Approximating the makespan distribution of stochastic schedules

Thomas McSweeney*

28th September 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. This can be done cheaply, although it should be noted that the related problem of adding the minimal number of necessary disjunctive edges is NP-hard [citation].

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.

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

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 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 .

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

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

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

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 Gaussian, 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 $\underline{s_1} = \overline{s_1} = Var[\pi_1]$ 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 straightforward. Since the maximization can only decrease the variance...

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].

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

5 Updating longest path estimates

Rather than considering how we can compute the longest path before runtime, we focus here instead on the related problem of how

Suppose we are following a static schedule for which we have already computed an estimate of the makespan, how do we efficiently update the estimate dynamically at runtime, perhaps in response to a large delay? This is straightforward for deterministic static schedules since we can just recompute the critical path length of the schedule DAG using observed costs for those task that have already been processed and estimates for those that have not yet been processed. The same basic approach can also be used for stochastic schedules of course, but the picture is slightly more complex, as illustrated below.

For a given task DAG and target platform, suppose we have a static schedule that dictates what tasks each processor should execute and in what order, which they will do greedily (i.e., without artificial delays). Before runtime we work through the DAG and estimate the makespan distribution up to when each of the tasks has completed. These makespan estimates are modeled as normal RVs in accordance with the central limit theorem so that for each task t_i we have an associated makespan estimate $F_i \sim N(\mu_i, \sigma_i^2)$, which is computed through

$$F_i = W_i + \max_{h \in P_i} \{F_h + W_{hi}\}, \tag{7}$$

where W_i and W_{hi} here represent the relevant computation and communication costs (since the processors are fixed by the schedule). We make the standard assumption that all of the computation and communication costs are independent so the summations are done using the rule for normal RVs (3) with the correlation coefficient assumed to be zero. We use Clark's equations (5) and (6) to estimate the distribution of the maximization. The terms within are not generally independent because the parent tasks they represent may have common ancestors, so we can either ignore all correlations (which corresponds to Sculli's method) or estimate the correlation coefficients (as in CorLCA and Cordyn).

We are concerned with the situation at any given point during runtime when some tasks may have completed execution while others are still to be done. Let f_h be the realization of the makespan estimate F_h once task t_h has actually been completed. In order to update the makespan estimate we need to move through the DAG and make use of the realizations of previous task makespans. Consider a generic task t_i and suppose that some of its parent tasks have been processed but others have not. The most straightforward way to update F_i is to simply use f_h instead of F_h in equation (7) for all those parents that have been realized. However, it would perhaps be better if we could use the realized parent makespans to update the makespan estimates F_h for those parents that haven't yet finished. Let $f_m = \max_{h \in P_i} f_h$ be the greatest makespan of those parent tasks that have been processed. For all other parent tasks t_h that have *not* yet been executed we

want to estimate the remaining cost, given that it has not been completed but t_m has been—i.e., we want to estimate $F'_h = F_h - f_m$ given that we know $f_h > f_m$. This can be done by applying 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 (which would be the case if we used CorLCA or Cordyn), we can use the definition $\rho_{uv} = \Sigma_{uv}/\sigma_u\sigma_v$ to find Σ_{uv}. For compactness of notation let $\rho_{hm} = \rho_{mh}$ be the linear correlation coefficient between F_h and F_m. Then we have

$$\begin{aligned} F'_h &\sim N\left(\mu_h + \frac{\rho_{hm}\sigma_h\sigma_m}{\sigma_m^2}(f_m - \mu_m),\ \ \sigma_h^2 - \frac{\rho_{hm}\sigma_h\sigma_m \cdot \rho_{mh}\sigma_m\sigma_h}{\sigma_m^2}\right) \\ &\sim N\left(\mu_h + \frac{\rho_{hm}\sigma_h}{\sigma_m}(f_m - \mu_m),\ \ (1 - \rho_{hm}^2)\sigma_h^2\right). \end{aligned}$$

The idea is that by using F'_h rather than F_h for those parent tasks that have not yet completed we can make greater use of the data available and should therefore obtain a superior estimate of the new makespan.

6 Results

We created a software framework to study the problem of approximating the makespan distribution of stochastic DAGs...

6.1 Benchmarking

We don't spend too much time on this since it was done much more thoroughly by Canon and Jeannot but we consider Sculli, CorLCA, possibly canonical and Cordyn for the Cholesky DAGs sets (i.e., we focus on a single example)...

6.2 Update rule

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

7 Conclusions

What we really want to know: is there enough justification for considering the correlations in the context of a stochastic scheduling heuristic?

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.