Probabilistic Timing Analysis of Asynchronous Systems with Moments of Delays

Supratik Chakraborty
Dept. of Computer Science and Engineering
Indian Institute of Technology, Bombay, India
supratik@cse.iitb.ac.in

Rohan Angrish *
Computer Science Dept.
Stanford University, Stanford, CA 94305, USA
angrish@cs.stanford.edu

Abstract

Finding time separation of events is a fundamental problem in the analysis of asynchronous systems. When component delays have statistical variations, it is both interesting and useful to compute moments of time separation of events. Traditionally, Monte Carlo simulation has been used for this purpose. However, Monte Carlo simulation requires knowledge of the probability distributions of component delays, which is often difficult to ascertain. Much more easily available are parameters like the statistical mean and variance of component delays. Unfortunately, with only these parameters, Monte Carlo simulation cannot be reliably applied. Yet another disadvantage of Monte Carlo simulation is the large number of runs needed before the error term becomes small enough to be acceptable. This paper describes a polynomial-time algorithm for computing bounds on the first two moments of times of occurrence of events in an acyclic timing constraint graph, given only means and variances of component delays. We present experimental results demonstrating the effectiveness of our algorithm.

1 Introduction

The behavior of asynchronous systems is commonly described in terms of events and their interactions. A central problem in the analysis of such systems is computing time separation between events. Depending on the application, different flavors of the problem may be of interest. For example, in interface timing verification, component delays are assumed to vary between given bounds, and bounds on time separation of events are calculated. In performance analysis and probabilistic timing verification, however, component delays are considered random variables and distributions or moments of time separations are of interest.

There is a rich body of work on analyzing time separation of events in systems with bounded component delays [6, 2, 10, 18, 13, 4]. In [1], Hulgaard and Amon described a method for analyzing time separation of events with symbolic delays. Williams analyzed the latency and throughput of pipelines in [14]. Algorithms for estimating the average performance of asynchronous circuits from average component delays were given by Burns [3] and Lee [9]. In [7], Greenstreet and Steiglitz gave bounds on the utilization of pipelines under various assumptions for delay distribution.

For systems in which component delays and state transitions are specified using probability distributions, Monte Carlo simulation and Markovian analysis are commonly used to study steady-state behavior. Probabilistic state transition diagrams lend themselves naturally to Markovian analysis. In Monte Carlo methods, all random variables in the system description are sampled according to their respective probability distributions. For each set of sample points thus obtained, the behavior of the system is simulated. By repeating the process a large number of times, statistical parameters of the system can be estimated. This is a powerful technique that is particularly useful when the parameters of interest are hard to determine analytically.

In [15], Xie and Beerel used symbolic techniques for performance analysis of asynchronous systems, using average time separation of events as the metric. In [17, 16], the same authors presented a methodology for bounding average time separation of events in stochastic timed Petri Nets – both with and without choice. For *marked graphs*, i.e., Petri Nets with a single predecessor transition and a single successor transition for each place, an interesting result proved in Xie and Beerel's work is the sufficiency of analyzing a finite unfolding for obtaining bounds on the average time separation of any pair of events. Their proof exploits the fact that an infinite unfolding of a finite graph has a repeating structure. This result effectively allows us to focus only on acyclic Petri Nets (resulting from finite unfoldings of a cyclic Petri Net) in the current work. Xie et al. also presented a method to identify the degree of unfolding sufficient to find the required bounds, and proved that such a finite unfolding always exists. Once the degree of unfolding is determined, they assumed probability distributions of individual component delays and performed Monte Carlo simulation to determine the required bounds. In [11], yet another method, called *stochastic simulation*, has been used to obtain average cycle time of asynchronous systems described using probabilistic *event-rule* graphs.

While Monte Carlo simulation is a powerful technique, a typical problem faced in practice is the large number of simulations necessary to obtain statistically meaningful results. This can pose serious problems when analyzing large systems repeatedly, for example, in a design-analyze-redesign loop. Yet

*This work was done when R. Angrish was at the Dept. of Computer Sc. and Engineering, IIT Bombay.

another problem is that Monte Carlo simulation requires knowledge of the probability distributions of individual component delays. Typically, component delays are assumed to be Gaussian, exponential or uniformly distributed, since exact delay distributions are hard to determine. Unfortunately, the results of Monte Carlo simulation are only as good as the probability models used. Errors in probability models, such as modeling a Gaussian variable by an exponential distribution, can lead to serious errors in the final results when non-linear functions of random variables are involved. In contrast, parameters such as the statistical mean and variance of component delays are much more easily determined, and often available directly from data sheets. This motivates the investigation of probabilistic timing analysis techniques for asynchronous systems in which only a few statistical parameters (and not complete probability distributions) of component delays are known. In this paper, we address this problem assuming that only the first two moments of component delays are available.

The remainder of this paper is organized as follows. Section 2 describes a formalism for representing timing constraints between events and formalizes the problem. Section 3 gives a graphical characterization of the problem of statistical dependence when performing probabilistic timing analysis with unknown distributions. Section 4 describes a simple procedure to compute initial bounds of the first and second moments of time separations of all pairs of events. Section 5 describes a polynomial-time algorithm to compute more refined bounds of the first and second moments of times of occurrence of all events. We present some experimental results in Section 6, and conclude the paper in Section 7.

2 Problem Formalization

We view an asynchronous system as a collection of components interacting through *events*. Timing and causality constraints between events are represented by a *timing constraint graph*. Vertices in the graph represent events and constraints between events are represented by directed, weighted edges.

Let i and j denote events, with times of occurrence τ_i and τ_j respectively. Let f denote an event that occurs after both i and j have occurred, potentially waiting for different durations after i and j. In particular, let $\tau_f = \max(\tau_i + d_1, \tau_j + d_2)$. We represent this synchronization constraint in a timing constraint graph as shown in Fig. 1a. If, however, f waits only for i, and occurs d_1 time units after i, we represent the corresponding sequencing constraint as shown in Fig. 1b. Thus, timing constraint graphs enable us to represent synchronization and sequencing constraints. In this sense, they are equivalent to *marked graphs* [16]. Like marked graphs, timing constraint graphs are cyclic in general, describing constraints between events that repeat over time. However, as shown by Xie and Beerel, bounds on the average time separation of events in stochastic marked graphs can be obtained by analyzing only a finite unfolding of the graph. Therefore, we restrict our analysis in this paper to acyclic timing constraint graphs obtained after finite unfoldings of cyclic graphs. For simplicity, we also assume that the acyclic graph thus obtained has a single source event (no incoming edges) and a single sink event (no outgoing edges). Extensions to our algorithm for analyzing multi-source and/or multi-sink graphs are discussed in the concluding section.

While marked graphs or timing constraint graphs do not permit representation of choice (non-determinism) or conflict, the literature contains several examples of interesting systems modeled with marked graphs, and hence representable using timing constraint graphs. Thus, although our analysis technique is restricted in its applicability to systems free of choice and conflict, we believe the class of systems that can be modeled using timing constraint graphs is large enough to make such analysis useful in practice.

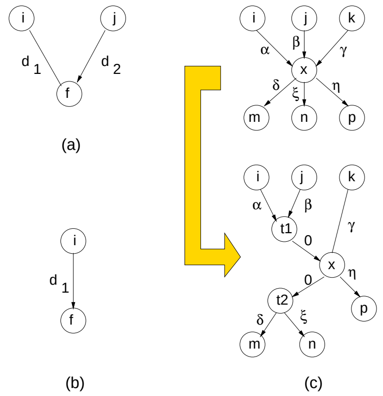

Figure 1. Timing constraint graphs.

Since a synchronization or max constraint between more than 2 events (as in Fig. 1c) can be decomposed into a sequence of max constraints, each between at most two events, we will henceforth assume that the in-degree of each vertex is at most 2. Similarly, for each vertex with out-degree (or fanout) greater than 2, we can use a binary fanout tree to ensure that all timing constraints in the original graph are correctly represented while no vertex has out-degree greater than 2. Thus, the maximum out-degree of each vertex may also assumed to be 2. An example of such transformations converting an arbitrary timing constraint graph to one with maximum in-degree and out-degree 2 is illustrated in Fig. 1c. As can be seen from the figure, some vertices in the timing constraint graph (e.g. $t1$ and $t2$ in Fig. 1c) may no longer correspond to events in the original system. Similarly, edges with zero delay may be introduced.

In general, edge delays in a timing constraint graph correspond to component delays in the system being modeled. The problem we wish to address can now be formalized as follows: *Given a single-source, single-sink acyclic timing constraint graph with first and second moments of all edge delays, we wish to compute bounds on the first and second moments of times of occurrence of all events in the graph relative to the time of occurrence of the source event.*

Bounds on moments thus computed are useful both for performance analysis and timing verification of asynchronous systems. For example, if the processing of a job is modeled by an acyclic timing constraint graph, bounds on the mean time of occurrence of the event signifying the end of processing gives bounds on the average job latency. Similarly, for certain cyclic graphs (representing events that repeat over time), it might be possible to unfold the graph a finite number of times and compute bounds on the average cycle time of a repeating event. In the context of verification, bounds on the first and second moments of time separations can yield bounds on the probability of timing failures. For example, if we represent the expected value of a random variable X by $\overline{X}$, *Chebychev's inequality* states that $Pr[|X - \overline{X}| \geq a] \leq \frac{Var(X)}{a^2}$, where $Var(X)$ is the variance of X. Thus, bounds on the variance of X enable us to bound the probability that X deviates from its mean by more than a specified amount, even when we do not have exact probability distributions.

3 A Graphical Characterization

In the following discussion, we assume that time is measured relative to the time of occurrence of the source event in a timing constraint graph. Thus, the time of occurrence of any event is an expression E derived from the following grammar: $E \rightarrow$ edge_delay$|\max(E, E)|E + E|(E)$, where edge_delay represents edge delays in the timing constraint graph. Henceforth, such expressions will be called *timing expressions*. Now suppose $Z = \max(X, Y)$, where X, Y and Z are timing expressions. It can be shown that $\max(\overline{X}, \overline{Y}) \leq \overline{Z} \leq \frac{1}{2}\left(\overline{X} + \overline{Y} + \sqrt{\overline{X^2} + \overline{Y^2} - 2\overline{XY}}\right)$. Hence, bounds of $\overline{Z}$ can be computed from bounds of the first and second moments of X and Y, and from bounds of $\overline{XY}$. If X and Y are independent, $\overline{XY}$ equals $\overline{X}.\overline{Y}$; otherwise, without knowledge of the joint distribution of X and Y, we can at best compute conservative bounds of $\overline{XY}$. One way to compute such bounds is to use Cauchy-Schwarz inequality, which states that for any random variables X and Y, $(\overline{XY})^2 \leq \overline{X^2}.\overline{Y^2}$. This gives $-\sqrt{\overline{X^2}.\overline{Y^2}} \leq \overline{XY} \leq \sqrt{\overline{X^2}.\overline{Y^2}}$.

Since $Z = \max(X, Y)$, it can also be shown that $\overline{Z^2} = \frac{1}{2}\left(\overline{X^2} + \overline{Y^2} + \overline{(X+Y)|X-Y|}\right)$. In general, $(X + Y)$ and $|X - Y|$ are statistically dependent since each is a function of both X and Y. Therefore, without knowledge of the joint distribution of X and Y, we are again forced in the general case to obtain conservative bounds of $\overline{(X+Y)|X-Y|}$ – e.g., by applying Cauchy-Schwarz inequality. Using this inequality and simplifying, we get

$$\left(U - V.\sqrt{\frac{2U}{V} - 1}\right) \leq 2.\overline{Z^2} \leq \left(U + V.\sqrt{\frac{2U}{V} - 1}\right)$$

where $U = \overline{X^2} + \overline{Y^2}$ and $V = \overline{(X - Y)^2} = \overline{X^2} + \overline{Y^2} - 2\overline{XY}$. Once again, $\overline{XY}$ can be computed easily if X and Y are independent.

The above discussion suggests that it is desirable to have independent arguments for every max subexpression in a timing expression. This motivates the following definition:

Definition 1 A simple timing expression *is one in which the arguments of every max subexpression are statistically independent. Otherwise, a timing expression is called* complex.

Notice that some complex timing expressions can be rendered simple by rearranging terms, e.g. $\max(d_1 + d_2, d_1 + d_3)$ is complex, but $d_1 + \max(d_2, d_3)$ is not. However, if all timing expressions for τ_f are complex, then τ_f is called an *inherently complex* timing expression. As an example, consider the graph in Fig. 2. Here $\tau_t - \tau_s = \max(d_2 + d_4, d_5 + \max(d_1, d_2 + d_3))$. It can

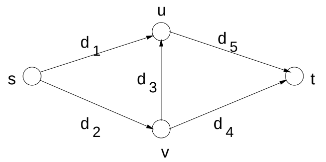

Figure 2. Eye structure.

be easily verified that there does not exist any rearrangement of terms such that all max subexpressions have independent arguments. Hence, $\tau_t - \tau_s$ is an inherently complex timing expression. The following lemma characterizes timing constraint graphs that give rise to inherently complex expressions for the time of occurrence of events:

Lemma 1 *Let G be a timing constraint graph without any constant delay edges. Let s be the source vertex (no incoming edges), and u, v and t be other distinct vertices in G. Let $\pi_{i,j}$ denote a path from i to j. Then $\tau_t - \tau_s$ is inherently complex iff there exist paths $\pi_{s,u}$, $\pi_{s,v}$, $\pi_{v,u}$, $\pi_{v,t}$ and $\pi_{u,t}$ such that (i) $\pi_{s,u}$ does not include vertex v, and (ii) $\pi_{v,t}$ does not include vertex u.*

Proof: We assume that the vertices in the acyclic graph are topologically indexed, such that the index of any vertex is greater than those of its predecessors. In addition, each edge is assumed to have a non-constant delay, i.e. delay with non-zero variance.

If part: If the conditions on the paths are satisfied, the situation is as shown in Fig. 2. In general, some of the edges may be replaced by paths of more than one edge. In addition, we may have some overlap between $\pi_{s,v}$ and $\pi_{s,u}$ but $\pi_{s,u}$ cannot pass through v. Similarly, $\pi_{v,t}$ may partially overlap with $\pi_{u,t}$ but $\pi_{v,t}$ cannot pass through u. Let the delays along paths $\pi_{s,u}$, $\pi_{s,v}$, $\pi_{v,u}$, $\pi_{v,t}$ and $\pi_{u,t}$ be d_1, d_2, d_3, d_4 and d_5 respectively. It follows that $\tau_t - \tau_s = \max(D_{\langle s,u,v,t\rangle}, D_{other})$, where $D_{\langle s,u,v,t\rangle}$ equals $\max(d_2 + d_4, d_5 + \max(d_1, d_2 + d_3))$, and D_{other} denotes the maximum delay along all other paths from s to t. By "other" paths, we mean all paths other than (i) $\pi_{s,u}$ followed by $\pi_{u,t}$, or (ii) $\pi_{s,v}$ followed by $\pi_{v,t}$, or (iii) $\pi_{s,v}$ followed by $\pi_{v,u}$ followed by $\pi_{u,t}$. Since we must determine the maximum delay along all paths from s to t in order to obtain $\tau_t - \tau_s$, the delays of the above three paths must be compared in any expression for $\tau_t - \tau_s$. However, as argued in the discussion for Fig. 2, the expression $D_{\langle s,u,v,t\rangle}$, representing the maximum delay along the three paths referred to above, is an inherently complex timing expression. It follows that $\tau_t - \tau_s$ is inherently complex.

Only if part: Suppose the timing expression for $\tau_t - \tau_s$ is inherently complex. In general, there exists a non-empty subset of vertices that appear in every path from s to t. Let $M = \{m_0 = s, m_1, \ldots m_k, m_{k+1} = t\}$ be the largest such subset with the vertices indexed in topological order. In other words, there exists a path from m_i to m_{i+1} for each i in 0 to k. Then $\tau_t - \tau_s = \sum_{i=0}^{k} D_i$, where $D_i = \tau_{m_{i+1}} - \tau_{m_i}$. Note that since M is the largest subset of vertices appearing in every path from s to t, no vertex on a path from m_i to m_{i+1}, other than m_i and m_{i+1}, can be present in M. Now, if each D_i is a simple timing expression, then so is $\tau_t - \tau_s$. For the latter to be inherently complex, there must exist an i^* such that D_{i^*} is inherently complex. To complete the proof, we now show that there exist paths from m_{i^*} to m_{i^*+1} satisfying the conditions of the lemma. For ease of notation, we will subsequently refer to m_{i^*} as a and m_{i^*+1} as b.

If b has a single predecessor, say l, then both b and l must be present in the set M, since all paths that pass through b also pass through l. This implies that vertex a is the same as vertex l, and there exists a single path from a to b. This contradicts our assumption that $\tau_b - \tau_a$ is inherently complex. Thus, b must have two predecessors, say l and m. Let the delays of edges (l, b) and (m, b) be d_1 and d_2 respectively. Then, $\tau'_b = \max(\tau'_l + d_1, \tau'_m + d_2)$, where τ'_x stands for $\tau_x - \tau_a$. In order for this expression to be inherently complex, there must exist an edge (n_1, n_2) with delay δ, such that δ appears in both arguments of the max expression. In other words, this edge appears in a path from a to l and also in a path from a to m. In general, multiple edges might satisfy this condition. From among them, let us choose the edge (n_1^*, n_2^*) with the highest topological index of n_2^*. Let the delay of this edge be δ^*. Thus, we have the situation depicted in bold in Fig. 3.

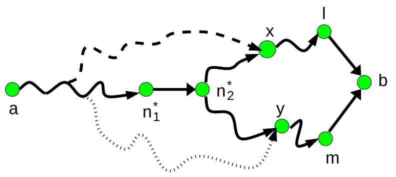

Figure 3. Graph structure.

Now, if both arguments of the max expression for τ'_b are of the form $\delta^* + D$, where D is a timing expression, it would imply that the edge (n_1^*, n_2^*) appears in all paths from a to b. Therefore, n_1^* and n_2^* must be in the set M, and cannot be in the subgraph between vertices a and b. Therefore, to be consistent with our assumptions, δ^* must appear in a max subexpression in at least one of τ'_l or τ'_m. Without loss of generality, let this be τ'_l. Since δ^* is nested within the max subexpression for τ'_l, there must exist one path π_1 from a to l that includes (n_1^*, n_2^*), and another path π_2 between the same vertices that does not include this edge. Let π'_1 be the subpath of π_1 from n_2^* to l, and let x be the vertex with the smallest topological index where π_2 intersects π'_1. Note that π_2 cannot intersect any path from n_2^* to m, except possibly at n_2^*. Otherwise n_2^* would not be the vertex with the highest topological index such that (n_1^*, n_2^*) lies on a path from a to l and also on a path from a to m. We now have two possibilities.

1. Path π_2 intersects subpath π'_1 at a vertex other than n_2^*. This gives rise to the graph structure shown in bold and dashed lines in Fig. 3. Clearly, this structure satisfies the conditions of the lemma.

2. All paths π_2 from a to l pass through n_2^*. In this case, there must exist at least one path from a to m which does not pass through n_2^*, as otherwise, n_2^* would have been in the set M. Let this path be π_3. We now have the graph structure shown in bold and dotted lines in Fig. 3. Once again, this satisfies the conditions of the lemma.

■

Henceforth, we will refer to a set of four vertices satisfying the conditions of Lemma 1 as an "eye" structure.

4 Computing Bounds of Moments

There are two obvious ways to address the problem of dependent max arguments due to eye structures in a timing constraint graph. First, we can modify the graph such that all eye structures are removed. The goal here is to modify the graph minimally and in a way that still gives correct bounds of moments of time of occurrence of events in the original graph. Alternatively, we can preserve the structure of the graph and try to compute as good bounds as possible, making conservative approximations when confronted with max subexpressions with dependent arguments.

In the first approach, there are two ways to modify a graph to remove an eye structure. These are shown in Fig. 4a and b, and are called *forward* and *backward split* respectively. In Fig. 4a, vertex u is split and the path from u to t (forming part of the eye) is replicated. We introduce a new delay variable, d_6, and require that d_6 and d_5 (delay of path from u to t in the eye) be independent and identically distributed random variables. In the modified graph, $\tau_t - \tau_s = \max(d_1 + d_5, d_2 + \max(d_4, d_3 + d_6))$; so this is a simple timing expression. A similar analysis for the backward split graph shown in Fig. 4b shows that $\tau_t - \tau_s = \max(d_2 + d_4, d_5 + \max(d_1, d_7 + d_3))$, where d_7 and d_2 are independent and identically distributed. Once again, $\tau_t - \tau_s$ is a simple expression.

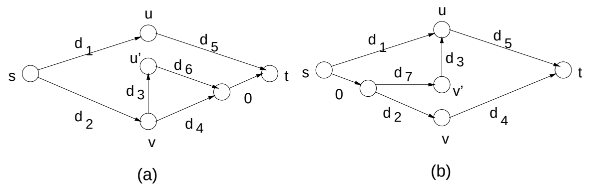

Figure 4. (a) Forward split. (b) Backward split.

Using Cauchy-Schwarz inequality and algebraic simplifications, it can be shown that bounds on moments of $\tau_t - \tau_s$ obtained from either a forward or backward split eye structure are conservative. In other words, $\overline{\tau_t - \tau_s}$ and $\overline{(\tau_t - \tau_s)^2}$ in the original timing constraint graph lie within the bounds computed from the modified graph. This justifies splitting the eye structure in the manner depicted in Fig. 4a and b.

By splitting eye structures, we have effectively created a timing constraint graph that is different from the original graph, but is easier to analyze. It would be desirable to modify the original graph minimally, and yet remove all eye structures. Unfortunately, for large graphs, this approach tends to alter the original graph significantly. We conjecture that this can lead to overly conservative bounds for the mean and variance of the times of occurrence of events.

In the second approach, we use elementary algebra and probability theory to compute bounds of the first two moments of time separations of events, without modifying the timing constraint graph. The algorithm consists of inspecting the expression for the time separation of every pair of events, and applying properties of expectation of random variables to bound the first and second moments of these expressions. Whenever we are confronted with a max expression with dependent arguments, we make use of Cauchy-Schwarz inequality to determine conservative bounds of the required moments. The acyclic nature of the timing constraint graph is also exploited to determine dependency patterns, so that bounds are computed in the order in which they are needed to compute further bounds. The details of the procedure are described below.

For a graph with n events, we use three $n \times n$ matrices, A, B and C, to store bounds of the first and second moments of time separation of events. We initialize the diagonal entries of each matrix with 0. Non-diagonal entries in A and B are initialized with $+\infty$, while those in C are initialized with 0. Thus, for arbitrary events f and s in the timing constraint graph, we have $-A[s,f] \leq \overline{\tau_s - \tau_f} \leq A[f,s]$ and $C[f,s] \leq \overline{(\tau_s - \tau_f)^2} \leq B[f,s]$ after initialization of the matrices. We maintain these inequalities at all times, as we update the matrices with more refined estimates of bounds. Since $(\tau_f - \tau_s)^2 = (\tau_s - \tau_f)^2$, it follows that $B[f,s] = B[s,f]$ and $C[f,s] = C[s,f]$.

The following results are obtained by applying Cauchy-Schwarz inequality and algebraic simplifications to the expressions for time separation of events. The details are omitted due to lack of space.

(I) Let event f have a single predecessor i in the timing constraint graph. Let the delay of edge (i,f) be δ_1. Then for an arbitrary event s, we have $\tau_f - \tau_s = \tau_i - \tau_s + \delta_1$. It follows that:

- $-A[i,s] + \overline{\delta_1} \leq \overline{\tau_f - \tau_s} \leq A[s,i] + \overline{\delta_1}$.
- $C[i,s] + \overline{\delta_1^2} + 2.W_L \leq \overline{(\tau_f - \tau_s)^2} \leq B[i,s] + \overline{\delta_1^2} + 2.W_U$, where W_L and W_U denote lower and upper bounds of $W = \overline{(\tau_i - \tau_s).\delta_1}$. Bounds on W are obtained as follows:
  - If there are no paths from f to s, $-A[i,s].\overline{\delta_1} \leq W \leq A[s,i].\overline{\delta_1}$.
  - If f is a dominator of s (all paths to s pass through f),

$$-\overline{\delta_1^2} - A[f,s].\overline{\delta_1} \leq W \leq -\overline{\delta_1^2} + A[s,f].\overline{\delta_1}.$$

  - Otherwise, $-\sqrt{B[i,s].\overline{\delta_1^2}} \leq W \leq \sqrt{B[i,s].\overline{\delta_1^2}}$.

(II) Let event f have two predecessors, i and j, in the timing constraint graph. Let the delays of edges (i,f) and (j,f) be δ_1 and δ_2 respectively. Then, for an arbitrary event s, $\tau_f - \tau_s = \max(\tau_i - \tau_s + \delta_1,\ \tau_j - \tau_s + \delta_2)$. It follows that:

- $\max(-A[i,s] + \overline{\delta_1}, -A[j,s] + \overline{\delta_2}) \leq \overline{\tau_f - \tau_s} \leq \frac{1}{2}\cdot\left(A[s,i] + A[s,j] + \overline{\delta_1} + \overline{\delta_2} + \sqrt{F_1(i,j,f)}\right)$, where

$$\begin{aligned} F_1(i,j,f) &= B[i,j] + \overline{\delta_1^2} + \overline{\delta_2^2} - 2.\overline{\delta_1}.\overline{\delta_2} + \\ &\quad 2.F_2(i,j,f).(\overline{\delta_1} - \overline{\delta_2}), \text{ and} \quad (1) \\ F_2(i,j,f) &= \begin{cases} A[j,i] & \text{if } \overline{\delta_1} \geq \overline{\delta_2} \\ -A[i,j] & \text{otherwise} \end{cases} \quad (2) \end{aligned}$$

- To compute bounds of $\overline{(\tau_f - \tau_s)^2}$, we first compute bounds of $U = \overline{X^2} + \overline{Y^2}$ and $V = \overline{(X-Y)^2}$, where $X = \tau_i - \tau_s + \delta_1$ and $Y = \tau_j - \tau_s + \delta_2$. $F_1(i,j,f)$ from Eqn. (1) gives an upper bound of V. A lower bound of V is obtained by replacing $B[i,j]$ by $C[i,j]$, $A[j,i]$ by $-A[i,j]$ and $-A[i,j]$ by $A[j,i]$ in Eqns. (1) and (2). To obtain bounds of U, we bound $\overline{X^2}$ and $\overline{Y^2}$ separately. Computing bounds of $\overline{X^2}$ is similar to computing bounds of $\overline{(\tau_f - \tau_s)^2}$ when f has a single predecessor i (case (I) above). As in the single predecessor case, three different scenarios need to be considered: (i) no path from f to s, (ii) f dominates s, and (iii) none of (i) or (ii). In the first and third cases, the expressions for bounds of $\overline{X^2}$ are identical to those obtained when considering f with a single predecessor i. In the second case, however, the expression is different. In particular, if f dominates s, then $\overline{(\tau_i - \tau_s).\delta_1}$ is bounded above by $A[s,f].\overline{\delta_1} + \min\left(-\overline{\delta_1^2},\ \overline{\delta_1}.(A[j,i] - \overline{\delta_2})\right)$ and below by $-A[f,s].\overline{\delta_1} - \sqrt{\overline{\delta_1^2}.B[i,f]}$. Having obtained bounds of $\overline{(\tau_i - \tau_s).\delta_1}$, we can now bound $\overline{X^2}$. Bounds of $\overline{Y^2}$ are computed similarly. These are then combined to obtain bounds for U. Let $\alpha \leq U \leq \beta$ and $\gamma \leq V \leq \eta$. It can then be shown that $\kappa \leq \overline{Z^2} \leq \lambda$, where

$$\kappa = \begin{cases} \frac{1}{2}\cdot\left(\alpha - \eta\cdot\sqrt{(\frac{2\alpha}{\eta} - 1)}\right) & \text{if } \eta < \alpha \\ \frac{1}{2}\cdot\left(\beta - \gamma\cdot\sqrt{(\frac{2\beta}{\gamma} - 1)}\right) & \text{if } \gamma > \beta \\ 0 & \text{otherwise} \end{cases}$$

$$\lambda = \begin{cases} \frac{1}{2}\cdot\left(\beta + \eta\cdot\sqrt{(\frac{2\beta}{\eta} - 1)}\right) & \text{if } \eta < \beta \\ \frac{1}{2}\cdot\left(\beta + \gamma\cdot\sqrt{(\frac{2\beta}{\gamma} - 1)}\right) & \text{if } \gamma > \beta \\ \beta & \text{otherwise} \end{cases}$$

Thus, we have derived simple bounds for $\overline{\tau_f - \tau_s}$ and $\overline{(\tau_f - \tau_s)^2}$ that can be used to update the corresponding entries in the A, B and C matrices. Observe that the roles of f and s in the above discussion are not identical. In particular, we expressed τ_f in terms of the times of occurrence of f's predecessors, whereas this was not the case for τ_s. Event f therefore plays a special role, and is called the *primary event*, in the above derivations. However, there is nothing special about the choice of f: bounds for $\overline{\tau_f - \tau_s}$ and $\overline{(\tau_f - \tau_s)^2}$ can be computed along similar lines by choosing s as the primary event as well. Thus, each bound ($A[f,s]$, $B[f,s]$ and $C[f,s]$) can be computed in two different ways: once using f as the primary event, and then using s as the primary event.

An inspection of the expressions for the bounds derived above reveals the following dependencies between the elements of matrices A, B and C. In the following, we assume that i and j are predecessors of f in the timing constraint graph.

D1: When computing $A[f,s]$ and $A[s,f]$ with f as the primary event, values of $A[i,s]$, $A[s,i]$, $A[j,s]$, $A[s,j]$, $A[i,j]$, $A[j,i]$ and $B[i,j]$ are needed.

D2: When computing $B[f,s]$ or $C[f,s]$ with f as the primary event, values of all matrix elements listed in D1 above are needed. In addition, values of $B[i,s]$, $B[j,s]$, $C[i,s]$, $C[j,s]$, $A[f,s]$, $A[s,f]$ and $C[i,j]$ are also needed. Further, if f dominates s, the elements $B[i,f]$ and $B[j,f]$ are also needed to compute $C[f,s]$.

Layer 0	0,0	0,1	0,2	0,3
Layer 1	1,0	1,1	1,2	1,3
Layer 2	2,0	2,1	2,2	2,3
Layer 3	3,0	3,1	3,2	3,3

Figure 5. Layers of a matrix.

These dependencies suggest a layer-wise computation of matrices A, B and C, similar to that used in [4]. To recapitulate, all events are assumed to be topologically indexed. Thus, if events i and j are predecessors of event f in the timing constraint graph, then $i < f$ and $j < f$. Layer f of a matrix, say A, is composed of elements $A[f,s]$ and $A[s,f]$ with $s \leq f$. An $n \times n$ matrix can thus be viewed as composed on n layers numbered 0 through $n-1$ (see Fig. 5). We start with layer 0 of each matrix, i.e., $A[0,0]$, $B[0,0]$ and $C[0,0]$ which are trivially 0, and proceed by computing the elements in each successive layer, until all elements have been computed. Within layer f of a matrix, say A, the elements $A[f,s]$ and $A[s,f]$ are computed in order of increasing s, i.e., from 0 to $f-1$. Dependency D1 ensures that all elements in layer f of A can be computed using elements in layers $f-1$ or less of A and B, if f is used as the primary event. Similarly, dependency D2 ensures that in computing elements in layer f of B and C, we only need elements in layers $f-1$ or less of the same matrices and elements in layer f of A, if f is used as the primary event. Since $s \leq f$ within a layer, the case of f dominating s does not arise. Thus, by computing elements of each matrix in order of increasing layers, and by computing layer f of A before the corresponding layer of B and C, we can ensure that all matrix elements are computed and updated before being used to compute other elements.

The above discussion treated f as the primary event when computing elements like $A[f,s]$ or $B[s,f]$ in layer f of matrices A, B and C. We now consider using s as the primary event. Once again, dependencies D1 and D2 above ensure that by proceeding in order of increasing s within a layer f, all matrix elements are computed and updated before being used to compute other elements. This suggests a simple algorithm, called InitialBounds and outlined in Fig. 6, for computing initial bounds of moments of time separation of events. For each f in 0 through $n-1$, we first compute elements in layer f of matrix A and then compute the corresponding elements in layer f of matrices B and C, using f as the primary event. We then recompute all bounds in layer f in the same order using s as the primary event, and retain the better of the two bounds for each element of each matrix. For a timing constraint graph with n events, the complexity of algorithm InitialBounds is $O(n^2)$. This excludes the complexity of determining dominator relations and reachability relations between vertices in the graph. Recall that these relations are needed to determine the right formula to use to update the matrix entries.

Initialization:
for each f in 0 to $n-1$
  for each s in 0 to $n-1$
    if ($f == s$) $A[f,f] = B[f,f] = C[f,f] = 0$
    else $A[f,s] = B[f,s] = +\infty$; $C[f,s] = 0$

InitialBounds (G : graph; A, B, C : matrices)
$n =$ number of events in G.
for each f in 0 to $n-1$
  (a) **for each** s in 0 to $f-1$
    Compute $A[f,s]^{new}$, $A[s,f]^{new}$ with primary event f
    $A[f,s] = \min(A[f,s], A[f,s]^{new})$
    $A[s,f] = \min(A[s,f], A[s,f]^{new})$
    Compute $B[f,s]^{new}$, $C[s,f]^{new}$ with primary event f
    $B[f,s] = B[s,f] = \min(B[f,s], B[f,s]^{new})$
    $C[f,s] = C[s,f] = \max(C[f,s], C[f,s]^{new})$
  (b) **for each** s in 0 to $f-1$
    Repeat step (a) with primary event s.

Figure 6. Pseudo-code of algorithm InitialBounds.

5 Exploiting the Graph Structure

We now describe an algorithm to compute bounds that are at least as good as, and potentially better than, those obtained by one invokation of InitialBounds. In other words, upper bounds obtained by the algorithm are no larger than those computed by InitialBounds. Similarly, lower bounds are no smaller than those computed by InitialBounds. In developing this algorithm, we view a timing constraint graph with source s and sink t merely as a representation of the timing expression for $\tau_t - \tau_s$. The correspondence between vertices in the graph and events in the system are not necessarily preserved during the execution of the algorithm.

5.1 Intuition

Consider the eye-structure in Fig. 2, and suppose we wish to compute bounds on moments of $\tau_t - \tau_s$. Algorithm InitialBounds proceeds by first computing bounds on moments of $\tau_u - \tau_s$, $\tau_v - \tau_s$ and $\tau_u - \tau_v$, and then using these bounds to derive bounds on moments of $\tau_t - \tau_s$. This is "similar" to analyzing the eye structure after splitting it backward, in the following sense: In a backward split eye, as shown in Fig. 4b, one proceeds by computing bounds on moments of $\tau_v - \tau_s$ and $\tau_u - \tau_s$, and then combining them to obtain bounds on moments of $\tau_t - \tau_s$. The difference between analyzing the backward split eye and applying algorithm InitialBounds to the original eye lies in the treatment of $\tau_u - \tau_s$ and $\tau_v - \tau_s$ as independent in the former case, and dependent (in general) in the latter. In contrast, there is no obvious correspondence between the operation of algorithm InitialBounds on the eye structure of Fig. 2, and the analysis of the forward split eye in Fig. 4a. Thus, algorithm InitialBounds ignores, in the above sense, the option of splitting the eye forward when computing bounds on moments of $\tau_t - \tau_s$.

Interestingly, if we reverse all edge directions in Fig. 2, we obtain a new eye structure. On splitting this new structure backward, and reversing all edges again, we obtain a structure similar to the forward split eye of Fig. 4a. Thus if algorithm InitialBounds is applied to the structure obtained by reversing all edges in Fig. 2, we effectively end up considering (in the sense described above) the forward split eye. Since $\tau_t - \tau_s$ in the original eye structure equals $\tau_s - \tau_t$ in the reversed eye structure, by applying algorithm InitialBounds to the original structure and then to the reversed one, and by taking the better of the two results, we have the benefit of considering both a forward and a backward split eye.

The above observation is formalized in the following lemma.

Lemma 2 *Let G be a timing constraint graph with source s and sink t. Let G' be the graph obtained by reversing all edges of G. The time of occurrence of t relative to s in G is given by the same timing expression as the time of occurrence of s relative to t in G'.*

Since every path from s to t in G is converted to a path from t to s in G', and since the time separation between the sink and source events is simply the maximum delay along all such paths, the lemma is easily proved. Thus, to compute bounds on moments of $\tau_t - \tau_s$, we can apply algorithm InitialBounds to the timing constraint graph G and also to the reversed graph G', and then choose the better of the two bounds. In fact, this strategy can be applied recursively. For example, in Fig. 2, algorithm InitialBounds uses bounds on moments of $\tau_u - \tau_s$ and $\tau_v - \tau_s$ to compute bounds on moments of $\tau_t - \tau_s$. The computation of $\tau_u - \tau_s$ can itself be done by analyzing the subgraph with source s and sink u, once with edge directions unchanged, and then with all edge directions reversed. The same holds true for the computation of $\tau_v - \tau_s$.

5.2 A polynomial-time algorithm

We now formalize the intuition of the previous subsection in an algorithm. To illustrate how the algorithm works, consider the timing constraint graph G in Fig. 7. Let G' be the graph obtained by reversing the direction of all edges of G. We will refer to a subgraph of G with source u and sink v by $G_{u,v}$. Similarly, a subgraph of G' with source u and sink v will be referred to as $G'_{u,v}$.

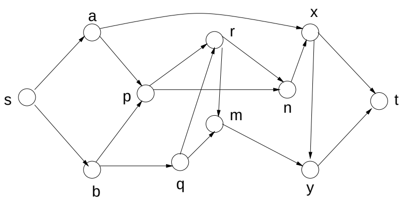

Figure 7. General timing constraint graph.

To determine bounds on moments of $\tau_t - \tau_s$ in $G_{s,t}$ (i.e., G), we proceed by computing bounds on moments of $\tau_x - \tau_s$ and $\tau_y - \tau_s$, where x and y are the predecessors of t in G. The bounds thus computed are used to initialize the corresponding entries in the A, B and C matrices of algorithm InitialBounds (see Fig. 6). Algorithm InitialBounds is then invoked on graph $G_{s,t}$ to compute bounds on moments of $\tau_t - \tau_s$. Next, we reverse graph $G_{s,t}$ to obtain $G'_{t,s}$, determine bounds on moments of $\tau_a - \tau_t$ and $\tau_b - \tau_t$ in the reversed graph, and use these bounds to initialize the corresponding entries in matrices A', B' and C'. The primed matrices have the same interpretation and dimensions as A, B and C referred to above. However, while matrices A, B and C are used by algorithm InitialBounds when analyzing $G_{s,t}$, the primed matrices are used when analyzing $G'_{s,t}$. Finally, algorithm InitialBounds is invoked on graph $G'_{t,s}$ to compute bounds on moments of $\tau_s - \tau_t$ in $G'_{t,s}$. Thus, to compute bounds on moments of $\tau_t - \tau_s$ in $G_{s,t}$, we need to compute bounds on moments of $\tau_x - \tau_s$ and $\tau_y - \tau_s$ in graph $G_{s,t}$, and bounds on moments of $\tau_a - \tau_t$ and $\tau_b - \tau_t$ in graph $G'_{s,t}$. Further, we can compute each of these bounds recursively in a similar manner. For example, to compute bounds on $\tau_x - \tau_s$ in $G_{s,t}$, we first analyze the subgraph $G_{s,x}$ consisting of only those edges and vertices of $G_{s,t}$ that lie on some path from s to x. Thereafter, the graph $G'_{x,s}$ obtained by reversing all edges of $G_{s,x}$ is analyzed to determine bounds on moments of $\tau_s - \tau_x$. The better of the bounds computed by the two approaches is then used to bound moments of $\tau_x - \tau_s$.

Let forward(s,t) be a procedure for computing bounds of the first and second moments of $\tau_t - \tau_s$ in the timing constraint subgraph $G_{s,t}$ with source s and sink t. The directions of all edges in $G_{s,t}$ are assumed to be the same as in the original timing constraint graph. Similarly, let backward(t,s) be a procedure for computing bounds of the first and second moments of $\tau_s - \tau_t$ in the subgraph $G'_{t,s}$ with source t and sink s. The directions of all edges in $G'_{t,s}$ are assumed to be the reverse of those in $G_{s,t}$. It follows from the discussion in the previous paragraph that one invokation of forward(s,t) on the timing constraint graph of Fig. 7 leads to invokations of forward(s,x), forward(s,y), backward(t,a) and backward(t,b). The results of the two forward invokations are used in algorithm InitialBounds to compute $\tau_t - \tau_s$ in graph $G_{s,t}$. The results of the two backward invokations are used by algorithm InitialBounds to compute $\tau_s - \tau_t$ in graph $G'_{t,s}$. The better of these two bounds is then chosen for bounding moments of $\tau_t - \tau_s$. Fig. 8 shows a portion of the call graph resulting from one invokation of forward(s,t) on graph $G_{s,t}$ of Fig. 7.

The above discussion suggests the pseudocodes shown in Fig. 9 for procedures forward and backward. Given a timing

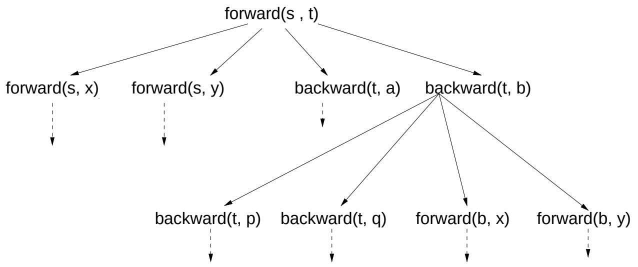

Figure 8. Call graph for Fig. 7.

constraint graph G with source s and sink t, bounds on moments of $\tau_t - \tau_s$ are computed by invoking $\mathsf{forward}(s,t)$. To avoid re-computations, we use memoization to remember the computed bounds and reuse them later. Since the source and sink events along with the direction of edges (same as original or reversed compared to original) uniquely identify the subgraph being analyzed, these three parameters are sufficient to memoize computed bounds and reuse them later. Observe that this technique can be used to compute bounds on moments of $\tau_t - \tau_s$ for an arbitrary event t and the source event s. It cannot, in general, be used to compute bounds on moments of $\tau_i - \tau_j$ for arbitrary events i and j in the graph.

It is easy to see that because of memoization, each time procedure forward is invoked, a different source-sink pair is used. The same holds true for procedure backward as well. Thus, the graph $G_{i,j}$ that is analyzed in one invokation of forward or backward differs from the graph analyzed in every other invokation of the same procedure. This suggests the need for representing the timing constraint graph in a manner that allows easy addition and deletion of edges, as we move up or down the procedure call graph. In our implementation, this is achieved by numbering all edges and representing the validity of edges by a bit vector. The number of bits in the vector equals the number of edges in the graph. Bit i is set to indicate that edge i is a valid edge; it is reset to imply that edge i must be considered deleted (or invalid). As successive procedure invokations and call returns occur, appropriate bits are set or reset in the bit vector to indicate the graph modifications. For a graph with source s and sink t, the initial invokation of $\mathsf{forward}(s,t)$ is made with the entire graph (i.e., all bits in the vector are set). Before each subsequent procedure call, the bit vector is modified by removing those edges that are no longer relevant for calls below the current procedure in the call graph. Thus, deleting and reinstating edges in the timing constraint graph reduces to the operations of setting or resetting bits in a bitvector.

Because of memoization, the number of calls to procedure forward or backward is at most the number of source-sink pairs in the graph. For a graph with n events, this is $O(n^2)$. In each call, the graph structure is modified 4 times – twice for the subsequent forward calls and twice for the subsequent backward calls. Each modification involves setting or resetting bits in a bitvector of length e, where e is the number of edges in the original graph. So this takes $O(e)$ time. Each invokation of algorithm InitialBounds takes $O(n^2)$ time. Thus, the total complexity of the algorithm is $O(n^2.(n^2+e))$. One run of the algorithm calculates bounds on moments of $\tau_v - \tau_s$ for all vertices

/* *$\tau_j - \tau_i$ is computed with respect to graph $G_{i,j}$.* */
/* *The return value is a four-tuple (a,b,c,d), where* */
/* *$a \leq \overline{\tau_j - \tau_i} \leq b$ and $c \leq \overline{(\tau_i - \tau_j)^2} \leq d$.* */

forward(i : event num; j : event num)
1. **if** $(i == j)$ **return** $(0,0,0,0)$
2. **if** $G_{i,j}$ consists only of the edge (i,j),
   return $(\overline{\delta_{i,j}}, \overline{\delta_{i,j}}, \overline{\delta^2_{i,j}}, \overline{\delta^2_{i,j}})$.
3. Let u and v be predecessors of j in $G_{i,j}$
   (a) f_1 = $\mathsf{forward}(i,u)$ or memoized value if computed
   (b) Initialize $A[i,u]$, $A[u,i]$, $B[i,u]$, $C[i,u]$ from f_1.
   (c) $f_2 = \mathsf{forward}(i,v)$ or memoized value if computed
   (d) Initialize $A[i,v]$, $A[v,i]$, $B[i,v]$, $C[i,v]$ from f_2.
   (e) Initialize all other non-diagonal entries in A, B to $+\infty$.
   Initialize all other non-diagonal entries in C to 0.
   Initialize all diagonal entries in A, B, C to 0.
   (f) $\mathsf{InitialBounds}(G_{i,j}, A, B, C)$
4. Let x and y be successors of i in $G_{i,j}$.
   (a) b_1 = $\mathsf{backward}(j,x)$ or memoized value if computed.
   (b) Initialize $A'[j,x]$, $A'[x,j]$, $B'[j,x]$, $C'[j,x]$ from b_1.
   (c) $b_2 = \mathsf{backward}(j,y)$ or memoized value if computed.
   (d) Initialize $A'[j,y]$, $A'[y,j]$, $B'[j,y]$, $C'[j,y]$ from b_2.
   (e) Initialize all other non-diagonal entries in A', B' to $+\infty$.
   Initialize all other non-diagonal entries in C' to 0.
   Initialize all diagonal entries in A', B', C' to 0.
   (f) $\mathsf{InitialBounds}(G'_{j,i}, A', B', C')$
5. val = $(\max(-A[j,i], -A'[i,j]), \min(A[i,j], A'[j,i]),$
   $\max(B[i,j], B'[i,j]), \min(C[i,j], C'[i,j]))$.
6. Memoize and **return** val

backward(i : event num; j : event num)
Similar to $\mathsf{forward}(i,j)$ with $G'_{i,j}$ used in place of $G_{i,j}$ and $G_{j,i}$ in place of $G'_{j,i}$.

Figure 9. Pseudo-code of forward and backward.

v in the original timing constraint graph, where s is the source event of the original graph. Therefore, the amortized complexity of the algorithm is $O(n.(n^2+e))$ per event.

In fact, it is possible to optimize the complexity further by observing that algorithm InitialBounds needs to be called only once for each combination of edge direction (same as in original graph or reversed compared to original graph) and source event. Let u and v be predecessors of t in graph $G_{s,t}$. The A, B and C matrix entries that are initialized from f_1 and f_2 in steps 3(b) and 3(d) of procedure forward are used only at the end when computing $A[s,t]$, $A[t,s]$, $B[s,t]$ and $C[s,t]$ in algorithm InitialBounds. If we do not initialize these entries from f_1 and f_2 before invoking algorithm InitialBounds, and instead compute $A[s,t]$, $A[t,s]$, $B[s,t]$ and $C[s,t]$ after InitialBounds returns by using the values from f_1 and f_2 and the values of $A[u,v]$, $A[v,u]$, $B[u,v]$ and $C[u,v]$ returned by algorithm InitialBounds, we obtain the same results. The same argument holds for the A', B' and C' entries initialized from b_1 and b_2 in steps 4(b) and 4(d), and also for procedure backward. Using this optimization, we need only $O(n)$ invokations of InitialBounds. Thus, the amortized complexity becomes $O(n.(n+e))$.

Graph	# events	# edges	CPU time (s)		$\theta_{1,max}$	$\theta_{1,av}$	$\theta_{2,max}$	$\theta_{2,av}$	$\theta_{3,max}$	$\theta_{3,av}$
			Our algo	Avg. MC						
M-4	42	57	0.06	38.06	0.068	0.014	0.055	0.035	0.106	0.065
M-8	82	113	0.49	78.28	0.082	0.011	0.056	0.043	0.107	0.082
M-16	162	225	3.84	163.92	0.078	0.009	0.056	0.048	0.108	0.093
H-4	105	128	0.71	71.78	0.036	0.022	0.001	0.0004	0.002	0.001
H-8	209	256	14.82	146.46	0.034	0.023	0.001	0.0003	0.002	0.001
H-12	313	384	72.58	215.72	0.036	0.022	0.001	0.0003	0.001	0.001
Diffeq	219	273	7.83	152.68	0.024	0.014	0.721	0.102	0.850	0.152
Arch	46	60	0.04	21.73	0.042	0.012	0.247	0.023	0.394	0.041

Table 1. Table of results.

6 Experimental Results

We have implemented the algorithm described in the previous section, along with the optimization for $O(n)$ invocations of InitialBounds. In order to test the effectiveness of the algorithm, we applied it to a set of timing constraint graphs. Our graphs are obtained either directly from the literature or by unfolding cyclic graphs in the literature a fixed number of times. The examples in the literature give bounds on the delay of each edge. We used these bounds to derive the mean and variance of each edge delay by assuming that the mean lies half-way between the specified lower and upper bounds, and the spread between the bounds represents 6 standard deviations. This corresponds to the usual 3σ spread around the mean. All edge delays are assumed to be statistically independent.

In order to compare our results with those of Monte Carlo simulation, we also ran 1000000 runs of Monte Carlo simulation on each timing constraint graph. Ideally, the number of runs required for Monte Carlo simulation should be determined on a case-by-case basis. However, we fixed the number of runs at 1000000 to simplify the error analysis for Monte Carlo simulation. For each of our benchmarks, we found that the statistical parameters of interest (moments of time separation of events) stabilized within 1000000 runs. Since the running time of Monte Carlo simulation grows linearly with the number of runs, appropriate scaling of the run-time can be easily done for fewer iterations of the simulation.

Since Monte Carlo simulation requires knowledge of the probability distribution of each edge delay, we had to use different strategies for assigning distributions to edge delays, while maintaining the mean and variance as computed above. In the first strategy, we assigned (possibly shifted and scaled) Gaussian distributions to all edge delays. In the second strategy, uniform distribution was assigned to the edge delays; in the third, we assigned gamma distributions to all edge delays, and in the fourth, we used (possibly shifted) exponential distributions for the delays. In the final strategy, we randomly chose between Gaussian, uniform, exponential and gamma for the distribution of each edge delay. Since the bounds computed by our algorithm hold regardless of the type of distribution of edge delays, we evaluated the accuracy of our bounds by comparing them with the largest and smallest values of the appropriate moments obtained from all the above strategies. Since we make no assumptions about the distributions of edge delays, this is an appropriate metric for evaluating the accuracy of our technique. Thus, for each time separation of the form $\tau_t - \tau_s$, where s is the source event of the graph and t is any arbitrary event, we recorded the maximum value of $\overline{\tau_t - \tau_s})$ obtained from the different Monte Carlo simulations. Let this be α. We then determined the upper bound of $\overline{\tau_t - \tau_s}$ computed by our algorithm. Let this be β. Our metric of accuracy for this bound is then given by $\theta_1 = |\frac{\beta-\alpha}{\alpha}|$. This metric, maximized over all events in the graph, gives the maximum relative "error" in our computed bound. In fact, since our bounds hold for all distributions of edge delays, while the Monte Carlo simulations use specific delay distributions, the "error" computed above is really an upper bound of the actual error. We define relative "error" metrics for the lower bound of $\overline{\tau_v - \tau_s}$ and for the upper bound of $\overline{(\tau_v - \tau_s)^2}$ in a similar manner. The maximum relative error of each type, maximized over all events in the graph, gives an indication of the accuracy of our algorithm.

Table 1 shows the results of our experiments. The graphs labeled M-n are obtained by unrolling n times a cyclic timing constraint graph from [12]. The graphs labeled H-n are similarly obtained from [8]. Benchmark Diffeq is a simplified timing constraint graph of the differential equation solver described in [19]. Benchmark Arch is obtained from [5] by treating each vertex in the timing constraint graph to be of max-type. For each benchmark, the number of vertices and edges gives an indication of the size of the graph. To compare the efficiency of our technique vis-a-vis Monte Carlo techniques, we present the times taken by one run of our algorithm on each benchmark and the average time taken by the various Monte Carlo (MC) simulations (1000000 runs) on the same benchmark. The runtimes are for a 500 MHz Pentium 686 processor with 128 MB RAM and running Linux 7.1. For each benchmark, we also give the accuracy of our bounds using the metrics described above. The metrics $\theta_{1,max}$ and $\theta_{2,max}$ give the maximum relative error of the lower and upper bounds of $\overline{\tau_v - \tau_s}$ computed by our algorithm. The maximization is done over all events v in the graph. The metrics $\theta_{1,av}$ and $\theta_{2,av}$ give the average value of the corresponding relative errors, where the averaging is done over all events in the graph. Metrics $\theta_{3,max}$ and $\theta_{3,av}$ give the maximum and average relative error of the upper bound of $\overline{(\tau_v - \tau_s)^2}$, considering all events v in the graph.

Lower bounds of $\overline{(\tau_v - \tau_s)^2}$ computed by our algorithm have high relative errors for a few events v, but are otherwise low. We are currently investigating the reason for this behavior. Nevertheless, one is typically interested in bounding the variance of a random variable from above. For this purpose, the relatively good upper bounds of $\overline{(\tau_v - \tau_s)^2}$ given by our algorithm are of practical significance.

We conclude from the results in Table 1 that our algorithm succeeds in achieving a fair degree of accuracy while remaining efficient. Moreover, the accuracy seems to vary little with the number of unfoldings of a cyclic graph. The most interesting aspect of our technique is that we are able to achieve this efficiency and accuracy without requiring exact distribution of the edge delays. Hence our results hold for all delay distributions that preserve the mean and variance of edge delays.

7 Conclusion

In this paper, we presented a polynomial-time algorithm for computing bounds on the first and second moments of times of occurrence (relative to a source event) of events in a timing constraint graph. Our method does not require knowledge of the edge delay distributions, which are often hard to determine or speculate. Instead, our algorithm works simply with knowledge of the first two moments of each edge delay — parameters that are often readily available from data sheets. Thus, the proposed algorithm enables one to do probabilistic timing analysis even without knowing the exact probability distributions. This has interesting applications in both performance analysis and probabilistic timing verification of asynchronous systems. The efficiency of our technique makes it suitable for use in a design-analyze-redesign loop, where multiple passes of analysis are required.

An apparent drawback of the current method is the requirement of a single-source, single-sink timing constraint graph. For a graph with multiple source events, we can create a new source event, and draw edges from the new source to each one of the original source events. The delays on these edges must be assigned in a way that reflects the time separation between the original source events. For a graph with multiple sink events, we simply create a new sink event and draw zero-delay edges from each of the original sink events to the new sink event. Thus, multi-source, multi-sink acyclic timing constraint graphs can be analyzed by the proposed algorithm with slight modifications. Yet another drawback of our algorithm is the relatively large memory requirements because of deep recursions in procedures forward and backward. We are currently investigating ways to address this problem.

We believe that techniques for probabilistic timing analysis like those proposed by Xie and Beerel [15, 17, 16], along with techniques like the one proposed in this paper are essential tools for the analysis of real-life asynchronous systems.

Acknowledgments

We would like to thank Prof. Kenneth Y. Yun of Univ. of California, San Diego, for initial discussions on this problem. We are thankful to Mr. Abhishek Deb of Institute of Technology, Banaras Hindu University, India, for helping us with Monte Carlo simulations. Finally, we are thankful to the anonymous reviewers for their helpful comments.

References

[1] T. Amon and H. Hulgaard. Symbolic Time Serparation of Events. In *Proc. of ASYNC*, pages 83–93, Apr. 1999.

[2] J. A. Brzozowski, T. Gahlinger, and F. Mavaddat. Consistency and satisfiability of waveform timing specifications. *Networks*, Jan. 1991.

[3] S. M. Burns. *Performance Analysis and Optimization of Asynchronous Circuits*. PhD thesis, California Institute of Technology, 1991.

[4] S. Chakraborty. *Polynomial-Time Techniques for Approximate Timing Analysis of Asynchronous Systems*. PhD thesis, Stanford University, Aug. 1998.

[5] S. Chakraborty and D. L. Dill. Approximate time separation of events in practice. In *Proc. of TAU worskhop*, pages 77–82, Dec. 1997.

[6] T. Gahlinger. *Coherence and Satisfiability of Waveform Timing Specifications*. PhD thesis, University of Waterloo, 1990.

[7] M. R. Greenstreet and K. Steiglitz. Bubbles can make self-timed pipelines fast. *Journal of VLSI Signal Processing*, 2(3):139–148, Nov. 1990.

[8] H. Hulgaard, S. M. Burns, T. Amon, and G. Borriello. An algorithm for exact bounds on time separation of events in concurrent systems. Technical report, Dept. of Computer Sc. and Engg., Univ. of Washington, Feb. 1994. Technical Report No. UW-CSE-94-02-02.

[9] T. K. Lee. *A General Approach to Performance Analysis and Optimization of Asynchronous Circuits*. PhD thesis, California Institute of Technology, 1995.

[10] K. L. McMillan and D. L. Dill. Algorithms for Interface Timing Verification. In *Proc. of ICCD*, pages 48–51, Oct. 1992.

[11] E. G. Mercer and C. J. Myers. Stochastic cycle period analysis in timed circuits. In *Proc. of ISCAS*, pages 172–175, 2000.

[12] C. J. Myers and T. H.-Y. Meng. Synthesis of timed asynchronous circuits. *IEEE Transactions on VLSI Systems*, 1(2):106–119, June 1993.

[13] E. A. Walkup. *Optimization of Linear Max-Plus Systems with Application to Timing Analysis*. PhD thesis, University of Washington, Seattle, 1995.

[14] T. E. Williams. Latency and throughput tradeoffs in self-timed asynchronous pipelines and rings. Technical Report CSL-TR-90-431, Stanford University, Aug. 1990.

[15] A. Xie and P. A. Beerel. Symbolic techniques for performance analysis of timed systems based on average time separation of events. In *Proc. of ASYNC*, pages 64–75, Apr. 1997.

[16] A. Xie and P. A. Beerel. Performance analysis of asynchronous circuits and systems using stochastic timed Petri nets. In A. Yakovlev, L. Gomes, and L. Lavagno, editors, *Hardware Design and Petri Nets*, pages 239–268. Kluwer Academic Pub., Mar. 2000.

[17] A. Xie, S. Kim, and P. A. Beerel. Bounding average time separations of events in stochastic timed Petri nets with choice. In *Proc. of ASYNC*, pages 94–107, Apr. 1999.

[18] T.-Y. Yen, A. Ishii, A. Casavant, and W. Wolf. Efficient algorithms for interface timing verification. *Formal Methods in System Design*, 12(3):241–265, Apr. 1998.

[19] K. Y. Yun, P. A. Beerel, V. Vakilotojar, A. E. Dooply, and J. Arceo. The design and verification of a high-performance low-control-overhead asynchronous differential equation solver. *IEEE Transactions on VLSI Systems*, 6(4):643–655, Dec. 1998.