An Efficient Algorithm for Time Separation of Events in Concurrent Systems

Peggy B. McGee and Steven M. Nowick
Department of Computer Science
Columbia University
New York, NY 10027
Email: {pmcgee,nowick}@cs.columbia.edu

***Abstract*— The time separation of events (TSE) problem is that of finding the maximum and minimum separation between the times of occurrence of two events in a concurrent system. It has applications in the performance analysis, optimization and verification of concurrent digital systems. This paper introduces an efficient polynomial-time algorithm to give exact bounds on TSE's for choice-free concurrent systems, whose operational semantics obey the max-causality rule. A choice-free concurrent system is modeled as a strongly-connected marked graph, where delays on operations are modeled as bounded intervals with unspecified distributions. While previous approaches handle acyclic systems only, or else require graph unfolding until a steady-state behavior is reached, the proposed approach directly identifies and evaluates the asymptotic steady-state behavior of a cyclic system via a graph-theoretical approach. As a result, the method has significantly lower computational complexity than previously-proposed solutions. A prototype CAD tool has been developed to demonstrate the feasibility and efficacy of our method. A set of experiments have been performed on the tool as well as two existing tools, with noticeable improvement on runtime and accuracy for several examples.**

I. Introduction

This paper addresses the problem of finding the maximum and minimum time separation of events (TSE) in concurrent systems.

A concurrent system is considered as a set of interacting processes which communicate through channels. When a process initiates a communication with one or more other processes, it waits for all parties to respond before it proceeds. Such operating semantics is said to obey the *max-causality rule*, or to operate under the *max timing constraint*. This model can be applied to problems in a wide range of domains. A "process" can correspond to the transition of a signal at the circuit level, or to a partition of functional units at the system level.

Several delay models have been used in modelling these systems. A "stochastic model" is often used for the performance analysis of these systems [11], [15]. However, for verification, a "bounded-delay" model, where the computation time of each process is assumed to be bounded below (min) and above (max) by non-negative real numbers, with no distribution specified, is much more useful. In the special case where the upper and lower bounds of the computation time are identical, a "fixed-delay" model is said to be used.

This paper targets concurrent systems under a bounded-delay model. To make the problem amenable for analysis, the system is assumed to be *decision-free*; such systems can be modeled by marked graphs [5], which are commonly used to capture concurrent behavior.

The TSE problem for bounded-delay systems has applications to performance analysis, optimization of verification of concurrent systems. In this case, while it is generally not possible to provide accurate average case performance metrics, since no delay distribution is given, one can usefully predict best case and worst case performance metrics, such as system throughput. For the restricted case of fixed-delay systems, TSE also becomes a performance measure of the system, as in this case the distribution of the delay is trivially known (i.e. exactly one delay value per event in the system).

This work was partially supported by NSF ITR Award No. NSF-CCR-0086036 and by an Initiatives in Science and Engineering (ISE) grant from Columbia University (from the Office of the Executive Vice President for Research).

Timing information for bounded delay systems can be exploited for the optimization of asynchronous circuits, e.g. in lazy transition systems [6], in which absolute knowledge of what event can happen before or after another is used to reduce circuitry. The TSE problem also has an important application in system-level timing verification. Correct operation of a circuit often depends on strict ordering of certain events. Verifying the maximum and minimum delay between two events against a set of constraints specifications ensures correct operation of the system.

There have been several previous approaches proposed for solving the TSE problem for concurrent systems. Early research was limited to systems with finite behavior, i.e., those that can be modeled by *acyclic* graphs [7], [12], [3]. Basic approaches assume a max-causality operating semantics, while some also handles min-causality, where an event is triggered by the earliest arriving pre-condition. More recent research in this area focuses on decision-free systems with infinitely-repeating behavior, i.e., those that can be modeled by cyclic graphs. Chakraborty et al. [4] proposed a heuristic method to give approximate bounds for cyclic systems operating under both max- and min-causality rules. Hulgaard et al. [8] gave exact bounds on TSE's for cyclic systems operating under the max-causality rule. In [9], Hulgaard extended his approach to handle also systems with linear timing constraints.

Like [4], [8] and [9], the approach proposed in this paper applies to decision-free cyclic systems, and like [8], it gives exact bounds for systems operating under the max-causality rule. However, there is an important difference in the methods: while previous approaches are based on *graph unfolding*, which performs a symbolic execution of the system until the desired TSE number finally converges, the proposed approach *entirely avoids graph unfolding*. For unfolding-based methods, convergence to steady-state behavior can be a significant bottleneck: (i) in cases, the ramp-up from an initial configuration to steady-state may require large numbers of unfoldings; and (ii) the system may not settle into a single fixed sequence of events, but rather reach a stable periodic behavior, presenting difficulties both in the identification of, and convergence to, steady-state behavior.

The key contributions of this paper are as follows. First, a novel approach is proposed for solving the TSE problem for cyclic systems. Unlike previous methods, the steady-state behavior of the system is directly identified and used to solve the TSE problem, hence no graph unfolding is required. Second, an integrated solution framework for the TSE problem is proposed, in which solutions to general cases of the problem are built upon those to the more restricted cases: the approach handles both fixed-delay systems as well as the general case of bounded-delay systems. Two classes of the problem, namely "single-token systems" and "multi-token systems", are highlighted (definitions of these terms will be provided in Section IV of this paper). Efficient polynomial-time algorithms are presented. Finally, a prototype CAD tool has been developed to demonstrate the feasible and efficacy of our method.

There are several advantages of the proposed approach over existing work, beyond the avoidance of graph unfolding. Compared to previous exact methods [8], our approach has low computational complexity, while the previous approach has no clear complexity

bound. Our approach is also significantly more efficient *multiple TSE queries* on the same graph: basic analysis of the system is performed only once, and then TSE's between different pairs of events can be obtained with little additional cost, whereas [8] performs the analysis (and therefore the unfolding) from scratch for each TSE query on the same graph. Compared to previous heuristic methods [4], the proposed approach can handle a larger class of designs. In particular, multi-token systems *cannot be directly handled* by [4], pre-processing must first be performed to transform them into single-token systems. In addition, unlike the proposed exact method, the method in [4] can produce inaccurate results, both as a inherent consequence of a heuristic method and because the initial configuration of the system is not taken into account.

A set of benchmarks has been run on the prototype tool, as well as on those from [4] and [8]. Our tool shows significant improvement in runtime over a previous exact method [8] in cases where the latter requires a large amount of graph unfolding. Compared to the heuristic tool from [4], the proposed approach provides more accurate results in some cases. It is also shown that tool from [4] cannot directly handle multi-token benchmarks.

The remainder of this paper is organized as follows: Section II gives background on marked graphs, which is used as a modeling formalism in this paper. Section III gives an informal overview of the TSE problem, including examples which demonstrate some challenges of the problem. Section IV presents the theoretical framework of our approach. Sections V, VI and VII shows the algorithms for single-token fixed-delay systems, multi-token fixed-delay systems and the general case of bounded-delay systems, respectively. Section VIII describes some experimental results, and finally, Section IX concludes the paper.

II. Background

This section reviews technical background on modeling necessary for the development of this paper.

Marked graphs, a subclass of Petri nets for modeling concurrent systems without choice, are used for modeling concurrent systems in this paper [5]. Other models have been used in related papers, such as process graphs in [8], event-rule systems in [14] and timing constraint graphs in [9]. Marked graphs are behaviorally equivalent to these models for modeling max-causality systems, meaning that a system modeled by these other models can be translated to a marked graph without losing any information. Though less succinct than these other models, marked graphs have the advantage that a large body of proven results exists, upon which new theories can be built.

In the remainder of this section, the structural components and semantics of basic marked graphs are first reviewed in Section II-A, followed by an extension to *timed* marked graphs in II-B. Section II-C shows an example of a marked graph, and Section II-D reviews some of their useful properties.

A. Marked graph basics

A marked graph (G, M_0) is a directed graph $G = (N, E)$ where N is a set of *nodes* and E a set of *directed edges*, and an initial marking M_0, which is an assignment of zero or one token on each edge $(u, v) \in E(G)$. In Petri net formalism, a node in a marked graph can also be referred as a *transition*, and an edge a *place*. In this paper, the convention of [5] is followed, and the structural components of a marked graph are called nodes and edges.

A node in a marked graph is enabled to fire if there is a token in each of its input edges. When a node fires, it removes one token from each of its input edges and deposits one token into each of its output edges. In a discrete-event system, the firing of a node corresponds to the occurrence of an event. A graph G, together with an initial assignment of tokens M_0, (G, M_0), defines the set of reachable markings of the system. Note that the firing semantics of a marked graph corresponds the the so-called *max-causality* system [12], where the arrival of the last token to an input edge of an event determines when it is fired. A marked graph is *live* if there always exists a firing sequence to fire each node in the graph. A marked graph is *bounded* if there exists an upper bound on the number of tokens that can reside on an edge simultaneously. In this paper, only marked graphs which are live and bounded are considered.

B. Timed marked graphs

A *timed* marked graph $(G, M_0, \delta, \lambda)$ is a marked graph with a vector δ, which is an assignment of a delay variable $\delta(u, v)$ to each edge $(u, v) \in E(G)$ and λ, which is a vector of real numbers associated with each token in the initial marking M_0, and describes timing information which is relevant only in the startup operation of the marked graph. δ and λ are also called *holding time* and *lag time*, respectively. In more detail, when a token arrives at edge (u, v) with holding time $\delta(u, v)$ from node u, it must reside there for $\delta(u, v)$ time units before it contributes the enabling of the output node v. Informally, the holding time can be interpreted as the time it takes for a token to "travel" from the input node to the output node of an edge. When all the holding time of all its input tokens expire, a node must fire immediately. More formally, the firing rule of a timed marked graph can be described as follows:

$$t(v) = \max_{u \in pred(v)} (t(u) + \delta(u, v)) \quad (1)$$

where each u is a distinct predecessor node to v, and $t(u)$ and $t(v)$ the firing times of nodes u and v, respectively. A token with lag time λ_j contributes to the enabling of the output node of the edge it resides on λ_j time units after the system begins running at time 0. In our model, $\delta(u, v)$ for each edge (u, v) is defined by a range $[d(u, v), D(u, v)]$, where $d(u, v)$ and $D(u, v)$ are non-negative real number and $d \leq D$. The exact distribution of $\delta(u, v)$ is unspecified. A system in which $d(u, v) = D(u, v)$ for all edges is a called a *fixed-delay system*.

C. A marked graph example

The concepts and definitions from Sections II-A and II-B are now illustrated with an example. Figure 1 shows an example of a marked graph. The interval labelled $[d, D]$ on each edge indicates its bounded holding time, i.e., the min and max times it takes for a token to travel from the input node to the output node of the edge. Suppose the initial lag time is zero on all tokens in the current marking of the graph; then nodes b and d are immediately enabled to fire, since there is a token in each of their input edges. Node c is only enabled to fire after both nodes b and d have fired, and after a delay which is equal to the maximum of 1 to 2 time units after b fires, *and* 1 to 3 time units after d fires.

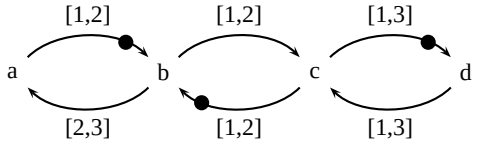


Fig. 1. Example Marked Graph

D. Properties of marked graphs

Two well-known properties regarding marked graphs, which will be used in the formulation of the TSE solution in this paper, are shown in Theorems 1 and 2. Proofs of these theorems can be found in [13].

Theorem 1: For a marked graph, the token count in any simple cycle is invariant under any firing.

Theorem 2: For a connected marked graph, a firing sequence leads back to the initial marking *iff* every node in the marked graph has fired an equal number of times.

III. Overview

An overview of the TSE problem is presented in this section. A formal description of the problem is first given in Section III-A, followed by a basic example in Section III-B. Section III-C then presents a set of examples which highlight some challenges of the problem, and gives insights on how they are tackled, which will be formalized in Section IV, and presented as an algorithm in Sections V through VII later in the paper.

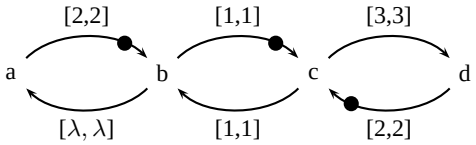
A. Problem formulation

The *time-separation of events* (TSE) problem can be formally described as follows.

Definition 1: **TSE problem:** Given a timed marked graph $(G(N,E), M_0, \delta, \lambda)$, find the minimum and maximum difference between the times of occurrence of any pair of events in the system, under all possible legal firing sequences, and assuming each edge can take on an arbitrary delay within its specified range every time it receives a token.

B. Basic example

Figure 2 shows an simple example of a timed marked graph with fixed delays. Its steady-state behavior shows a fixed pattern in which every node fire at a fixed rate.



time	0	1	2	3	4	5	6	7	8	9	10	....
event	c	b	a	d	*	c	b	a	d	*	c	....

Fig. 2. Example: fixed-delay system exhibiting simple behavior

Suppose the initial lag time is $\lambda = 1$ on edge (b, a). A simulation result of the system is shown underneath the figure. A * symbol in a given time slot indicates that no event takes place. To see how part of the simulation result is derived, consider the following. Suppose that in the initial marking shown in the figure, nodes c and b are assigned lag times 0 and 1, respectively. After nodes c and b fires, nodes a and d can fire after the delay specified on the corresponding edges. However, after node a fires at time $t = 2$, and a token arrives at node b via on edge (a, b) at time $t = 4$, b cannot immediately fire, because it must also wait for the arrival of a token from node c, which in turn must wait for the arrival of a token from node d.

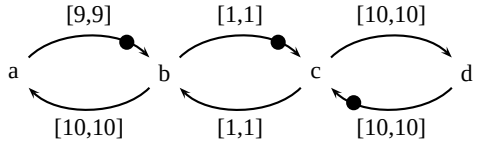
A few observations can be made on the asymptotic behavior of the simulation result. First, with the chosen initial configuration, the firing sequence $...cbad...$ repeats every 5 time units. In fact, though not shown in this simulation run, no matter what initial condition this example system starts with, it always settles to the same firing pattern. Second, the firing time of each transition of the system is constrained by the cycle with the longest delay, i.e., the critical cycle, which has a delay 5 (cycle $C(cd)$). Tokens arriving on edges bc and ab must always wait for 3 and 2 time units, respectively, before their output nodes fires. The amount of waiting time is called *local slack*. Formal definitions of critical cycle and local slack will be given in Section IV.

C. Challenges

This section presents three examples to highlight some challenges of the TSE problem, namely, the slow convergence to a steady-state behavior, the sensitivity of the steady-state behavior to the initial condition, and the exhibition of an oscillating, periodic behavior at the steady state. In each case, intuition is give on how these challenges are dealt with in the proposed formal solutions to the problem.

1) Fixed-delay systems: slow convergence: The example in Figure 3 highlights a challenge for TSE methods based on unfolding. The model of the system has the same structure as the one in Figure 2, except that delay values are different. As shown by the simulation result, it takes a much longer time for the firing sequence to finally converge, i.e., to reach asymptotic steady state. With the unfolding method, this entire simulation with slow convergence must be completed with a large amount of overhead. The proposed method, which uses knowledge of the asymptotic behavior the system to derive the solution, does not suffer from this problem.

2) Fixed-delay systems: sensitivity to initial condition: The example in this section shows how the asymptotic behavior of a system can be sensitive to its initial timing configuration i.e., the lag time of tokens in the initial marking (which in turn determines



time	0	10	20	28	40	46	60	64	...	82	100	101	120...
event	c	b	c	b	c	b	c	b	...	b	c	b	c...

Fig. 3. Example: fixed-delay system with slow convergence time

the initial firing order of nodes). It highlights the importance of any solution to the TSE problem to take into account the initial timing configuration of the system. Sensitivity to the initial condition arises when there is more than one critical cycle in the system. Suppose we assign a value of 3 for λ on edge (b, a) in the example in Figure 2. Both cycles $C(a, b)$ and $C(c, d)$ now have a cycle time of 5. Two possible runs for the system are shown below. As can be seen from two simulations runs, the system settles into two different timing behaviors, depending on whether node b or node c fires first.

Run 1:

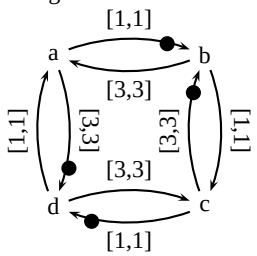
time	0	1	2	3	4	5	6	7	8	9	10	....
event	c	b	*	d	a	c	b	*	d	a	c	....

Run 2:

time	0	1	2	3	4	5	6	7	8	9	10	....
event	b	c	*	a	d	b	c	*	a	d	b	....

In contrast, in the example shown in Figure 2, which has only one critical cycle, the asymptotic behavior is independent of the initial configuration.

3) Fixed-delay systems: periodic behavior: A final challenge of the TSE problem is that a system may not converge to a single steady state wherein every node fire at the same rate, but rather settles into a *periodic pattern*. In unfolding-based methods, not only does the amount of unfolding increases in such cases, it is difficult to determine a terminating criteria for the unfolding process [8].



Run 1:

time	0	1	2	3	4	5	6	7	8	9	10	11	12	....
event	b,d	*	*	a,c	*	*	b,d	*	*	a,c	*	*	b,d	....

Run 2:

time	0	1	2	3	4	5	6	7	8	9	10	11	12	....
event	b	*	d	a	*	c	d	*	b	c	*	a	b	....

Fig. 4. Marked graph exhibiting periodic behavior

In the first run, the system settles into a steady-state behavior where the same firing sequence repeats every 6 time units. However, in second run, where the initial condition is different (d firing 2 time units after b), the system settles into a periodic behavior, in which the same node is visited alternately every 4 or 8 time units. The mean cycle time of the system is still the same as the first firing sequence. However, it takes a total of 12 time units before the the same firing sequence to repeat itself.

IV. Theoretical foundations

This section presents the theoretical foundations for the proposed solution of the TSE problem.

The focus is on properties for a restricted class of designs, namely those that can be modeled by marked graphs in which there is *exactly one token* in each simple cycle, i.e., *single-token systems*, with a fixed-delay timing model, are discussed. Examples of these systems can be found in Figures 2 and 3 in Section III. In later sections, it will be shown how these properties can be readily extended to fixed delay systems with multiple tokens in simple cycles, i.e.,

multi-token systems (Section VI), and finally to the general case of bounded delay systems (Section VII). In particular, solutions to multi-token fixed delay systems can be formulated as the superposition of several instances of solutions to a single-token fixed delay problem. Solutions for bounded delay systems can be formulated as two distinct instances of the fixed delay problem. Section IV-A first presents some definitions of terms. Section IV-B shows the behavior of a single-token fixed-delay system always converges to a steady state. Section IV-C shows that how the steady-state behavior of these systems can then be derived directly from the timed marked graph.

A. Definitions

Four definitions useful for the discussion in the rest of this section are now presented.

Definition 2: **Timed firing sequence:** A timed firing sequence of a timed marked graph is a mapping of events (firing of nodes) to real numbers (firing time).

Definition 3: **Steady state:** The behavior of a timed marked graph is said to have reached steady-state if its timed firing sequence exhibits a repeating pattern, i.e., each event of the same type repeats at a fixed time interval, or at intervals that repeats every k time units, where k is a fixed number.

Definition 4: **Local slack:** The local slack of an edge (u, v) in a timed marked graph is the duration of time between the expiration of the holding time of a token arriving at (u, v) from node u, to the removal of the token from (u, v) by the firing of node v.

Figure 5 shows a graphical depiction of the definition of local slack.

Definition 5: **Critical cycle:** A critical cycle $\mathcal{C}$ of a timed marked graph $(G, M_0, \delta, \lambda)$ is the set of all edges and nodes in a simple cycle with the maximum cycle mean, which is defined as:

$$\tilde{T} = \max_k \frac{\sum_{(u,v)\in C_k} d(u, v)}{N_k}$$

where $\sum d(u, v)$ is the sum of delays along all edges in a simple cycle C_k, and N_k is number of tokens on the cycle.

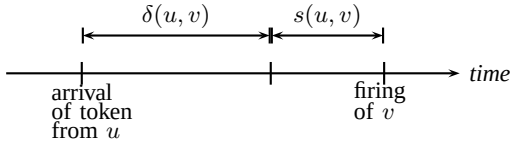


Fig. 5. Local slack

B. System behavior always converges

This section presents the first important property for single-token fixed-delay systems: their behavior always converges to a steady state.

Theorem 3: **Steady state:** Regardless of the initial condition, a system modeled by a single-token timed marked graph converges to a steady-state behavior in which all nodes fire at a repeated timed firing sequence, with cycle time equal to that of the critical cycle(s). In addition, given a particular initial condition, the steady-state behavior is unique.

The proof for Theorem 3 is broken into three parts: Lemma 1 shows that regardless of the initial condition, the system naturally converges to a behavior in which there is *no slack* on the critical cycle. Lemma 2 shows that the system also converges to a state in which all nodes fire at the same rate, i.e. with the same cycle time. Hence, effectively, the entire system converges to a fixed steady-state behavior, where the critical cycle precisely defines the operating rate of every node in the system. Finally, Lemma 3 shows that once a system enters a steady-state, its behavior remains the same, i.e., all future behavior of the system conforms to a repeated pattern.

Lemma 1: **No slack on critical cycle:** Regardless of initial configuration, slack on a critical cycle of a single-token fixed-delay system converges to zero asymptotically.

Proof: Suppose that all nodes in the the timed marked graph has fired at least twice already, i.e., all nodes obey the relationship

$$t(v) = \max_{u\in pred(v)} (t(u) + \delta(u, v))$$

with respect to their predecessors, then starting from any arbitrary node in the firing sequence, one can trace back a path consisting only of edges with no slack. Since there are only finite number of nodes in the sequence, eventually this path forms a loop. We claim that this loop is a critical cycle in the graph. The similarity of the firing rule to Bellman's equation for solving the shortest path problem [10] should remind the reader that the action of "unfolding" of the graph for a couple iterations is essentially carrying out the steps to find the longest path of an acyclic graph, i.e., the critical cycle of the corresponding cyclic graph. □

Lemma 2: **All nodes fire with same cycle time:** Regardless of initial configuration, the system converges to a state where all nodes fire at the same cycle time asymptotically.

Proof: Let nodes u and v be in the same simple cycle in G. Since there is only one token circulating in the simple cycle, v can only fire as many time as u asymptotically, i.e., they have the same firing rates. Since G is strongly-connected, all nodes are in the same simple cycle as some other node. Therefore they all have the same asymptotic firing rate. □.

Lemma 3: **Convergence:** Once a single-token system enters the steady-state (two consecutive identical timed firing sequences), it remains in the steady state.

Proof: Recall from Theorems 1 and 2 in Section II that the token count in any simple cycle in a marked graph is invariant, and that a marking can only return to itself if and only if all nodes in the graph has fired an equal number of times. In addition, it can easily be shown that for a single-token marked graph, a marking returns to itself if and only if all nodes in the graph has fired exactly once.

Suppose the system has gone through two iterations, $i-2$ and $i-1$ of identical timed firing sequences in which all nodes fire at exactly the same cycle time T, we show that all future firing of the nodes follow exactly the same time sequence with the same cycle time.

Consider node v at the beginning of the timed firing sequence of iteration $i-1$. The next time it fires (in iteration i) is $\max_{u\in pred(v)} (t(u)+ \delta(u, v))$ where $t(u)$ is the firing time of all predecessors nodes u of v in iteration $i-1$. Since the distances of all predecessors is the same as in iteration $i-2$, and $\delta(u, v)$ is constant, the next time v fire is exactly T. It can easily be seen the same applies to every other node in the timed firing sequence. □.

The proof for Theorem 3 follows directly from the results of Lemmas 1 to 3. □.

C. Deriving steady-state behavior

Having shown that the behavior of a single-token fixed-delay system always converges to a steady state, it is now shown that the steady-state behavior is deterministic, and can be quantitatively derived directly from the given timed marked graph. Before presenting this result formally in Theorem 4, an intuitive overview is first given.

The steady-state behavior can be represented as the firing time of every node in the graph relative to a single "reference node" on the critical cycle. The time difference between the firing of the reference node and the firing of any node in the graph can be directly derived.

In particular, first consider a naive solution to the problem: simply to compute the *longest simple path* in the graph from the reference node to the given node. This path is the critical time difference between firing the reference and the given node. The idea is that, when the reference node fires, tokens are "emitted" on all outgoing edges. The longest simple path from the reference node to the given node represents the longest walk, and is therefore the time separation between the reference node and the given node. Once the firing time of each node, compared to the reference node, is computed, any steady-state TSE between two arbitrary nodes is easily obtained by

subtracting the relative firing times (with respect to the reference node) of one node from another.

However, this naive approach does not always work. In particular, at any snapshot of the system, the fundamental challenge is that there may be *multiple tokens* lying on a simple path from the reference node to the given node. These other tokens are generated by *earlier instances* of firings of the reference node. Hence, it is inaccurate to treat the longest "path distance" from reference to given node as their critical time separation. In fact, the given node may be enabled by a mix of tokens, on its different input edges, generated from *different instances* of the firing of the reference node. A long path with several tokens on it will fire much earlier: when the lead token enables the given node. These earlier-fired tokens must be taken into account for a correct solution.

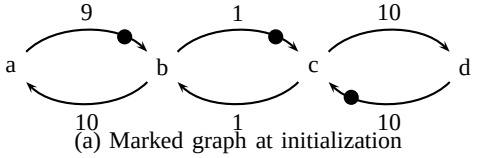
To track the tokens from different iterations, a new representation of the system in steady state, called the steady-state graph, is introduced. Intuitively, the steady-state graph captures the enabling conditions (tokens) from different iterations of firing by the weights of edges on the graph. It tracks how a given node is enabled by a mix of tokens from different adjacent iterations of the system.

A formal definition and a graphical example of a steady-state graph is shown in Definition 6 and Figure 6, respectively. For each edge with a token in the initial configuration of the original marked graph, a quanitity T (the critical cycle time of the system) is subtracted from the holding time of the edge ($\delta(u, v)$), representing that the enabling condition is caused be a previous iteration of firing. By capturing the enabling conditions of the marked graph in the steady state in the edge weights instead of in tokens, the marked graph is simplified to a cyclic directed graph.

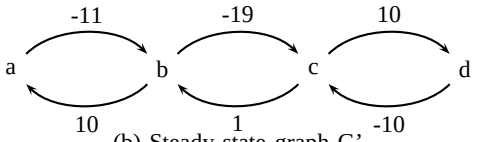
Definition 6: **steady-state graph:** Given a fixed-delay timed marked graph $(G, M_0, \delta, \lambda)$ and the cycle time T of its critical cycle, its steady-state graph G' has the same nodes and edges as G. The weight of each edge $e(G') \in E(G')$ is given by:

$$\begin{aligned} \delta(e(G')) &= \delta(e(G)) - T \text{ if there is a token on } e(G) \text{ in } M_0 \\ &= \delta(e(G)) \qquad \text{otherwise} \end{aligned}$$

An example of a timed marked graph and its steady state graph G' is shown in Figure 6. Three edges in this figure have different weights from those in the original marked graph: ab, bc, and dc. In each case, the critical cycle of 20 has been subtracted off to account for the initial token placement, which effectively "shortens" any path length calculation involving these edges.



(a) Marked graph at initialization



(b) Steady-state graph G'

Fig. 6. A marked graph and its steady-state graph

We now present the main technical result of this section.

Theorem 4: Given a single-token fixed-delay timed-marked graph with a single critical cycle, the firing time of every node relative to a source node on a critical cycle in the steady state is equivalent to the longest path between them in the steady-state graph G'.

□

In the above presentation, the problem of deriving the steady-state behavior of a single-token fixed-delay system has been reduced to a single-source longest path problem for cyclic graphs. There exists many efficient, polynomial-time algorithms for this problem. In particular, the signs of all edges in G' can be inverted to obtain a graph G'', such that a longest path problem in G' can be treated as a shortest-path problem in G''. Details will be provided in Section V.

For cases where is more than one critical cycle, each critical cycle can span a longest-path tree, i.e., a time firing sequence. In cases where one critical cycle is triggered to fire at initialization before others, the asymptotic behavior of the system is determined by the timed firing sequence of that cycle. In cases when more than one critical cycle is triggered to fire at the same time, the timed firing sequence of the entire system is the result of a superposition of the individual firing sequences for each critical cycle under the "max" operator. More formally, Let $\mathbf{x_i}$ be a vector of the timed firing sequence for each critical cycle C_i in G, then the

$$\mathbf{x} = \max(\mathbf{x_i}) \tag{2}$$

Intuitively, the timed firing sequence can be understood as the "clock tree" of the system, initiated by a reference node on the critical cycle and spanning out to all nodes in the graph. A key property of this clock tree is that there no local slack on any of its paths.

V. SINGLE-TOKEN FIXED-DELAY SYSTEMS

This section presents the algorithm for finding the TSE of any two pair of events in a fixed-delay system described by single-token marked graph. The TSE algorithm is divided into three major steps: (i) finding the critical cycle(s); (ii) finding the longest path tree from each critical cycle and combining the results; and (iii) finding the TSE.

Step 1: Critical cycles: This step takes as an input a timed marked graph with a fixed delay assignment, and outputs the cycle time $\tilde{T}$ of the graph, as well as the set of all critical cycles.

This is a similar problem to finding the cycle mean of a digraph, for which many efficient algorithms exists. A method similar to that for solving the minimum cost-to-time ratio cycle problem based on binary search presented by Lawler [10] is summarized below.

A cycle time for the system, τ, halfway between an interval user-supplied interval $[a, b]$ is first used as a "guess", and a corresponding steady-state graph G' for the system constructed. The sign of the weight of all edges on G' is then flipped to transform it to a graph G'', so that the longest-path problem can be conveniently transformed to a shortest-path problem, for which efficient solutions exists. The Floyd-Warshall algorithm, with the distances between all pairs of nodes initialized to ∞, is used for computing the shortest path between all pairs of nodes in G''. The result is presented in a square matrix. In particular, the cycle time of the longest path between any node to itself, i.e., the longest self-loop, can be read off as the diagonal entries of the matrix.

There are three scenarios:

Case 1: τ is too small: in this case all diagonal entries of the matrix are smaller than 0. *Case 2:* τ is too big: in this case all diagonal entries of the matrix are larger than 0. *Case 3:* τ is equal to the cycle time: in this case one or more diagonal entries of the matrix is equal to 0.

In case 1 and 2, a new "guess" value τ is refined by halving the interval $[a, b]$ and picking a new value from the appropriate half, and the Floyd-Warshall algorithm is applied again using the new τ value. In case 3, the critical cycle value is found and Step 2 of the procedure can be proceeded.

The complexity of the Floyd-Warshall algorithm used in each iteration of the "guess" is $O(n^3)$, where n is the number of nodes in the graph. As shown in [10], the complexity of the entire algorithm is $O(n^3 log n)$.

Step 2: Compute timed firing sequence: This step takes in result from Step 1 of the algorithm, as well as the initial configuration (λ), and returns a vector of the timed firing sequence of the nodes in the marked graph. As a side-product of the Floyd-Warshall algorithm used in Step 1, one can read off the shortest distance from any node to any other node in the graph G''. The initial configuration (lag time and initial token placing) can be used to determine which critical cycle fires first. If the system has only one critical cycle, an arbitrary

Step 1: $Find\_critical\_cycles(G, M_0, \delta), a, b$
1 do
2 $\tau \leftarrow (a+b)/2$
3 Construct graph G''
4 Use Floyd-Warshall to find all pairs shortest paths for G''
5 if all diagonal entries of shortest-path matrix < 0
6 $\tau \leftarrow (a+(a+b)/2)/2$
7 if all diagonal entries of shortest-path matrix > 0
8 $\tau \leftarrow ((a+b)/2+b)/2$
9 while (all diagonal entries of shortest-path matrix $\neq 0$)
10 $C \leftarrow$ all loops with 0 diagonal entries

Step 2: $Compute\_firing\_sequence(\lambda)$
11 foreach critical cycle $C_i \in C$ with earliest firing time
12 construct longest-path tree $\mathbf{x_i}$
13 $\mathbf{x} \leftarrow \mathbf{max}(\mathbf{x_i})$

Step 3: $Find|_T SE(u, v)$
14 return $\mathbf{x(v)} - \mathbf{x(u)}$

TABLE I
PSEUDOCODE FOR TSE FOR SINGLE-TOKEN FIXED-DELAY SYSTEM

reference node from this cycle is picked as a source node. Its shortest distance to every other node in the graph is then read off the matrix and represented in a vector. The signs of the values in the vector are then reverted to form the longest path timed firing sequence. If there is more than one critical cycle, and one fires earlier that all others, then the reference node is picked from the earliest-firing critical cycle. Finally, if more than one critical cycle fires at the same time, their corresponding timed firing sequences are combined using the *max* operator, as shown in Equation 2 in Section IV.

Step 3: Finding TSE's: The result of Step 2 is a timed firing sequence, i.e., a vector $\mathbf{x}$ of the firing time of all nodes in G relative to a reference node. Given two input events u and v, their TSE can simply be computed from the timed firing sequence by subtracting the relative firing time of one from the other:

$$TSE(u, v) = \mathbf{x}(v) - \mathbf{x}(u)$$

The pseudocode of the algorithm is shown in Table I. The complexity for the algorithm to solve the single-token fixed-delay TSE problem is dominated by the step for finding the critical cycle, which is $O(n^3 logn)$.

VI. MULTI-TOKEN FIXED-DELAY SYSTEMS

In this section, we show how the solution framework can be extended to handle the general case of a system which is modeled by a marked graph with more than one token in a simple cycle.

A. Theoretical foundations

Section IV presented a theoretical framework for single-token fixed-delay systems. A key theoretical result for multi-token fixed-delay systems is now presented.

Theorem 5: The steady-state behavior of a multi-token system is the overlapping of K timed firing sequences of single-token systems, where K is the number of tokens on the critical cycle(s) of the graph.

Instead of a formal proof, intuition on how this works is given via an example.

Example: Recall the example in Figure 4 in Section III-C. The critical cycle time is 6 for cycle $abcd$. Assuming a source node at b, the timed firing sequence given by the longest path tree is $\mathbf{x_1} = [3\ 0\ 9\ 6]$. Run 1 is two of these timed sequences superimposed on each other, with a displacement of -3 units for the second sequence: $\mathbf{x} = [3\ 0\ 9\ 6] \| (-6) + [3\ 0\ 9\ 6]$, where the parallel operator $\|$ denotes the superposition of two firing sequences. The single firing sequence shown in Figure 4 can thus be considered a superposistion of the two independent timed firing sequences shown below.

Run 1:

time	0	1	2	3	4	5	6	7	8	9	10	....
sequence 1	b	*	*	a	*	*	d	*	*	c	*	....
sequence 2	d	*	*	c	*	*	b	*	*	a	*	....

Run 2 in Figure 4 is similarly composed of two timed firing sequences, with a displacement of 2 for the second sequence: $\mathbf{x} = [3\ 0\ 6\ 9] \| 2 + [3\ 0\ 6\ 9]$.

Step 1: $Find\_critical\_cycles(G, M_0, \delta), a, b$
1 use same algorithm as in single-token systems

Step 2: $Compute\_firing\_sequence(\lambda)$
2 $T_2 \leftarrow$ 2nd largest cycle time in G' from shortest-path matrix
3 foreach critical cycle $C_i \in C$ with earliest firing time
4 construct longest-path tree $\mathbf{x_i}$
5 $c_0 \leftarrow 0$
6 foreach token t_k in initial marking M_0
7 $c_k \leftarrow \sum(\lambda + d)$ for all edges between t_k and t_{k-1}
8 if $c_k < T_2$
9 $c_k \leftarrow T_2$; $c_k - 1 \leftarrow -(T_2 - c_k)$
10 $\mathbf{x} \leftarrow x_1 \| c_1 + \mathbf{x_1} \| ... \| c_k + \mathbf{x_1}$

Step 3: $Find|_T SE(u, v)$
11 Min_TSE $\leftarrow \min(\mathbf{x}(v) - \mathbf{x}(u))$ for all adjacent pairs of $u, v \in \mathbf{x}$
12 Max_TSE $\leftarrow \max(\mathbf{x}(v) - \mathbf{x}(u))$ for all adjacent pairs of $u, v \in \mathbf{x}$

TABLE II
PSEUDOCODE FOR TSE FOR MULTI-TOKEN FIXED-DELAY SYSTEM

Two sequences run independently of each other as long as the following rule is observed: The minimum displacement between any two timed firing sequences must be greater than or equal to the second largest critical cycle in the graph. This rule ensures that the firing sequences are spaced far apart enough such that all tokens other than the critical one arrives in time at a node before the critical token arrives. A proof similar to that for Lemma 1 can be used to show that if this rule is violated during the initial configuration, the system asymptotically converges to a state such that the rule is observed. If the rule is not violated in the initial configuration, the spacing of tokens at initialization remains constant during the entire operation of the system.

B. Algorithm

The algorithm for multi-token fixed-delay systems is now presented. Recall from Section VI-A that the behavior of a multi-token system can be interpreted as more than one single-token system superimposed on each other by overlapping their timed firing sequences by a displacement. The algorithm can be seen as an extension of the algorithm for single-token system presented in Section V. The pseudocode of the algorithm is shown in Table I.

VII. BOUNDED-DELAY SYSTEMS

Now that an algorithm the TSE for fixed-delay systems is presented, it is shown in this section how it is used as a foundation for the TSE algorithm for bounded-delay systems. In particular, it is shown how finding the TSE of bounded-delay systems can be considered solving two instances of a fixed-delay problem. The algorithm is made up of two passes: in each pass, the system is considered a fixed delay system with delay values on each edge carefully assigned so as to generate extreme TSE values.

A. Overview

Before getting into further details of the algorithm, an example is shown to give some intuitions on the method. Consider the example shown in Figure 7.

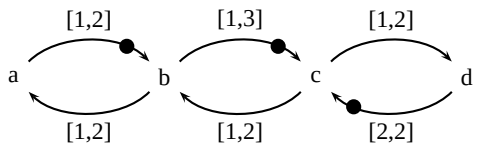


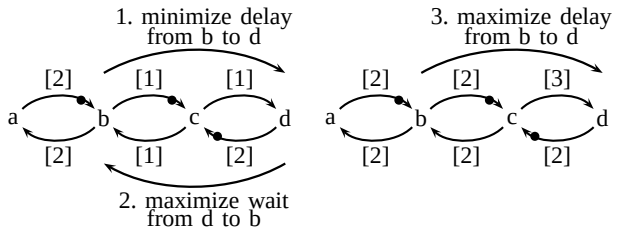
Fig. 7. Bounded-delay systems

Suppose we want to find the maximum separation between two consecutive firings of node d, i.e., maximizing Δdd. Intuitively, we want the first firing of d to happen as *early* as possible with respect to some reference event, and the second firing of d to happen as *late* as possible after. An example of this is shown in the simulation below.

$time$	0	1	2	3	4	5	6	7	8	9	....
$event$	b	c	d,a	*	b	*	a	c	*	d	....

To see how the simulation run is derived, consider the following. Suppose b fires at time 0. To make the first firing of d as early as possible, we use the minimum delay values of 1 on edges (b, c) and (c, d) to enable d to fire at time $t = 2$. To make the second firing of d as late as possible, we use maximum delay values on all other edges to make d fire at time $t = 9$. This results in a maximum separation of $\Delta dd = 7$.

Figure 8 shows how the simulation can be thought of as two separate runs of a *fixed-delay* system, each with a different delay assignment.



(a) Run 1 (b) Run 2

Fig. 8. Finding maximum separation of events

Suppose now we want to solve the opposite problem of finding the minimum separation between two consecutive firings of d. Using similar reasoning, we would want the first firing of d to happen as late as possible with respect to b, and the second firing to happen as early as possible after. The simulation run is shown below. This results in a minimum separation of $\delta dd = 3$.

$time$	0	1	2	3	4	5	6	7	8	9	10	11	....
$event$	b	a	*	c	b	d	*	c	d	*	....		

The key observation here is that the problem of finding the maximum and minimum TSE in a bounded-delay system can be treated as two separate runs of finding the TSE for a fixed-delay system. In the first run, delay values are chosen so as to make the source event happen as early as possible with respect to a reference event, and in the second, the target event to happen as late as possible. A second observations is that the reference point should be picked from a point on the critical cycle, which has the convenient property that its delay does not change between the two runs.

B. Theoretical foundations

The above observations are now formalized. Lemmas 4 and 5 show how delays on a path to a node on the critical cycle and a node from the critical cycle can be maximized. Theorem 6 then shows how the results of these two lemmas can be used to justify a procedure for finding the maximum and minimum TSE of two events in a bounded-delay system.

Lemma 4: For any pair of nodes u not on the critical cycle $\mathcal{C}$ i and $v \in \mathcal{C}$, the maximum TSE between u and v, $\Delta(u, v)$ is achieved if the delay value $\delta(i, j)$ on every edge $(i, j) \in \mathcal{C}$ is equal to its maximum value of $D(i, j)$, and $\delta(i, j)$ on every edge (i, j) on every cycle $C \neq \mathcal{C}$ that intersects p_{uv} equal to its minimum value of $d(i, j)$.

Proof: Suppose by contradiction, there exists an edge $(i, j) \in p_{uv}$ such that

$$d(i, j) + s_{max}(i, j) < D(i, j) \tag{3}$$

Let the cycle mean of the critical cycle $\mathcal{C}$ be $\mathcal{T}$, and the summation of minimum delays values on all edges in the simple cycle C which (i, j) belongs to be $\sum d$. Then the maximum slack the edge (i, j) can attain is given by $s_{max}(i, j) = \mathcal{T} - \sum_i d$. Putting the expression for $s_{max}(i, j)$ into Equation 3 gives

$$\begin{aligned} d(i, j) + \mathcal{T} - \sum d &< D(i, j) \\ D(i, j) - d(i, j) + \sum d &> \mathcal{T} \end{aligned}$$

This is a contradiction, because this would imply that the simple cycle (i, j) is on a cycle with cycle time larger than $\mathcal{T}$. □

Lemma 5: For any pair of nodes u not on the critical cycle $\mathcal{C}$ and $v \in \mathcal{C}$, the maximum TSE between v and u, $\Delta(v, u)$ is achieved if the delay value $\delta(i, j)$ on every edge (i, j) on every cycle $\mathcal{C}$ that intersects p_{uv} is equal to its maximum value of $D(i, j)$.

Proof: From Theorem 4, the firing time of node u relative to node v is equal to the longest path between them in the steady-state graph G'. Therefore the maximum value of $\Delta(v, u)$ is achieved if every edge $(i, j) \in p_{vu}$ attains its maximum delay value $D(i, j)$. □

Theorem 6: The maximum TSE between two events i, j, $\Delta(i, j)$, is equal to $\Delta(i, k) + \Delta(k, j)$, where k is a node on a critical cycle $\mathcal{C}$. $\Delta(i, k)$ is found by setting $\delta(u, v)$ to its maximum value of $D(u, v)$ for every edge $(u, v) \in \mathcal{C}$, and $\delta(i, j)$ to its minimum value of $d(i, j)$ on every edge $(i, j) \notin \mathcal{C}$. $\Delta(k, v)$ is found by setting $\delta(i, j)$ to $D(i, j)$ for every edge $(u, v) \in G$.

Proof: It follows from Lemmas 1 and 2 that for any two events i, k, where $i \notin \mathcal{C}$ and $v \in \mathcal{C}$, the maximum TSE between i, k $\Delta(i, k) = \sum_{p_{i,i+1\ldots,k}} (d(i, i + 1) + s(i, i + 1))$, and for any two events k, j, where $k \in \mathcal{C}$ and $j \notin \mathcal{C}$, the maximum TSE between k, j $\Delta(k, j) = \sum_{p_{k,k+1\ldots,j}} D(k, k + 1)$, □

C. Algorithm

An algorithm for finding the TSE between two events in a bounded-delay system is now presented.

The algorithm consists of two passes: In the first pass, the time separation between i and a reference event k is found, and in the second, the time separation between k and j. In each pass, a preprocessing step is used to pick a delay value from the range of possible values on each edge, and a single value is assigned, thus effectively turning the system in a fixed-delay one. The algorithm for finding the TSE for fixed delay systems is then invoked to compute the TSE. The results of the two passes are then added up.

During pass one, delay values are assigned so as to maximize the delay between the source event i and a reference event k, which is a node on the critical cycle of the graph. The goal is to use maximum delay values $D(u, v)$ for each edge (u, v) in the critical cycle, and minimum delay values $d(u, v)$ for each edge (u, v) not on the critical cycle, thereby maximizing the local slack on each edge leading from the source event i to the reference event k on the critical cycle. Unfortunately, this is not a straight forward step: without pre-assigned delay values, it is not possible to find the critical cycle; Moreover, the assignment of delay values, and thus the location of the critical cycle, is dependent on where the source event i is in the graph.

The algorithm takes care of this problem by using a search strategy. Each edge (u, v) in the graph is initially assigned their maximum delay value $D(u, v)$. Starting from the source node i, the algorithm flips the delay value of each incoming edge (u, i) to i to its minimum value $d(u, i)$, and computes the critical cycle of the system. If any of the input edges (u, i) originates from the critical cycle, the search terminates, the critical cycle is returned, and the delay value on each edge on the graph which is not on the critical cycle is flipped to its minimum delay value $d(u, v)$, otherwise the search continues in a breadth-first search fashion for all incoming edges to i.

In the second pass of the algorithm, delay values are assigned so as the maximize the delay between the reference event k, which is node on the critical cycle, and the target event j. This is achieved simply by assigning the maximum delay value $D(u, v)$ for each edge on the graph. There is no need to run all three steps of the fixed-delay algorithm. Only step 3 is needed. This is because the critical cycle has already been found in Pass 1 of the algorithm. The only work done in this pass is to sum up the maximum delay value $D(u, v)$ on each edge from k to j.

The pseudocode of the algorithm is shown in Table III.

VIII. Experimental results

A CAD tool has been implemented to demonstrate the feasibility and efficacy of our method. The tool was implemented in C. In

```
Max_TSE(G, M0, D, i, j)
1    for each edge (u, v) ∈ E(G)
2      δ(u, v) ← D(u, v)
3    do
4      for each k ∈ in[i]
5        δ(k, i) ← d(k, i) //Assign minimum delay
6    while k ∉ C(G)
7    for each edge (k, v) ∈ E(G) ∉ C(ki)
8      δ(k, v) ← d(k, v)
9      Δik ← TSE algoithm from Table II
9      Δkj ← TSE algoithm from Table II
12   return Δij ← (TSE(i, k) + TSE(k, j))

Min_TSE(G, M0, D, i, j)
1    max ← Max_TSE(G, M0, D, i, j)
2    return T̃ × 2 − max
```

TABLE III

PSEUDOCODE FOR TSE FOR BOUNDED-DELAY SYSTEMS

addition tools from [8] and [4] were used. All experiments were run on a PowerPC G4 CPU at 1.33 GHz with 768 MB of memory. The experimental results are shown in Table IV.

Four sets of test cases were used: (i) different variations of asynchronous FIFO designs; (ii) an asynchronous Huffman decoder design from [2]; (iii) designs from the published literature [1], [8] (iv) variants of the examples shown in Section III of this paper.

In many cases, since the examples are small and simple, the runtimes of the three tools do not show significant discrepancies, and the results match. However, for several examples, there are noticeable differences in runtime; also, an inaccuracy of results for the heuristic method [4] is highlighted. Also, it is shown that several examples cannot be handled directly by [4]. Finally, in a multiple TSE query example, the benefit in runtime of our method over a previous method [8] is shown.

A few interesting cases are highlighted below.

Test cases 10 and 11 illustrates slow convergence in previous approaches. Test case 10 is the example in Figure 3 in Section III-C of this paper, and test case 11 is a variation of the example, with the delay on each edge amplified 10000 times. In this example, there are two cycles with very close cycle times, which can lead to a large number of iterations in unfolding-based method. As can be seen from the table of results, test case 11 does not present a significant runtime increase in our tool or in the heuristic method [4], over test case 10, but does show an increase in runtime for [8].

Test case 15 illustrates a multiple TSE query problem. This uses the same design as in test case 13; in test case 15, TSE's between all 10 pairs of events are queried, while in test case 13, only one pair of events is queried. Method [8] needs to be run 10 times to obtain all the results, resulting in nearly a 7X increase in runtime over test case 13 (which has one TSE query). In our tool, there is no noticeable runtime increase. This example shows how our method can have significant runtime benefits for multiple TSE queries.

Test cases 3, 4, 13, 14 and 15 are cases that cannot be handled by [4]. These examples are modeled by multi-token systems. To be handled by [4], these specifications must be preprocessed to turn them into single-token systems.

Test case 7 is also multi-token system. In this case, the original specification cannot be handled by [4] (as indicated by the N/A result in the table), and therefore results are presented by a manually-preprocessed single-token version provided with the tool for [4].

Test case 12 shows the sensitivity of the accuracy of the TSE to initial conditions. It uses the example in Section III-C.2 of this paper. Since the method in [4] ignores initial conditions, the result of this test case is inaccurate. In particular, this reported TSE is twice as large as the actual result.

	Test case	Runtime (mS)		
		this paper	[8]	[4]
1	linear FIFO	12	12	15
2	FIFO ring (one token)	15	15	30
3	FIFO ring (concurrent)	20	20	N/A
4	FIFO ring w/ split merge	22	20	N/A
5	Huffman decoder [2]	15	14	30
6	[8] Fig. 1	12	14	20
7	[1] Fig. 4	12	12	20 (N/A)
8	[1] Fig. 5	15	20	20
9	[1] Fig. 6	14	13	30
10	Fig. 3	20	30	30
11	Fig. 3 (amplified edge weights)	25	60	30
12	Fig. 2 with $\lambda = 3$	20	20	30
13	Fig. 4	30	40	N/A
14	Fig. 4 (increased period to 5)	30	60	N/A
15	Fig. 4 (multiple queries)	30	200	N/A

TABLE IV

EXPERIMENTAL RESULTS

IX. CONCLUSIONS

We have addressed the problem of computing the maximum and minimum time separation between any pair of events in a concurrent system, which is modeled as decision-free cyclic marked graph, and whose operation semantics obeys the max-causality rule. A polynomial-time algorithm is proposed, which requires no graph unfolding, and a CAD tool built to demonstrate the feasibility and efficacy of the method.

Acknowledgments. The authors thank Henrik Hulgaard of Configit Software and Supratik Chakraborty of the Indian Institute of Technology for providing their tools and testcases for experiments.

REFERENCES

[1] T. Amon, H. Hulgaard, S. M. Burns, and G. Borriello. An algorithm for exact bounds on the time separation of events in concurrent systems. In *Proceedings of ICCD*, pages 166–173, 1993.

[2] M. Benes, S. Nowick, and A. Wolfe. A fast asynchronous Huffman decoder for compressed-code embedded processors. In *Proceedings of ASYNC*, 1998.

[3] S. Chakraborty and D. L. Dill. Approximate algorithms for time separation of events. In *Proceedings of ASYNC*, 1997.

[4] S. Chakraborty, K. Y. Yun, and D. L. Dill. Timing analysis of asynchronous systems using time separation of events. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 18(8), Aug. 1999.

[5] F. Commoner, A. Holt, S. Even, and A. Pnueli. Marked directed graphs. *Journal of Computer System and Sciences*, 5:511–523, 1971.

[6] J. Cortadella, M. Kishinevsky, S. Burns, A. Kondratyev, L. Lavagno, K. Stevens, A. Taubin, and A. Yakovlev. Lazy transition systems and asynchronous circuit synthesis with relative timing assumptions. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 21(2), Feb. 2002.

[7] P. Girodias, E. Cerny, and W. Older. Solving linear, min, and max constraint systems using CLP based on relational interval arithmetic. *Journal of Theoretical Computer Science*, 173:253 – 281, 1997.

[8] H. Hulgaard, S. M. Burns, T. Amon, and G. Borriello. An algorithm for exact bounds on the time separation of events in concurrent systems. *IEEE Transactions on Computers*, 44(11), Nov. 1995.

[9] F. Jin, H. Hulgaard, and E. Cerny. Maximum time separation of events in cyclic systems with linear and latest timing constraints. In *Formal Methods in Computer-Aided Design*, pages 167–184, 1998.

[10] E. L. Lawler. *Combinatorial Optimization: Networks and Matroids*. Holt, Rinehart and Winston, 1976.

[11] P. B. McGee, S. M. Nowick, and E. G. Coffman, Jr. Efficient performance analysis of asynchronous systems based on periodicity. In *CODES+ISSS '05: Proceedings of the 3rd IEEE/ACM/IFIP international conference on Hardware/software codesign and system synthesis*, 2005.

[12] K. McMillan and D. Dill. Algorithms for interface timing verification. In *Proceedings of Int. Conf. Computer Design: VLSI in Computers and Processors*, 1992.

[13] T. Murata. Petri nets: properties, analysis and applications. *Proceedings of the IEEE*, 77(4):541–580, Apr. 1989.

[14] C. Myers and T.-Y. Meng. Synthesis of timed asynchronous circuits. *IEEE Transaction of VLSI Systems*, 1(2):106–119, Jun. 1993.

[15] A. Xie, S. Kim, and P. A. Beerel. Bounding average time separations of events in stochastic timed petri nets with choice. In *Proceedings of ASYNC*, 1999.