# Estimating the Probability of Meeting a Deadline in Hierarchical Plans

**Liat Cohen** and **Solomon Eyal Shimony** and **Gera Weiss**
Computer Science Department
Ben Gurion University of The Negev
Beer-Sheva, Israel 84105
{liati,shimony,geraw}@cs.bgu.ac.il

## Abstract

Given a hierarchical plan (or schedule) with uncertain task times, we may need to determine the probability that a given plan will satisfy a given deadline. This problem is shown to be NP-hard for series-parallel hierarchies. We provide a polynomial-time approximation algorithm for it. Computing the expected makespan of an hierarchical plan is also shown to be NP-hard. We examine the approximation bounds empirically and demonstrate where our scheme is superior to sampling and to exact computation.

## 1 Introduction

Numerous planning tools produce plans that call for executing tasks non-linearly. Usually, such plans are represented as a tree, where the leaves indicate primitive tasks, and other nodes represent compound tasks consisting of executing their sub-tasks either in parallel (also called "concurrent" tasks [Gabaldon, 2002]) or in sequence. [Erol *et al.*, 1994; Nau *et al.*, 1998; 2003; Kelly *et al.*, 2008].

Given such a hierarchical plan representation, it is frequently of interest to evaluate its desirability in terms of resource consumption, such as fuel, cost, or time. The answer to such questions can be used to decide which of a set of plans, all valid as far as achieving the goal(s) are concerned, is better given a user-specified utility function. Another reason to compute these distributions is to support runtime monitoring of resources, generating alerts to the execution software or human operator if resource consumption in practice has a high probability of surpassing a given threshold.

While most tools aim at good average performance of the plan, in which case one may ignore the full distribution and consider only the expected resource consumption [Bonfietti *et al.*, 2014], our paper focuses on providing guarantees for the probability of meeting deadlines. This type of analysis is needed, e.g., in Service-Level-Agreements (SLA) where guarantees of the form: "response time less than 1mSec in at least 95% of the cases" are common [Buyya *et al.*, 2011] Section 8 discusses additional related work.

We assume that an hierarchical plan is given in the form of a tree, with uncertain resource consumption of the primitive actions in the network, provided as a probability distribution. The problem is to compute a property of interest of the distribution for the entire task network. In this paper, we focus mainly on the issue of computing the probability $P(t < T)$ of satisfying a deadline $T$ (i.e. that the *makespan* $t$ of the plan is less than a given value). Since in the above-mentioned applications for these computations, one needs results in real-time (for monitoring) or multiple such computations (in comparing candidate plans), efficient computation here is crucial, and is more important than in, e.g., off-line planning.

We show that computing $P(t < T)$ is NP-hard (see Section 6) even for the simple sum of independent random variables (r.v.s) , the first contribution of this paper. A deterministic polynomial-time approximation scheme for this problem is proposed, the second contribution of this paper. Error bounds are analyzed and are shown to be tight. For discrete r.v.s with finite support, finding the distribution of the maximum can be done in low-order polynomial time. However, when compounded with errors generated due to approximation in subtrees, handling this case requires careful analysis of the resulting error. The approximations developed for both sequence and parallel nodes are combined into an overall algorithm for task trees, with an analysis of the resulting error bounds, yielding a polynomial-time (additive error) approximation scheme for computing the probability of satisfying a deadline for the complete network, another contribution of this paper.

We also briefly consider computing *expected* makespan. Since for discrete r.v.s, in parallel nodes one can compute an exact distribution efficiently, it is easy to compute an *expected makespan* in this case as well as for sequence nodes. Despite that, we show that for trees with both parallel and sequence nodes, computing the expected makespan is NP-hard.

Experiments are provided in order to examine the quality of approximation in practice when compared to the theoretical error bounds. A simple sampling scheme is also provided as a yardstick, even though the sampling does not come with error guarantees, but only bounds in probability. Finally, we examine our results in light of related work in the fields of planning and scheduling, as well as probabilistic reasoning.

## 2 Problem statement

We are given a hierarchical plan represented as a task tree consisting of three types of nodes: primitive actions as leaves, sequence nodes, and parallel nodes. Primitive action nodes

contain distributions over their resource consumption. Although other resources can be handled with our approach (e.g. memory: tasks running in parallel use the sum of the memory space of each of the tasks; if they run in sequence, only the maximum thereof is needed), we will assume henceforth, in order to be more concrete, that the only resource of interest is time. A sequence node $v_s$ denotes a task that has been decomposed into subtasks, represented by the children of $v_s$, which must be executed in sequence in order to execute $v_s$. We assume that a subtask of $v_s$ begins as soon as its predecessor in the sequence terminates. Task $v_s$ terminates when its last subtask terminates. A parallel node $v_p$ also denotes a decomposed task, but subtasks begin execution in parallel immediately when task $v_p$ begins execution; $v_p$ terminates as soon as all of the children of $v_p$ terminate.

Resource consumption is uncertain, and described as probability distributions in the leaf nodes. We assume that the distributions are independent (but *not* necessarily identically distributed). We also assume initially that the r.v.s are discrete and have finite support (i.e. number of values for which the probability is non-zero). As the resource of interest is assumed to be completion time, let each leaf node $v$ have a completion-time distribution $P_v$, in some cases represented as a cumulative distribution function form (CDF) $F_v$.

The main computational problem tackled in this paper is: Given a task tree $\tau$ and a deadline $T$, compute the probability that $\tau$ satisfies the deadline $T$ (i.e. terminate in time $t \leq T$). We show that this problem is NP-hard and provide an approximation algorithm. The above *deadline problem* reflects a step utility function: a constant positive utility $U$ for all $t$ less than or equal to a deadline time $T$, and 0 for all $t > T$. We also briefly consider a linear utility function, requiring computation of the expected completion time of $\tau$, and show that this *expectation problem* is also NP-hard.

## 3 Sequence nodes

Since the makespan of a sequence node is the sum of durations of its components, the deadline problem on sequence nodes entails computation of (part of) the CDF of a sum of r.v.s which is an NP-hard problem (shown in Section 6). Thus, there is a need for an approximation algorithm for sequence nodes. The main ingredient in our approximation scheme is the $\mathrm{Trim}$ operator specified as follows:

**Definition 1 (The $\mathrm{Trim}$ operator).** *For a discrete r.v. $X$ and a parameter $\varepsilon > 0$, consider the sequence of elements in the support of $X$ defined recursively by: $x_1 = \min \mathrm{support}(X)$ and, if the set $C = \{x > x_i \colon Pr(x_i < X \leq x) > \varepsilon\}$ is not empty, $x_{i+1} = \min\{x > x_i \colon Pr(x_i < X \leq x) > \varepsilon\}$. Let $l$ be the length of this sequence, i.e., let $l$ be the first index for which $C$ is empty. For notational convenience, define $x_{l+1} = \infty$. Now, define $X' = \mathrm{Trim}(X, \varepsilon)$ to be the random variable specified by:*

$$Pr(X'{=}x) := \begin{cases} Pr(x_i \leq X < x_{i+1}) & \text{if } x{=}x_i \in \{x_1, \ldots, x_l\}; \\ 0 & \text{otherwise.} \end{cases}$$

For example, if $X$ is a r.v. such that $Pr(X{=}1){=}0.1, Pr(X{=}2){=}0.1$ and $Pr(X{=}4){=}0.8$, the r.v. $X'{=}\mathrm{Trim}(X, 0.5)$ is given by $Pr(X'{=}1){=}0.2$ and $Pr(X'{=}4){=}0.8$. Intuitively, the $\mathrm{Trim}$ operator removes consecutive domain values whose accumulated probability is less than $\varepsilon$ and adds their probability mass to the element in the support that precedes them.

Using the $\mathrm{Trim}$ operator, we are now ready to introduce the main operator of this section:

**Definition 2.** *Let* $\mathrm{Sequence}(X_1, \ldots, X_n, \varepsilon)$ *be* $\mathrm{Trim}(\mathrm{Sequence}(X_1, \ldots, X_{n-1}, \varepsilon) + X_n, \varepsilon)$ *and* $\mathrm{Sequence}(X_1, \varepsilon) = \mathrm{Trim}(X_1, \varepsilon)$.

$\mathrm{Sequence}$ takes a set of r.v.s and computes a r.v. that represents an approximation of their sum by applying the $\mathrm{Trim}$ operator after adding each of the variables. The parameter $\varepsilon$, (see below) specifies the accuracy of approximation.

The $\mathrm{Sequence}$ operator can be implemented by the procedure outlined in Algorithm 1. The algorithm computes the distribution $\sum_{i=0}^{n} X_i$ using convolution (the Convolve() operator in line 3) in a straightforward manner. Computing the convolution is itself straightforward and not further discussed here. However, since the support of the resulting distribution may be exponential in the number of convolution operations, the algorithm must trim the distribution representation to avoid this exponential blow-up. This decreases the support size, while introducing error. The trick is to keep the support size under control, while making sure that the error does not increase beyond a desired tolerance. Note that the size of the support can also be decreased by simple "binning" schemes, but these do not provide the desired guarantees. In the algorithm, the PDF of a r.v. $X_i$ is represented by the list $D_{X_i}$, which consists of $(x, p')$ pairs, where $x \in \mathrm{support}(X_i)$ and $p'$ is the probability $Pr(X_i{=}x)$, the latter denoted by $D_{X_i}[x]$. We assume that $D_{X_i}$ is kept sorted in increasing order of $x$.

**Algorithm 1:** Sequence $(X_1, \ldots, X_n\ , \varepsilon)$

```
1  D = ((0,1)) // Dummy random var.: 0 with prob. 1
2  for i = 1 to n do
3      D=Convolve(D, D_{X_i})
4      D=Trim(D, ε)
5  return D
6  Procedure Trim(D,ε)
7      D' = (), p = 0
8      d_0 = d_prev = min support(D)
9      foreach d ∈ support(D) \ {d_0} in ascending order
       do
10         if p + D[d] ≤ ε then
11             p = p + D[d]
12         else
13             Append (d_prev, D[d_prev] + p) to D'
14             d_prev = d, p = 0
15     Append (d_prev, D[d_prev] + p) to D'
16     return D'
```

We proceed to show that Algorithm 1 indeed approximates the sum of the r.v.s, and analyze its accuracy/efficiency tradeoff. A notion of approximation relevant to deadlines is:

**Definition 3.** *For r.v.s $X'$, $X$, and $\varepsilon \in [0, 1]$ we say $X' \approx_\varepsilon X$ if $0 \leq Pr(X' \leq T) - Pr(X \leq T) \leq \varepsilon$ for all $T > 0$.*

Note that this definition is asymmetric, because, as shown below, our algorithm never underestimates the exact value. For the proof of the Lemma 1 below, we establish the following technical claim (can be proven by induction on $n$):

**Claim 1.** *Let $(a_i)_{i=1}^n$ and $(b_i)_{i=1}^n$ be sequences of real numbers such that $\sum_{i=1}^k a_i \geq 0$ for all $1 \leq k \leq n$ and $b_1 \geq b_2 \geq \cdots \geq b_n \geq 0$, then $\sum_{i=1}^n a_i b_i \geq 0$*

We now bound the approximation error of sums of r.v.s:

**Lemma 1.** *For discrete random variables $X_1, X_1', X_2, X_2'$ and $\varepsilon_1, \varepsilon_2 \in [0,1]$, if $X_1' \approx_{\varepsilon_1} X_1$ and $X_2' \approx_{\varepsilon_2} X_2$, then $X_1' + X_2' \approx_{\varepsilon_1+\varepsilon_2} X_1 + X_2$.*

*Proof.* $Pr(X_1' + X_2' \leq T) - Pr(X_1 + X_2 \leq T)$

$$= \sum_{j=1}^{T} Pr(X_1'=j) \underbrace{Pr(X_2' \leq T-j)}_{\leq Pr(X_2 \leq T-j)+\varepsilon_2} - \sum_{j=1}^{T} Pr(X_1=j) Pr(X_2 \leq T-j)$$

$$\leq \sum_{j=1}^{T} (Pr(X_1'=j) - Pr(X_1=j)) \underbrace{Pr(X_2 \leq T-j)}_{\in[0,1]} + \varepsilon_2 \sum_{j=1}^{T} Pr(X_1'=j)$$

$$\leq \underbrace{\sum_{j=1}^{T} (Pr(X_1'=j) - Pr(X_1=j))}_{\leq \varepsilon_1} + \varepsilon_2 \underbrace{\sum_{j=1}^{T} Pr(X_1'=j)}_{\in[0,1]} \leq \varepsilon_1 + \varepsilon_2.$$

Finally, we show that the difference is also nonnegative:

$$Pr(X_1' + X_2' \leq T) - Pr(X_1 + X_2 \leq T)$$

$$= \sum_{j=1}^{T} Pr(X_1'=j) Pr(X_2' \leq T-j) - \sum_{j=1}^{T} Pr(X_1=j) Pr(X_2 \leq T-j)$$

$$= \sum_{j=1}^{T} (Pr(X_1'=j) - Pr(X_1=j)) Pr(X_2 \leq T-j) + \sum_{j=1}^{T} Pr(X_1'=j)(Pr(X_2' \leq T-j) - Pr(X_2 \leq T-j))$$

The first term here is non-negative by Claim 1, the second is nonnegative because it is a sum of nonnegative numbers. □

We now show that $\texttt{Trim}(X, \varepsilon)$ is an $\varepsilon$-approximation of $X$:

**Lemma 2.** $\texttt{Trim}(X, \varepsilon) \approx_\varepsilon X$

*Proof.* Let $X' = \texttt{Trim}(X, \varepsilon)$. Let $x_1 < \cdots < x_m$ be the support of $X'$ and let $l = \max\{i \colon x_i \leq T\}$. We have,

$$Pr(X' = x_i) = Pr(x_i \leq X < x_{i+1}) \tag{1}$$

because, after Trim, the probabilities of elements that were removed from the support are assigned to the element that precedes them. From Equation (1) we get:

$$Pr(X' \leq T) - Pr(X \leq T)$$

$$= \sum_{i=0}^{l-1} (Pr(X'=x_i) - Pr(x_i \leq X < x_{i+1})) + (Pr(X'=x_l) - Pr(x_l \leq X \leq T))$$

$$= Pr(X'=x_l) - (Pr(x_l \leq X < x_{l+1}) - Pr(T < X < x_{l+1}))$$

$$= Pr(T < X < x_{l+1}) \in (0, \varepsilon]$$

The inequality $Pr(T < X < x_{l+1}) \leq \varepsilon$ follows from the observation that, for all $i$, $Pr(x_i < X < x_{i+1}) < \varepsilon$, because $p$ is never greater than $\varepsilon$ in Algorithm 1. □

To bound the amount of memory needed for our approximation algorithm, the next lemma bounds the size of the support of the trimmed r.v.:

**Lemma 3.** $|\operatorname{support}(\texttt{Trim}(X, \varepsilon))| \leq 1/\varepsilon$

*Proof.* Let $X' = \texttt{Trim}(X, \varepsilon)$ and let $x_1 < \cdots < x_m$ be the support of $X'$. And, for notational convenience, let $x_{m+1} = \infty$. Let $p_i = \sum_{x_i < x < x_{i+1}} Pr(X=x)$. Then, $1 = \sum_{i=1}^m Pr(X'=x_i) = Pr(X=x_1) + \sum_{i=1}^{m-1}(p_i + Pr(X=x_{i+1})) + p_m$. According to algorithm 1, lines 11-12, $p_i \leq \varepsilon$ and $p_i + Pr(X=x_{i+1}) > \varepsilon$ for all $0 \leq i < m$. Therefore, $1 = \sum_{i=1}^m Pr(X'=x_i) > Pr(X'=x_1) + (m-1) \cdot \varepsilon + p_m$. Using the fact $Pr(X'=x_1) > 0$, we get: $(m-1) \cdot \varepsilon < 1$, therefore $m \leq 1/\varepsilon$. □

These lemmas highlight the main idea behind our approximation algorithm: the Trim operator trades off approximation error for a reduced size of the support. The fact that this trade-off is linear allows us to get a linear approximation error in polynomial time, as shown below:

**Theorem 1.** *If $X_i' \approx_{\varepsilon_i} X_i$ for all $i \in \{1, \ldots, n\}$ and $\hat{X} = Sequence(X_1', \ldots, X_n', \varepsilon)$ then $\hat{X} \approx_e \sum_{i=1}^n X_i$, where $e = \sum_{i=1}^n \varepsilon_i + n\varepsilon$.*

*Proof.* (outline) For $n$ iterations, from Lemma 1, we get an accumulated error of $\varepsilon_1 + \cdots + \varepsilon_n$. From Lemma 2, we get an additional error of at most $n\varepsilon$ due to trimming. □

**Theorem 2.** *Assuming that $m \leq 1/\varepsilon$, the procedure $Sequence(X_1', \ldots, X_n', \varepsilon)$ can be computed in time $O((nm/\varepsilon)\log(m/\varepsilon))$ using $O(m/\varepsilon)$ memory, where $m$ is the size of the largest support of any of the $X_i'$s.*

*Proof.* From Lemma 3, the size of list $D$ in Algorithm 1 is at most $m/\varepsilon$ just after the convolution, after which it is trimmed, so the space complexity is $O(m/\varepsilon)$. Convolve thus takes time $O((m/\varepsilon)\log(m/\varepsilon))$, where the logarithmic factor is required internally for sorting. Since the runtime of the Trim operator is linear, and the outer loop iterates $n$ times, the overall run-time of the algorithm is $O((nm/\varepsilon)\log(m/\varepsilon))$. □

**Example 1.** *The error bound provided in Theorem 1 is tight, i.e. $Sequence(X_1, \ldots, X_n, \varepsilon/n)$ may results in error $\varepsilon$: Let $0 \leq \varepsilon < 1$ and $n \in \mathbb{N}$ such that $1-\varepsilon > \varepsilon/n$. Consider, for very small $\delta > 0$, the r.v. $X_1$ defined by:*

$$Pr(X_1=x) = \begin{cases} \delta & x = 0, \\ \varepsilon/(n(1-\delta)^x) & x \in \{1, \ldots, n\}, \\ 1-\delta-\sum_{x=1}^n \frac{\varepsilon}{n(1-\delta)^x} & x = n+1, \\ 0 & \textit{otherwise} \end{cases}$$

*and, for $i = 2, \ldots, n$, let the r.v.s $X_i$ be such that $Pr(X_i=0) = 1-\delta$, $Pr(X_i=n^2) = \delta$, and zero otherwise.*

## 4 Parallel nodes

Unlike sequence composition, the deadline problem for parallel composition is easy to compute, since the execution time of a parallel composition is the maximum of the durations:

$$Pr(\max_{i\in[1:n]} X_i \le T) = Pr(\bigwedge_{i=1}^{n} X_i \le T) = \prod_{i=1}^{n} Pr(X_i \le T) \quad (2)$$

where the last equality follows from independence of the r.v.s. We denote the construction of the CDF using Equation (2) by $\texttt{Parallel}(X_1,\ldots,X_n)$. If the r.v.s are all discrete with finite support, $\texttt{Parallel}(X_1,\ldots,X_n)$ incurs linear space, and computation time $O(nmlog(n))$.

If the task tree consists only of parallel nodes, one can compute the exact CDF, with the same overall runtime. However, when the task tree contain both sequence and parallel nodes we may get only approximate CDFs as input, and now the above straightforward computation can compound the errors. When the input CDFs are themselves approximations, we bound the resulting error:

**Lemma 4.** *For discrete r.v.s $X'_1,\ldots X'_n$, $X_1,\ldots,X_n$, if for all $i=1,\ldots,n$, $X'_i \approx_{\varepsilon_i} X_i$ and $0 \le \varepsilon_i \le \frac{1}{n(Kn+1)}$ for some $K>0$, then, for any $\varepsilon \ge \varepsilon_i$, we have: $\max_{i\in[1:n]} X'_i \approx_e \max_{i\in[1:n]} X_i$ where $e = \sum_{i=1}^{n} \varepsilon_i + \varepsilon/K$.*

*Proof.*

$$\begin{aligned}
&Pr(\max_{i\in[1:n]} X'_i \le T) - Pr(\max_{i\in[1:n]} X_i \le T) \\
&\le \prod_{i=1}^{n}(Pr(X_i \le T)+\varepsilon_i) - \prod_{i=1}^{n} Pr(X_i \le T) \\
&\le \prod_{i=1}^{n}(1+\varepsilon_i) - 1 \le 1 + \sum_{i=1}^{n}\varepsilon_i + \sum_{k=2}^{n}\binom{n}{k}\varepsilon^k - 1 \\
&\le \sum_{i=1}^{n}\varepsilon_i + \underbrace{\sum_{k=2}^{n} n^k\varepsilon^k}_{\text{sum of a geo. series}} \le \sum_{i=1}^{n}\varepsilon_i + \frac{n^2\varepsilon^2}{1-n\varepsilon} \le \sum_{i=1}^{n}\varepsilon_i + \varepsilon/K
\end{aligned}$$

Since $Pr(X'_i \le T) > Pr(X_i \le T)$ for each $i$, this expression is nonnegative. □

## 5 Task trees: mixed sequence/parallel

Given a task tree $\tau$ and a accuracy requirement $0<\varepsilon<1$, we generate a distribution for a r.v. $X'_\tau$ approximating the true duration distribution $X_\tau$ for the task tree. We introduce the algorithm and prove that the algorithm indeed returns an $\varepsilon$-approximation of the completion time of the plan. For a node $v$, let $\tau_v$ be the sub tree with $v$ as root and let $child_v$ be the set of children of $v$. We use the notation $|\tau|$ to denote the total number of nodes in $\tau$.

Algorithm 2, that implements the operator `Network`, is a straightforward postorder traversal of the task tree. The only remaining issue is handling the error, in an amortized approach, as seen in the proof of the following theorem.

**Theorem 3.** *Given a task tree $\tau$, let $X_\tau$ be a r.v. representing the true distribution of the completion time for the network. Then* $\texttt{Network}(\tau,\varepsilon) \approx_\varepsilon X_\tau$.

**Algorithm 2:** Network($\tau$, $\varepsilon$)

```
Let v be the root of τ // Hence, τ_v = τ
n_v = |child_v|
if v is a Primitive node then
    return the distribution of v
if v is a Sequence node then
    for c ∈ child_v do
        X_c = Network (τ_c, |τ_c|ε / |τ_v|)
    return Sequence ({X_c}_{c∈child_v}, ε / (n_v |τ_v|))
if v is a Parallel node then
    for c ∈ child_v do
        X_c = Network (τ_c, min(|τ_c|ε / |τ_v|, 1 / (n_v(|τ_v|n_v+1))))
    return Parallel ({X_c}_{c∈child_v})
```

*Proof.* By induction on $|\tau|$. Base: $|\tau|=1$, the node must be primitive, and `Network` will just return the distribution unchanged which is obviously an $\varepsilon$-approximation of itself. Suppose the claim is true for $1 \le |\tau| < n$. Let $\tau$ be a task tree of size $n$ and let $v$ be the root of $\tau$. If $v$ is a Sequence node, by the induction hypothesis that $X_c \approx_{|\tau_c|\varepsilon/|\tau_v|} X_{\tau_c}$, and by Theorem 1, the maximum accumulated error is $\sum_{c\in child_v} |\tau_c|\varepsilon/|\tau_v| + \varepsilon/|\tau_v| = (n-1)\varepsilon/|\tau_v| + \varepsilon/|\tau_v| = \varepsilon$ for $v$, therefore, $\texttt{Sequence}(\{X_c\}_{c\in child_v}, \varepsilon/n) \approx_\varepsilon X_\tau$ as required. If $v$ is a Parallel node, by the induction hypothesis that $X_c \approx_{e_c} X_{\tau_c}$, where $e_c = \min(\frac{|\tau_c|\varepsilon}{|\tau_v|}, \frac{1}{n_v(|\tau_v|n_v+1)})$ so $\sum_{c\in child_v} e_c \le \sum_{c\in child_v} \frac{|\tau_c|\varepsilon}{|\tau_v|} \le \varepsilon - \varepsilon/|\tau_v|$. Then, by Lemma 4, using $K = |\tau_v|$ and $n = n_v$, we get that $\texttt{Parallel}(\{X_c\}_{c\in child_v}) \approx_\varepsilon X_\tau$ as required. □

**Theorem 4.** *Let $N$ be the size of the task tree $\tau$, and $M$ the size of the maximal support of each of the primitive tasks. If $0 \le \varepsilon \le \frac{1}{N(N^2+1)}$ and $M < N/\varepsilon$, the* `Network` *approximation algorithm runs in time $O((N^5/\varepsilon^2)\log(N^3/\varepsilon^2))$, using $O(N^3/\varepsilon^2)$ memory.*

*Proof.* The run-time and space bounds can be derived from the bounds on `Sequence` and on `Parallel`, as follows. In the `Network` algorithm, the trim accuracy parameter is less than or equal to $\varepsilon/N$. The support size (called $m$ in Theorem 2) of the variables input to `Sequence` are $O(N^2/\varepsilon)$. Therefore, the complexity of the `Sequence` algorithm is $O((N^4/\varepsilon^2)\log(N^3/\varepsilon^2))$ and the complexity of the `Parallel` operator is $O((N^3/\varepsilon)\log(N))$. The time and space for sequence dominate, so the total time complexity is $N$ times the complexity of `Sequence` and the space complexity is that of `Sequence`. □

If the constraining assumptions on $M$ and $\varepsilon$ in Theorem 4 are lifted, the complexity is still polynomial: replace one instance of $1/\varepsilon$ by $max(m, 1/\varepsilon)$, and the other by $max(1/\varepsilon, N(N^2+1))$ in the runtime complexity expression.

## 6 Complexity results

We show that the deadline problem is NP-hard, even for a task tree consisting only of primitive tasks and one sequence node, i.e. *linear plans*.

**Lemma 5.** *Let $Y = \{Y_1, \ldots, Y_n\}$ be a set of discrete real-valued r.v.s specified by probability mass functions with finite supports, $T \in \mathbb{Z}$, and $p \in [0, 1]$. Then, deciding whether $Pr(\sum_{i=0}^{n} Y_i < T) > p$ is NP-Hard.*

*Proof.* By reduction from *SubsetSum* [Garey and Johnson, 1990, problem number SP13]. Recall that *SubsetSum* is: given a set $S = \{s_1, \ldots, s_n\}$ of integers, and integer target value $T$, is there a subset of $S$ whose sum is exactly $T$? Given an instance of *SubsetSum*, create the two-valued r.v.s $Y_1, \ldots, Y_n$ with $Pr(Y_i = s_i) = 1/2$ and $Pr(Y_i = 0) = 1/2$. By construction, there exists a subset of $S$ summing to $T$ if and only if $Pr(\sum_{i=0}^{n} Y_i = T) > 0$.

Suppose that algorithm $A(Y, T, p)$ can decide $Pr(\sum_{i=0}^{n} Y_i < T) > p$ in polynomial time. Then, since the r.v.s $Y_i$ are two-valued uniform r.v.s, the only possible values of $p$ are integer multiples of $1/2^n$, and we can compute $p = Pr(\sum_{i=0}^{n} Y_i < T)$ using a binary search on $p$ using $n$ calls to $A$. To determine whether $Pr(\sum_{i=0}^{n} Y_i = T) > 0$, simply use this scheme twice, since $Pr(\sum_{i=0}^{n} Y_i = T) > 0$ is true if and only if $Pr(\sum_{i=0}^{n} Y_i < T) < Pr(\sum_{i=0}^{n} Y_i < T + 1)$. □

**Theorem 5.** *Finding the probability that a task tree satisfies a deadline $T$ is NP-hard.*

*Proof.* Given a task tree consisting of leaf nodes, all being children of a single sequence node, its makespan is the sum of the completion times of the leaves. The theorem follows immediately from Lemma 5. □

Finally, we consider the linear utility function, i.e. the problem of computing an expected makespan of a task network. Note that although for linear plans the *deadline problem* is NP-hard, the *expectation problem* is trivial because the expectation of the sum of r.v.s $X_i$ is equal to the sum of the expectations of the $X_i$s. For *parallel nodes*, it is easy to compute the CDF and therefore also easy to compute the expected value. Despite that, for task networks consisting of *both* sequence nodes and parallel nodes, these methods cannot be effectively combined, and in fact, we have:

**Theorem 6.** *Computing the expected completion time of a task network is NP-hard.*

*Proof.* By reduction from subset sum. Construct r.v.s ("primitive tasks") $Y_i$ as in the proof of Lemma 5, and denote by $X$ the r.v. $\sum_{i=1}^{n} Y_i$. Construct one parallel node with two children, one being the a sequence node having the completion time distribution defined by $X$, the other being a primitive task that has a completion time $T_j$ with probability 1. (We will use more than one such case, which differ only in the value of $T_j$, hence the subscript $j$). Denote by $M_j$ the r.v. that represents the completion time distribution of the parallel node, using this construction, with the respective $T_j$. Now consider computing the expectation of the $M_j$ for the following cases: $T_1 = T + 1/2$ and $T_2 = T + 1/4$. Thus we have, for $j \in \{1, 2\}$, by construction and the definition of expectation:

$$\begin{aligned} E[M_i] &= T_j Pr(X \leq T_j) + \sum_{x > T_j} x \, Pr(X = x) \\ &= T_j Pr(X \leq T) + \sum_{x \geq T+1} x \, Pr(X = x) \end{aligned}$$

where the second equality follows from the $Y_i$ all being integer-valued r.v.s (and therefore $X$ is also integer valued). Subtracting these expectations, we have $E[M_1] - E[M_2] = \frac{1}{4} P(X \leq T)$. Therefore, using the computed expected values, we can compute $P(X \leq T)$, and thus also $P(X = T)$, in polynomial time. □

## 7 Empirical Evaluation

We examine our approximation bounds in practice, and compare the results to exact computation of the CDF and to a simple stochastic sampling scheme. Three types of task trees are used in this evaluation: task trees used as execution plans for the ROBIL team entry in the DARPA robotics challenge (DRC simulation phase, http://in.bgu.ac.il/en/Pages/news/dar_pa.aspx), linear plans (seq), and plans for the Logistics domain (from IPC2 http://ipc.icaps-conference.org/). The primitive task distributions were uniform distributions discretized to $M$ values. For every entry of $M$ in Table 1 the first line is the runtime in seconds, the second line presents the estimation error.

In the Logistics domain, packages are to be transported by trucks or airplanes. Hierarchical plans were generated by the JSHOP2 planner [Nau *et al.*, 2003] for this domain and consisted of one parallel node (packages delivered in parallel), with children all being sequential plans. The duration distribution of all primitive tasks is uniform but the support parameters were determined by the type of the task, in some tasks the distribution is fixed (such as for load and unload) and in others the distribution depends on the velocity of the vehicle and on the distance to be travelled.

After running our approximation algorithm we also ran a variant that uses an inverted version of the `Trim` operator, providing a *lower* bound of the CDF, as well as the upper bound generated by Algorithm 2. Running both variants allows us to bound the actual error, costing only a doubling of the run-time. Despite the fact that our error bound is theoretically tight, in practice and with actual distributions, according to Table 1, the resulting error in the algorithm was usually much better than the theoretical $\varepsilon$ bound.

We ran the exact algorithm, our approximation algorithm with $\varepsilon \in \{0.1, 0.01, 0.001\}$, and a simple simulation with $10^3$ to $10^7$ samples (number of samples is denoted by $s$ in the table), on networks from the DRC implementation, sequence nodes with 10, 20, and 50 children (number of nodes denoted by $N$ in the table), and 20 Logistics domain plans, and several values of $M$ ($M, N$ are as in Theorem 4). Results for a typical indicative subset (regretfully reduced due to page limits) are shown in table 1.

The exact algorithm times out in some cases. Both our approximation algorithm and the sampling algorithm handle

| Task Tree | $M$ | Exact | Approximation | | Sampling | |
|---|---|---|---|---|---|---|
| | | | $\varepsilon$=0.1 | $\varepsilon$=0.01 | $s$=$10^4$ | $s$=$10^6$ |
| DRC-Drive ($N$=47) | 2 | 1.49 | 0.141 | 1.14 | 1.92 | 190.4 |
| | | 0 | [-0.005, 0.009] | [-0.0004, 0.0004] | 0.0072 | 0.0009 |
| | 4 | 18.9 | 0.34 | 7.91 | 2.1 | 211.5 |
| | | 0 | [-0.0096, 0.019] | [-0.0009, 0.0013] | 0.0075 | 0.0011 |
| | 10 | >2h | 1.036 | 32.94 | 2.81 | 279.1 |
| | | 0 | [-0.014, 0.028] | [-0.0014, 0.0025] | 0.0083 | 0.0015 |
| Seq ($N$=10) | 4 | 0.23 | 0.003 | 0.02 | 0.545 | 54.22 |
| | | 0 | [-0.03, 0.04] | [-0.003, 0.004] | 0.008 | 0.0016 |
| | 10 | 10.22 | 0.008 | 0.073 | 0.724 | 72.4 |
| | | 0 | [-0.03, 0.06] | [0.003, 0.007] | 0.0117 | 0.001 |
| Logistics ($N$=45) | 4 | 373.3 | 0.2 | 7 | 2.5 | 256 |
| | | 0 | [-0.004,0.004] | [-0.0004,0.0004] | 0.008 | 0.0006 |
| | 10 | >4h | 2.19 | 120 | 3.12 | 314 |
| | | 0 | [-0.005,0.006] | [-0.0004,0.0006] | 0.013 | 0.001 |

Table 1: Runtime and estimation errors comparison

all these cases, as our algorithm's runtime is polynomial in $N$, $M$, and $1/\varepsilon$ as is the sampling algorithm's (time linear in number of samples).

The advantage of the approximation algorithm is mainly in providing bounds with certainty as opposed to the bounds in-probability provided by sampling. Additionally, as predicted by theory, accuracy of the approximation algorithm improves linearly with $1/\varepsilon$ (and almost linear in runtime), whereas accuracy of sampling improves only as a square root of the number of samples. Thus, even in cases where sampling initially outperformed the approximation algorithm, increasing the required accuracy for both algorithms, eventually the approximation algorithm overtook the sampling algorithm.

## 8 Discussion and Related Work

Numerous issues remain unresolved, briefly discussed below. Trivial improvements to the `Trim` operator are possible, such as the inverse version of the operator used to generate a lower bound for the empirical results. Other candidate improvements are not performing trimming (or even stopping a trimming operation) if the current support size is below $1/\varepsilon$, which may increase accuracy but also the runtime. Another point is that in the combined algorithm, space and time complexity can be reduced by adding some `Trim` operations, especially after processing a parallel node, which is not done in our version. This may reduce accuracy, a trade-off yet to be examined. Another option is, when given a specific threshold, trying for higher accuracy in just the region of the threshold, but how to do that is non-trivial. For *sampling* schemes such methods are known, including adaptive sampling [Bucher, 1988; Lipton *et al.*, 1990], stratified sampling, and other schemes. It may be possible to apply such schemes to deterministic algorithms as well - an interesting issue for future work.

Extension to continuous distributions: our algorithm can handle them by pre-running a version of the `Trim` operator on the primitive task distribution. Since one cannot iterate over support values in a continuous distribution, start with the smallest support value (even if it is $-\infty$), and find the value at which the CDF increases by $\varepsilon$. This requires access to the inverse of the CDF, which is available, either exactly or approximately, for many types of distributions.

We showed that the expectation problem is also NP-hard. A natural question is on approximation algorithms for the expectation problem, but the answer here is not so obvious. Sampling algorithms may run into trouble if the target distribution contains major outliers, i.e. values very far from other values but with extremely low probability. Our approximation algorithm can also be used as-is to estimate the CDF and then to approximate the expectation, but we do not expect it to perform well because our current `Trim` operator only limits the amount of probability mass moved at each location to $\varepsilon$, but does not limit the "distance" over which it is moved. The latter may be arbitrarily bad for estimating the expectation. Possibly adding simple binning schemes to the `Trim` operator in addition to limiting the moved probability mass to $\varepsilon$ may work, another issue for future research.

Related work on computing makespan distributions includes [Hong, 2013], which examines sum of Bernoulli distributed r.v.s. Other work examines both deterministic [Mercier, 2007] and Monte-Carlo techniques [Bucher, 1988; Lipton *et al.*, 1990]. Distribution of maximum of r.v.s was studied in [Devroye, 1980], with a focus mostly on continuous distributions.

Complexity of finding the probability that the makespan is under a given threshold in task networks was shown to be NP-hard in [Hagstrom, 1988], even when the completion time of each task has a Bernoulli distribution. Nevertheless, our results are orthogonal as the source of the complexity in [Hagstrom, 1988] is in the graph structure, whereas in our setting the complexity is due to the size of the support. In fact for linear plans (an NP-hard case in our setting), the probability of meeting the deadline can be computed in low-order polynomial time for Bernoulli distributions, using straightforward dynamic programming. Makespan distributions in series parallel networks in the i.i.d. case was examined in [Gutjahr and Pflug, 1992], without considering algorithmic issues. There is also a significant body of work on estimating the makespan of plans and schedules [Herroelen and Leus, 2005; Fu *et al.*, 2010; Beck and Wilson, 2007], within a context of a planner or scheduler. The analysis in these papers is based on averaging or on limit theorems, and does not provide a guaranteed approximation scheme. Temporal planing and in particular TPNs (temporal plan network) are presented in [Kim *et al.*, 2001], the model is similar to ours, but the focus is on lower/upper bounds, rather than probability distributions. Hierarchical constraint-based plans in MAPGEN [Ai-Chang *et al.*, 2004] allow for more general dependencies than series-parallel, providing additional expressive power but making the deadline problem even harder.

Computing the distribution of the makespan in trees is a seemingly trivial problem in probabilistic reasoning [Pearl, 1988]. Given the task network, it is straightforward to represent the distribution using a Bayes network (BN) that has one node per task, and where the *children* of a node $v$ in the task network are represented by BN nodes that are *parents* of the BN node representing $v$. This results in a tree-shaped BN, where it is well known that probabilistic reasoning can be done in time linear in the number of nodes, e.g. by belief propagation (message passing) [Pearl, 1988; Kim and Pearl, 1983]. The difficulty is in the potentially ex-

ponential size of variable domains, which our algorithm, essentially a limited form of approximate belief propagation, avoids by trimming.

Looking at makespan distribution computation as probabilistic reasoning leads to interesting issues for future research, such as how to handle task completion times that have dependencies, represented as a BN. Since reasoning in BNs is NP-hard even for binary-valued variables [Dagum and Luby, 1993; Cooper, 1990], this is hard in general. But for cases where the BN toplogy is tractable, such as for BNs with bounded treewidth [Bodlaender, 2006], or directed-path singly connected BNs [Shimony and Domshlak, 2003], a deterministic polynomial-time approximation scheme for the makespan distribution may be achievable. The research literature contains numerous *randomized* approximation schemes that handle depenencies [Pearl, 1988; Yuan and Druzdzel, 2006], especially for the case with *no evidence*. In fact, our implementation of the sampling scheme in ROBIL handled dependent durations. It is unclear whether such sampling schemes can be adapted to handle dependencies *and* arbitrary evidence, such as: "the completion time of compound task $X$ in the network is known to be exactly 1 hour from now". Finally, one might consider additional commonly used utility functions, such as a "soft" deadline: the utility is a constant $U$ before the deadline $T$, decreasing linearly to 0 until $T + G$ for some "grace" duration $G$, and 0 thereafter.

**Acknowledgments.** This research was supported by the ROBIL project, by the EU, by the ISF, and by the Lynne and William Frankel Center for Computer Science.

## References

[Ai-Chang *et al.*, 2004] Mitchell Ai-Chang, John Bresina, Len Charest, Adam Chase, JC-J Hsu, Ari Jonsson, Bob Kanefsky, Paul Morris, Kanna Rajan, Jeffrey Yglesias, et al. Mapgen: mixed-initiative planning and scheduling for the mars exploration rover mission. *Intelligent Systems, IEEE*, 19(1):8–12, 2004.

[Beck and Wilson, 2007] J. Christopher Beck and Nic Wilson. Proactive algorithms for job shop scheduling with probabilistic durations. *J. Artif. Intell. Res.(JAIR)*, 28:183–232, 2007.

[Bodlaender, 2006] Hans L. Bodlaender. Treewidth: Characterizations, applications, and computations. In *WG*, pages 1–14, 2006.

[Bonfietti *et al.*, 2014] Alessio Bonfietti, Michele Lombardi, and Michela Milano. Disregarding duration uncertainty in partial order schedules? Yes, we can! In *CPAIOR*, pages 210–225. 2014.

[Bucher, 1988] Christian G. Bucher. Adaptive sampling: an iterative fast Monte Carlo procedure. *Structural Safety*, 5(2):119–126, 1988.

[Buyya *et al.*, 2011] Rajkumar Buyya, Saurabh Kumar Garg, and Rodrigo N. Calheiros. SLA-oriented resource provisioning for cloud computing: Challenges, architecture, and solutions. In *Cloud and Service Computing (CSC)*, 2011.

[Cooper, 1990] Gregory F. Cooper. The computational complexity of probabilistic inference using Bayesian belief networks. *Artificial Intelligence*, 42 (2-3):393–405, 1990.

[Dagum and Luby, 1993] Paul Dagum and Michael Luby. Approximating probabilistic inference in Bayesian belief networks is NP-hard. *Artificial Intelligence*, 60 (1):141–153, 1993.

[Devroye, 1980] Luc Devroye. Generating the maximum of independent identically distributed random variables. *Computers & Mathematics with Applications*, 6(3):305–315, 1980.

[Erol *et al.*, 1994] Kutluhan Erol, James Hendler, and Dana S. Nau. HTN planning: Complexity and expressivity. In *AAAI*, 1994.

[Fu *et al.*, 2010] Na Fu, Pradeep Varakantham, and Hoong Chuin Lau. Towards finding robust execution strategies for RCPSP/max with durational uncertainty. In *ICAPS*, pages 73–80, 2010.

[Gabaldon, 2002] Alfredo Gabaldon. Programming hierarchical task networks in the situation calculus. In *AIPS02 Workshop on On-line Planning and Scheduling*, 2002.

[Garey and Johnson, 1990] Michael R. Garey and David S. Johnson. *Computers and Intractability; A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., NY, USA, 1990.

[Gutjahr and Pflug, 1992] W. J. Gutjahr and G. Ch Pflug. Average execution times of series–parallel networks. *Séminaire Lotharingien de Combinatoire*, 29:9, 1992.

[Hagstrom, 1988] Jane N. Hagstrom. Computational complexity of PERT problems. *Networks*, 18(2):139–147, 1988.

[Herroelen and Leus, 2005] Willy Herroelen and Roel Leus. Project scheduling under uncertainty: Survey and research potentials. *European journal of operational research*, 165(2):289–306, 2005.

[Hong, 2013] Yili Hong. On computing the distribution function for the poisson binomial distribution. *Computational Statistics & Data Analysis*, 59:41–51, 2013.

[Kelly *et al.*, 2008] John Paul Kelly, Adi Botea, and Sven Koenig. Offline planning with Hierarchical Task Networks in video games. In *AIIDE*, pages 60–65, 2008.

[Kim and Pearl, 1983] Jin H. Kim and Judea Pearl. A computation model for causal and diagnostic reasoning in inference systems. In *IJCAI*, 1983.

[Kim *et al.*, 2001] Phil Kim, Brian C. Williams, and Mark Abramson. Executing reactive, model-based programs through graph-based temporal planning. In *IJCAI*, pages 487–493, 2001.

[Lipton *et al.*, 1990] Richard J. Lipton, Jeffrey F. Naughton, and Donovan A. Schneider. *Practical selectivity estimation through adaptive sampling*, volume 19. ACM, 1990.

[Mercier, 2007] Sophie Mercier. Discrete random bounds for general random variables and applications to reliability. *European j. of operational research*, 177(1):378–405, 2007.

[Nau *et al.*, 1998] Dana S. Nau, Stephen J. Smith, Kutluhan Erol, et al. Control strategies in HTN planning: Theory versus practice. In *AAAI/IAAI*, pages 1127–1133, 1998.

[Nau *et al.*, 2003] Dana S. Nau, Tsz-Chiu Au, Okhtay Ilghami, Ugur Kuter, J. William Murdock, Dan Wu, and Fusun Yaman. SHOP2: An HTN planning system. *J. Artif. Intell. Res. (JAIR)*, 20:379–404, 2003.

[Pearl, 1988] Judea Pearl. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. Morgan Kaufmann, San Mateo, CA, 1988.

[Shimony and Domshlak, 2003] Solomon E. Shimony and Carmel Domshlak. Complexity of probabilistic reasoning in directed-path singly connected Bayes networks. *Artificial Intelligence*, 151:213–225, 2003.

[Yuan and Druzdzel, 2006] Changhe Yuan and Marek J. Druzdzel. Importance sampling algorithms for Bayesian networks: Principles and performance. *Mathematical and Computer Modelling*, 43(910):1189 – 1207, 2006.