision-making and planning literature, involve deadlines [11–16,7]. In these problems, the goal is to estimate the probability that a random variable gets a value that is below a given threshold (a deadline). If the random variable is X and the deadline is T, this is nothing more than evaluating the CDF, $F_X(T)$. This computation may not be straightforward in cases where the distribution of X is not given explicitly, e.g., X is specified as s sum of random variables.

One of the motivations for this research was to find ways to estimate the probability of meeting deadlines in situations that involve multiple random variables, such as those described in previous studies [13–16,7]. Specifically, we were interested in situations where X represents the probability distribution of the time it takes to complete a complex schedule, and it is not practical to maintain the full table of X's probability mass function. In these cases, we want to use an alternative random variable X' whose probability mass table is much smaller than X's (usually, the size is specified) but still, as accurately as possible, reflects the probabilities for meeting deadlines as compared to X.

The Supplier Assignment for Meeting a Deadline (SAMD) problem [17] was another key motivation for this research. This problem involves assigning suppliers to tasks in large projects with the goal of maximizing the probability of meeting the project deadline. In real-world projects, tasks must be completed by a single supplier with the necessary skills to perform the task, and the time it takes for each supplier to complete their tasks is typically uncertain and follows a known probability distribution. The SAMD problem involves finding the optimal way to assign suppliers to tasks to maximize the chances of meeting the overall project deadline, given this uncertainty about task completion times.

The relationship between the Kolmogorov distance and the deadline problem is clear. The Kolmogorov distance between two probability distributions, F_X and $F_{X'}$, measures how similar they are in terms of estimating probabilities. In the context of deadlines, $F_X(T)$ represents the probability of meeting the deadline T, so the distance between F_X and $F_{X'}$ tells us how close X and X' are in terms of estimating the probability of meeting deadlines. It is important to use the one-sided version of the Kolmogorov distance in this context, as we commonly want to be conservative in our estimations. The need for such *pessimistic* estimations means that it is acceptable to *underestimate* the probabilities of meeting deadlines, but overestimation is not allowed. Conversely, some situations, such as admissible heuristics, require *optimistic* estimations, in which it is only acceptable to *overestimate* the probabilities. In Section 4, we provide example applications for both underestimation and overestimation. For simplicity, unless otherwise stated, we hereinafter relate to overestimation as the default direction of one-sided Kolmogorov distance.

To define the specific type of approximation that we are targeting, we introduce the term "optimal m-approximation" as follows:

Definition 1. *A random variable X' is an optimal m-approximation of a random variable X if $|X'| \leq m$ and there is no random variable X'' such that $|X''| \leq m$ and $d_K(X, X'') < d_K(X, X')$.*

For the one-sided case we define "optimal m-one-sided-approximation":

Definition 2. *A random variable X' is an optimal m-one-sided-approximation of a random variable X if $|X'| \leq m$ and $F_X(t) \leq F_{X'}(t)$ for all t and there is no random variable X'' such that $|X''| \leq m$ and $F_X(t) \leq F_{X''}(t)$ for all t and $d_K(X, X'') < d_K(X, X')$.*

Using the above terms, the main contributions of this paper are two efficient algorithms for approximating random variables. The first algorithm takes a random variable X and a parameter m as inputs, and produces an optimal m-approximation of X. The second algorithm takes the same inputs but produces an optimal m-one-sided-approximation of X. For clarity, we present multiple algorithms for each type of approximation, with varying run-time and memory complexity. Our best-performing algorithms for both approximation types have log-linear run-time and constant memory. We provide theoretical analysis and experimental evaluation to support our results. In addition, for the m-one-sided approximation, we present an inferior algorithm in terms of run-time and memory, but it has the advantage of producing a set of optimal solutions rather than just a single optimal solution.

This article presents the results of two previous conference papers and includes new findings that have not been published before. The first optimal one-sided Kolmogorov approximation algorithm, which generates multiple solutions, was presented at the AAAI 2018 conference [18]. The efficient optimal algorithms for the two-sided variant were presented at the AAAI 2019 conference [19]. The efficient optimal algorithms for one-sided approximation are being published for the first time in this article. In addition to the algorithms, this article also includes an expanded review of related work, motivating examples, more elaborate proofs, and a comprehensive experimental evaluation. As a result, this article provides a comprehensive overview of the optimal Kolmogorov approximation problem and efficient methods for solving it.

The structure of the remainder of the paper is outlined as follows. In Section 2, we discuss how our work relates to other algorithms and problems in the literature. Section 3 focuses on the classical (two-sided) variant of Kolmogorov approximation and presents several optimal algorithms for that problem. In Section 4, we present two artificial intelligence applications in which the one-sided variant of the Kolmogorov approximation is required. Section 5 presents an algorithm for one-sided Kolmogorov approximation that returns a set of optimal solutions. Section 6 introduces more efficient algorithms for the one-sided variant. For all the algorithms presented, both one-sided and two-sided, we provide analyses of their properties and formal proofs. In Section 7, we conduct a thorough empirical evaluation for both the one-sided and two-sided variants. Finally, we conclude with a discussion of our findings and suggestions for future work in Section 8.

2. Related work

Given N independent (not necessarily identically distributed) random variables $X_1, \ldots, X_N$ and a number T, the problem of computing the probability $\sum_{i=1}^{N} X_i \leq T$ is considered fundamental, and as such has been investigated in the literature. The decision problem $(\sum_{i=1}^{N} X_i \leq T) > p$ has been proved to be NP-hard [1,2]; the counting problem to calculate the probability has been proved to be $\#P$-hard [20,7]. In the paper by Li and Shi [21], the authors focus on an integer T and present a fully polynomial-time approximation scheme (FPTAS) to estimate the probability up to a relative error of $(1 \pm \varepsilon)$ using run-time of $O(N^3/\varepsilon^2 \log N \log T)$. They use dynamic programming techniques similar to those commonly used for approximate counting knapsack solutions and therefore the complexity result is not only dependent on N and ε, but also on T. Different from the study of Li and Shi that focuses on a *relative* error, in the present paper our focus is on an *additive* error. Moreover, the complexity of our proposed algorithms, as will be elaborated later in the paper, is in the size of the support of the random variable and not in the number of random variable addends as in [21].

A similar notion to that of one-sided Kolmogorov approximation was proposed in the work by Cohen, Shimony, and Weiss [7], where a polynomial-time (additive error) approximation scheme was suggested; the scheme was applied to a problem of task trees with deadlines. Since the deadline problem of a complete task tree is NP-hard, they suggested an approximation algorithm. A key component in the suggested algorithm was a procedure for "trimming" the support of random variables. The Trim operator proposed in [7] gets as input a random variable, X, and an error bound, ε, and returns as output a new random variable, X', such that X' is a one-sided Kolmogorov approximation of X ($F_X(t) \leq F_{X'}(t)$ for all t) with up to ε Kolmogorov distance ($d_K(X, X') \leq \varepsilon$). This is similar to what we do in this paper. However, the Trim operator is not optimal in the sense that there may exist a different random variable X'' with the same support size as X' such that $d_K(X, X'') < d_K(X, X')$. In this paper, we show that it is possible to find an optimal approximation in polynomial time.

At the technical level, the problem examined in this paper bears a resemblance to the challenge of approximating a set of 2-D points using a step function. This study was originally motivated by query optimization and histogram constructions in database management systems [22–27], as well as computational geometry [28,29]. However, there are two notable distinctions between the problem studied in the context of databases and the problem discussed in this paper. The first difference lies in the fact that, when approximating random variables, the step function (which corresponds to the cumulative distribution function in our case) must be a monotonically increasing function that concludes with a value of one. The second difference is related to the handling of the first step in the approximation process. Unlike the database context, where a value is typically placed in the support of the approximated variable to generate the first step, in our case, this initial step is not counted. Consequently, the approach of simply adding a constant (e.g., two) to m cannot address these differences, as the presence of the first step and the monotonicity requirement impose constraints on the set of eligible step functions.

Another relevant prior work is the theory of Sparse Approximation (or Sparse Representation) which deals with sparse solutions for systems of linear equations. Given a matrix $D \in \mathbb{R}^{n \times p}$ and a vector $x \in \mathbb{R}^n$, the most studied sparse representation problem is finding

$$\min_{\alpha \in \mathbb{R}^p} \|\alpha\|_0 \text{ subject to } x = D\alpha$$

where $\|\alpha\|_0 := |\{i \in [p] : \alpha_i \neq 0\}|$ is the ℓ_0 pseudo-norm, counting the number of non-zero coordinates of α. This problem is known to be NP-hard with a reduction to subset selection problems. In these terms, also using the ℓ_∞ norm that represents the maximal coordinate and the ℓ_1 norm that represents the sum of the coordinates, our problem can be phrased as:

$$\min_{\alpha \in [0,\infty)^p} \|x - D\alpha\|_\infty \text{ subject to } \|\alpha\|_0 = m \text{ and } \|\alpha\|_1 = 1 \tag{1}$$

where D is the lower unitriangular matrix, x is related to X such that the ith coordinate of x is $F_X(x_i)$ where $\text{support}(X) := \{x_1 < \cdots < x_n\}$, and α is related to X' such that the ith coordinate of α is $f_{X'}(x_i)$. The functions F_X and $f_{X'}$ represent, respectively, the cumulative distribution function of X and the mass distribution function of X', i.e., the coordinates of x are positive and monotonically increasing and its last coordinate is one.

The present work is also related to the research on binning in statistical inference. Consider, for example, the problem of credit scoring [30] that deals with separating good applicants from bad applicants; there, the Kolmogorov–Smirnov statistic (KS) is used as a standard measure. The KS comparison is often preceded by a procedure called binning in which small values in the probability mass function are moved to nearby values. There exist many methods for binning [31–34]. In this context, our algorithm can be considered a binning strategy that provides optimality guarantees with respect to the Kolmogorov distance.

Another related problem is that of "order reduction", which was studied by Vidyasagar [5]. In that problem, an information-theoretic-based distance between discrete random variables is defined; following that, Vidyasagar investigates the problem of finding a random variable whose support is of size m and its distance from X is as small as possible (where X and m are given). The main difference between the problem studied by Vidyasagar and the problem studied in the present paper is that Vidyasagar considers a different notion of distance in which the entropy of the joint distribution is minimized. Vidyasagar proves that the problems of computing the distance (that he considers) between two probability distributions and computing the optimal approximation are both NP-hard problems; he proves that through reduction to nonstandard bin-packing problems. Subsequently, he develops efficient greedy approximation algorithms. In contrast, our study shows that there are efficient solutions to these problems when the Kolmogorov distance is considered.

Compact representations of distributions are mentioned in the literature in various contexts. The present study is also a continuation of the work by Pavlikov and Uryasev [6], in which a procedure to produce a random variable X' that optimally approximates a random variable X is presented. Their approximation scheme, achieved using convex and linear programming, is designed for a different notion of distance (called CVaR). The new contribution of the present work, in this context, is that our method is direct, not using linear or convex programming, thus allowing tighter analysis of time and memory complexity. Also, our method is designed to approximate the above-described Kolmogorov distance. A very common use of discrete approximation is for representing continuous random variables, as in the so-called, three-point approximations [35,36]. Other proposed approximation approaches include the approaches presented by Miller III and Rice [37] and by Hammond and Bickel [38], in which the support of a distribution is partitioned and the mean or the median of every partition is chosen to be in the support of the discrete approximation. Another approach is to compute a discrete variable that has the same moments as the original distribution [39]. An approach that is applicable when the input distribution is not specified completely involves resolving ambiguities in the definition of the discrete approximation of size m using maximization of the entropy [40].

The contribution of the present paper to the above literature is an approximation scheme similar to those proposed in [37] and [38]. The similarity is in the fact that our proposed algorithms are also based on a (consecutive) partition of the support of the input variable (discrete in our case). The difference is that we are proposing to take the minimum of each partition as a representative to be included in the support of the approximation. As with the moments preserving approximations [37,39], in which the motivation is that certain objective functions depend mostly on the first moments, our motivation is to discretize such that, e.g. probabilities of meeting deadlines are preserved. As we proceed to show in the next sections, a partition that yields an optimal approximation in this sense can be found in polynomial time. Our proposed algorithms improve on the Trim algorithm [7] in that they find an optimal approximation; yet, the run-time of Trim is linear in the size of the support [7], whereas the run-times of our algorithms are of higher complexity (with the best variant being log-linear).

A substantial body of work has also been done by Cicalese et al. in the area of Information Theory [41–45]. The motivation and applications are different than those presented in this paper and obviously strongly related to Information Theory; however, the goal is similar, which is suggesting an alternative, compact representation of a random variable while losing as little information as possible. In their most relevant work to ours, Cicalese and Vaccaro [43] consider the problem of efficiently finding the contiguous m-aggregation of maximum entropy. They define m-aggregation as follows: given a random variable X with support $x_1, \ldots, x_n$, probabilities $p = (p_1, \ldots, p_n)$, and an integer $1 \le m < n$, they say that $q = (q_1, \ldots, q_m)$ is contiguous m-aggregation of p if there exist indices $0 = i_0 < i_1 < \cdots < i_{m-1} < i_m = n$ such that for each $j = 1, \ldots, m$ it holds that $q_j = \sum_{k=i_{j-1}+1}^{k=i_j} p_k$. In other words, they search for a partition of the support of X, where each class i of the partition consists of consecutive elements of X. With the goal of finding the contiguous m-aggregation of maximum entropy, they design an optimal dynamic programming algorithm that runs in $O(n^2 m)$. They also provide more efficient, yet sub-optimal, greedy algorithms. However, their study differs from ours in two important aspects. First, they focus on the maximum entropy, whereas we are interested in the error itself and the Kolmogorov metric. Second, their approach does not consider the values of random variables as numerical quantities (e.g., task durations), and as such, it is not appropriate for approximating probabilities of events such as missing a deadline.

3. Algorithms for optimal Kolmogorov approximation

In this section, we focus on the two-sided Kolmogorov approximation, i.e., the m-approximation problem. We begin with presenting an algorithm for solving a problem that is dual to the m-approximation problem. First lets recall that a random variable X' is an optimal m-approximation of a random variable X if $|X'| \le m$ and there is no random variable X'' such that $|X''| \le m$ and $d_K(X, X'') < d_K(X, X')$. The input to solve this problem is X, m, and the output is a minimal difference, an error ε. While the dual problem of optimal m-approximation is: given a random variable X and $0 \le \varepsilon \le 1$, find a new random variable X' such that $d_K(X, X') \le \varepsilon$ and $|X'|$ is minimal. The input to solve this problem is X, ε, and the output is a minimal support size, m.

We assume that the input to Algorithm 1 that solves the dual problem is a representation of the variable X as a sorted CDF, i.e., as an array $X_{CDF} := \{(x_i, c_i)\}_{i=1}^n$ such that $c_i := Pr(X \le x_i)$ and $\text{support}(X) := \{x_1 < \cdots < x_n\}$. In line 1, we perform a single pass over X_{CDF} starting from the first index to find the last index where the cumulative distribution function of X is less than ε; this index is stored in the variable f. In line 2, we start from the last index of X_{CDF}, which is n, and seek in decreasing order for the last index where the cumulative distribution function of X is less than $1-\varepsilon$; this index is stored at the variable l. The last part of the algorithm is a pass on all elements of X_{CDF} between the indices f and l in order to construct the random variable X'. In lines 5-6, we check if the difference between the first element in the current set of indices (which we call b) and the last element in the current set of indices (which we call $e+1$) is less than 2ε. If yes, we add the pair $(x_b, (c_b + c_e)/2 - s)$ to the set S, and then, we increment e. In line 9, we add the last element to the set S, and finally, in line 10, we return a CDF representation of X. See running Example 3.

Algorithm 1: $dual(\{(x_i, c_i)\}_{i=1}^n, \varepsilon)$.

```
1  $f \leftarrow \min\{i : c_i \ge \varepsilon\}$
2  $l \leftarrow 1 + \max\{i : c_i \le 1 - \varepsilon\}$
3  $S \leftarrow \emptyset$, $b \leftarrow f$, $e \leftarrow f$
4  while $e < l$ do
5      while $c_{e+1} - c_b \le 2\varepsilon \wedge e < l$ do
6          $e \leftarrow e + 1$
7      $S \leftarrow S \cup \{(x_b, (c_b + c_e)/2)\}$
8      $b \leftarrow e + 1$, $e \leftarrow e + 1$
9  $S \leftarrow S \cup \{(x_l, 1)\}$
10 return A r.v. $X'$ as CDF such that $Pr(X' \le x) = c$ if there is $c$ such that $(x, c) \in S$.
```

Example 3. *When* $dual$ *is called with* $X = \{(1, 0.3), (2, 0.7), (3, 0.9), (4, 1)\}$ *and* $\varepsilon = 0.1$*, the following steps occur:*

1. *The variables f and l are set to 1 and 4, respectively, in lines 1 and 2.*
2. *After the first iteration of the main loop (line 4), the set S is updated to contain the element* $(1, 0.3)$*, and the values of b and e are both set to 2.*
3. *After the second iteration, S is updated to contain the elements* $(1, 0.3)$ *and* $(2, 0.8)$*, and b and e are both set to* 4.
4. *At the end of the function, the set S contains the elements* $(1, 0.3)$, $(2, 0.8)$, *and* $(4, 1)$.

Proposition 4. *If* $\{(x_i, c_i)\}_{i=1}^n$ *is such that* $c_i = Pr(X \le x_i)$ *and* $\text{support}(X) = \{x_1 < \cdots < x_n\}$ *then*

$$dual(\{(x_i, c_i)\}_{i=1}^n, \varepsilon) \in \underset{X' \in \bar{B}(X,\varepsilon)}{\arg\min} |X'|$$

where $\bar{B}(X, \varepsilon) := \{X' : d_K(X, X') \le \varepsilon\}$.

Proof. The number of level sets of the CDF of X' is minimal: (1) The first and the last sets are of maximal length; (2) By construction, an extension of any of the other sets to the right will generate a random variable whose Kolmogorov distance from X is bigger than ε; and (3) Extension of a level set to the left may either leave the number of level sets unchanged or combine it with the previous set, which will enlarge the Kolmogorov distance because it is equivalent to extending the previous set to the right. Thus, there is no random variable whose Kolmogorov distance from X is smaller or equal to ε and its support is smaller than $|X'|$. □

Proposition 5. $dual(\{(x_i, c_i)\}_{i=1}^n, \varepsilon)$ *runs in time* $O(n)$*, using* $O(n)$ *memory.*

Proof. This algorithm describes a single pass over $\{(x_i, c_i)\}_{i=1}^n$. Lines 1 and 2 are easy to follow; each takes $O(n)$ in the worst case. Lines 4-8 also describe a single pass since the counter e is updated to $e+1$ at most n times. Altogether, we get run-time complexity of $O(n)$. We are constructing the set S, which is of size n in the worst case; therefore, memory complexity is $O(n)$.[1] □

[1] By using an alternative implementation, one may suggest an $O(1)$ memory usage, for example by printing to the screen or to a file.

The binsApprox(X, m) algorithm, listed below, is the first algorithm we propose for approximating a random variable X with a new random variable X' such that the size of the support of X' is at most m. It is based on an algorithm previously proposed by D'iaz-B'anez and Mesa [28], but with some significant changes. The input is a random variable X and an integer m. The algorithm first converts X into a sorted cumulative distribution function representation, called X_{CDF}, which takes $O(n \log n)$ time. Then, it calculates the possible errors between X and all approximations of X, X', whose support is a subset of the support of X. These errors are sorted and a binary search is performed over them, running the $dual(X, \varepsilon)$ algorithm at each step to find the optimal m. If the size of the support of the resulting approximation is larger than m, the error is too small and the search is extended to the right side of the set of errors. If it is smaller, the error is too large and the search is extended to the left. If it is equal, the optimal m has been found. The search is run twice to handle the case where the optimal m is not within the set of possible errors. The optimal m'' is found in the second round by setting m to the value m' found in the first round.

Algorithm 2: binsApprox(X, m).

```
1  Let $\{(x_i, c_i)\}_{i=1}^n$ be such that $c_i = Pr(X \le x_i)$ and support$(X) = \{x_1 < \cdots < x_n\}$.
2  $E \leftarrow \emptyset$
3  for $i \leftarrow 1$ to $n-1$ do
4      for $j \leftarrow i+1$ to $n-1$ do
5          if $i = 1$ then
6              $E \leftarrow E \cup \{c_j\}$
7          $E \leftarrow E \cup \{(c_j - c_i)/2\}$
8      $E \leftarrow E \cup \{1 - c_i\}$
9  Let $e_1 < \cdots < e_{n'}$ be such that $E = \{e_1, \ldots, e_{n'}\}$
10 for $i \leftarrow 1$ to $2$ do
11     $l \leftarrow 1$, $r \leftarrow n'$, $k \leftarrow 1$, $k' \leftarrow 0$
12     while $k \ne k'$ do
13         $k' \leftarrow k$, $k \leftarrow \lceil (l+r)/2 \rceil$
14         $m' \leftarrow |dual(X, e_k)|$
15         if $m' < m$ then $l \leftarrow k$
16         if $m' > m$ then $r \leftarrow k-1$
17         if $m' = m$ then $r \leftarrow k$
18     $m \leftarrow m'$
19 return $dual(X, e_k)$
```

Proposition 6. binsApprox$(X, m) \in \underset{|X'| \le m}{\arg\min}\, d_K(X, X')$.

Proof. Let $E = \{Pr(x_1 < X \le x_2) : x_1, x_2 \in \text{support}(X) \cup \{-\infty, \infty\}, x_1 < x_2\}$ and let $E' = \{d_K(X, X') : |X'| \le m\}$. Since lines 10-18 describe a binary search to find the smallest $e \in E$ such that $|dual(X, e)| \le m$, we only need to prove that $\min E' \in E$.

Let $\varepsilon = \min E'$ and assume by contradiction that exists $\varepsilon \notin E$. Let $X' = dual(X, \varepsilon)$. By definition, there is $x \in \text{support}(X)$ in which $F_{X'}(x) - F_X(x) = \varepsilon$. Let $x_- = \max\{x' : x' \le x, x' \in \text{support}(X')\}$. If $x \in \text{support}(X')$ we have that $x_- = x$. Otherwise we have that $Pr(x_- < X' \le x) = 0$. In both cases we get that $F_{X'}(x_-) = F_{X'}(x)$. Plugging this in the previous inequality gives us that $F_{X'}(x_-) - F_X(x) = \varepsilon$. Since X' is constructed by the *dual* algorithm, $x_- \in \text{support}(X)$ and $\exists t \in \text{support}(X) : F_{X'}(x_-) = F_X(t)$. Therefore, $F_{X'}(x_-) - F_X(x) \in E$ in contradiction to $\varepsilon \notin E$. □

Proposition 7. *The* binsApprox(X, m) *algorithm runs in time* $O(n^2 log(n))$*, using* $O(n^2)$ *memory, where* $n = |X|$.

Proof. The first part of the algorithm, lines 2-8, takes $O(n^2)$ time to construct the set E. The second part, line 9, takes $O(n^2 \log n)$ time to sort the set E. The third part, lines 10-19, performs a binary search on the set E, and in each step of the search runs the *dual* algorithm, which takes $O(n \log n)$ time. This part runs twice, so it takes $O(2n \log n)$ time. In total, the algorithm takes $O(n^2 \log n + n^2 \log n + 2n \log n) = O(n^2 \log n)$ time and has a memory complexity of $O(n^2)$ for storing the set E. □

Towards an improved algorithm let us introduce the matrix $E_m = (e_{i,j})_{i,j=1}^{\infty}$ defined by:

$$e_{i,j} = \begin{cases} 1 - c_{n+1-j} & \text{if } j \le n \wedge i = n; \\ c_i & \text{if } i \le n \wedge j = n+1; \\ (c_i - c_{n+1-j})/2 & \text{if } i < n \wedge j \le n \wedge i+j \ge n; \\ 0 & \text{if } i+j < n; \\ 1 & \text{otherwise} \end{cases}$$

where $c_1, \ldots, c_n$ are as in the first line of Algorithm 2. It is easy to see that the set of values in matrix E_m are exactly the elements of the set E in Algorithm 2. An additional useful fact is that E_m is a sorted matrix:

Lemma 8. *If $i \le i'$ and $j \le j'$ then $e_{i,j} \le e_{i',j'}$.*

Proof. Since the c_is are monotonically increasing, $\forall i, j,\ 0 \le c_i - c_{n+1-j} \le c_i - c_{n+1-(j+1)}$ and $\forall i, j,\ 0 \le c_i - c_{n+1-j} \le c_{i+1} - c_{n+1-j}$; moreover, 0 is the minimal value of E_m. The elements in the last row and the last column in E_m are keeping the terms of the sorted matrix. It suffices to compare row n with row $n-1$ and show that $1 - c_{n+1-j} \ge (c_{n-1} - c_{n+1-j})/2$. After some manipulation we get that $2 - c_{n+1-j} \ge c_{n-1}$, which is true because $0 \le c \le 1$. Next, it is suffice to compare column n with only column $n+1$ and show that $c_i \ge (c_i - c_{n+1-n})/2$. After additional manipulation, we get that $2c_i \ge c_i - c_1$, which is true because $0 \le c_1 \le c_i \le 1$. □

The fact that matrix E_m is sorted allows us to apply a saddleback search algorithm, listed as Algorithm 3. The algorithm starts at the top right entry of the matrix $(e_{i,j})_{i=1..n, j=1..(n+1)}$ and traverses it as follows. If it hits an entry e such that $|dual(X, e)| \le m$ it goes left, otherwise it goes down. This assures that the minimal e such that $|dual(X, e)| \le m$ is visited after at most $n+1$ steps. The optimal random variable is found by brute-force search over the visited entries.

Algorithm 3: sdlbkApprox(X, m).

```
1 Let $\{(x_i, c_i)\}_{i=1}^{n}$ be such that $c_i = Pr(X \le x_i)$ and support$(X) = \{x_1 < \cdots < x_n\}$.
2 $i \leftarrow 1,\ j \leftarrow n{+}1,\ S \leftarrow \emptyset$
3 while $i < n \wedge j \ge 1$ do
4     $m' \leftarrow |dual(X, e_{i,j})|$
5     if $m' \le m$ then $j \leftarrow j - 1$
6     if $m' > m$ then $i \leftarrow i + 1$
7     if $m' \ge m$ then $S \leftarrow S \cup (m', e_{i,j})$
8 $e \leftarrow \min\{e : (m', e) \in \arg\max_{(m',e)\in S} m'\}$
9 return $dual(X, e)$
```

Proposition 9. sdlbkApprox$(X, m) \in \underset{|X'|\le m}{\arg\min}\, d_K(X, X')$

Proof. The algorithm traverses all the frontier between those es that satisfy $|dual(X, e)| \le m$ and those that do not satisfy it. Since all the entries that satisfy the condition are recorded in S and considered in the brute-force phase in line 7, the minimal satisfying e is found. □

Proposition 10. *The* sdlbkApprox(X, m) *algorithm runs in time $O(n^2)$, using $O(n)$ memory, where $n = |X|$.*

Proof. At each step of the loop in lines 3-6, either j decreases or i increases; thus, the loop can be executed at most $2n$ times. Since we execute *dual* once in a loop, the total time complexity is $O(n^2)$. Storing visited states in S (line 7) requires $O(n)$ memory. □

The problem with the sdlbkApprox algorithm is that it needs to run *dual* at every step, so it has a quadratic time complexity. Since we cannot find the required entry of the matrix in less than n steps, we can only reduce the complexity by proposing an algorithm that does not execute *dual* in all the steps but rather only in $\log(n)$ of them. Such an algorithm, based on Section 2.1 of [27], is listed as Algorithm 4. The algorithm, termed linApprox, maintains a set S of sub-matrices of $(e_{i,j})_{i=1..2^{\lceil \log_2(n)\rceil}, j=1..2^{\lceil \log_2(n+1)\rceil}}$. At each round of its execution, each sub-matrix is split into four, and then about three-quarters of the matrix is discarded. In the end, at most four scalar matrices remain to contain the index of the entry we seek. Note that the above procedure requires a square matrix of a power-of-two order. This can be achieved by padding the matrix E_m with zero values, which can, in the worst case, almost quadruple the matrix's number of entries; the padded matrix is termed E_{pm}. Obviously, the padding of the matrix does not affect the asymptotic complexity. However, when dealing with small random variables, the saddleback algorithm may turn out to be advantageous due to its use of the original (unpadded) matrix E_m. An interesting observation is that we do not physically store a matrix as a data structure but use the indices i, j to note the value in a specific location.

One technical note regarding the linApprox algorithm: lines 5 and 6 are looking for a median in a set; the formal definition of median states that in case of an even number of observations (i.e., there is no distinct middle value), the median is defined to be the arithmetic mean of the two middle values. In our case, we need the median to be part of the set and not an arithmetic mean of two middle values, and therefore, we define it to be the lower value for e^- and the upper value for e^+.

Theorem 11. linApprox$(X, m) \in \underset{|X'|\le m}{\arg\min}\, d_K(X, X')$.

Proof. In line 10, we only discard sub-matrices whose minimal entry (at the top left) is larger than an entry that we prefer. In line 12, we only discard sub-matrices whose maximal entry (at the bottom right) is smaller or equal to an entry that does not meet our condition. □

Algorithm 4: linApprox(X, m).

```
Let $\{(x_i, c_i)\}_{i=1}^n$ be such that $c_i = Pr(X \le x_i)$ and support$(X) = \{x_1 < \cdots < x_n\}$.
$S \leftarrow \{((1,1),(2^{\lceil \log_2(n) \rceil}, 2^{\lceil \log_2(n+1) \rceil}))\}$, $S' \leftarrow \emptyset$
while $S \neq S'$ do
    $S' \leftarrow S$, $S \leftarrow \bigcup_{s \in S} split(s)$
    $e^- \leftarrow median(\{e_{i_1,j_1} : ((i_1,j_1),(i_2,j_2)) \in S\})$
    $e^+ \leftarrow median(\{e_{i_2,j_2} : ((i_1,j_1),(i_2,j_2)) \in S\})$
    for $e \in \{e^-, e^+\}$ do
        $m' \leftarrow |dual(X, e)|$
        if $m' \le m$ then
            $S \leftarrow S \setminus \{((i_1,j_1),(i_2,j_2)) \in S : e_{i_1,j_1} > e\}$
        else
            $S \leftarrow S \setminus \{((i_1,j_1),(i_2,j_2)) \in S : e_{i_2,j_2} \le e\}$
$S'' \leftarrow \{(|dual(X, e_{i,j})|, e_{i,j}) : ((i,j),(i,j)) \in S\}$
$e \leftarrow \min\{e : (m', e) \in \arg\max_{(m',e) \in S''} m'\}$
return $dual(X, e)$

Function $split(((i_1,j_1),(i_2,j_2)))$ ()
    $j^- \leftarrow \lfloor (j_1 + j_2)/2 \rfloor$, $j^+ \leftarrow \lceil (j_1 + j_2)/2 \rceil$
    $i^- \leftarrow \lfloor (i_1 + i_2)/2 \rfloor$, $i^+ \leftarrow \lceil (i_1 + i_2)/2 \rceil$
    return $\{((i_1,j_1),(i^-,j^-)), ((i_1,j^+),(i^-,j_2)), ((i^+,j_1),(i_2,j^-)), ((i^+,j^+),(i_2,j_2))\}$
```

Theorem 12. *The* linApprox(X, m) *algorithm runs in time* $O(n\log(n))$*, using* $O(n)$ *memory, where* $n = |X|$.

Proof. The dimension of each matrix is halved in each round; thus, the loop is executed $O(\log(n))$ rounds. Since *dual* is called once in each round and a finite number of times in the end, the time complexity is $O(n\log(n))$. Note that there is no real matrix as a data structure but only i, j indices, and the evaluation of each cell in the abstract matrix is done locally, taking $O(1)$ memory. However, since we run *dual* and it is using $O(n)$ memory, then linApprox(X, m) is also in $O(n)$ memory.[2] □

4. Motivating examples for one-sided Kolmogorov approximation

Next, we will focus on the one-sided variant of the Kolmogorov approximation problem. We begin by presenting two example problems from artificial intelligence domains. These problems demonstrate the importance of the one-sided variant. We will then proceed, in later sections, to introduce algorithms for solving the one-sided Kolmogorov approximation problem.

4.1. Task trees with deadlines

Hierarchical planning is a well-established field in AI. Research on hierarchical planning goes back decades [46–48], but is still relevant nowadays [49,50]. A hierarchical plan is a method for representing problems of automated planning in which the dependency among tasks can be given in the form of networks.

We focus on hierarchical plans represented by task trees, in which the leaves are *primitive* actions (or tasks), and the internal nodes are either *sequence* or *parallel*. The plans we deal with are of stochastic nature, where the duration of a primitive action is given by a random variable. A sequence node denotes a series of tasks that should be performed consecutively, whereas a parallel node denotes a set of tasks that begin at the same time. A *valid* plan is one that is fulfilled before some given *deadline*, i.e., its *makespan* is less than or equal to the deadline. The objective in this context is to compute the probability that a given plan is valid, or more formally computing $P(X < T)$, where X is a random variable representing the makespan of the plan and T is the deadline. We note that deadlines correspond to the resource of *time*, where in fact this work may be applied (possibly with some adjustments) to other resources like fuel, cost, and memory consumption.

As noted above, resource consumption (task duration) is uncertain; it is described in the form of probability distributions in the leaf nodes. We assume that the distributions are independent but *not* necessarily identically distributed and that the random variables are discrete and have a finite support.

The problem of finding the probability that a task tree satisfies a deadline is known to be NP-hard. In fact, even the problem of summing a set of random variables is NP-hard [2].

Various hierarchical plans that can be represented as task trees with deadlines can be found in the JSHOP2 planner [51]. One example there is the domain of logistic problems consisting of packages that are to be transported by trucks or airplanes. Fig. 1 presents an example, which includes one parallel node (packages delivered in parallel), with all descendant task nodes being sequential plans. Given a specific task tree (generated by JSHOP2 or some other planner) and a deadline, our objective is to compute in a reasonable

[2] As mentioned in the complexity analysis of *dual*, by using an alternative implementation one may suggest an $O(1)$ memory usage for *dual*, e.g., by printing to the screen or to a file. Namely, allowing linApprox(X, m) use only $O(1)$ memory.

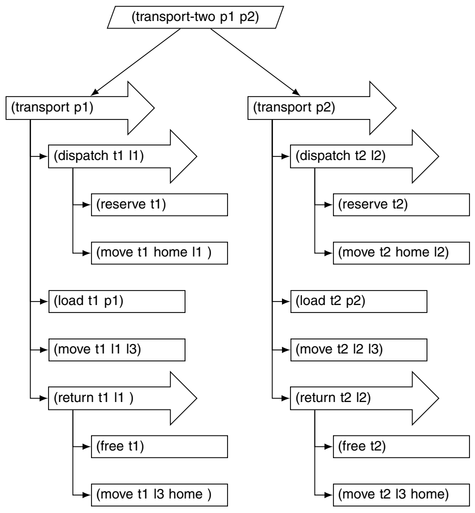

Fig. 1. A plan generated by the JSHOP2 algorithm. Arrow shapes represent sequence nodes, parallelograms represent parallel nodes, and rectangles represent primitive nodes.

time the probability that the plan represented by the task tree meets the deadline. These plans are used later on in our experimental evaluation, see Section 7. For this domain and type of application, we would be interested in *underestimating* the probabilities since our main goal is to provide a pessimistic prediction of staying below the deadline threshold.

4.2. Supplier assignment for meeting a deadline

Another motivating example is the Supplier Assignment for Meeting a Deadline problem (SAMD) [17]. The SAMD problem deals with assigning suppliers to tasks such that the chance to meet the overall deadline of the project is maximized. Most real-world projects have a deadline and consist of completing tasks. In this setting, each task needs to be executed by a single supplier, chosen from a subset of suppliers with the required proficiency to handle that task. The suppliers differ in their execution times, which are stochastically taken from known distributions. The problem is how to assign suppliers to tasks in a manner that maximizes the chance of meeting the overall project deadline. Several algorithms for solving SAMD have been proposed [17]: two simple algorithms for finding sub-optimal solutions and two optimal algorithms – one based on Branch-and-Bound and the other based on A^*. For the A^*-based algorithm, the authors proposed an admissible heuristic function, in which A^* is guaranteed to return an optimal solution; the heuristic performs multiple m-one-sided approximations. Experimentally, the authors compare all the developed algorithms over a range of distributions, including one created from real-world software projects. The results show that the A^*-based algorithm compares favorably with the other algorithms, delivering either better solution quality or better run-time in most cases. However, those optimal results are not convincing in terms of run-time as they use a less efficient method for m-one-sided approximation as part of the admissible heuristic. To this end, utilizing the most efficient algorithm for m-one-sided approximation, see Section 6, will enable solving the SAMD problem much more efficiently.

Note that the A^* algorithm for SAMD aims to find the maximal-cost path (contrary to the minimal-cost path that is commonly sought in A^*) because the problem herein is to maximize the probability. Therefore, to achieve an admissible (optimistic) heuristic regarding the probability of meeting the project deadline, we need to *overestimate* the probabilities.

To illustrate the difference between overestimation and underestimation, consider the following example:

Example 13. *The random variable* $X = \{(1,1/6),(2,1/3),(3,1/2),(4,2/3),(5,5/6),(6,1)\}$ *represents the CDF of a standard dice. If we underestimate it with a variable of size* $m = 3$ *we get* $X' = \{(2,1/3),(4,2/3),(6,1)\}$. *If we overestimate it with the same* m, *we get* $X' = \{(1,1/3),(3,2/3),(5,1)\}$. *In both cases, the approximation error is* $\varepsilon = 1/6$.

5. Generation of all one-sided Kolmogorov approximations with subset support

We introduce the first algorithm for solving the one-sided Kolmogorov approximation problem, which we have named OptTrim. While other algorithms with higher efficiency will be presented in Section 6, OptTrim stands out for two reasons. Firstly, it uses a unique method involving a reduction to the shortest path problem, distinct from the approaches of the other algorithms. Secondly, OptTrim is able to produce multiple optimal solutions for the m-one-sided Kolmogorov approximation, in contrast to the single solutions provided by the algorithms in Section 6. Specifically, OptTrim generates all solutions whose support is a subset of the support of the original random variable.

Recall Definition 2. In what follows, we will relate to the quality of the *optimal* m-one-sided Kolmogorov approximation of random variable X as $\varepsilon^*(X,m)$, i.e., $\varepsilon^*(X,m)$ represents the best-possible $d_K(X,X')$. We also provide the following notation for representing an m-one-sided-approximation relation between two random variables X and X':

Definition 14. *For two discrete random variables X and X', $X \preceq_{\varepsilon,m} X'$ means that $d_K(X, X') \le \varepsilon$ and $|\operatorname{support}(X')| \le m$.*

Subsequently, $X \preceq_{\varepsilon^*,m} X'$ represents the optimal m-one-sided Kolmogorov approximation, i.e., an approximation with distance $\varepsilon^*(X,m)$. Therefore, the algorithmic problem we focus on is: Given a random variable X with finite support and a natural number m, find a random variable X' such that $X \preceq_{\varepsilon^*,m} X'$.

We proceed to describe the OptTrim algorithm. Its main idea is the observation that we can focus the search for an optimal approximation of a variable X on a small set of candidate variables that we call consecutive approximations of X, defined as follows.

Definition 15. *A partition $P = \{B_1, \dots, B_n\}$ of a set $S \subseteq \mathbb{R}$ is called* consecutive *if $B_i = [\min B_i, \max B_i] \cap S$ for all i.*

Definition 16. *We say that a random variable X' is a* consecutive approximation *of a random variable X if there is a consecutive partition $P = \{B_1, \dots, B_n\}$ of* $\operatorname{support}(X)$ *such that the probability mass function (PMF) of X' is*

$$f_{X'}(t) = \begin{cases} Pr(X \in B_i) & \text{if } t = \min(B_i) \text{ for some } i; \\ 0 & \text{otherwise.} \end{cases}$$

In terms of CDFs, for a given t, we identify the B_i, $1 \le i < n$, if exists, such that $t \in [\min(B_i), \min(B_{i+1}))$ and define $F_{X'}(t) = F_X(\max(B_i))$. If $t < \min(B_1)$ or $t \ge \min(B_n)$, we define $F_{X'}(t) = 0$ and $F_{X'}(t) = 1$, respectively.

Example 17. *To demonstrate the above notions, we list two possible consecutive approximations of the random variable X defined by its PMF, as follows.*

$$f_X(t) = \begin{cases} 1/3 & \text{if } t = 1 \text{ or } t = 2; \\ 1/6 & \text{if } t = 3 \text{ or } t = 4; \\ 0 & \text{otherwise.} \end{cases}$$

We have $\operatorname{support}(X) = \{1, 2, 3, 4\}$. *One consecutive partition of it is $P = \{\{1, 2\}, \{3, 4\}\}$. The corresponding consecutive approximation is the random variable X_P whose PMF is:*

$$f_{X_P}(t) = \begin{cases} 2/3 & \text{if } t = 1; \\ 1/3 & \text{if } t = 3; \\ 0 & \text{otherwise.} \end{cases}$$

One may also define a different consecutive partition $P' = \{\{1, 2, 3\}, \{4\}\}$ that gives $X_{P'}$ whose PMF is:

$$f_{X_{P'}}(t) = \begin{cases} 5/6 & \text{if } t = 1; \\ 1/6 & \text{if } t = 4; \\ 0 & \text{otherwise.} \end{cases}$$

Alternatively, one can use the CDF notation for $F_{X_P}(t)$ and $F_{X'_P}(t)$ as follows:

$$F_{X_P}(t) = \begin{cases} 0 & \text{if } t < 1; \\ 2/3 & \text{if } 1 \le t < 3; \\ 1 & \text{if } t \ge 3. \end{cases} \qquad F_{X_{P'}}(t) = \begin{cases} 0 & \text{if } t < 1; \\ 5/6 & \text{if } 1 \le t < 4; \\ 1 & \text{if } t \ge 4. \end{cases}$$

The next theorem establishes that there is an optimal approximation that is also consecutive. It is key to the correctness proof of the OptTrim algorithm that efficiently scans all possible consecutive approximations to find an optimal one, as we will elaborate on later.

Theorem 18. *For any discrete random variable X and any $m \in \mathbb{N}$, there exists a consecutive approximation X_P of X such that $X \preceq_{\varepsilon^*,m} X_P$.*

Proof. Let X' be such that $X \preceq_{\varepsilon^*,m} X'$. Specifically, for all t,

$$F_X(t) \le F_{X'}(t) \le F_X(t) + \varepsilon^* \tag{2}$$

The proof is constructive and consists of two steps: (1) we first construct a variable X'' from X' that approximates X as X' does, i.e., $X \preceq_{\varepsilon^*,m} X''$, but also has the property that its support is a subset of the support of X; (2) then, from X'', we construct another random variable, X''', that in addition to being an approximation of X with the same parameters is also equal to X_P for some consecutive partition P.

Assume that $t_1, \dots, t_n$ are all the elements in the support of X in ascending order. Define the random variable X'' by

$$f_{X''}(t) = \begin{cases} Pr(X' \leq t_1) & \text{if } t{=}t_1; \\ Pr(t_{i-1} < X' \leq t_i) & \text{if } t{=}t_i \text{ for some } i{\neq}1; \\ 0 & \text{otherwise.} \end{cases}$$

In words, given X and X', we construct X'' by defining $f_{X''}(t_i) = Pr(t_{i-1} < X' \leq t_i)$ for all $i = 1, \dots, n$ (using $t_0 = -\infty$) and $f_{X''}(t) = 0$ for all other ts. Note that $Pr(t_{i-1} < X' \leq t_i)$ may be zero in which case $t_i \notin \text{support}(X'')$.

We will show now that: (1) $\text{support}(X'') \subseteq \text{support}(X)$; (2) $X \preceq_{\varepsilon^*,m} X''$. Since we only assign a non-zero probability to $f_{X''}(t)$ if $t = t_1$ or if $t = t_i$ for some i, i.e., only if t is in the support of X, we have that $\text{support}(X'') \subseteq \text{support}(X)$. Furthermore, if $t_i \in \text{support}(X'')$ then $Pr(t_{i-1} < X' \leq t_i) \neq 0$ which means that there is some $t_{i-1} < t' \leq t_i$ such that $t' \in \text{support}(X')$. To also handle the case where $i = 0$, we denote $t_{-1} = -\infty$. This (unique) mapping gives us that $|\text{support}(X'')| \leq |\text{support}(X')| \leq m$. To complete the proof of the properties of X'', we will show now that $F_X(t) \leq F_{X''}(t) \leq F_{X'}(t)$ for all t by examining the different ts as follows:

(1.1) Case $t < t_1$**:** $F_{X''}(t) = F_X(t) = 0$. Since $F_{X'}(t) \geq 0$ for all t, we get that $F_X(t) \leq F_{X''}(t) \leq F_{X'}(t)$.
(1.2) Case $t = t_i$**:** $F_{X'}(t) = F_{X''}(t)$ and $F_X(t) \leq F_{X'}(t)$ by Eq. (2).
(1.3) Case $t_{i-1} < t < t_i$**:** $F_{X''}(t) = F_{X''}(t_{i-1})$ and $F_X(t) = F_X(t_{i-1})$. Since we already have that $F_X(t_{i-1}) \leq F_{X''}(t_{i-1}) \leq F_{X'}(t_{i-1})$ from Case 1.2, we get that $F_X(t) \leq F_{X''}(t) \leq F_{X'}(t_{i-1})$. By monotonicity of CDF, $F_{X'}(t_{i-1}) \leq F_{X'}(t)$ therefore $F_X(t) \leq F_{X''}(t) \leq F_{X'}(t)$.
(1.4) Case $t > t_n$**:** $F_X(t) = F_{X''}(t) = 1$ and, by Eq. (2), since CDFs are always bounded by one, also $F_{X'}(t) = 1$.

From the four different cases of t, as we already established that $|\text{support}(X'')| \leq m$, we get that $X \preceq_{\varepsilon^*,m} X''$.

Therefore,

$$F_X(t) \leq F_{X''}(t) \leq F_{X'}(t) \leq F_X(t) + \varepsilon^* \tag{3}$$

Let $s_1, \dots, s_k$ be the elements in the support of X'' in ascending order $k \leq m$. Define the random variable X'''

$$f_{X'''}(t) = \begin{cases} Pr(s_i \leq X < s_{i+1}) & \text{if } t{=}s_i \text{ for some } i{<}k; \\ Pr(X \geq s_k) & \text{if } t{=}s_k; \\ 0 & \text{otherwise.} \end{cases}$$

We will show that: (1) $X \preceq_{\varepsilon^*,m} X'''$; (2) There exists a partition P such that $X''' = X_P$. Again, we will show that $F_X(t) \leq F_{X'''}(t) \leq F_{X''}(t)$ for all t by examining the different values of t as follows:

(2.1) Case $t < s_1$**:** $F_{X'''}(t) = F_{X''}(t) = 0$ and $F_X(t) \leq F_{X''}(t)$ from Eq. (3).
(2.2) Case $t = s_i$**:** First, $F_X(t) \leq F_{X'''}(t)$ since $F_{X'''}(s_i) = F_X(s_i) + Pr(s_i < X < s_{i+1})$. Second we show that $F_{X'''}(t) \leq F_{X''}(t)$. Since $X \preceq_{\varepsilon^*,m} X''$, $F_X(s_i) + Pr(s_i < X < s_{i+1}) \leq Pr(X'' < s_{i+1})$. As $s_1, \dots, s_k$ is the support of X'', $Pr(X'' < s_{i+1}) = F_{X''}(s_i)$. By definition $F_{X'''}(s_i) = F_X(s_i) + Pr(s_i < X < s_{i+1})$. Together we get that $F_{X'''}(s_i) \leq F_{X''}(s_i)$. For the case $t = s_k$, the argument holds with the notation $k + 1 = \infty$.
(2.3) Case $s_{i-1} < t < s_i$**:** $F_{X''}(t) = F_{X''}(s_{i-1})$ and $F_{X'''}(t) = F_{X'''}(s_{i-1})$ therefore by Case 2.2, $F_{X'''}(t) \leq F_{X''}(t)$. Also, $F_X(t) \leq Pr(X < s_i) = F_{X'''}(t)$.
(2.4) Case $t > s_k$**:** $F_{X''}(t) = F_{X'''}(t) = 1$. Since CDFs are always smaller or equal to one, also $F_X(t) \leq 1$.

From the different cases of t, because $\text{support}(X'') = \text{support}(X''')$, we established that $X \preceq_{\varepsilon^*,m} X'''$. The next step is to prove that $X''' = X_P$, by presenting a partition P. As shown before, $\text{support}(X) = \{t_1, \dots, t_n\}$, $\text{support}(X''') = \{s_1, \dots, s_k\}$, and $\text{support}(X'') \subseteq \text{support}(X)$ since $\text{support}(X'')$ is equal to $\text{support}(X''')$ then $\text{support}(X''') \subseteq \text{support}(X)$. In addition, $\forall 0 \leq i \leq k, Pr(X''' = s_i) = Pr(s_i \leq X \leq s_{i+1})$ therefore we get that $X''' = X_P$ is a consecutive approximation of X with the partition $P = \{[s_i, s_{i+1}) \cap \text{support}(X) : i = 1, \dots, n-1\}$. □

To demonstrate how the above proof works, the following example shows one possible list of the random variables constructed in the different stages of the proof.

Example 19. *Consider the following two random variable X and X' such that $X \preceq_{\varepsilon^*,m} X'$ where $m = 3$:*

$$f_X(t) = \begin{cases} 1/3 & \textit{if } t = 1 \textit{ or } t = 2; \\ 1/6 & \textit{if } t = 3 \textit{ or } t = 4; \\ 0 & \textit{otherwise.} \end{cases} \qquad f_{X'}(t) = \begin{cases} 1/6 & \textit{if } t = 0.9; \\ 1/2 & \textit{if } t = 1; \\ 1/3 & \textit{if } t = 2; \\ 0 & \textit{otherwise.} \end{cases}$$

In this case $\varepsilon^* = 1/3$ and indeed $X \preceq_{\varepsilon^*,m} X'$. The steps in the proof of Theorem 18 transform X' to a partitioned random variable X''' with the same approximation qualities:

$$f_{X''}(t) = \begin{cases} 2/3 & \text{if } t = 1; \\ 1/3 & \text{if } t = 2; \\ 0 & \text{otherwise.} \end{cases} \qquad f_{X'''}(t) = \begin{cases} 2/3 & \text{if } t = 1; \\ 1/3 & \text{if } t = 3; \\ 0 & \text{otherwise.} \end{cases}$$

Note that this example shows that it may be that X'' is not a consecutive approximation of X. We only guarantee that $\text{support}(X'') \subseteq \text{support}(X)$ *and that $X \preceq_{\varepsilon^*,m} X''$. Only the variable X''', in addition to these properties, is guaranteed to be a consecutive approximation. In this case, with the partition $P = \{\{1,2\}\{3,4\}\}$.*

Theorem 18 gives us that there is a consecutive approximation that is also optimal. This observation allows to restrict the focus of the OptTrim algorithm to only search for the best consecutive approximation.

Chakravarty, Orlin, and Rothblum [52] proposed a polynomial-time method that, given certain objective functions (additive), finds an optimal consecutive partition. Their method involves the construction of a graph such that the (consecutive) set partitioning problem is reduced to the problem of finding the shortest path in that graph.

The OptTrim algorithm (Algorithm 5) starts by constructing a directed weighted graph G similar to the method of Chakravarty, Orlin, and Rothblum [52]. The nodes V consist of the support of X together with an extra node ∞ for technical reasons, whereas the edges E connect every pair of nodes in one direction (lines 1 and 2). The weight w of each edge $e = (i, j) \in E$ is determined by the probability of X to get a value between i and j, non-inclusive (lines 3 and 4), i.e., $w(e) = Pr(i < X < j)$. The values taken are non-inclusive since we are interested only in the error value. The source node of the shortest path problem at hand corresponds to the minimal value in $\text{support}(X)$, and the target node is the extra node ∞. The set of all solution paths in G, i.e., those starting at s and ending in ∞ with at most m edges, is called $paths(G)$. The goal is to find the path in $paths(G)$ with the lightest bottleneck (lines 5 and 6). This can be achieved by using the $Bellman - Ford$ algorithm with two tweaks. The first is to iterate the graph G in order to find only paths with lengths of at most m edges. The second is to find the lightest bottleneck as opposed to the traditional objective of finding the shortest path. This is performed by modifying the manner of “relaxation” to $bottleneck(x) = min[max(bottleneck(v), w(e))]$, done also in [53]. Consequently, we find the lightest maximal edge in a path of length $\le m$, which represents the minimal error, ε^*. X_P is then derived from the resulting path (lines 7 and 8).

Algorithm 5: OptTrim(X, m).

```
1 $S = \text{support}(X) \cup \{\infty\}$
2 $G = (V, E) = (S, \{(i, j) \in S^2 : j > i\})$
3 foreach $e = (i, j) \in E$ do
4     $w(e) = Pr(i < X < j)$
5 /* The following can be obtained, e.g., using the Bellman-Ford algorithm */
6 $l^* = \underset{l \in paths(G), |l| \le m}{\text{argmin}} \max\{w(e) : e \in l\}$
7 foreach $e = (i, j) \in l^*$ do
8     $f_{X'}(i) = Pr(i \le X < j)$
9 return $X'$
```

Now we proceed to prove the correctness (optimality) of the OptTrim algorithm.

Theorem 20. $X \preceq_{\varepsilon^*,m}$ OptTrim(X, m).

Proof. According to Theorem 18 there exists a consecutive partition P for which $X \preceq_{\varepsilon^*,m} X_P$. For every consecutive partition P there is a path l, $l \in paths(G), |l| \le m$, such that its corresponding partitioned random variable X_P satisfies $X \preceq_{\varepsilon,m} X_P$ where $\varepsilon = \max\{w(e) : e \in l\}$. By using, for instance, the Bellman-Ford algorithm (lines 5-6), we obtain the optimal path l^* containing the minimal edge among all maximal edges of all the paths in $paths(G)$. The optimal consecutive partition P^* associated with this “lightest” path l^*, results in $X' = X_{P^*}$ (lines 7-8). Thus, $X' = \text{OptTrim}(X, m)$ and $X \preceq_{\varepsilon^*,m} X'$. □

Next, we analyze the run-time and memory complexity of the OptTrim algorithm. Note that G is acyclic, therefore the shortest path can be found in $O(E + V)$ time using topological sorting [54].

Theorem 21. *The* OptTrim(X, m) *algorithm runs in time $O(n^3)$, using $O(n^2)$ memory, where $n = |\text{support}(X)|$.*

Proof. Constructing the graph G takes $O(n^3)$. The number of edges is $O(E) \approx O(n^2)$ and for every edge, the weight is at most the sum of all probabilities between the source node s and the target node ∞, $Pr(s < X < \infty)$, $O(n^3)$. The construction is also the only stage that requires memory allocation, specifically $O(E + V) = O(n^2)$. Finding the shortest path takes $O(m(E + V)) \approx O(mn^2)$. Since G acyclic finding shortest path takes $O(E + V)$. We only need to find paths of length $\le m$, which takes $O(m(E + V))$. Deriving the new random variable X' from the computed path l^* takes $O(mn)$. For every node in l^* (at most m nodes), calculating the probability $P(s < X < \infty)$ takes at most n. To conclude, the worst case run-time complexity is $O(n^3 + mn^2 + mn) = O(n^3)$ and memory complexity is $O(E + V) = O(n^2)$. □

Although both operators Trim and OptTrim return a one-sided Kolmogorov approximation for a given random variable, they set different parameters in the process. The Trim operator sets the error bound ε and provides guarantees regarding the support size, whereas OptTrim sets the support size and provides a minimal error. In order to compare the two operators we show a connection between the error bound ε and the support size m, this will come into effect in the empirical evaluation section.

Lemma 22. $\varepsilon^*(X, m) \leq 1/m$.

Proof. According to Lemma 2 in [1] $X' = \text{Trim}(X, 1/m)$. Lemma 3 in that paper states that for a given error ε the support size m is bounded by $1/\varepsilon$, thus $\varepsilon \leq 1/m$. Turning to the terminology of the present work, this means that $X \preceq_{1/m,m} X'$. Since $\varepsilon^*(X, m)$ is the minimal distance between any random variable with support size m and the random variable X, we get $\varepsilon^*(X, m) \leq 1/m$. □

Cohen, Shimony, and Weiss [1] use a slightly different notation for the error between X and X', $X \preceq_\varepsilon X'$, relating only to the approximation error, without the support size. The following theorem relates OptTrim to this notation:

Theorem 23. *If* $X' = \text{OptTrim}(X, 1/\varepsilon)$ *then* $X \preceq_\varepsilon X'$.

Proof. Following the optimality of OptTrim (Theorem 20), $X \preceq_{\varepsilon^*(X,1/\varepsilon),1/\varepsilon} X'$. Now, given Lemma 22, $X \preceq_{\varepsilon,1/\varepsilon} X'$, and consequently $X \preceq_\varepsilon X'$. □

The above properties enable a fair and sound comparison between the OptTrim and Trim operators. Following Lemma 22 we set the support size given as the input of OptTrim to $m = 1/\varepsilon$, whereas ε is the input of Trim. From Theorem 23 we establish that by setting the support size to be $1/\varepsilon$ we bound the error of OptTrim to at most ε, resulting in the same error guarantees (bounds) by both operators. It remains to determine the actual error resulting from each operator. Both the Trim and OptTrim are used as operators for achieving one-sided Kolmogorov approximation within the approximation algorithm. The following example illustrates the difference between the Trim and OptTrim operators.

Example 24. *Consider the variable X and its approximation, X_{Trim}, using* Trim *with $\varepsilon = 1/3$:*

$$f_X(t) = \begin{cases} 0.1 & \text{if } t \in \{1,2,3,4\}; \\ 0.2 & \text{if } t = 5; \\ 0.4 & \text{if } t = 6; \\ 0 & \text{otherwise.} \end{cases} \qquad f_{X_{\text{Trim}}}(t) = \begin{cases} 0.4 & \text{if } t = 1; \\ 0.2 & \text{if } t = 5; \\ 0.4 & \text{if } t = 6; \\ 0 & \text{otherwise.} \end{cases}$$

In this case we get that $d_K(X_{\text{Trim}}, X) = 0.3$. To improve this, using OptTrim *with $m = 3$, we construct X_{OptTrim} such that $d_K(X_{\text{OptTrim}}, X) = 0.2$:*

$$f_{X_{\text{OptTrim}}}(t) = \begin{cases} 0.3 & \text{if } t = 1; \\ 0.3 & \text{if } t = 4; \\ 0.4 & \text{if } t = 6; \\ 0 & \text{otherwise.} \end{cases}$$

Underestimation The analysis presented in this section overestimates the probability of meeting the deadline. In cases where one might be interested in underestimating the probability, the algorithm requires a few adjustments. All are straightforward, and no further mathematical analysis is needed. First, ∞ is replaced with $-\infty$; second, the graph is defined as $G := (V, E) = (S, \{(i, j) \in S^2 : j < i\})$; and third, the search starts from the maximal node with the goal being the minimal node. Apart from these adjustments, the rest of the algorithm remains unchanged.

6. Efficient optimal one-sided Kolmogorov approximation

In the previous section, we introduced the OptTrim algorithm, which provides m-one-sided approximations. However, its run-time complexity yields it inapplicable in complex scenarios, such as those described in Section 4. Therefore, we revisit the methodology used for the two-sided case presented in Section 3 in search of more efficient solutions to the m-one-sided-approximation problem. The main differences when considering one-sided approximation are in the *dual* algorithm and in the set E and its derived matrices E_m and E_{pm}. Once these required changes are applied, the algorithms binsApprox, sdlbkApprox, and linApprox can be easily adapted to their respective one-sided variations binsApproxOneSided, sdlbkApproxOneSided, and linApproxOneSided. In this section, we focus on the changes to $dual$ and E; full descriptions of the m-one-sided approximation algorithms are given in Appendix A.

First, we revisit $dual$ and present its one-sided variation, $dualOneSided$, in Algorithm 6.

Algorithm 6: $dualOneSided(\{(x_i, c_i)\}_{i=1}^n, \varepsilon)$.

```
1 $f \leftarrow 1, e \leftarrow 1$
2 while $e < n$ do
3     while $c_{e+1} - c_f \leq \varepsilon \wedge e < n$ do
4         $e \leftarrow e + 1$
5     $S \leftarrow S \cup \{(x_f, c_e)\}$
6     $f \leftarrow e + 1, e \leftarrow e + 1$
7 if $e = n$ then
8     $S \leftarrow S \cup \{(x_n, 1)\}$
9 return A r.v. $X'$ as CDF such that $Pr(X' \leq x) = c$ if there is $c$ such that $(x, c) \in S$.
```

To illustrate the operation of $dualOneSided$, consider the following example (based on Example 3):

Example 25. *In the $dualOneSided$ algorithm, when given the parameters $X = \{(1, 0.3), (2, 0.7), (3, 0.9), (4, 1)\}$ and $\varepsilon = 0.1$, the following occurs:*

1. *Line 1 sets $f = 1$ and $e = 1$.*
2. *After the first iteration of the main loop (line 2), $S = \{(1, 0.3)\}$ and $f = e = 2$.*
3. *After the second iteration, $S = \{(1, 0.3), (2, 0.7)\}$ and $f = e = 3$.*
4. *The third and final iteration results with $S = \{(1, 0.3), (2, 0.7), (3, 1)\}$.*

In this example, lines 7 and 8 are not used because the loop has already constructed a complete CDF.

Proposition 26. *If $p = \{(x_i, c_i)\}_{i=1}^n$ is such that $c_i = Pr(X \leq x_i)$ and $\text{support}(X) = \{x_1 < \cdots < x_n\}$ then*

$$dualOneSided(\{(x_i, c_i)\}_{i=1}^n, \varepsilon) \in \underset{X' \in \bar{B}(X,\varepsilon)}{\arg\min} |X'|$$

where $\bar{B}(X, \varepsilon) = \{X' : d_K(X, X') \leq \varepsilon$ and $F_X(x) < F_{X'}(x)$ for all $x\}$.

Proof. The CDF of X' has a minimal number of level sets: (1) The first set has the maximum length; (2) By design, any extension of the other sets to the right will produce a random variable with a Kolmogorov distance from X larger than ε. Therefore, there is no random variable whose Kolmogorov distance from X is equal to or less than ε and whose support is smaller than $|X'|$. Additionally, we maintain the invariant $f \leq e$ and, since X is given as a CDF, $c_f \leq c_e$ is maintained throughout the algorithm. □

Proposition 27. *$dualOneSided(\{(x_i, c_i)\}_{i=1}^n, \varepsilon)$ runs in time $O(n)$, using $O(n)$ memory.*

Proof. Similar to the proof of Proposition 5. □

The other change required for one-sided approximation is in the construction of the set E; we denote the modified set by $\overline{E}$. When we constructed the set E for two-sided approximation (Algorithm 2), we considered that the error can spread up and down to establish minimized two-sided error; for that reason, we divided each possible error by two. Now, in $\overline{E}$, we are handling one-sided errors; therefore, all possible distances are only shifted towards a single direction, so the division by two (line 7, Algorithm 2) is not needed. An outcome of this change is that the error calculated for the one-sided approximation is twice the error of the two-sided version. An additional change in $\overline{E}$ is that the complementing probability (line 8, Algorithm 2) is no longer required. These changes are depicted in Algorithm 7 in Appendix A.

Similarly to the case of E and E_m, $\overline{E}$ can also be transformed to its matrix form $\overline{E_m} := (e_{i,j})_{i,j=1}^{\infty}$:

$$e_{i,j} = \begin{cases} c_i & \text{if } i \leq n \wedge j = n+1; \\ c_i - c_{n+1-j} & \text{if } i < n \wedge j \leq n \wedge i+j \geq n; \\ 0 & \text{if } i+j < n; \\ 1 & \text{otherwise.} \end{cases}$$

As in the respective two-sided case, the set of values in matrix $\overline{E_m}$ are exactly the elements of the set $\overline{E}$. Moreover, $\overline{E_m}$ is a sorted matrix:

Table 1
The errors of OptTrim and linApproxOneSided vs. Trim on randomly generated random variables with original support size $n = 100$.

m	OptTrim/linApproxOS	Trim	Relative error
2	0.491	0.493	0.4%
4	0.242	0.247	2.1%
8	0.118	0.123	4.4%
10	0.093	0.099	6%
20	0.043	0.049	15%
50	0.013	0.019	45.4%

Lemma 28. *If $i \leq i'$ and $j \leq j'$ then $e_{i,j} \leq e_{i',j'}$.*

Proof. Since the c_is are monotonically increasing, $\forall i, j,\ 0 \leq c_i - c_{n+1-j} \leq c_i - c_{n+1-(j+1)}$ and $\forall i, j,\ 0 \leq c_i - c_{n+1-j} \leq c_{i+1} - c_{n+1-j}$, moreover, 0 is the minimal value of $\overline{E_m}$. The elements in the last row and the last column in $\overline{E_m}$ are keeping the terms of the sorted matrix. It is suffice to compare column n with only column $n+1$ and show that $c_i \geq (c_i - c_{n+1-n})$. After some manipulation, we get that $c_1 \geq 0$ which is true. □

A similar padding process to that of E_{pm} can also be applied to $\overline{E_m}$, resulting in $\overline{E_{pm}}$. The matrices $\overline{E_m}$ and $\overline{E_{pm}}$ are respectively used in the algorithms sdlbkApproxOneSided and linApproxOneSided, see Algorithms 8 and 9 in Appendix A.

Underestimation The analysis presented in this section overestimates the probability of meeting the deadline. In cases where one might be interested in underestimating the probability, the algorithms require a few adjustments. All are straightforward, and no further mathematical analysis is needed. The idea is to find the correct differences (the set E) but also adjust $dualOneSided$ (Algorithm 6). Details of the required adjustments for underestimation are given in Appendix B.

7. Experimental evaluation

The experiments in this section evaluate the performance of the suggested approximation algorithms, both one-sided and two-sided, on various domains. All the algorithms were implemented in Python and the experiments were executed on hardware comprised of an Intel(R) Core(TM) i7-10610U CPU @ 1.80GHz, 16GB System Memory. The Trim algorithm was implemented as in [1].

The first set of experiments focuses on single-step approximation, meaning that we evaluate the accuracy and run-time in practice when applying the approximation only once on a single random variable. The Second set of experiments focuses on the use of approximation as an intermediate computation step when performing convolutions to calculate the sum of random variables; this is required when solving the deadline problem in complex instances. Finally, we relate to Monte Carlo sampling and linear programming and discuss the shortcomings of these techniques through use-case examples.

Note that all the algorithms presented in Section 3 output the same m-approximation and all the algorithms presented in Section 6 output the same m-one-sided-approximation. Therefore, in all experiments that relate to the output (error comparisons), linApprox and linApproxOneSided will, respectively, represent those groups of algorithms.

Single-step support minimization – error comparison We start with the evaluation of solution quality in one-sided approximation, focusing on the error gaps between Trim and the proposed optimal algorithms (OptTrim or linApproxOneSided); error in this sense is the Kolmogorov distance $d_K(X, X')$ between the original random variable X and the approximated random variable X'. We consider in each instance a single random variable with an original support size of either $n = 100$ or $n = 1000$ and various support sizes m of the approximated random variable. In each instance of this experiment, a random variable is randomly generated by choosing the probabilities of each element in the support from a uniform distribution and then normalizing these probabilities so that they sum to one.

Tables 1 and 2 present the error produced by OptTrim, linApproxOneSided, and Trim on random variables with supports sizes of $n = 100$ and $n = 1000$, respectively. The depicted results in these tables are averages over several instances of random variables for each entry (50 instances in Table 1 and 10 instances in Table 2). The two central columns in each table show the average error of each method, whereas the right column presents the average percentage of the relative error of the Trim operator with respect to the error of the optimal approximation provided by OptTrim/linApproxOneSided; the relative error of each instance is calculated by (Trim / OptTrim) − 1. According to the depicted results, it is evident that increasing the support size of the approximation m reduces the error, as expected, in both methods. However, the interesting phenomenon is that the relative error percentage of Trim grows with the increase of m.

Next, we consider the error in the case of two-sided approximation. In this context, linApprox provides the optimal (minimal) error. However, OptTrim and linApproxOneSided restrict themselves to one-sided approximation and are therefore no longer optimal when an error is allowed in both directions. Naturally, Trim is also not optimal in this context. Therefore, in the following experiment,

Table 2
The errors of OptTrim and linApproxOneSided vs. Trim on randomly generated random variables with original support size $n = 1000$.

m	OptTrim/linApproxOS	Trim	Relative error
50	0.0193	0.0199	3.4%
100	0.0093	0.0099	7.1%
200	0.0043	0.0049	15.7%

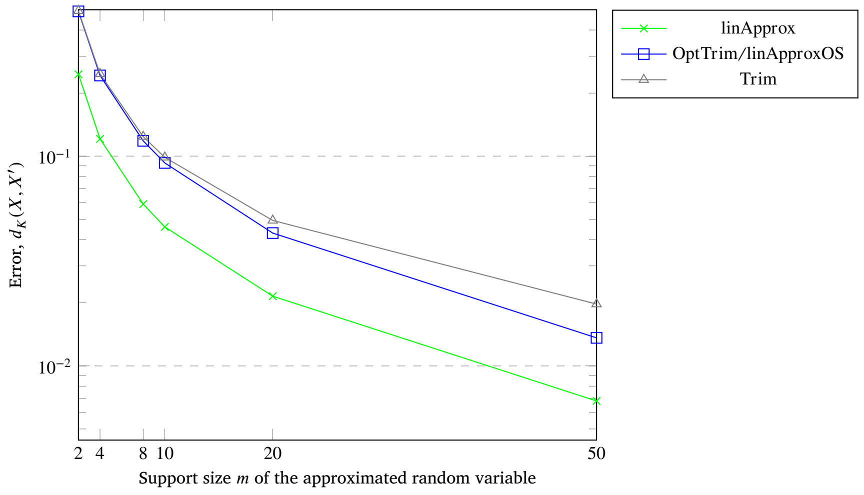

Fig. 2. Error comparison between linApprox, OptTrim, linApproxOneSided, and Trim on randomly generated random variables with original support size $n = 100$ as a function of the support size m of the approximated random variable.

we evaluate the two-sided error of the different algorithms. Each instance in that experiment is a random variable with a support size of $n = 100$ and various support sizes m of the approximated random variable are considered.

Fig. 2 presents the error produced by the above methods. The depicted results are averages over 50 instances of random variables. The curves in the figure show the average error of OptTrim/linApproxOneSided and Trim algorithms compared to the average error of the optimal approximation provided by linApprox as a function of m. It is evident from these graphs that increasing the support size of the approximation m reduces the error, as expected, in all three methods. However, the errors produced by the linApprox are significantly smaller; in fact, the error produced by linApprox is half of the error of OptTrim/linApproxOneSided following the phenomenon discussed in Section 6. Naturally, the error of Trim is even greater, with the error gap between Trim and OptTrim/linApproxOneSided growing as larger values of m are used; this is consistent with the trends shown in Tables 1 and 2. Nevertheless, when accuracy is not a crucial factor, one may prefer using Trim due to its speed and straightforward implementation. As shown in Fig. 2, for approximation with support size of $m = 10$ or smaller, the error of Trim is close to that of OptTrim/linApproxOneSided.

Single-step support minimization – run-time comparison We presented several optimal approximation algorithms for both the two- and one-sided variants of the problem; now, we focus on the comparison of the run-time performance of these optimal approximation algorithms. We start with the evaluation of the two-sided optimal approximation algorithms binsApprox, sdlbkApprox, and linApprox. We consider instances with varying support sizes of $n = [100..1000]$; all instances are approximated to random variables with support size $m = 50$. Fig. 3 presents the run-time performance of the above methods. The depicted results are averages over 30 instances of random variables. The experiment shows that linApprox is the most efficient algorithm of the three, and the depicted results coincide with the theoretical run-time complexities of the algorithms.

Fig. 4 presents the results of a similar experiment performed on the optimal one-sided approximation algorithms. Again, the run-times in the experiment are consistent with the theoretical run-time complexities of the algorithms. In fact, OptTrim was not able to solve the problems with $n = 1000$ due to memory limitations.

Repetitive support minimization – error comparison A common use of support size minimization is for computations that involve repetitive summations of random variables. Without a reduction of the support size, the convolutions performed in such computations lead to an exponential growth in the support size, which results in slowdowns and/or lack of memory issues [1]. A key action for handling this issue is a reduction of the support size after each summation; this is done by replacing the summed random variable with another random variable of a smaller support size that approximates it. Such reduction has been performed in past research via the sub-optimal Trim algorithm [7].

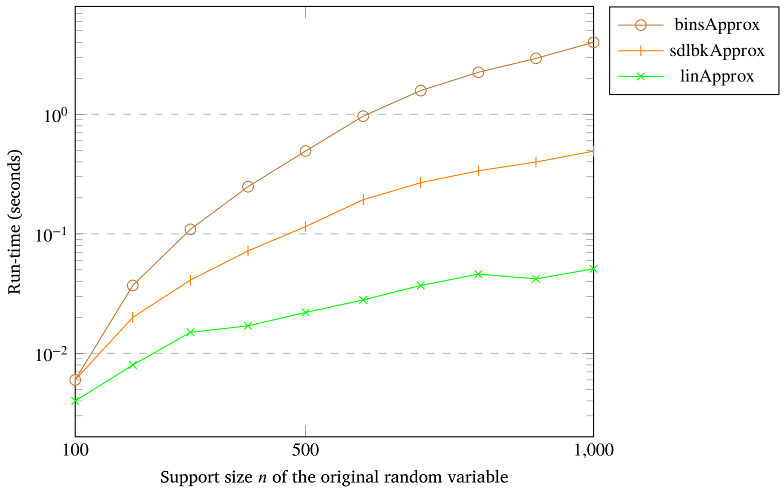

Fig. 3. Run-time comparison between binsApprox, sdlbkApprox, and linApprox on randomly generated random variables with varying support sizes n and support size $m = 50$ of the approximated random variable.

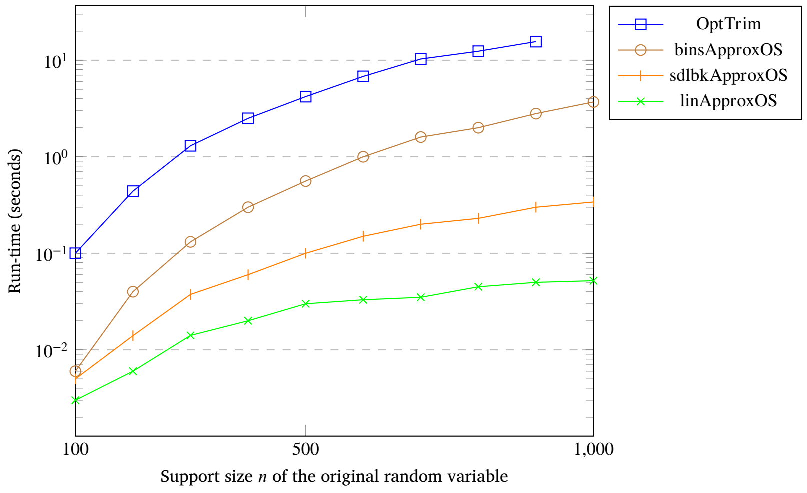

Fig. 4. Run-time comparison between OptTrim, binsApproxOneSided, sdlbkApproxOneSided, and linApproxOneSided on randomly generated random variables with varying support sizes n and support size $m = 50$ of the approximated random variable.

As we proved in Sections 3 and 6, given m, a single step of linApprox or linApproxOneSided guarantees an optimal m-approximation or m-one-side-approximation, respectively. However, in settings that require repetitive activations of the algorithm, the optimality of the eventually obtained random variable is not guaranteed. In light of this, we tested the accuracy of repetitive activations of linApprox, linApproxOneSided, OptTrim, and Trim to see the effect of the chosen approximation algorithm on the accumulated error. The benchmarks we used in the following experiments are taken from the area of task trees with deadlines, a sub-area of the well-established hierarchical planning [46,49,50] that was also used in [7]. Note that such problems with deadlines require a one-sided approximation, so applying a two-sided approximation yields inapplicable outcomes. Nonetheless, we include the results of linApprox to also witness the effect of repetitive activations of the two-sided approximation algorithms. Also note that we separately display the accumulative errors of linApproxOneSided and OptTrim. In a single step, the two algorithms return the same error, but potentially via different approximated random variables (in case there are several optimal approximations). However, their accumulated error after repetitive activations may be different, so we present both these algorithms.

For evaluation of the accumulated error after repetitive approximations, we consider three types of task trees. The first type includes logistic problems of transporting packages by trucks and airplanes (from IPC2 http://ipc.icaps-conference.org/). Hierarchical plans of those logistic problems were generated by the JSHOP2 planner [51], one parallel node with all descendant task nodes being in sequence. The second type consists of task trees used as execution plans for the ROBIL team entry in the DARPA robotics challenge (DRC simulation phase), and the third type is linear plans (sequential task trees). The primitive tasks in all the trees are modeled as discrete random variables with support of size n obtained by discretization of uniform distributions over various intervals. The number of tasks in a tree is denoted by N. All experiments are small enough (notice the small values of n) so that we could also run the exact computation as a reference to the true accumulated error.

Table 3
Error comparison between linApproxOneSided, OptTrim, Trim, and linApprox with respect to the reference exact computation on various task trees with deadlines.

Task Tree	n	linApproxOS	OptTrim	Trim	linApprox
		m/N=10	m/N=10	$\varepsilon \cdot N$=0.1	m/N=10
Logistics	2	0	0	0.0019	0
(N=34)	4	0.0046	0.0046	0.0068	0.0024
DRC-Drive	2	0.004	0.004	0.009	0.0014
(N=47)	4	0.008	0.008	0.019	0.001
Sequential	2	0.015	0.015	0.024	0.0093
(N=10)	4	0.024	0.024	0.04	0.008

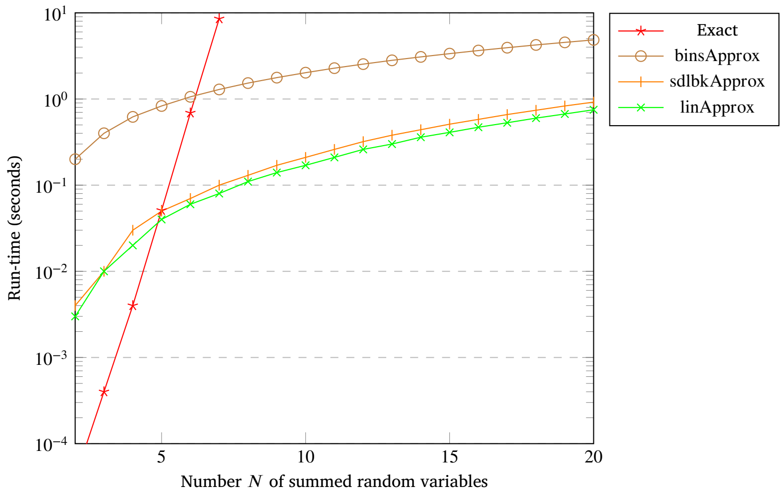

Fig. 5. Run-time comparison between an exact computation, binsApprox, sdlbkApprox, and linApprox on repetitive summations of a varying number N of randomly generated random variables with support size $n = 10$ and support size $m = 10$ of the approximated random variables after each step.

Table 3 shows the results of the experiment. Naturally, linApproxOneSided and OptTrim operators are better (more accurate) than Trim since Trim is sub-optimal. Also, linApprox is more accurate than linApproxOneSided and OptTrim and all other one-sided approximations since one-sided approximation is evaluating on average twice the error than the known Kolmogorov metric. However, recall that two-sided approximations are inapplicable in such scenarios that involve deadlines. Interestingly enough, linApproxOneSided and OptTrim result in the same accumulated error in all experiments, even though they potentially use different approximations (of equal quality) in each stage.

Repetitive support minimization – run-time comparison Similarly to the single-step case, we also conducted an empirical evaluation to examine the run-times of the optimal approximation algorithms in repetitive settings. Fig. 5 presents a comparison of the run-time performance of an exact computation and all the proposed optimal m-approximation algorithms – linApprox, sdlbkApprox, and binsApprox. Fig. 6 presents similar comparison of the m-one-sided-approximation algorithms – OptTrim, binsApproxOneSided, sdlbkApproxOneSided, and linApproxOneSided. The computation is a summation of a sequence of random variables, each with a support size of $n = 10$, where the number N of summed random variables varies from 2 to 20. In this experiment, the values and probabilities of the random variables are randomly generated. We executed the approximation algorithms with a parameter $m = 10$ after performing each convolution between two random variables, in order to maintain a support size of 10 in all intermediate computations. The presented results are averages of 20 independent runs.

The exponential blowup of the exact computation is evident; it is caused by the repetitive convolution between every two consecutive random variables. In fact, in the experiment with $N = 8$, the exact computation ran out of memory. Due to memory limitations, we could accurately calculate the summation of up to 7 random variables (support of size 10^7); however, for the next convolution operation to obtain the summation of 8 random variables, which results in a new random variable of 10^8 support size – the execution of computation failed to complete. Contrary to that, Figs. 5 and 6 show that our most efficient approximation algorithms (both two- and one-sided) manage to complete repetitive summations of more random variables in less than a second. These results illuminate the advantage of the proposed linApprox and linApproxOneSided algorithms, which successfully balance solution quality and run-time performance.

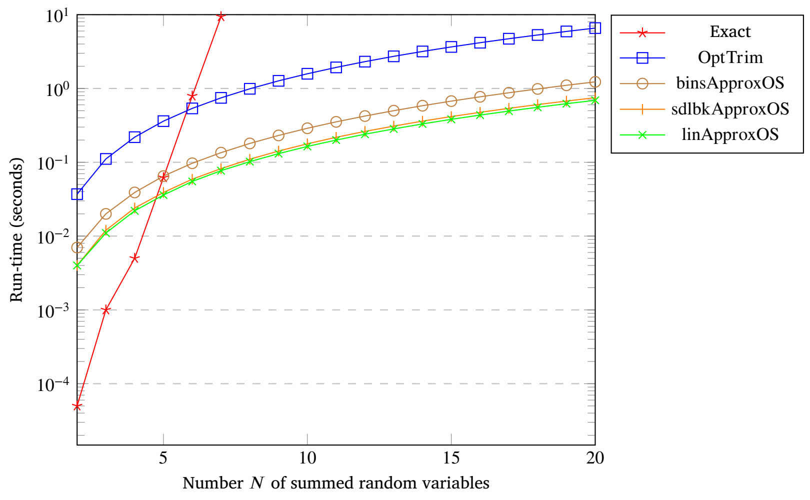

Fig. 6. Run-time comparison between an exact computation, OptTrim, binsApproxOneSided, sdlbkApproxOneSided, and linApproxOneSided on repetitive summations of a varying number N of randomly generated random variables with support size $n = 10$ and support size $m = 10$ of the approximated random variables after each step.

Table 4
Error comparison between linApprox and Sampling with respect to the reference exact computation on various task trees.

Task Tree	n	linApprox	Sampling	
		m/N=10	s=10^4	s=10^6
Logistics	2	0	0.007	0.0009
(N=34)	4	0.0024	0.0057	0.0005
DRC-Drive	2	0.0014	0.0072	0.0009
(N=47)	4	0.001	0.0075	0.0011
Sequential	2	0.0093	0.0063	0.0008
(N=10)	4	0.008	0.008	0.0016

Comparison to Monte Carlo sampling Monte Carlo sampling is a common method for approximation. In fact, it is straightforward to produce an additive PRAS (polynomial-time randomized approximation scheme) for the approximation problem at hand via the Monte Carlo method. By using the standard Chernoff bound, one can see that with K polynomial in $1/\varepsilon$ samples, the estimate is within an additive error ε from the true value with constant probability. To get a reasonable multiplicative approximation factor close to 1, we need to set the additive error at the order of the sum itself, so the number of samples might get exponentially large [21]. In our experiment, we use a stochastic sampling scheme in which we sample the distribution of every primitive task and aggregate the duration according to the tree structure. This approach enables us to derive a distribution over all of the duration results. Sampling may induce error on both sides, so we compare the errors of Sampling to those obtained by linApprox.

Table 4 shows the results of the experiment. The quality of the solutions obtained with the linApprox algorithm is in most cases better than that obtained by the sampling technique with $s = 10^4$ samples. However, in the case of $s = 10^6$ samples, there is an advantage to the sampling technique in most cases. Still, the linApprox approximation algorithm comes with an inherent advantage of providing exact quality guarantees, as opposed to sampling, for which at most probabilistic guarantees may be provided.

Regarding run-time, sampling with $s = 10^4$ runs in less than a second, whereas with $s = 10^6$ it takes up to 30 seconds.

Comparison to linear programming We also compared the run-time of linApprox with a linear programming (LP) algorithm that guarantees optimality, as described and discussed by Pavlikov and Uryasev [6]. We used the “Minimize” function of Wolfram Mathematica as an implementation of linear programming, encoding the problem by the LP problem $\min_{\alpha\in\mathbb{R}^n} \|x - \alpha\|_\infty$ subject to $\|\alpha\|_0 \leq m$ and $\|\alpha\|_1$=1. The run-time comparison results were clear and persuasive: linApprox significantly outperforms the LP algorithm. For a random variable with support size $n = 10$ and $m = 5$, the LP algorithm’s run-time was 850 seconds, whereas the run-time of linApprox was negligible (less than a few milliseconds). For $n = 100$ and $m = 5$, the run-time of linApprox was 0.14 seconds, while the LP algorithm took more than a day to run this problem. Since it is not trivial to formally analyze the run-time of the LP algorithm, we conclude from the reported experiment that in this case, the LP algorithm might not be sufficiently efficient.

Table 5
Summary of algorithms for approximating random variables under the Kolmogorov metric, comparing optimality guarantee, run-time complexity, and memory complexity. The variables n, m, and ε are defined as follows: n is the original size of the support, m is the desired size of the support, and ε is the error bound.

Algorithm	Opt	Run-time	Memory
binsApprox(X, m) (Section 3)	Yes	$O(n^2 \log(n))$	$O(n^2)$
sdlbkApprox(X, m) (Section 3)	Yes	$O(n^2)$	$O(n)$
linApprox(X, m) (Section 3)	Yes	$O(n \log(n))$	$O(n)$
Trim(X, ε) ([7])	No	$O(n/\varepsilon)$	$O(n/\varepsilon)$
OptTrim(X, m) (Section 5)	Yes	$O(n^3)$	$O(n^2)$
binsApproxOneSided(X, m) (Appendix A)	Yes	$O(n^2 \log(n))$	$O(n^2)$
sdlbkApproxOneSided(X, m) (Appendix A)	Yes	$O(n^2)$	$O(n)$
linApproxOneSided(X, m) (Appendix A)	Yes	$O(n \log(n))$	$O(n)$

8. Discussion and future work

We have created a range of algorithms for solving both one-sided and two-sided variations of the problem under consideration, with all of the new algorithms ensuring optimal approximations. The performance characteristics of each algorithm, including their optimality guarantee, run-time complexity, and memory usage complexity, are summarized in Table 5. The algorithms for the two-sided variant are listed above those for the one-sided variant in the table.

The Trim algorithm, which appears in the table, was originally developed in a previous study [7] to solve the one-sided variant of the problem. However, it does not guarantee to find an optimal solution. In the present study, we have modified and improved upon the Trim algorithm to ensure that the algorithms always produce optimal approximations. Note that in all of the algorithms, we assume the random variable or the distribution (as CDF or PDF) is given sorted. If it is not sorted, then a log-linear overhead that accounts for preprocessing sorting is added to the complexity; for instance, running Trim(X, ε) on an unsorted random variable X leads to a total run-time of $O(n/\varepsilon \log(n/\varepsilon))$ [7].

In this study, the approximation used is additive, meaning that the maximum deviation from the optimal solution is bounded by a value ε that is added to the cost of the solution. This is in contrast to other known techniques, such as the PTAS (polynomial-time approximation scheme) presented in [21], which use a relative approximation, where the maximum deviation from the optimal solution is bounded by a value ε that is multiplied by the cost of the solution.

There are a few potential advantages to using an additive approximation over a relative approximation in certain scenarios. First, additive approximations may be easier to understand and interpret for certain types of problems. For example, in a scenario involving deadlines, the intuitive concept of error may be the distance from the actual probability of meeting the deadline, which is an additive error. In contrast, a relative error may be more challenging to interpret in this context, as it would depend on the size of the solution's cost. Second, additive approximations may be more robust in scenarios where the optimal solution has a large range of possible costs. For example, consider a problem where the optimal solution has a cost that ranges from 1 to 1000. A relative approximation of, say, 10% error would result in an error range of 0.1 to 100, which may be much larger than the desired error tolerance. In contrast, an additive approximation of, say, 10 units of error would result in an error range of 10 to 1010, which may be more reasonable in this case. Of course, there may also be scenarios where a relative approximation is more appropriate. It is ultimately up to the algorithm designer to choose the type of approximation that is most suitable for the problem at hand.

The linApprox and linApproxOneSided algorithms are the most efficient among the algorithms discussed in the paper, with log-linear run-time and constant memory usage. These characteristics make them particularly well-suited for applications requiring repeated approximation of random variables, such as hierarchical planning and supplier assignment in projects. These types of scenarios often involve complex and computationally intensive calculations, and having algorithms with fast run-time and low memory usage can significantly improve the efficiency and scalability of the overall system.

One potential direction for future work in this area could be incorporating approximation layers into neural networks. When implementing neural networks with DSS (digital signal processing systems), it is often necessary to use approximated computation due to constraints on resources such as memory and processing power. Adding approximation layers after each layer of the network may be a way to improve the quality of the digital version of the neural network, even when using a limited number of bits to represent the weights and activations. Initial studies in this area have shown promising results, suggesting that this could be a promising direction for future research.

The techniques and approaches developed in this study for approximating random variables under the Kolmogorov metric may also be applicable to other measures of approximation. For example, similar techniques may be used to approximate random variables under the total variation distance [55] or the Wasserstein distance [9]. These distances are commonly used to measure the similarity between two probability distributions, and efficient algorithms for approximating random variables under these measures could have a range of potential applications. It is worth noting, however, that while the techniques developed in this study may be applicable to other measures of approximation, it is still necessary to carefully consider the specific characteristics and requirements of each measure in order to design algorithms that are effective and efficient for approximating random variables under each measure.

CRediT authorship contribution statement

Liat Cohen: Conceptualization, Formal analysis, Investigation, Software, Writing – original draft, Writing – review & editing. **Tal Grinshpoun:** Conceptualization, Formal analysis, Investigation, Supervision, Writing – original draft, Writing – review & editing. **Gera Weiss:** Conceptualization, Formal analysis, Funding acquisition, Investigation, Supervision, Writing – original draft, Writing – review & editing.

Declaration of competing interest

The authors declare the following financial interests/personal relationships which may be considered as potential competing interests: Liat Cohen reports financial support was provided by Horizon Europe (grant agreement no. 952215). Liat Cohen reports financial support was provided by Eric and Wendy Schmidt Fund for Strategic Innovation. Liat Cohen reports financial support was provided by Council for Higher Education of Israel. Liat Cohen reports financial support was provided by Israel Science Foundation (grant #2714/19). Gera Weiss reports a relationship with Israel Science Foundation that includes: funding grants (grant #2714/19).

Data availability

The authors do not have permission to share data.

Acknowledgement

Liat Cohen was partially supported by TAILOR, a project funded by the EU Horizon 2020 research and innovation program (grant agreement no. 952215), by the Eric and Wendy Schmidt Fund for Strategic Innovation, and by the Council for Higher Education of Israel. This research was partially funded by the Israel Science Foundation grant #2714/19.

Appendix A. Efficient one-sided Kolmogorov approximation algorithms

We first present in Algorithm 7 the binsApproxOneSided algorithm, which is the one-sided variant of binsApprox (Algorithm 2). For clarity, the changes required for binsApproxOneSided are marked in red.

Algorithm 7: binsApproxOneSided(X, m). (For interpretation of the colors in algorithm(s), the reader is referred to the web version of this article.)

```
1  Let $\{(x_i, c_i)\}_{i=1}^n$ be such that $c_i = Pr(X \le x_i)$ and support$(X) = \{x_1 < \cdots < x_n\}$.
2  $\overline{E} \leftarrow \emptyset$
3  for $i \leftarrow 1$ to $n-1$ do
4      for $j \leftarrow i+1$ to $n-1$ do
5          if $i = 1$ then
6              $\overline{E} \leftarrow \overline{E} \cup \{c_j\}$
7          $\overline{E} \leftarrow \overline{E} \cup \{c_j - c_i\}$
8  Let $e_1 < \cdots < e_{n'}$ be such that $\overline{E} = \{e_1, \ldots, e_{n'}\}$
9  for $i \leftarrow 1$ to 2 do
10     $l \leftarrow 1,\ r \leftarrow n',\ k \leftarrow 1,\ k' \leftarrow 0$
11     while $k \neq k'$ do
12         $k' \leftarrow k, k \leftarrow \lceil (l+r)/2 \rceil$
13         $m' \leftarrow |dualOneSided(X, e_k)|$
14         if $m' < m$ then $l \leftarrow k$
15         if $m' > m$ then $r \leftarrow k-1$
16         if $m' = m$ then $r \leftarrow k$
17     $m \leftarrow m'$
18 return $dualOneSided(X, e_k)$
```

As can be seen, the differences between binsApproxOneSided and binsApprox are only in the construction process of $\overline{E}$ (E in Algorithm 2) and calls to $dualOneSided$ instead of $dual$. As described in Section 6, the differences between $\overline{E}$ and E are the removal of the division by two (line 7, Algorithm 2) and the removal of the complementing probability (line 8, Algorithm 2).

Proposition 29. binsApproxOneSided$(X, m) \in \underset{|X'| \le m, \forall t. F_{X'}(t) \ge F_X(t)}{\arg\min} d_K(X, X')$.

Proof. Let $\overline{E} = \{Pr(x_1 < X \le x_2) : x_1, x_2 \in \text{support}(X) \cup \{-\infty, \infty\}, x_1 < x_2\}$ and let $\overline{E'} = \{d_K(X, X') : |X'| \le m, F_X(x) \le F_{X'}(x)$ for all $x\}$. Since lines 10-18 describe a binary search for finding the smallest $e \in \overline{E}$ such that $|dualOneSided(X, e)| \le m$, we only need to prove that $\min \overline{E'} \in \overline{E}$.

This proof is similar to the proof of Proposition 6 only using $dualOneSided$ and not $dual$, which entails some delicate changes. Let $\varepsilon = \min \overline{E'}$ and assume by contradiction that exists $\varepsilon \notin \overline{E}$. Let $X' = dualOneSided(X, \varepsilon)$. By definition, there is $x \in \text{support}(X)$ in which $F_{X'}(x) - F_X(x) = \varepsilon$. Let $x_- = \max\{x' : x' \leq x, x' \in \text{support}(X')\}$. If $x \in \text{support}(X')$ we have that $x_- = x$. Otherwise we have that $Pr(x_- < X' \leq x) = 0$. In both cases we get that $F_{X'}(x_-) = F_{X'}(x)$. Plugging this in the previous inequality gives us that $F_{X'}(x_-) - F_X(x) = \varepsilon$. Since X' is constructed by the $dualOneSided$ algorithm, $x_- \in \text{support}(X)$ and $\exists t \in \text{support}(X) : F_{X'}(x_-) = F_X(t)$. Therefore, $F_{X'}(x_-) - F_X(x) \in \overline{E}$ in contradiction to $\varepsilon \notin \overline{E}$. □

Proposition 30. *The* binsApproxOneSided(X, m) *algorithm runs in time* $O(n^2 \log(n))$*, using* $O(n^2)$ *memory where* $n = |X|$.

Proof. Similar to the proof of Proposition 7. □

Next, we present sdlbkApproxOneSided, listed as Algorithm 8, which is the one-sided variation of sdlbkApprox. The algorithm uses the $\overline{E_m}$ matrix that was defined in Section 6. The fact that $\overline{E_m}$ is sorted (Lemma 28) enables applying the same saddleback procedure as in the two-sided case. In fact, the only changes in Algorithm 8 with respect to Algorithm 3 are the calls to $dualOneSided$ instead of $dual$.

Algorithm 8: sdlbkApproxOneSided(X, m).

```
1  Let $\{(x_i, c_i)\}_{i=1}^n$ be such that $c_i = Pr(X \leq x_i)$ and support$(X) = \{x_1 < \cdots < x_n\}$.
2  $i \leftarrow 1,\ j \leftarrow n+1,\ S \leftarrow \emptyset$
3  while $i < n \wedge j \geq 1$ do
4      $m' \leftarrow |dualOneSided(X, e_{i,j})|$
5      if $m' \leq m$ then $j \leftarrow j - 1$
6      if $m' > m$ then $i \leftarrow i + 1$
7      if $m' \geq m$ then $S \leftarrow S \cup (m', e_{i,j})$
8  $e \leftarrow \min\{e : (m', e) \in \arg\max_{(m',e) \in S} m'\}$
9  return $dualOneSided(X, e)$
```

Proposition 31. sdlbkApproxOneSided$(X, m) \in \underset{|X'| \leq m, \forall t. F_{X'}(t) \geq F_X(t)}{\arg\min} d_K(X, X')$.

Proof. Identical to the proof of Proposition 9. □

Proposition 32. *The* sdlbkApproxOneSided(X, m) *algorithm runs in time* $O(n^2)$*, using* $O(n)$ *memory, where* $n = |X|$.

Proof. Similar to the proof of Proposition 10. □

Finally, we present linApproxOneSided, listed as Algorithm 9, which is the one-sided variation of linApprox. The algorithm uses the padded matrix $\overline{E_{pm}}$ in the same manner that E_{pm} was used in Algorithm 4. Again, the only changes in Algorithm 9 with respect to Algorithm 4 are the calls to $dualOneSided$ instead of $dual$.

Algorithm 9: linApproxOneSided(X, m).

```
1  Let $\{(x_i, c_i)\}_{i=1}^n$ be such that $c_i = Pr(X \leq x_i)$ and support$(X) = \{x_1 < \cdots < x_n\}$.
2  $S \leftarrow \{((1,1), (2^{\lceil \log_2(n) \rceil}, 2^{\lceil \log_2(n+1) \rceil}))\},\ S' \leftarrow \emptyset$
3  while $S \neq S'$ do
4      $S' \leftarrow S,\ S \leftarrow \bigcup_{s \in S} split(s)$
5      $e^- \leftarrow median(\{e_{i_1,j_1} : ((i_1,j_1),(i_2,j_2)) \in S\})$
6      $e^+ \leftarrow median(\{e_{i_2,j_2} : ((i_1,j_1),(i_2,j_2)) \in S\})$
7      for $e \in \{e^-, e^+\}$ do
8          $m' \leftarrow |dualOneSided(X, e)|$
9          if $m' \leq m$ then
10             $S \leftarrow S \setminus \{((i_1,j_1),(i_2,j_2)) \in S : e_{i_1,j_1} > e\}$
11         else
12             $S \leftarrow S \setminus \{((i_1,j_1),(i_2,j_2)) \in S : e_{i_2,j_2} \leq e\}$
13 $S'' \leftarrow \{(|dualOneSided(X, e_{i,j})|, e_{i,j}) : ((i,j),(i,j)) \in S\}$
14 $e \leftarrow \min\{e : (m', e) \in \arg\max_{(m',e) \in S''} m'\}$
15 return $dualOneSided(X, e)$
```

Theorem 33. $\text{linApproxOneSided}(X, m) \in \underset{|X'| \le m, \forall t. F_{X'}(t) \ge F_X(t)}{\arg\min} d_K(X, X')$.

Proof. Identical to the proof of Proposition 11. □

Theorem 34. *The* $\text{linApproxOneSided}(X, m)$ *algorithm runs in time* $O(n \log(n))$*, using* $O(1)$ *memory, where* $n = |X|$.

Proof. Similar to the proof of Proposition 12. □

Appendix B. Underestimation

As we elaborate in Section 4, some applications require the underestimation of random variables instead of their overestimation. The main differences when considering underestimation are in the *dualOneSided* algorithm, which should be replaced by *dualOneSidedUnder*, and in the set $\overline{E}$ and its derived matrices $\overline{E_m}$ and $\overline{E_{pm}}$, which should be replaced by the respective $\overline{\overline{E}}$, $\overline{\overline{E_m}}$, and $\overline{\overline{E_{pm}}}$. We next provide details on these changes.

The change in *dualOneSidedUnder*, listed as Algorithm 10, is in the direction the random variable is traversed: instead of traversing it from bottom to top for overestimation (Algorithm 6), the underestimation version needs to traverse the random variable from top to bottom.

Algorithm 10: *dualOneSidedUnder*$(\{(x_i, c_i)\}_{i=1}^n, \varepsilon)$.

```
1 $l \leftarrow n$, $e \leftarrow n$
2 while $e > 1$ do
3     while $c_l - c_{e-1} \le \varepsilon \wedge e > 1$ do
4         $e \leftarrow e - 1$
5     $S \leftarrow S \cup \{(x_l, c_l)\}$
6     $l \leftarrow e - 1$, $e \leftarrow e - 1$
7 if $x_1 > \varepsilon$ then
8     $S \leftarrow S \cup \{(x_1, c_1)\}$
9 return A r.v. $X'$ as CDF such that $Pr(X' \le x) = c$ if there is $c$ such that $(x, c) \in S$.
```

To best illuminate the required changes for underestimation in the set $\overline{\overline{E}}$ and its derivatives, we present the matrix form $\overline{\overline{E_m}} := (e_{i,j})_{i,j=1}^{\infty}$:

$$e_{i,j} = \begin{cases} 1 - c_{n+1-j} & \text{if } j \le n \wedge i = n; \\ c_i - c_{n+1-j} & \text{if } i < n \wedge j \le n \wedge i+j \ge n; \\ 0 & \text{if } i+j < n; \\ 1 & \text{otherwise.} \end{cases}$$

The underestimation variants of binsApproxOneSided, sdlbkApproxOneSided, and linApproxOneSided, can now be derived by respectively using $\overline{\overline{E}}$, $\overline{\overline{E_m}}$, and $\overline{\overline{E_{pm}}}$, calling *dualOneSidedUnder*, and applying straightforward technical adjustments that stem from these changes.

References

[1] L. Cohen, S.E. Shimony, G. Weiss, Estimating the probability of meeting a deadline in hierarchical plans, in: IJCAI, 2015, pp. 1551–1557.
[2] R. Möhring, Scheduling under uncertainty: bounding the makespan distribution, Comput. Discrete Math. (2001) 79–97.
[3] A.N. Pettitt, M.A. Stephens, The Kolmogorov-Smirnov goodness-of-fit statistic with discrete and grouped data, Technometrics 19 (2) (1977) 205–210, http://www.jstor.org/stable/1268631.
[4] A.C. Miller, T.R. Rice, Discrete approximations of probability distributions, Manag. Sci. 29 (3) (1983) 352–362, http://www.jstor.org/stable/2631060.
[5] M. Vidyasagar, A metric between probability distributions on finite sets of different cardinalities and applications to order reduction, IEEE Trans. Autom. Control 57 (10) (2012) 2464–2477.
[6] K. Pavlikov, S. Uryasev, CVaR distance between univariate probability distributions and approximation problems, Tech. Rep. 2015-6, University of Florida, 2016.
[7] L. Cohen, S.E. Shimony, G. Weiss, Estimating the probability of meeting a deadline in schedules and plans, Artif. Intell. 275 (2019) 329–355.
[8] H.W. Lilliefors, On the Kolmogorov-Smirnov test for normality with mean and variance unknown, J. Am. Stat. Assoc. 62 (318) (1967) 399–402.
[9] S. Vallender, Calculation of the Wasserstein distance between probability distributions on the line, Theory Probab. Appl. 18 (4) (1974) 784–786.
[10] J.D. Gibbons, S. Chakraborti, Nonparametric statistical inference, in: International Encyclopedia of Statistical Science, Springer, 2011, pp. 977–979.
[11] S.S. Shperberg, A. Coles, B. Cserna, E. Karpas, W. Ruml, S.E. Shimony, Allocating planning effort when actions expire, in: Proceedings of the AAAI Conference on Artificial Intelligence, vol. 33, 2019, pp. 2371–2378.
[12] S.S. Shperberg, A. Coles, E. Karpas, W. Ruml, S.E. Shimony, Situated temporal planning using deadline-aware metareasoning, in: Proceedings of the International Conference on Automated Planning and Scheduling, vol. 31, 2021, pp. 340–348.

[13] W. Herroelen, R. Leus, Project scheduling under uncertainty: survey and research potentials, Eur. J. Oper. Res. 165 (2) (2005) 289–306.
[14] J.C. Beck, N. Wilson, Proactive algorithms for job shop scheduling with probabilistic durations, J. Artif. Intell. Res. 28 (2007) 183–232.
[15] N. Fu, P. Varakantham, H.C. Lau, Towards finding robust execution strategies for RCPSP/max with durational uncertainty, in: ICAPS, 2010, pp. 73–80.
[16] R. Buyya, S.K. Garg, R.N. Calheiros, SLA-oriented resource provisioning for cloud computing: challenges, architecture, and solutions, in: International Conference on Cloud and Service Computing (CSC), 2011, pp. 1–10.
[17] L. Cohen, T. Grinshpoun, R. Stern, Assigning suppliers to meet a deadline, in: Proceedings of the International Symposium on Combinatorial Search, vol. 10, 2019, pp. 170–171.
[18] L. Cohen, T. Grinshpoun, G. Weiss, Optimal approximation of random variables for estimating the probability of meeting a plan deadline, in: Proceedings of the AAAI Conference on Artificial Intelligence, vol. 32, 2018, pp. 6327–6334.
[19] L. Cohen, G. Weiss, Efficient optimal approximation of discrete random variables for estimation of probabilities of missing deadlines, in: Proceedings of the AAAI Conference on Artificial Intelligence, vol. 33, 2019, pp. 7809–7815.
[20] J. Kleinberg, Y. Rabani, É. Tardos, Allocating bandwidth for bursty connections, in: Proceedings of the Twenty-Ninth Annual ACM Symposium on Theory of Computing, 1997, pp. 664–673.
[21] J. Li, T. Shi, A fully polynomial-time approximation scheme for approximating a sum of random variables, Oper. Res. Lett. 42 (3) (2014) 197–202.
[22] S. Guha, Posterior simulation in the generalized linear mixed model with semiparametric random effects, J. Comput. Graph. Stat. 17 (2) (2008) 410–425, https://doi.org/10.1198/106186008X319854.
[23] S. Guha, K. Shim, A note on linear time algorithms for maximum error histograms, IEEE Trans. Knowl. Data Eng. 19 (7) (2007) 993–997.
[24] S. Guha, K. Shim, J. Woo, Rehist: relative error histogram construction algorithms, in: VLDB, 2004, pp. 300–311.
[25] H.V. Jagadish, N. Koudas, S. Muthukrishnan, V. Poosala, K.C. Sevcik, T. Suel, Optimal histograms with quality guarantees, in: VLDB, vol. 98, 1998, pp. 24–27.
[26] P. Karras, D. Sacharidis, N. Mamoulis, Exploiting duality in summarization with deterministic guarantees, in: SIGKDD, 2007, pp. 380–389.
[27] H. Fournier, A. Vigneron, Fitting a step function to a point set, Algorithmica 60 (1) (2011) 95–109, https://doi.org/10.1007/s00453-009-9342-z.
[28] J.M. Díaz-Bánez, J.A. Mesa, Fitting rectilinear polygonal curves to a set of points in the plane, Eur. J. Oper. Res. 130 (1) (2001) 214–222.
[29] H. Fournier, A. Vigneron, Fitting a step function to a point set, in: European Symposium on Algorithms, Springer, 2008, pp. 442–453.
[30] G. Zeng, A comparison study of computational methods of Kolmogorov–Smirnov statistic in credit scoring, Commun. Stat., Simul. Comput. 46 (10) (2017) 7744–7760.
[31] E. Mays, Handbook of Credit Scoring, Global Professional Publishi, 2001.
[32] M. Refaat, Credit Risk Scorecard: Development and Implementation Using SAS, Lulu.com, 2011.
[33] C. Bolton, et al., Logistic regression and its application in credit scoring, Ph.D. thesis, University of Pretoria, 2010.
[34] N. Siddiqi, Credit Risk Scorecards: Developing and Implementing Intelligent Credit Scoring, vol. 3, John Wiley & Sons, 2012.
[35] D.L. Keefer, S.E. Bodily, Three-point approximations for continuous random variables, Manag. Sci. 29 (5) (1983) 595–609.
[36] D.L. Keefer, Certainty equivalents for three-point discrete-distribution approximations, Manag. Sci. 40 (6) (1994) 760–773.
[37] A.C. Miller III, T.R. Rice, Discrete approximations of probability distributions, Manag. Sci. 29 (3) (1983) 352–362.
[38] R.K. Hammond, J.E. Bickel, Reexamining discrete approximations to continuous distributions, Decis. Anal. 10 (1) (2013) 6–25.
[39] J.E. Smith, Moment methods for decision analysis, Manag. Sci. 39 (3) (1993) 340–358.
[40] E. Rosenblueth, Karmeshu, H.P. Hong, Maximum entropy and discretization of probability distributions, Probab. Eng. Mech. 2 (2) (1987) 58–63.
[41] F. Cicalese, L. Gargano, U. Vaccaro, Bounds on the entropy of a function of a random variable and their applications, IEEE Trans. Inf. Theory 64 (4) (2017) 2220–2230.
[42] F. Cicalese, L. Gargano, U. Vaccaro, Minimum entropy joint distributions with fixed marginals: algorithms and applications, in: Tenth Workshop on Information Theoretic Methods in Science and Engineering, vol. 56, 2017, p. 9.
[43] F. Cicalese, U. Vaccaro, Maximum entropy interval aggregations, in: 2018 IEEE International Symposium on Information Theory (ISIT), IEEE, 2018, pp. 1764–1768.
[44] F. Cicalese, L. Gargano, U. Vaccaro, An information theoretic approach to probability mass function truncation, in: 2019 IEEE International Symposium on Information Theory (ISIT), IEEE, 2019, pp. 702–706.
[45] F. Cicalese, L. Gargano, U. Vaccaro, Minimum-entropy couplings and their applications, IEEE Trans. Inf. Theory 65 (6) (2019) 3436–3451.
[46] T. Dean, R.J. Firby, D. Miller, Hierarchical planning involving deadlines, travel time, and resources, Comput. Intell. 4 (3) (1988) 381–398, https://doi.org/10.1111/j.1467-8640.1988.tb00287.x.
[47] K. Erol, J. Hendler, D.S. Nau, HTN planning: complexity and expressivity, in: AAAI, vol. 94, 1994, pp. 1123–1128.
[48] K. Erol, J. Hendler, D.S. Nau, Complexity results for HTN planning, Ann. Math. Artif. Intell. 18 (1) (1996) 69–93.
[49] R. Alford, V. Shivashankar, M. Roberts, J. Frank, D.W. Aha, Hierarchical planning: relating task and goal decomposition with task sharing, in: IJCAI, 2016, pp. 3022–3029.
[50] Z. Xiao, A. Herzig, L. Perrussel, H. Wan, X. Su, Hierarchical task network planning with task insertion and state constraints, in: IJCAI, 2017, pp. 4463–4469.
[51] D.S. Nau, T.-C. Au, O. Ilghami, U. Kuter, J.W. Murdock, D. Wu, F. Yaman, SHOP2: an HTN planning system, J. Artif. Intell. Res. 20 (2003) 379–404.
[52] A. Chakravarty, J. Orlin, U. Rothblum, A partitioning problem with additive objective with an application to optimal inventory groupings for joint replenishment, Oper. Res. 30 (5) (1982) 1018–1022.
[53] E. Shufan, H. Ilani, T. Grinshpoun, A two-campus transport problem, in: MISTA, 2011, pp. 173–184.
[54] N. Christofides, Graph Theory: An Algorithmic Approach, Academic Press, 1975.
[55] C. Daskalakis, I. Diakonikolas, R. ODonnell, R.A. Servedio, L.-Y. Tan, Learning sums of independent integer random variables, in: 2013 IEEE 54th Annual Symposium on Foundations of Computer Science, IEEE, 2013, pp. 217–226.