
Kolmogorov Approximation

Anonymous Author(s)

Affiliation

Address

email

1 Introduction

There are many report in the literature of the need to approximate probability distributions. Usually a continuous distribution is approximated by a discrete one with a small support (number of outcomes), but the same is relevant when the distribution to be approximated is given as a large discrete table. The methods differ in the type of approximation they seek.

In the statistical literature, the distance between distributions is often considered to be the distances between the corresponding commutative distribution functions (cdf). One of the most widely known distances is the, so called, Kolmogorov-Smirnov distance, which leads to the corresponding goodness of fit test, see for instance ? and ?. This distance is based on a single point where the absolute difference between two cdfs is maximized, and equals to the corresponding value of the absolute difference.

approximations are sometimes employed.

In this work, motivated by the problem of estimating the probability of meeting deadlines, we focus on the Kolmogorov distance $d_k(X, X') = \sup_t |F_X(t) - F_{X'}(t)|$ where F_X and $F_{X'}$ are the CDFs of X and X' , respectively.

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

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 $\text{support}(X') \subseteq \text{support}(X)$.

Lemma 2. For any discrete random variable X and any $m \in \mathbb{N}$, there is an m -optimal-approximation X' of X such that $\text{support}(X') \subseteq \text{support}(X)$.

Proof. Assume there is a random variable X'' with support size m such that $d_K(X, X'')$ is minimal but $\text{support}(X'') \not\subseteq \text{support}(X)$. We will show how to transform X'' support such that it will be contained in $\text{support}(X)$. Let v' be the first $v' \in \text{support}(X'')$ and $v' \notin \text{support}(X)$. Let $v = \max\{i : i < v' \wedge i \in \text{support}(X)\}$. 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 . \square

Observation 3. $\max\{|a|, |b|\} \geq |a - b|/2$

• 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 4. For $x_1, x_2 \in \text{support}(X) \cup \{-\infty, \infty\}$ 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 $\hat{x} = \max\{x \in \text{support}(X) \cap \{-\infty, \infty\} : x < x_2\}$. 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 3, $d_k(X, X') \geq 1/2|F_X(x_1) -$
 $F_X(\hat{x}) + F_{X'}(\hat{x}) - F_{X'}(x_1)|$. Since it is given that $F_{X'}(\hat{x}) - F_{X'}(x_1) = P(x_1 < X' < x_2) = 0$,
 $d_k(X, X') \geq 1/2|F_X(x_1) - F_X(\hat{x})| = P(x_1 < X \leq \hat{x})/2 = P(x_1 < X < x_2)/2$. \square

• The next lemma strengthen the lower bound.

Lemma 5. For $x_1, x_2 \in \text{support}(X) \cup \{-\infty, \infty\}$ 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 $\hat{x} = \max\{x \in \text{support}(X) \cap \{-\infty, \infty\} : x < x_2\}$. 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
 $d_k(X, X') \geq |F_X(x_1) - 1| = |1 - F_X(\hat{x})| = P(x_1 < X \leq \hat{x}) = P(x_1 < X < x_2)$. \square

Definition 6. For $x_1, x_2 \in \text{support}(X) \cup \{-\infty, \infty\}$ 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 7. For $S = \{x_1 < \dots < x_m\} \subseteq \text{support}(X)$

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

where, to simplify notations, we assume that $x_0 = -\infty$ and $x_{m+1} = \infty$.

Proposition 8. There is no X' such that $\text{support}(X') = S$ 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 4. If $i = 0$ or $i = n + 1$ the same follows from Lemma 5. \square

Proposition 9. There exists X' such that $\text{support}(X') = S$ 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 $i = 1, \dots, m$ and
 $f_{X'}(x) = 0$ for $x \notin S$. \square

Chakravarty, Orlin, and Rothblum Chakravarty et al. (1982) 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 KolmogorovApprox algorithm (Algorithm 2) starts by constructing a directed weighted graph G similar to the method of Chakravarty, Orlin, and Rothblum Chakravarty et al. (1982). 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 one 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 Shufan et al. (2011). 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 then iterate the path l^* to find 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) is 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. $j == \infty$) then node i gets the full weight $w(i, j)$ in addition to what was already accumulated such that $f_{X'}(i) = f_X(i) + w(i, j)$ (lines 13-14).

Algorithm 1: KolmogorovApprox(X, m)

```

1  $S = \text{support}(X) \cup \{\infty, -\infty\}$ 
2  $G = (V, E) = (S, \{(x, y) \in S^2 : x < y\})$ 
3  $l = \text{argmin}_{l \in paths(G, -\infty, \infty), |l| \leq m} \max\{w(e) : e \in l\}$ 
4 foreach  $e = (x, y) \in l$  do
5   if  $x \neq -\infty \wedge y \neq \infty$  then
6      $f_{X'}(j) = f_X(j) + Pr(i \leq X < j)$ 
7   else if  $j == \infty$  then
8      $f_{X'}(i) = f_X(i) + Pr(i \leq X < j)$ 
9   else
10     $f_{X'}(i) = f_X(i) + Pr(i \leq X < j)/2$ 
11     $f_{X'}(j) = f_X(j) + Pr(i \leq X < j)/2$ 
12 return  $X'$ 

```

Algorithm 2: KolmogorovApprox(X, m)

```
1  $S = \text{support}(X) \cup \{\infty, -\infty\}$ 
2  $G = (V, E) = (S, \{(x, y) \in S^2 : x < y\})$ 
3 foreach  $e = (x, y) \in E$  do
4   if  $i = \infty$  OR  $j = -\infty$  then
5      $w(e) = \text{Pr}(i < X < j)$ 
6   else
7      $w(e) = \text{Pr}(i < X < j)/2$ 
8 /* The following can be obtained, e.g., using the Bellman-Ford algorithm */
9  $l^* = \text{argmin}_{l \in \text{paths}(G, -\infty, \infty, |l| \leq m)} \max\{w(e) : e \in l\}$ 
10 foreach  $e = (i, j) \in l^*$  do
11   if  $i = -\infty$  then
12      $f_{X'}(j) = f_X(j) + \text{Pr}(i \leq X < j)$ 
13   else if  $j = \infty$  then
14      $f_{X'}(i) = f_X(i) + \text{Pr}(i \leq X < j)$ 
15   else
16      $f_{X'}(i) = f_X(i) + \text{Pr}(i \leq X < j)/2$ 
17      $f_{X'}(j) = f_X(j) + \text{Pr}(i \leq X < j)/2$ 
18 return  $X'$ 
```

92 **Theorem 10.** *The KolmogorovApprox(X, m) algorithm runs in time $O(mn^2)$, using $O(n^2)$ mem-*
93 *ory where $n = |\text{support}(X)|$.*

94 *Proof.* Constructing the graph G takes $O(n^2)$. The number of edges is $O(E) \approx O(n^2)$ and for every
95 edge the weight is at most the sum of all probabilities between the source node $-\infty$ and the target
96 node ∞ , which can be done efficiently by aggregating the weights of already calculated edges. The
97 construction is also the only stage that requires memory allocation, specifically $O(E + V) = O(n^2)$.
98 Finding the shortest path takes $O(m(E + V)) \approx O(mn^2)$. Since G is DAG (directed acyclic graph)
99 finding shortest path takes $O(E + V)$. We only need to find paths of length $\leq m$, which takes
100 $O(m(E + V))$. Deriving the new random variable X' from the computed path l^* takes $O(mn)$. For
101 every node in l^* (at most m nodes), calculating the probability $P(s < X < \infty)$ takes at most n .
102 To conclude, the worst case run-time complexity is $O(n^2 + mn^2 + mn) = O(mn^2)$ and memory
103 complexity is $O(E + V) = O(n^2)$. \square

104 3 Experiments and Results

105 In the first experiment we focus on the problem of task trees with deadlines, and consider three
106 types of task trees. The first type includes logistic problems of transporting packages by trucks and
107 airplanes (from IPC2 <http://ipc.icaps-conference.org/>). Hierarchical plans of those logistic problems
108 were generated by the JSHOP2 planner Nau et al. (2003) (see example problem, Figure 1). The
109 second type consists of task trees used as execution plans for the ROBIL team entry in the DARPA
110 robotics challenge (DRC simulation phase), and the third type is of linear plans (sequential task trees).
111 The primitive tasks in all the trees are modeled as discrete random variables with support of size M
112 obtained by discretization of uniform distributions over various intervals. The number of tasks in a
113 tree is denoted by N .

114 We implemented the approximation algorithm for solving the deadline problem with four different
115 methods of approximation. The first two are for achieving a one-sided Kolmogorov approximation –

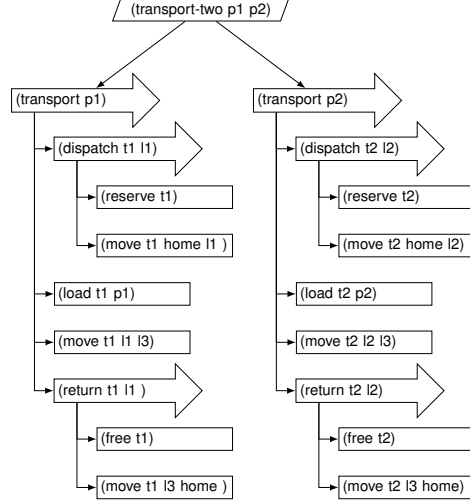


Figure 1: A plan generated by the JSHOP2 algorithm. Arrow shapes represent sequence nodes, parallelograms represent parallel nodes, and rectangles represent primitive nodes.

the OptTrim and the Trim operators, and a simple sampling scheme which we used as comparison to the Kolmogorov approximation with the KolmogorovApprox algorithm. The parameter m of OptTrim and KolmogorovApprox corresponds to the inverse of ε given to the Trim operator. Note 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 KolmogorovApprox as its operator with $m = 10 \cdot N$, the 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 | 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.

123

Table 1 shows the results of the main experiment. The quality of the solutions provided by using the 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 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 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

| m | 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: OptTrim vs. Trim on randomly generated random variables with original support size $M = 100$.

| m | 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: OptTrim vs. Trim on randomly generated random variables with original support size $M = 1000$.

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 2 and 3 present the error produced by OptTrim and Trim on random variables with support 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 OptTrim; the relative error of each instance is calculated by $(\text{Trim} / \text{OptTrim}) - 1$. 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 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 OptTrim, for which the time complexity is only polynomial (Theorem 10), 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 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 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 OptTrim algorithm that balances between solution quality and run-time performance – while there exist other, faster, methods (e.g., Trim), 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.

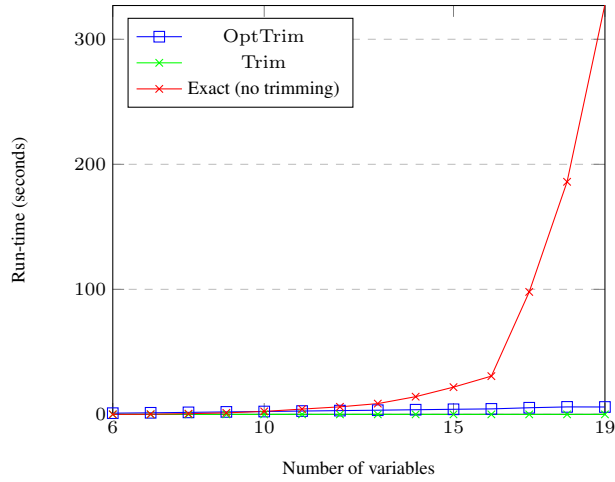


Figure 2: Run-time of a long computation with OptTrim, with Trim, and without any trimming (exact computation).

References

- Chakravarty, A., Orlin, J., and Rothblum, U. (1982). A partitioning problem with additive objective with an application to optimal inventory groupings for joint replenishment. *Operations Research*, 30(5):1018–1022.
- Nau, D. S., Au, T.-C., Ilghami, O., Kuter, U., Murdock, J. W., Wu, D., and Yaman, F. (2003). SHOP2: An HTN planning system. *Journal of Artificial Intelligence Research*, 20:379–404.
- Shufan, E., Ilani, H., and Grinshpoun, T. (2011). A two-campus transport problem. In *MISTA*, pages 173–184.