Kolmogorov Approximation

Anonymous Author(s)
Affiliation
Address
email

1 Introduction

Many different approaches to approximation of probability distributions are studied in the literature [3, 5, 6]. This is on on approximating discrete distributions with ones that are simpler to store and to manipulate. This is needed, for example, when a discrete distribution is given as a large data-set, obtained, e.g., by sampling, and we want to represent it approximately with a small table.

The main contribution of this paper is an efficient algorithm for computing the best possible approximation of a given random variable with a random variable whose complexity is not above a prescribed threshold, where the measures of the quality of the approximation and the complexity of the random variable are as specified in the following two paragraphs.

We measure the quality of an approximation is usually measured by the distance between the original variable and the approximate one. Specifically, we use the Kolmogorov distance which is one of the most used in statistical practice and literature. Given two random variables X and X' whose cumulative distribution functions (cdfs) 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., [2]. We say taht X' is a good approximation of X if $d_K(X, X')$ is small.

The complexity of a random variable is measured by the size of its support, the number of values that it can take, $|\operatorname{support}(X)| = |\{x\colon Pr(X = x) \neq 0\}|$. When distributions are maintained as explicit tables, as done in many implementations of statistical software, the size of the support of a variable is proportional to the amount of memory needed to store it and to the complexity of the computations around it.

In summary, the exact notion of optimality of the approximation is:

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

The main contribution of the paper is a constructive proof of:

Theorem 2. *Given a random variable X a number m, there exists an algorithm with memory and time complexity $O(|\operatorname{support}(X)|^2 \cdot m)$ that computes an optimal m-approximation of X.*

Submitted to 32nd Conference on Neural Information Processing Systems (NIPS 2018). Do not distribute.

2 An Algorithm for Optimal Approximation

- We now start our story: Given X and m how can we find X'?
- We first show that it is enough to limit our search to X's such that $\mathrm{support}(X') \subseteq \mathrm{support}(X)$.

Lemma 3. *For any discrete random variable X and any $m \in \mathbb{N}$, there is an m-optimal-approximation X' of X such that* $\mathrm{support}(X') \subseteq \mathrm{support}(X)$.

Proof. Assume there is a random variable X'' with support size m such that $d_K(X, X'')$ is minimal but $\mathrm{support}(X'') \nsubseteq \mathrm{support}(X)$. We will show how to transform X'' support such that it will be contained in $\mathrm{support}(X)$. Let v' be the first $v' \in \mathrm{support}(X'')$ and $v' \notin \mathrm{support}(X)$. Let $v = \max\{i : i < v' \wedge i \in \mathrm{support}(X)\}$. Every v' we will replace with v and name the new random variable X', we will show that $d_K(X, X'') = d_K(X, X')$. First, note that: $F_{X''}(v') = F_{X'}(v)$, $F_X(v') = F_X(v)$. Second, $F_{X'}(v') - F_X(v') = F_{X'}(v) - F_X(v)$. Therefore, $d_K(X, X'') = d_K(X, X')$ and X' is also an optimal approximation of X. □

Observation 4. $max\{|a|, |b|\} \geq |a - b|/2$

- The next lemma states a lower bound on the distance $d_K(X, X')$ when a range of elements is excluded from the support of X'.

Lemma 5. *For $x_1, x_2 \in \mathrm{support}(X) \cup \{-\infty, \infty\}$ such that $x_1 < x_2$, if $P(x_1 < X' < x_2) = 0$ then $d_k(X, X') \geq P(x_1 < X < x_2)/2$.*

Proof. Let $\hat{x} = \max\{x \in \mathrm{support}(X) \cap \{-\infty, \infty\} \colon x < x_2\}$. By definition, $d_k(X, X') \geq \max\{|F_X(x_1) - F_{X'}(x_1)|, |F_X(\hat{x}) - F_{X'}(\hat{x})|\}$. From Observation 4, $d_k(X, X') \geq 1/2|F_X(x_1) - F_X(\hat{x}) + F_{X'}(\hat{x}) - F_{X'}(x_1)|$. Since it is given that $F_{X'}(\hat{x}) - F_{X'}(x_1) = P(x_1 < X' < x_2) = 0$, $d_k(X, X') \geq 1/2|F_X(x_1) - F_X(\hat{x})| = P(x_1 < X \leq \hat{x})/2 = P(x_1 < X < x_2)/2$. □

- The next lemma strengthen the lower bound.

Lemma 6. *For $x_1, x_2 \in \mathrm{support}(X) \cup \{-\infty, \infty\}$ such that $x_1 = -\infty$ or $x_2 = \infty$, if $P(x_1 < X' < x_2) = 0$ then $d_k(X, X') \geq P(x_1 < X < x_2)$.*

Proof. Let $\hat{x} = \max\{x \in \mathrm{support}(X) \cap \{-\infty, \infty\} \colon x < x_2\}$. By definition $d_k(X, X') \geq \max\{|F_X(x_1) - F_{X'}(x_1)|, |F_X(\hat{x}) - F_{X'}(\hat{x})|\}$. If $x_1 = -\infty$ then $d_k(X, X') \geq \{|F_X(\hat{x}) - F_{X'}(\hat{x})|\}$ since $F_X(-\infty) = F_{X'}(-\infty) = 0$. Furthermore, $F_{X'}(\hat{x}) = P(x_1 < X' < x_2) = 0$. Therefore $d_k(X, X') \geq F_X(\hat{x}) = P(x_1 < X \leq \hat{x}) = P(x_1 < X < x_2)$. If $x_2 = \infty$ then $d_k(X, X') \geq \{|F_X(x_1) - F_{X'}(x_1)|\}$ since $F_X(\hat{x}) = F_{X'}(\hat{x}) = F_X(\infty) = F_{X'}(\infty) = 1$. Furthermore, $F_{X'}(x_1) = 1$ since it is given that $P(x_1 < X' < x_2) = 0$. Therefore we get that $d_k(X, X') \geq |F_X(x_1) - 1| = |1 - F_X(\hat{x}) - | = P(x_1 < X \leq \hat{x}) = P(x_1 < X < x_2)$. □

Definition 7. *For $x_1, x_2 \in \mathrm{support}(X) \cup \{-\infty, \infty\}$ let*

$$w(x_1, x_2) = \begin{cases} P(x_1 < X < x_2) & \text{if } x_1 = -\infty \text{ or } x_2 = \infty; \\ P(x_1 < X < x_2)/2 & \text{otherwise.} \end{cases}$$

Definition 8. *For $S = \{x_1 < \cdots < x_m\} \subseteq \mathrm{support}(X)$, $x_0 = -\infty$, and $x_{m+1} = \infty$, let*

$$\varepsilon(X, S) = \max_{i=0,\ldots,m} w(x_i, x_{i+1}).$$

- From here on, until the end of the section, S is fixed.

Proposition 9. *There is no X' such that* $\mathrm{support}(X') = S$ *and* $d_k(X, X') < \varepsilon(X, S)$.

Proof. Let i be the index that maximizes $w(x_i, x_{i+1})$. If $0 < i < n - 1$ then $d_k(X, X') \geq w(x_i, x_{i+1})$ by Lemma 5. If $i = 0$ or $i = n + 1$ the same follows from Lemma 6. □

Let X' to by $f_{X'}(x_i) = w(x_{i-1}, x_i) + w(x_i, x_{i+1}) + f_X(x_i)$ for $i = 1, \ldots, m$ and $f_{X'}(x) = 0$ for $x \notin S$.

Lemma 10. *For $i > 1$, if $F_{X'}(x_{i-1}) - F_X(x_{i-1}) = w(x_{i-1}, x_i)$ then $F_{X'}(x_i) - F_X(x_i) = w(x_i, x_{i+1})$.*

Proof.

$$
\begin{aligned}
& F_X(x_i) - F_{X'}(x_i) = && (1)\\
& f_X(x_i) - f_{X'}(x_i) + P(X < x_i) - P(X' < x_i) = && (2)\\
& f_X(x_i) - f_{X'}(x_i) + F_X(x_{i-1}) + P(x_{i-1} < X < x_i) - F_{X'}(x_{i-1}) = && \\
& f_X(x_i) - f_{X'}(x_i) + F_X(x_{i-1}) + 2w(x_{i-1}, x_i) - F_{X'}(x_{i-1}) =^* && (3)\\
& f_X(x_i) - f_{X'}(x_i) + 2w(x_{i-1}, x_i) - w(x_{i-1}, x_i) = && (4)\\
& - w(x_{i-1}, x_i) - w(x_i, x_{i+1}) + 2w(x_{i-1}, x_i) - w(x_{i-1}, x_i) = && (5)\\
& - w(x_i, x_{i+1}) && (6)
\end{aligned}
$$

$*$ by induction hypothesis. The probability $P(x_{i-1} < X < x_i) = 2w(x_{i-1}, x_i)$ by Definition 7, and $f_{X'}(x_i) - f_X(x_i) = w(x_{i-1}, x_i) + w(x_i, x_{i+1})$ by construction. □

Lemma 11. $F_{X'}(x_1) - F_X(x_1) = w(x_1, x_2)$.

Proposition 12. *There exists X' such that* $\mathrm{support}(X') = S$ *and* $d_k(X, X') = \varepsilon(X, S)$.

Proof. Define X' to by $f_{X'}(x_i) = w(x_{i-1}, x_i) + w(x_i, x_{i+1}) + f_X(x_i)$ for $i = 1, \ldots, m$ and $f_{X'}(x) = 0$ for $x \notin S$. We need to show that $F_X(x_i) - F_{X'}(x_i) = -w(x_i, x_{i+1})$. Assume this is true for every $j < i$, the induction hypothesis hereby: $F_X(x_{i-1}) - F_{X'}(x_{i-1}) = -w(x_{i-1}, x_i)$.

$$
\begin{aligned}
& F_X(x_i) - F_{X'}(x_i) =\\
& f_X(x_i) - f_{X'}(x_i) + P(X < x_i) - P(X' < x_i) =\\
& f_X(x_i) - f_{X'}(x_i) + F_X(x_{i-1}) + P(x_{i-1} < X < x_i) - F_{X'}(x_{i-1}) =\\
& f_X(x_i) - f_{X'}(x_i) + F_X(x_{i-1}) + 2w(x_{i-1}, x_i) - F_{X'}(x_{i-1}) =^*\\
& f_X(x_i) - f_{X'}(x_i) + 2w(x_{i-1}, x_i) - w(x_{i-1}, x_i) =\\
& - w(x_{i-1}, x_i) - w(x_i, x_{i+1}) + 2w(x_{i-1}, x_i) - w(x_{i-1}, x_i) =\\
& - w(x_i, x_{i+1})
\end{aligned}
$$

$*$ by induction hypothesis. The probability $P(x_{i-1} < X < x_i) = 2w(x_{i-1}, x_i)$ by definition 7, and $f_{X'}(x_i) - f_X(x_i) = w(x_{i-1}, x_i) + w(x_i, x_{i+1})$ by construction.

□

Chakravarty, Orlin, and Rothblum [1] 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 $\mathrm{KolmogorovApprox}$ algorithm (Algorithm 2) starts by constructing a directed weighted graph G similar to the method of Chakravarty, Orlin, and Rothblum [1]. The nodes V consist of the support of X together with an extra two nodes ∞ and $-\infty$ for technical reasons, whereas the edges E connect every pair of nodes in one direction (lines 1-2). The weight w of each edge $e = (i, j) \in E$ is determined by on of two cases. The first is where i or j are the source or target nodes respectively. In this case the weight is the probability of X to get a value between i and j, non inclusive, i.e., $w(e) = Pr(i < X < j)$ (lines 4-5). The second case is where i or j are not a source or target nodes, here the weight is the probability of X to get a value between i and j, non inclusive, divided by two i.e., $w(e) = Pr(i < X < j)/2$ (lines 6-7). 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 $-\infty$ node added to G in the construction phase, and the target node is the extra node ∞. The set of all solution paths in G, i.e., those starting at $-\infty$ and ending in ∞ with at most m edges, is called $paths(G, -\infty, \infty)$. The goal is to find the path l^* in $paths(G, -\infty, \infty)$ with the lightest bottleneck (lines 8-9). 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 length 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 [7]. Consequently, we find the lightest maximal edge in a path of length $\leq m$, which represents the minimal error, ε^*, defined in Definition **??**. X' is then derived from the resulting path l^* (lines 10-17). Every node $n \in l^*$ represent a value in the new calculated random variable X', we than iterate the path l^* to fine the probability of the event $f_{X'}(n)$. For every edge $(i, j) \in l^*$ we determine: if (i, j) is the first edge in the path l^* (i.e. $i == -\infty$), then node j gets the full weight $w(i, j)$ and it's own weight in X such that $f_{X'}(j) = f_X(j) + w(i, j)$ (lines 11-12). If (i, j) in not the first nor the last edge in path l^* then we divide it's weight between nodes i and j in addition to their own original weight in X and the probability that already accumulated (lines 16-17). If (i, j) is the last edge in the path l^* (i.e. $i == \infty$) then node i gets the full weight $w(i, j)$ in addition to what was already accumulated such that $f_{X'}(j) = f_{X'}(j) + w(i, j)$ (lines 13-14).

Algorithm 1: $\mathrm{KolmogorovApprox}(X, m)$

```
1  S = support(X) ∪ {∞, −∞}
2  G = (V, E) = (S, {(x, y) ∈ S²: x < y})
3  l = argmin_{l ∈ paths(G,−∞,∞), |l| ≤ m} max{w(e): e ∈ l}
4  foreach e = (x, y) ∈ l do
5      if x ≠ −∞ ∧ y ≠ ∞ then
6          f_X'(j) = f_X(j) + Pr(i ≤ X < j)
7      else if j == ∞ then
8          f_X'(i) = f_X'(i) + Pr(i ≤ X < j)
9      else
10         f_X'(i) = f_X'(i) + Pr(i ≤ X < j)/2
11         f_X'(j) = f_X(j) + Pr(i ≤ X < j)/2
12 return X'
```

Theorem 13. *The* $\mathrm{KolmogorovApprox}(X, m)$ *algorithm runs in time* $O(mn^2)$*, using* $O(n^2)$ *memory where* $n = |\mathrm{support}(X)|$.

Algorithm 2: $\mathrm{KolmogorovApprox}(X, m)$

$S = \mathrm{support}(X) \cup \{\infty, -\infty\}$
$G = (V, E) = (S, \{(x, y) \in S^2 \colon x < y\})$
foreach $e = (x, y) \in E$ **do**
  if $i = \infty$ *OR* $j = -\infty$ **then**
    $w(e) = Pr(i < X < j)$
  else
    $w(e) = Pr(i < X < j)/2$
/* The following can be obtained, e.g., using the Bellman-Ford algorithm */
$l^* = \mathrm{argmin}_{l \in paths(G, -\infty, \infty, |l| \leq m} \max\{w(e) \colon e \in l\}$
foreach $e = (i, j) \in l^*$ **do**
  if $i = -\infty$ **then**
    $f_{X'}(j) = f_X(j) + Pr(i \leq X < j)$
  else if $j == \infty$ **then**
    $f_{X'}(i) = f_{X'}(i) + Pr(i \leq X < j)$
  else
    $f_{X'}(i) = f_{X'}(i) + Pr(i \leq X < j)/2$
    $f_{X'}(j) = f_X(j) + Pr(i \leq X < j)/2$
return X'

Proof. Constructing the graph G takes $O(n^2)$. 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 $-\infty$ and the target node ∞, which can be done efficiently by aggregating the weights of already calculated edges. 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 id DAG (directed acyclic graph) finding shortest path takes $O(E + V)$. We only need to find paths of length $\leq m$, 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 $O(n^2 + mn^2 + mn) = O(mn^2)$ and memory complexity is $O(E + V) = O(n^2)$. □

3 Experiments and Results

In the first experiment we focus on the problem of task trees with deadlines, and 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 [4] (see example problem, Figure 1). 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 of linear plans (sequential task trees). The primitive tasks in all the trees are modeled as discrete random variables with support of size M obtained by discretization of uniform distributions over various intervals. The number of tasks in a tree is denoted by N.

We implemented the approximation algorithm for solving the deadline problem with four different methods of approximation. The first two are for achieving a one-sided Kolmogorov approximation – the $\mathrm{OptTrim}$ and the Trim operators, and a simple sampling scheme which we used as comparison to the Kolmogorov approximation with the $\mathrm{KolmogorovApprox}$ algorithm. The parameter m of $\mathrm{OptTrim}$ and $\mathrm{KolmogorovApprox}$ corresponds to the inverse of ε given to the Trim operator. Note

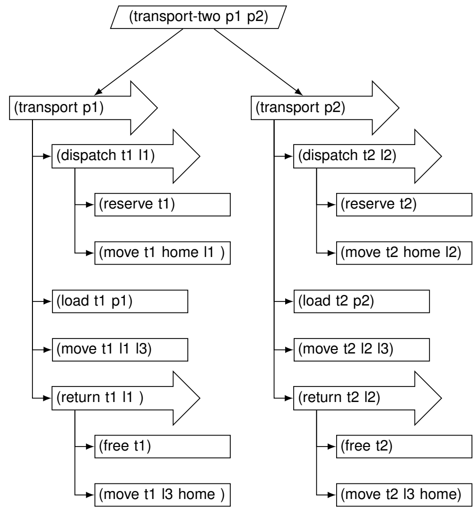

Figure 1: A plan generated by the JSHOP2 algorithm. Arrow shapes represent sequence nodes, parallelograms represent parallel nodes, and rectangles represent primitive nodes.

that in order to obtain some error ε, one must take into consideration the size of the task tree, N, therefore, $m/N = 1/(\varepsilon \cdot N)$. We ran the algorithm for exact computation as reference, the approximation algorithm using $\mathrm{KolmogorovApprox}$ as its operator with $m = 10 \cdot N$, the $\mathrm{OptTrim}$ as its operator with $m = 10 \cdot N$, the Trim as operator with $\varepsilon = 0.1/N$, and two simple simulations, with a different samples number $s = 10^4$ and $s = 10^6$.

Task Tree	M	$\mathrm{OptTrim}$	Trim	Sampling	
		m/N=10	$\varepsilon \cdot N$=0.1	s=10^4	s=10^6
Logistics ($N = 34$)	2	0	0.0019	0.007	0.0009
	4	0.0046	0.0068	0.0057	0.0005
Logistics (N=45)	2	0.0005	0.002	0.015	0.001
	4	0.003	0.004	0.008	0.0006
DRC-Drive (N=47)	2	0.004	0.009	0.0072	0.0009
	4	0.008	0.019	0.0075	0.0011
Sequential (N=10)	4	0.024	0.04	0.008	0.0016
	10	0.028	0.06	0.0117	0.001

Table 1: Comparison of estimation errors with respect to the reference exact computation on various task trees.

Table 1 shows the results of the main experiment. The quality of the solutions provided by using the $\mathrm{OptTrim}$ operator are better (lower errors) than those provided by the Trim operator, following the optimality guarantees, but is interesting to see that the quality gaps happen in practice in each of the examined task trees. However, in some of the task trees the sampling method produced better results than the approximation algorithm with $\mathrm{OptTrim}$. Nevertheless, the approximation algorithm comes with an inherent advantage of providing an exact quality guarantees, as opposed to the probabilistic guarantees provided by sampling.

In order to better understand the quality gaps in practice between $\mathrm{OptTrim}$ and Trim, we investigate their relative errors when applied on single random variables with different sizes of the support (M), and different support sizes of the resulting random variable approximation (m). 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.

m	$\mathrm{OptTrim}$	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%

Table 2: $\mathrm{OptTrim}$ vs. Trim on randomly generated random variables with original support size $M = 100$.

m	$\mathrm{OptTrim}$	Trim	Relative error
50	0.0193	0.0199	3.4%
100	0.0093	0.0099	7.1%
200	0.0043	0.0049	15.7%

Table 3: $\mathrm{OptTrim}$ vs. Trim on randomly generated random variables with original support size $M = 1000$.

Tables 2 and 3 present the error produced by $\mathrm{OptTrim}$ and Trim on random variables with supports sizes of $M = 100$ and $M = 1000$, respectively. The depicted results in these tables are averages over several instances of random variables for each entry (50 instances in Table 2 and 10 instances in Table 3). 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 $\mathrm{OptTrim}$; the relative error of each instance is calculated by $(\mathrm{Trim} \,/\, \mathrm{OptTrim}) - 1$. 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.

The above experiments display the quality of approximation provided by the $\mathrm{OptTrim}$ algorithm, but it comes with a price tag in the form of run-time performance. The time complexity of both the Trim operator and the sampling method is linear in the number of variables, resulting in much faster run-time performances than $\mathrm{OptTrim}$, for which the time complexity is only polynomial (Theorem 13), not linear. The run-time of the exact computation, however, may grow exponentially. Therefore, we examine in the next experiment the problem sizes in which it becomes beneficial in terms of run-time to use the proposed approximation.

Figure 2 presents a comparison of the run-time performances of an exact computation and approximated computations with $\mathrm{OptTrim}$ and Trim as operators. The computation is a summation of a sequence of random variables with support size of M=10, where the number N of variables varies from 6 to 19. In this experiment, we executed the $\mathrm{OptTrim}$ operator with m=10 after performing each convolution between two random variables, in order to maintain a support size of 10 in all intermediate computations. Equivalently, we executed the Trim operator with $\varepsilon = 0.1$. The results clearly show the exponential run-time of the exact computation, caused by the convolution between two consecutive random variables. In fact, in the experiment with N=20, the exact computation ran out of memory. These results illuminate the advantage of the proposed $\mathrm{OptTrim}$ algorithm that balances between solution quality and run-time performance – while there exist other, faster, methods (e.g., Trim), $\mathrm{OptTrim}$ provides high-quality solutions in reasonable (polynomial) time, which is especially important when an exact computation is not feasible, due to time or memory.

References

[1] A. Chakravarty, J. Orlin, and U. Rothblum. A partitioning problem with additive objective with an application to optimal inventory groupings for joint replenishment. *Operations Research*,

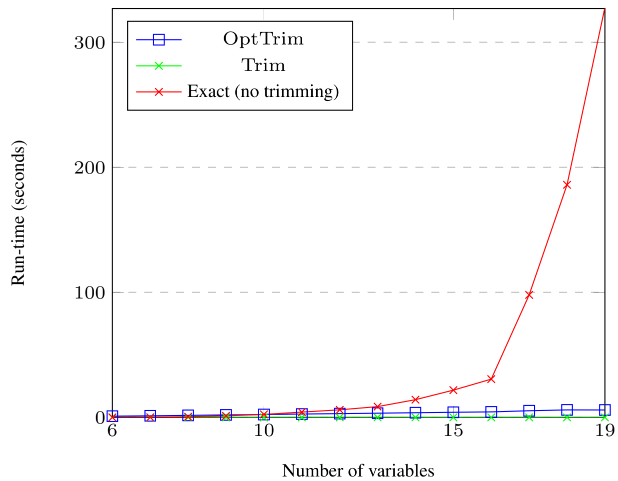

Figure 2: Run-time of a long computation with $\mathrm{OptTrim}$, with Trim, and without any trimming (exact computation).

30(5):1018–1022, 1982.

[2] J. D. Gibbons and S. Chakraborti. Nonparametric statistical inference. In *International encyclopedia of statistical science*, pages 977–979. Springer, 2011.

[3] A. C. Miller and T. R. Rice. Discrete approximations of probability distributions. *Management Science*, 29(3):352–362, 1983.

[4] D. S. Nau, T.-C. Au, O. Ilghami, U. Kuter, J. W. Murdock, D. Wu, and F. Yaman. SHOP2: An HTN planning system. *Journal of Artificial Intelligence Research*, 20:379–404, 2003.

[5] K. Pavlikov and S. Uryasev. CVaR distance between univariate probability distributions and approximation problems. Technical Report 2015-6, University of Florida, 2016.

[6] A. N. Pettitt and M. A. Stephens. The kolmogorov-smirnov goodness-of-fit statistic with discrete and grouped data. *Technometrics*, 19(2):205–210, 1977.

[7] E. Shufan, H. Ilani, and T. Grinshpoun. A two-campus transport problem. In *MISTA*, pages 173–184, 2011.