Approximating the makespan distribution of stochastic schedules

Thomas McSweeney*

15th October 2020

1 Introduction

For any optimization problem, we obviously need to be able to evaluate how good any given solution is with regards to the optimization criteria in order to find an optimal, or otherwise acceptable, solution. The scheduling problems we consider here are clearly no exception: we want to know how good a computed schedule is, whether there are other schedules which are better, and so on. Evaluating the makespan of a given schedule would appear to be a straightforward problem—and so it is, when the computation and communication costs are static. But if costs are *stochastic* then this may no longer be the case.

Now, as long as it specifies the execution order of tasks on processors, any schedule π for an application with task DAG G can be represented by another DAG G_π such that the longest path of G_π is equal to the makespan of π: G_π contains all the same vertices and edges as G, plus additional zero-weight *disjunctive* edges that indicate the execution order of the tasks on their chosen processors; the other weights of G_π are induced by the processor selections of π. There's some flexibility in how we add the disjunctive edges; the most straightforward way to do it is to simply add an edge between a task t_i and the task t_h which is executed immediately before t_i on its chosen processor if an edge does not already exist between the two.

For example, consider the schedule π from Figure X and the graph G in Figure Y. We can construct the associated graph G_π as shown in Figure Z. From now, we use the notations π_i and π_{ik} to represent the computation cost of task t_i and the communication cost between tasks t_i and t_k under the schedule π, respectively. Assuming, without loss of generality, that there is only one entry task t_1 and one exit task t_n, to compute the longest path through G_π—and therefore the

*School of Mathematics, University of Manchester, Manchester, M13 9PL, England (thomas.mcsweeney@postgrad.manchester.ac.uk).

makespan of π—we compute a sequence of numbers L_i defined by $L_1 = \pi_1$ and

$$L_i = \pi_i + \max_{h \in P_i} \{\pi_{hi} + L_h\} \tag{1}$$

for all other $i = 2, \ldots, n$. The longest path of G_π is then given by L_n. In the example above, we have... so we see that L_n is indeed equal to the schedule makespan. Computing the longest path using (1) is an $O(n + e) \approx O(n^2)$ operation, which depending on the size of the DAG may be expensive but is at least polynomial. (Of course, we could work backward through the DAG by setting $L_n = \pi_n$ and doing the maximization over the set of task children in (1) instead; the makespan would then be given by L_1 but the procedure is otherwise equivalent.)

Unfortunately, in practice, schedule costs are almost never known precisely before runtime. Typically, the best we can do is estimate the probability distribution that we believe they follow based on previous executions of the application and its constituent tasks, or similar data—i.e., we model the costs as random variables (RVs). But if costs are RVs rather than fixed scalars, it is obviously impossible to specify the precise time at which each task should begin execution so at this point we need to redefine what we mean by a schedule: here, we assume that a schedule π is a mapping from tasks to processors that specifies only which tasks each processor should execute and in what order; the processor then executes the next scheduled task as soon as it is able. Conceptually, we can view this as a processor being assigned an ordered queue of tasks before runtime and only being allowed to pop the task currently at the head of the queue.

This definition means that even though costs are stochastic, the disjunctive graph G_π has the same topology as it would in the static case. However, since all costs are RVs, the longest path is now also an RV and it is unclear how we should go about computing its distribution. If we attempt to apply (1) we soon run into difficulty because the L_h may be dependent and computing the maximum of a set of dependent RVs is intractable in the general case. Indeed, Hagstrom [9] proved that computing the longest path distribution, or even just its expected value, is a $\#P$-complete problem when all costs are discrete RVs, and there is no good reason to assume it is any easier for continuous RVs.

Given the difficulty of the problem, bounds and approximations of the longest path distribution—and therefore the schedule makespan distribution—are typically needed instead. In this chapter, we describe several existing heuristic methods for doing this, before focusing on a related problem: how do we quickly update a longest path estimate—i.e., an estimated schedule makespan—at runtime? This is a question that is particularly relevant when large, unexpected delays occur and decisions may need to be made as to whether to continue following the computed schedule at all. Although useful on its own merits, in the context of this research this chapter functions as a bridge between earlier chapters which focus on computing schedules when costs are assumed to be known exactly and

the later chapters in which the aim is to compute a schedule that is robust to the effects of uncertain cost estimates. We will see that many of the techniques discussed here underlie both existing stochastic scheduling heuristics and the new heuristic that we propose in the next chapter.

The problem of approximating the distribution of the longest path through a DAG with stochastic weights occurs in contexts other than scheduling, such as *program evaluation review technique* (PERT) network analysis [13] and digital circuit design [1]. Hence in this chapter we use the more general terminology—i.e., *longest path* rather than *makespan*—but the choice of methods that we focus on is often motivated by our wider aim of computing the initial schedules for a given stochastic task graph.

2 Bounds

Although computing the exact distribution of the longest path distribution or its expected value is usually infeasible, bounds may be computed much more cheaply. Depending on the context, these may be tight enough to be useful. Now, while bounds on the distribution itself have been proven—for example, Kleindorfer gives both upper and lower bounds [11], the first of which was improved on by Dodin [6]—these are typically based on graph reductions and fairly expensive, perhaps impractically so in the context of a scheduling heuristic, which is where our interest ultimately lies.

However, bounds on the expected value, or other moments, may be more practical. Indeed, we have already seen in the previous chapter that a lower bound u_n on the expected value of L_n (respectively u_1 and L_1 if working backward) can be computed in $O(n^2)$ operations by replacing all costs with their expected value and proceeding as in (1)—i.e., define $u_1 = \mathbb{E}[\pi_1]$ and

$$u_i = \mathbb{E}[\pi_i] + \max_{h \in P_i} \{\mathbb{E}[\pi_{hi}] + u_h\} \tag{2}$$

for all other $i = 2, \ldots, n$, then we have $u_i \leq \mathbb{E}[L_i]$ and in particular $u_n \leq \mathbb{E}[L_n]$. Furthermore, we also saw that a tighter bound can be found through Fulkerson's [8] alternative method, which was extended to continuous costs by Clingen [5] and later improved by Elmaghraby [7] and Robillard and Trahan [15].

All of those methods provide only lower bounds for the expected value. If all costs are normally distributed RVs, then Kamburowski [10] was able to prove both lower and upper bounds on the expected value, as well as a lower bound on the variance (and a conjectured upper bound). However, his method is conceptually very similar to the approach discussed in the following section, so will be presented in more detail there.

Rather than a formal bound, in many cases it may be more useful to simply approximate the longest path distribution (or its moments). To that end, many

heuristic methods have been proposed. We focus in this chapter largely on the family of heuristics described in the following section, although alternative methods are briefly discussed in Section 4.

3 Assuming normality

Fundamentally, the longest path is computed through a series of summations and maximizations of the cost RVs. By the Central Limit Theorem, sums of random variables are asymptotically normally distributed so if we assume that the effect of the maximizations is minor, then the longest path distribution is likely to be approximately normal, $L_n \approx N(\mu_n, \sigma_n)$. Indeed, this has often been observed empirically, even when all costs follow very different distributions [2]. If we assume further that all RVs can be characterized by their mean and variance (i.e., effectively that they are also normal), then sums can be computed though the well-known rule for summing two normal RVs $\epsilon \sim N(\mu_\epsilon, \sigma_\epsilon^2)$ and $\eta \sim N(\mu_\eta, \sigma_\eta^2)$,

$$\epsilon + \eta \sim N\big(\mu_\epsilon + \mu_\eta, \sigma_\epsilon^2 + \sigma_\eta^2 + 2\rho_{\epsilon\eta}\sigma_\epsilon\sigma_\eta\big), \tag{3}$$

where $\rho_{\epsilon\eta}$ is the linear correlation coefficient between the two distributions. Formulae for the first two moments of the maximization of two normal RVs—which is not itself normal—are less well-known but were first provided by Clark in the early 1960s [4]. Let

$$\phi(x) = \frac{1}{\sqrt{2\pi}} e^{-x^2/2} \quad \text{and} \quad \Phi(x) = \int_{-\infty}^{x} \phi(t)dt$$

be the unit normal probability density function and cumulative probability function, respectively, and define

$$\alpha = \sqrt{\sigma_\epsilon^2 + \sigma_\eta^2 - 2\rho_{\epsilon\eta}\sigma_\epsilon\sigma_\eta} \quad \text{and} \quad \beta = \frac{\mu_\epsilon - \mu_\eta}{\alpha}. \tag{4}$$

Then the first two moments $\mu_{\max}$ and $\sigma_{\max}$ of $\max(\epsilon, \eta)$ are given by

$$\mu_{\max} = \mu_\epsilon\Phi(\beta) + \mu_\eta\Phi(-\beta) + \alpha\phi(\beta), \tag{5}$$

$$\sigma_{\max}^2 = (\mu_\epsilon^2 + \sigma_\epsilon^2)\Phi(\beta) + (\mu_\eta^2 + \sigma_\eta^2)\Phi(-\beta) + (\mu_\epsilon + \mu_\eta)\alpha\phi(\beta) - \mu_{\max}^2. \tag{6}$$

Using (3) for summations, and (5) and (6) for maximizations, we can now move through the DAG and compute approximations μ_n and σ_n (or μ_1 and σ_1 if working backward) of the first two moments of the longest path distribution in a manner similar to (1).

This method appears to have first been proposed for estimating the completion time of PERT networks by Sculli [16], although there he assumed that all of the

correlation coefficients $\rho_{\epsilon\eta}$ in (4) were zero (see next section). While there are obviously no guarantees, the moment estimates obtained using Sculli's method tend to be fairly good [10], with performance improving as the cost distributions move closer to normality and the number of nodes in the graph increases, as we might expect. Furthermore, Sculli's method is typically much faster than the alternatives [3].

3.1 Including correlations

Sculli assumed that all correlations were zero, which is rarely the case for real graphs since common ancestors make the longest path at two nodes dependent, even if all costs themselves are independent. Computing the correlation coefficients efficiently is tricky. However, Canon and Jeannot [3] proposed two different heuristics which alternatively prioritize precision and speed. The first is a dynamic programming algorithm called Cordyn which recursively computes the correlations using formulae derived in Clark's original paper for the correlation coefficients between any normal RV τ and a summation or maximization of normal RVs ϵ and η,

$$\rho_{\tau,\,\mathrm{sum}(\epsilon,\eta)} = \frac{\sigma_\epsilon \rho_{\tau\epsilon} + \sigma_\eta \rho_{\tau\eta}}{\sigma_{\mathrm{sum}}} \quad \text{and} \quad \rho_{\tau,\mathrm{max}(\epsilon,\eta)} = \frac{\sigma_\epsilon \rho_{\tau\epsilon} \Phi(\beta) + \sigma_\eta \rho_{\tau\eta} \Phi(-\beta)}{\sigma_{\mathrm{max}}}.$$

Cordyn has time complexity $O(ne) \approx O(n^3)$, so is more expensive than Sculli's method, which is quadratic in n, however numerical experiments by Canon and Jeannot suggest that it is almost always more accurate.

In an effort to marry the speed of Sculli's method and the accuracy of Cordyn, Canon and Jeannot also proposed an alternative heuristic called CorLCA. The main idea is to construct a simplified version of the DAG called a *correlation tree* that has all the same nodes as the original but only retains a subset of the edges. In particular, where multiple edges are incident to a node—i.e., a maximization must be performed—only the edge which contributes most to the maximization is retained in the correlation tree. The motivation here is that the correlation coefficient between any two longest path estimates L_i and L_k can be efficiently approximated by finding the *lowest common ancestor* (LCA) t_a of the corresponding nodes t_i and t_k in the correlation tree: since $L_i \approx L_a + \eta$ and $L_k \approx L_a + \epsilon$ where η and ϵ are independent RVs representing the sums of the costs along the paths between t_a and t_i (resp. t_a and t_k) in the correlation tree, we have

$$\rho_{L_i,L_k} \approx \frac{\sigma^2_{L_a}}{\sigma_{L_i}\sigma_{L_k}}.$$

For every edge, we need to do a lowest common ancestor query, so the time complexity of CorLCA depends to a large extent on the cost of these queries.

Although they do not present a method, based on similar results in the literature, Canon and Jeannot hypothesize this can be done in $O(1)$ operations, giving an overall time complexity $O(e) \approx O(n^2)$ for the entire algorithm. At any rate, an extensive numerical comparison of several heuristic methods for approximating the longest path distribution by the original authors suggested that CorLCA is more efficient than Cordyn with only a relatively small reduction in accuracy [3]. It should however also be noted that it can do badly when longest path length estimates at two or more nodes with a common child are similar since only one of the respective edges to the child will be retained in the correlation tree.

Another method for estimating the longest path distribution that approximates correlations in a similar manner comes from the field of digital circuit design. In the so-called *canonical model* [18, 19], all RVs are expressed as the sum of their expected value and a weighted sum of standard normal distributions that characterize the variance,

$$\eta = \mu + \sum_i v_i \delta_i,$$

where all $\delta_i \sim N(0,1)$ and are independent of one another. The advantage of this is that evaluating summations and maximizations becomes much more straightforward. Let $\eta = \mu_\eta + \sum_i v_{\eta,i}\delta_i$ and $\epsilon = \mu_\epsilon + \sum_i v_{\epsilon,i}\delta_i$. Then

$$\omega = \eta + \epsilon = (\mu_\eta + \mu_\epsilon) + \sum_i (v_{\eta,i} + v_{\epsilon,i})\delta_i.$$

Suppose now that $\omega = \max(\eta, \epsilon)$. Let α and β be defined as in (4), and $\Phi(x) = \int_{-\infty}^{x} \phi(t)dt$ be the standard normal cdf. Note that computing β requires the linear correlation coefficient $\rho_{\eta\epsilon}$ which can be efficiently calculated as:

$$\rho_{\eta\epsilon} = \frac{\sum_i v_{\eta,i} v_{\epsilon,i}}{\sqrt{\sum_i v_{\eta,i}^2} \cdot \sqrt{\sum_i v_{\epsilon,i}^2}}.$$

By definition, $\mathbb{P}[\eta > \epsilon] = \Phi(\beta)$ and we can therefore approximate ω by

$$\begin{aligned}\hat{\omega} &= \Phi(\beta)\eta + \Phi(-\beta) \\ &= \Phi(\beta)\mu_\epsilon + \Phi(-\beta)\mu_\epsilon + \sum_i \big(\Phi(\beta)v_{\eta,i} + \Phi(-\beta)v_{\epsilon,i}\big)\delta_i.\end{aligned}$$

This is both similar and in some sense contrary to the Clark equation approach, in that the latter precisely computes the first two moments of the maximization of two normal RVs, whereas the canonical method approximates the distribution of the maximization of any two RVs using linear combinations of normal RVs. In their empirical comparison, Canon and Jeannot found that the canonical method tended to fall between Sculli's method and CorLCA in terms of both speed and approximation quality [3].

3.2 Kamburowski's bounds

When all costs are independent Gaussian RVs, Kamburowski was able to prove both upper and lower bounds on the first two moments[1] of the longest path distribution. Since normally distributed costs occur in many applications and the method is both cheap and somewhat similar to the approaches discussed previously in this section, we describe it in depth here.

The bounds are achieved by recursively computing four number sequences $\underline{m_i}$, $\overline{m_i}$, $\underline{s_i}$ and $\overline{s_i}$ such that $\underline{m_i} \leq \mu_{L_i} \leq \overline{m_i}$ and $\underline{s_i} \leq \sigma_{L_i} \leq \overline{s_i}$ for all $i = 1, \ldots, n$. Clearly, by taking $\underline{m_1} = \overline{m_1} = \mathbb{E}[\pi_1]$ and $\underline{s_1}^2 = \overline{s_1}^2 = Var[\pi_1]$ we can achieve the desired bounds for L_1. (As ever, we can work backward instead in which case the analogous results hold for the index n.) Now we suppose that all of the upper and lower bounds have been computed for all of the parents of a given node t_i and consider how we can construct $\underline{m_i}$, $\overline{m_i}$, $\underline{s_i}$ and $\overline{s_i}$. The variance bounds are relatively straightforward. The lower bound is given by

$$\underline{s_i^2} = \begin{cases} \underline{s_h^2} + Var[\pi_{hi}] + Var[\pi_i], & \text{if } P_i = \{s_h\}, \\ 0, & \text{otherwise,} \end{cases} \tag{7}$$

reflecting the fact that the variance, as computed by equation (6), can be reduced to an arbitrary extent by a maximization (which needs to be performed for multiple parents). The (conjectured) upper bound is similarly a result of this fact,

$$\overline{s_i^2} = \max_{h \in P_i} \{\overline{s_h^2} + Var[\pi_{hi}] + Var[\pi_i]\}. \tag{8}$$

The bounds on the expected value are somewhat more complex. First, define a function h by

$$h(\mu_i, \sigma_i, \mu_k, \sigma_k) = \mu_i \Phi(\overline{\beta}) + \mu_k \Phi(-\overline{\beta}) + \overline{\alpha}\phi(\overline{\beta}),$$

where $\overline{\alpha} = \sqrt{\sigma_i^2 + \sigma_k^2}$ and $\beta = (\mu_i - \mu_k)/\overline{\alpha}$. Per equation (5), h is the expected value of the maximization of two *independent* normally distributed RVs $X_i \sim N(\mu_i, \sigma_i^2)$ and $X_k \sim N(\mu_k, \sigma_k^2)$. Now, suppose that we have a set of (not necessarily independent) normally distributed RVs $X_1, X_2, \ldots, X_r$, where each $X_i \sim N(\mu_i, \sigma_i^2)$ and $\sigma_1 \leq \sigma_2 \leq \cdots \leq \sigma_r$. Define two functions $\underline{f}$ and $\overline{f}$ by the recursions,

$$\begin{aligned} &\underline{f}(X_1) = \overline{f}(X_1) = \mu_1, \\ &\underline{f}(X_1, X_2) = \overline{f}(X_1, X_2) = h(\mu_1, \sigma_1, \mu_2, \sigma_2), \\ &\underline{f}(X_1, \ldots, X_r) = h(\underline{f}(X_1, \ldots, X_{r-1}), 0, \mu_r, \sigma_r), \\ &\overline{f}(X_1, \ldots, X_r) = h(\overline{f}(X_1, \ldots, X_{r-1}), \sigma_{r-1}, \mu_r, \sigma_r). \end{aligned}$$

[1] As noted in Section 2, the upper bound on the variance technically remains a conjecture.

Then, for all $i = 2, \ldots, n$, if we define

$$\underline{m_i} = \underline{f}(\{\underline{X_h}\}_{h \in S_i}),$$

where

$$\underline{X_h} \sim N(\underline{m_h} + \mathbb{E}[\pi_{hi}] + \mathbb{E}[\pi_i],\ \underline{s_h^2} + Var[\pi_{hi}] + Var[\pi_i]),$$

and

$$\overline{m_i} = \overline{f}(\{\overline{X_h}\}_{h \in S_i}),$$

where

$$\overline{X_h} \sim N(\overline{m_h} + \mathbb{E}[\pi_{hi}] + \mathbb{E}[\pi_i],\ \overline{s_h^2} + Var[\pi_{hi}] + Var[\pi_i]),$$

we have

$$\underline{m_i} \leq \mu_{L_i} \leq \overline{m_i}.$$

(Here we are assuming that the sets $\{\underline{X_h}\}_{h \in S_i}$ and $\{\overline{X_h}\}_{h \in S_i}$ are ordered in such a way that the inequality constraints on the variances is satisfied.)

4 Other approximation methods

Monte Carlo methods have a long history in approximating the longest path distribution of PERT networks, dating back to at least the early 1960s [17]. The idea is to simulate the realization of all RVs and evaluate the longest path of the resulting deterministic graph. This is done repeatedly, giving a set of longest path instances whose empirical distribution function is guaranteed to converge to the true distribution by the Glivenko-Cantelli theorem [3]. Furthermore, analytical results allow us to quantify the approximation error for any given the number of realizations—and therefore the number of realizations needed to reach a desired accuracy. The major disadvantage is the cost, particularly when the number of realizations required is large, although modern architectures are well-suited to MC methods because of their parallelism so this problem may no longer be as acute as it once was. At any rate, in this chapter, we typically only use MC methods in order to obtain reference solutions.

If the graph is *series-parallel* then we can compute the exact distribution of the longest path length in polynomial-time through a series of reductions [6, 14]. If this isn't the case, similar methods have been proposed that give approximations to the distribution [6, 12], however these tend to be less accurate than Monte Carlo-based methods and more expensive than the those based on the normality assumption [3].

5 Updating makespan estimates

Estimating the length of a schedule with stochastic costs before its execution has been widely-studied, as the examples in previous sections illustrate. We therefore focus here on a different but related problem: how can schedule makespan estimates be quickly updated *during* execution? This would be particularly useful in situations such as when a large unexpected delay occurs. More precisely, the scenario we consider is as follows. Given a schedule π, we computed a complete set of task finish times estimates $L_i \sim N(\mu_i, \sigma_i^2)$ for all $i = 1, \ldots, n$ before runtime using one of the Clark equation-based methods discussed in Section 3. At some point in time T during the schedule execution we want to compute a new, more accurate estimate of the makespan based on the costs which have been observed thus far. How do we do this as quickly and accurately as possible?

Effectively, at time T the schedule graph G_s can be divided into two distinct subgraphs: one corresponding to those tasks that have been completed and the other to those that have not. Define C_T to be the set of indices corresponding to tasks that have completed by time T and U_T to be the set of indices corresponding to tasks that are still not done. Furthermore, define $C_T^* \subseteq C_T$ to be the set of indices of tasks which have been completed but at least one of their parents has not—i.e., in some sense the boundary of the realized and unrealized sections of the graph.

Since all cost RVs corresponding to tasks indexed by C_T—and the edges between them—have been realized, we can compute the realized finish times ℓ_i for the corresponding tasks in the usual manner for scalar costs. The question then remains of how we compute an updated set of longest path estimates L'_k for all $k \in U_T$, including a new final makespan estimate L'_n. We outline different possible approaches to this problem in the following three sections.

5.1 Estimating the remaining time

The most straightforward approach is to simply consider the unrealized portion of the graph separately and compute the longest path through it, starting from any of the boundary tasks indexed by C_T^*, using any heuristic method; a new longest path estimate which passes through the given boundary task is then the sum of the realized longest path up to and including the task, plus the estimated longest path through the remaining unrealized subgraph. More efficiently, rather than initially computing values L_i that represent longest path lengths from the source to a task (inclusive), we can compute a sequences of values R_i that represent the longest paths from the tasks to the sink (exclusive)—i.e., estimates of the remaining time until the entire schedule has been completed. In particular, we set $R_n = 0$, then work backward and recursively compute

$$R_i = \max_{k \in S_i} \{\pi_{ik} + \pi_k + R_k\}$$

for all other tasks, where the moments of the maximization are computed using the Clark formulae in a manner corresponding to whichever of Sculli's method, CorLCA or CorDyn we decide to use. Note that since the R_i do not include the cost of task t_i the full schedule makespan estimate is $R_1 + \pi_1$.

(Minor changes need to be made to CorLCA and CorDyn in order to estimate the correlations when working backward through the DAG. For example, in CorLCA, rather than the lowest common ancestors of two tasks, we want the deepest common descendants. However these changes are fairly straightforward. As an aside though, fairly significant differences are sometimes apparent between the makespan moment estimates computed forward and backward through the DAG for all three heuristics, particularly with regard to the variance. For example, we found that for Cholesky schedule DAGs, the variance was always smaller when the makespan is computed backward. We attribute this to the fact that the average number of task children exceeds the average number of task parents so the corresponding maximizations contain more terms; since these are computed pairwise and the effect of (6) is to decrease the variance, this may not be entirely surprising.)

The advantage of this approach is that at time T the estimated longest path which passes through task t_i, where $i \in C_T^*$, is simply given by $L_i' = \ell_i + R_i$ and, in particular, a new estimate of the makespan can be computed through

$$L_n' = \max_{i \in C_T^*} \{\ell_i + R_i\}.$$

This maximization can be done in the same manner as the calculation of the R_i, using the same set of correlation coefficients (or not, if using Sculli's method).

5.2 Updating the makespan directly

Another possible method for updating the final makespan estimate makes use of the following result.

Proposition 1 *Let u be a normally distributed vector with mean $\bar{\mu}$ and covariance Σ_u, $u \sim N(\bar{\mu}, \Sigma_u)$, and suppose that $u = (v, w)$. Then*

$$(w \mid v) \sim N\big(\bar{w} + \Sigma_{wv}\Sigma_{vv}^{-1}(v - \bar{v}),\ \ \Sigma_{ww} - \Sigma_{wv}\Sigma_{vv}^{-1}\Sigma_{vw}\big).$$

Note that if all of the linear correlation coefficients ρ_{uv} are known, we can use the definition $\rho_{uv} = \Sigma_{uv}/\sigma_u\sigma_v$ to find Σ_{uv}.

This proposition is useful because we have been implicitly assuming that there is a joint normal distribution across the entire set of finish times L_i due to their dependencies. Furthermore, we can estimate the correlation coefficients between finish times at any two tasks using, for example, the correlation tree approach of CorLCA. This means that we can update the final makespan estimate using only the observed finish times of the tasks indexed by C_T^*. For compactness of notation

let $\rho_{in} = \rho_{ni}$ be the linear correlation coefficient between L_i and L_n. Suppose that task t_i has finished at time ℓ_i. Then we can compute a new estimate L_n^i of the final makespan by applying the above result as follows,

$$\begin{aligned} L_n^i &\sim N\left(\mu_n + \frac{\rho_{ni}\sigma_n\sigma_i}{\sigma_i^2}(\ell_i - \mu_i),\ \ \sigma_n^2 - \frac{\rho_{ni}\sigma_n\sigma_i \cdot \rho_{in}\sigma_i\sigma_n}{\sigma_i^2}\right) \\ &\sim N\left(\mu_n + \frac{\rho_{ni}\sigma_n}{\sigma_i}(\ell_i - \mu_i),\ \ (1-\rho_{ni}^2)\sigma_n^2\right). \end{aligned}$$

We do this for all $i \in C_T^*$ and then compute their maximum (using the same method as before to estimate the correlation coefficients, if they are not already known) in order to obtain a new estimated final makespan L'_n,

$$L'_n = \max_{i \in C_T^*}\{L_n^i\}.$$

5.3 Propagating updates forward

Rather than updating the finish time of the exit task—and therefore the makespan—directly, we can instead move forward through the DAG and update the finish time estimates for all unrealized tasks. This is slightly more expensive of course but potentially more accurate. Now, in principle the updated finish times can be computed for each task t_i, where $i \in U_T$, using equation (1) in the usual manner. However, there are essentially three different cases that we need to consider, depending on the status of the terms in the maximization.

1. Both π_{hi} and L_h have been realized for all $h \in P_i$.
2. Both π_{hi} and L_h have been realized for some $h \in P_i$ but not others.
3. Both π_{hi} and L_h have been realized for none of the parents.

In the first instance, the sums of π_{hi} and L_h are now deterministic so the maximization is just done over a set of scalars and is therefore straightforward. Similarly, in the third case, all of the sums are unrealized RVs so we can just use Clark's equations again to compute the new finish time estimate.

The second case is more interesting. The problem basically reduces to how we compute the maximum of a set of RVs, some of which have been realized and others which have not. For notational ease, let $Z_h = \pi_{hi} + L_h$ and $M_i = \max_{h \in P_i}\{Z_h\}$. Let X_i be the maximum over the subset of parents for which Z_h has been realized. Now, we know that $M_i > X_i$ since at least some of the Z_h have not been realized yet. Furthermore, we have already computed an estimate of $L_i = \pi_i + M_i$, where M_i is assumed to be roughly normal, so we can also explicitly form $M_i \sim N(\mu_{M_i}, \sigma_{M_i}^2)$. We want to find a new finish time estimate

$$L'_i = \pi_i + (M_i | M_i > X_i). \tag{9}$$

Given that M_i is assumed to be Gaussian, we can model the term on the right as a truncated Gaussian. Let $a = (X_i - \mu_{M_i})/\sigma_{M_i}$ and $b = 1 - \Phi(a)$, where Φ is the unit normal cdf as before. Then the first two moments of $(M_i | M_i > X_i)$ are given by

$$\mathbb{E}[M_i \mid M_i > X_i] = \mu_{M_i} + \frac{\sigma_{M_i}\phi(a)}{b}$$

and

$$Var[M_i \mid M_i > X_i] = \sigma^2_{M_i}\left[1 + \frac{a\phi(a)}{b} - \left(\frac{\phi(a)}{b}\right)^2\right].$$

Using these expressions, we can now compute (9), and therefore work through the remainder of the DAG updating the finish time estimates for all tasks that have not yet completed.

6 Numerical experiments

In order to study the problem of estimating the longest path of a stochastic graph—i.e., the makespan distribution of stochastic schedules—we created a simple software package which implements several of the heuristic methods discussed so far in this chapter. As in previous chapters, the source code is written in `Python` and is available in its entirety on Github[2]. Much more sophisticated software along these lines already exists, such as the *Emapse* package from Canon and Jeannot (citation). However, we decided to create our own, both as a learning exercise and for ease of integration with other work in this thesis. In this section, we describe the results of small-scale numerical experiments performed in this framework concerning various aspects of the longest path estimation problem.

6.1 Before runtime

Exhaustive comparisons of heuristic methods for estimating the longest path distribution before any RVs have been realized have been done before in the literature, with the best example again probably being the work of Canon and Jeannot [3]. Rather than repeating those investigations, we take a narrower view and focus on a single family of graphs based on schedules for a widely-used application in scientific computing, namely Cholesky factorization, on an accelerated target platform.

More specifically, the graphs are constructed from schedules computed by the static HEFT heuristic for Cholesky task graphs with between 35 and 11480 tasks (corresponding to matrix tilings from 5×5 to 40×40, with the dimensions increasing in increments of 5). The target platform is the *Single GPU* platform from

[2] https://github.com/mcsweeney90/stochastic-longest-path

Chapter X, comprising 7 CPU resources and one GPU. The schedule is initially computed as in Chapter X, which then determines the topology of the schedule graph—i.e., where we add the disjunctive edges to the original task graph. The node and edge weights of the schedule graph are then modeled as random variables with means and variances given by the sample means and variances we observed for the relevant tasks in the experimental testing we described in Chapter X, with tile size 128 and no asynchronous data transfers being assumed.

Taking the main conclusions of the more wide-ranging study by Canon and Jeannot as given, at least in the general case, the overall aim of our small-scale investigation is basically to determine whether the improved moment estimates produced by the correlation-aware heuristic CorLCA rather than Sculli's method are worth the extra computational cost for the realistic example schedule graphs considered here. In addition, we also briefly consider how useful some of the cheaper bounds are in this case, with a view to their possible use for making scheduling decisions.

6.1.1 The normality assumption

Other studies have thoroughly investigated how close to normal the longest path distribution actually tends to be for randomly-generated schedule DAGs, so we only treat this topic very briefly here. Our conclusions for the Cholesky schedule graphs are that the longest path distribution does very much tend to normality as the number of tasks increases and the costs themselves are sampled from distributions closer to normal.

In order to estimate the true longest path distribution in this section—and the following two—we use the Monte Carlo method with 10^5 samples. Given the number of samples, these estimates are likely to be very good for the smallest graphs but perhaps slightly less accurate for the largest ones (although we expect them to still be at least fairly good).

First we assume that all individual costs are normally distributed. Now, there are several standard statistical tests for normality, such as Shapiro-Wilks, Anderson-Darling or Kolmogorov-Smirnoff, etc, but it is perhaps more instructive here to consider the longest path distribution visually. Figure X shows how the shape of empirical distribution changes as the number of tasks in the DAG increases... We see that the distribution does indeed more closely resemble the normal bell curve as the size of the DAG increases...

6.1.2 Kamburowski's bounds

Although there are many bounds on the longest path distribution in the literature, given our overriding interest in practical scheduling methods and our experience in the previous chapter with Fulkerson's bounds on the expected value—which were impractically expensive even for relatively small graphs—we decided

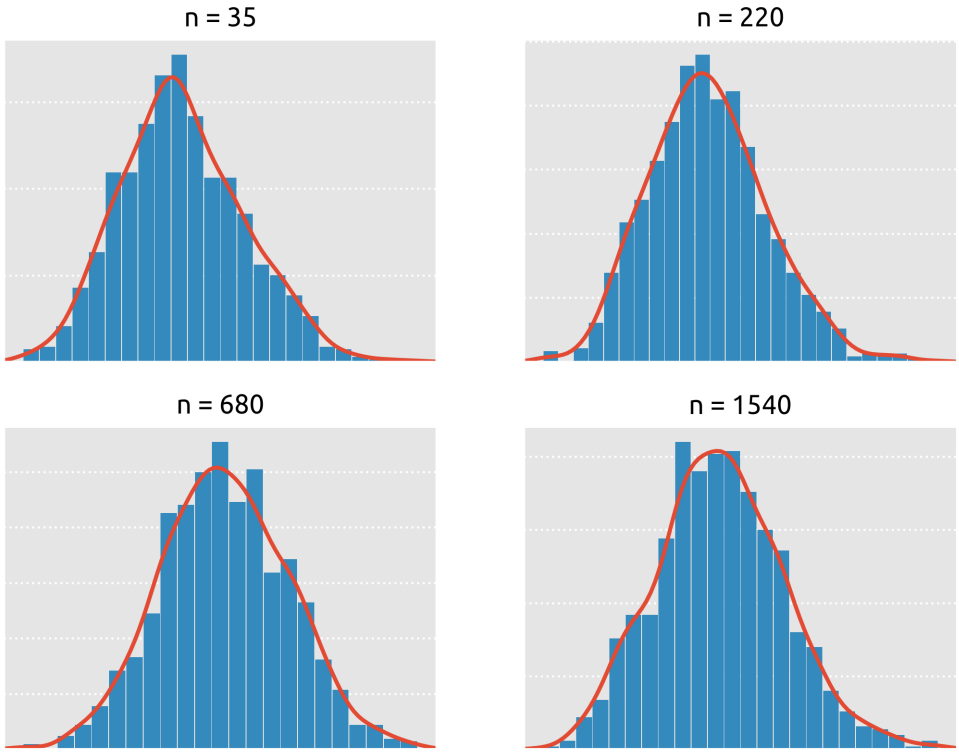

Figure 1: Placeholder–TODO.

Table 1: Tightness of bounds on the expected value for Cholesky DAGs, as a percentage of the reference expected value.

	DAG size							
Bound	35	220	680	1540	2925	4960	7770	11480
PERT-CPM	99.5	98.9	98.5	98.7	99.2	99.2	99.4	99.5
K. (lower)	99.9	99.2	98.8	99.0	99.4	99.4	99.5	99.6
K. (upper)	100.7	104.0	110.2	118.5	116.6	114.1	112.4	111.5

to eschew methods which require the evaluation of complex integrals. Similarly, bounds based on graphs reductions may be prohibitively expensive. Hence we decided to focus here on bounds that can be computed through simple numerical schemes, such as the classic PERT-CPM bound on the mean and Kamburowski's bounds on the first two moments (when costs are normally distributed). The latter does require the evaluation of the unit normal cdf, but this is a standard part of most good numerical software libraries, so can be computed efficiently.

Table 1 shows how tight the PERT-CPM and Kamburowski bounds on the expected value are for the Cholesky graph set, where the reference solution is computed using the Monte Carlo method with 10^5 samples. We see that the lower bounds are very tight, with Kamburowski only improving on PERT-CPM very slightly. This may be down to the topology and cost structure of the Cholesky graphs, which is relatively simple, although it should be noted that in general it is not the case that complex applications necessarily have complex task graphs. The upper bounds from Kamburowski are much looser, although even in the worst case no more than 19% greater than the true value.

Unfortunately, the variance bounds given by Kamburowski's method are considerably looser in both directions, potentially to the extent of being impractical when making scheduling decisions. Figure 2 illustrates how wide the bounds are for the Cholesky graphs...

6.1.3 Do correlations need to be considered?

The question we consider here is, although CorLCA generally obtains superior estimates of the makespan distribution, are these gains significant enough to be worthwhile in a scheduling heuristic?

6.2 Updating the finish time

Consider different permutations (fraction of tasks initially realized, distributions of the costs, etc) in a systematic manner and compare with MC estimates...

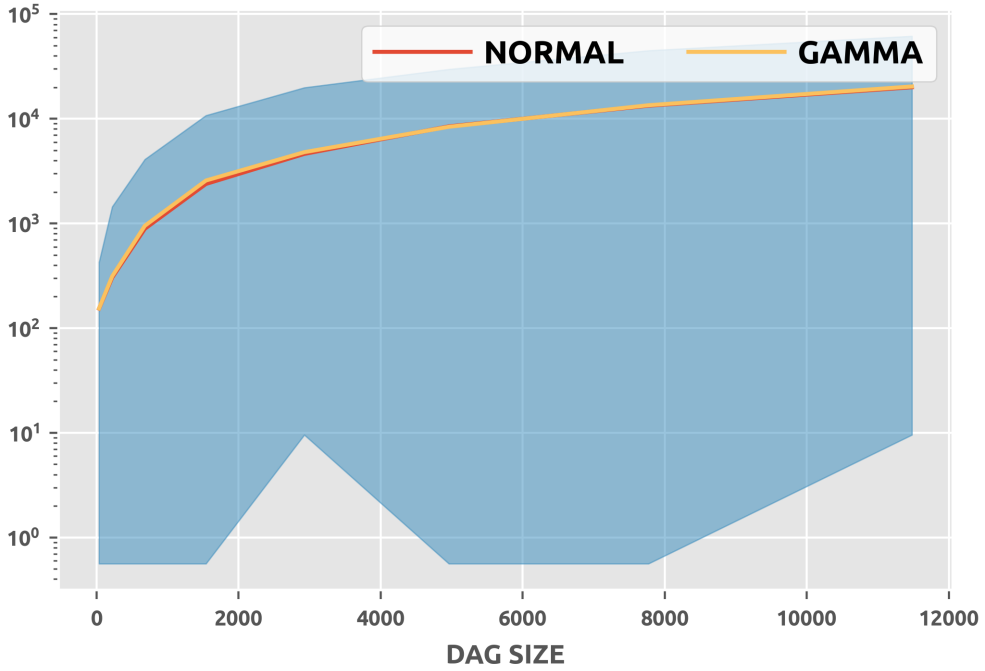

Figure 2: Kamburowski's bounds on the variance for the Cholesky graph set. Reference solutions for normal and Gamma distributed costs given by solid lines.

7 Conclusions

Conclusions go here.

References

[1] D. Blaauw, K. Chopra, A. Srivastava, and L. Scheffer. Statistical timing analysis: From basic principles to state of the art. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 27(4):589–607, 2008.

[2] L. Canon and E. Jeannot. Evaluation and optimization of the robustness of DAG schedules in heterogeneous environments. *IEEE Transactions on Parallel and Distributed Systems*, 21(4):532–546, 2010.

[3] Louis-Claude Canon and Emmanuel Jeannot. Correlation-aware heuristics for evaluating the distribution of the longest path length of a DAG with random weights. *IEEE Transactions on Parallel and Distributed Systems*, 27(11):3158–3171, 2016.

[4] Charles E. Clark. The greatest of a finite set of random variables. *Operations Research*, 9(2):145–162, 1961.

[5] C. T. Clingen. A modification of Fulkerson's PERT algorithm. *Operations Research*, 12(4):629–632, 1964.

[6] Bajis Dodin. Bounding the project completion time distribution in PERT networks. *Operations Research*, 33(4):862–881, 1985.

[7] Salah E. Elmaghraby. On the expected duration of PERT type networks. *Management Science*, 13(5):299–306, 1967.

[8] D. R. Fulkerson. Expected critical path lengths in PERT networks. *Operations Research*, 10(6):808–817, 1962.

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

[10] Jerzy Kamburowski. Normally distributed activity durations in PERT networks. *Journal of the Operational Research Society*, 36(11):1051–1057, 1985.

[11] George B. Kleindorfer. Bounding distributions for a stochastic acyclic network. *Operations Research*, 19(7):1586–1601, 1971.

[12] Arfst Ludwig, Rolf H. Möhring, and Frederik Stork. A computational study on bounding the makespan distribution in stochastic project networks. *Annals of Operations Research*, 102(1-4):49–64, 2001.

[13] D. G. Malcolm, J. H. Roseboom, C. E. Clark, and W. Fazar. Application of a technique for research and development program evaluation. *Operations Research*, 7(5):646–669, 1959.

[14] J. J. Martin. Distribution of the time through a directed, acyclic network. *Operations Research*, 13(1):46–66, 1965.

[15] Pierre Robillard and Michel Trahan. Technical note—expected completion time in PERT networks. *Operations Research*, 24(1):177–182, 1976.

[16] D. Sculli. The completion time of PERT networks. *Journal of the Operational Research Society*, 34(2):155–158, 1983.

[17] Richard M. Van Slyke. Letter to the editor—Monte Carlo methods and the PERT problem. *Operations Research*, 11(5):839–860, 1963.

[18] C. Visweswariah, K. Ravindran, K. Kalafala, S. G. Walker, S. Narayan, D. K. Beece, J. Piaget, N. Venkateswaran, and J. G. Hemmett. First-order incremental block-based statistical timing analysis. *IEEE Transactions on*

Computer-Aided Design of Integrated Circuits and Systems, 25(10):2170–2180, 2006.

[19] L. Zhang, W. Chen, Y. Hu, and C. C. Chen. Statistical static timing analysis with conditional linear MAX/MIN approximation and extended canonical timing model. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 25(6):1183–1191, 2006.