Accurate Performance Estimation for Stochastic Marked Graphs by Bottleneck Regrowing*

Ricardo J. Rodríguez and Jorge Júlvez

Dpto. de Informática e Ingeniería de Sistemas
Universidad de Zaragoza, Zaragoza, Spain
{rjrodriguez, julvez}@unizar.es

Abstract. The adequate system performance is usually a critical requirement to be checked during the verification phase of a system. Thus, accurately measuring the performance of current industrial systems, which are often modelled as a Discrete Event Systems (DES), is a need. Due to the state explosion problem, the performance evaluation of DES becomes increasingly difficult as the size of the systems increases. Some approaches, such as the computation of performance bounds, have been developed to overcome this problem. In this paper we propose a new method to produce performance bounds that are sharper than the ones that can be achieved with current methods. The core of our method is an iterative algorithm that initially considers the most constraining bottleneck cycle of the system and adds other cycles to it in each iteration. The proposed method is deeply explained and then applied to a broad set of Marked Graphs.

1 Introduction

One of the problems when dealing with the production of a new system is the verification of requirements. A requirement is a singular need of what the product (i.e, system, or service) should be or should perform. The requirements of a system can be divided in functional and non-functional requirements. The functional requirements involve calculations, technical details, data (or item) manipulation and processing or any functionality that defines what the system is supposed to do, while non-functional requirements define how the system is supposed to be. Some examples of non-functional requirements are constraints, usability, maintainability and performance.

Thus, correctly measuring the performance of an industrial system is eventually a need. Many of these artificial systems (e.g., logistic, manufacturing, traffic system, etc.) can be naturally modeled as Discrete Event Systems (DES). In practice, the increasing size of systems makes the exact computation of their performance a highly complex computational task. The main reason for this complexity is the state explosion problem, according to which the number of states

* This work has been supported by the European Community's Seventh Framework Programme under project DISC (Grant Agreement n.INFSO-ICT-224498) and by the project FORMALISM (TIN2007-66523).

of a system is exponential in the size of the system description. As a result, a task that requires an exhaustive state space exploration becomes unachievable in reasonable time for large systems.

There exist many works in the literature related to the performance evaluation of systems modeled as DES. Concerning works that compute exact analytical measures of performance of Timed Marked Graphs, which are the framework of this paper, the work in [1] determined the average throughput of a timed Marked Graph with deterministic firing delays in polynomial time. In [2], the average cycle time of events in Marked Graphs with fixed delays was calculated using linear programming. Other works [3,4,5] are focused on obtaining the Markov chain (continuous time, CTMC [3], or discrete time, DTMC [4,5]) in asynchronous circuits to calculate its stationary probability distribution. In [6], queuing models are used to avoid the state explosion problem and the performance of closed asynchronous ring structures is studied.

In addition, several approaches have been developed to overcome the state explosion problem. These approaches provide performance bounds [7,8,9,10,11] and avoid the necessity of calculating the whole state space. The main advantage of these approaches is that they reduce the running time needed for computing a performance bound. Unfortunately, a drawback is the difficulty to assess how *good*, i.e., accurate, the computed bound is with respect to the real system performance. In the particular case of Marked Graphs with deterministic firing delays, the obtained bound is equal to real performance. However, this is not the case if we are dealing with other probability distribution functions (e.g., exponential, uniform or normal) for the firing delays.

Linear programming techniques have been used to compute throughput bounds of Marked Graphs [7] and general Petri nets [8]. These bounds are calculated using the first order moment (i.e., the mean) of the distributions associated to the firing delay. A sharper performance bound that makes use of second order moments is proposed in [9]. The work in [10] bounds the average time separations of events in Stochastic Timed Petri Nets by deriving closed-form expressions for lower and upper bounds which are then evaluated using standard statistical methods (more precisely Monte-Carlo simulation). Perfomance bounds for interval Time Petri nets are also provided in [11].

This paper proposes an iterative algorithm to obtain performance bounds on Stochastic Marked Graphs that are sharper, i.e., closer to the real performance, than the ones we can currently compute with some of the works previously mentioned. In a few words, our method works as follows. First, the algorithm calculates the most restrictive cycle of the Marked Graph by applying well-known methodologies. Then, it adds to the bottleneck cycle those sets of places that are more likely to constraint the throughput of the system. The process of adding sets of places is repeated until the throughput of the resulting net does not vary significantly. Such throughput cannot increase during the addition process since more constraints are added to the net. The proposed algorithm produces the following outputs:

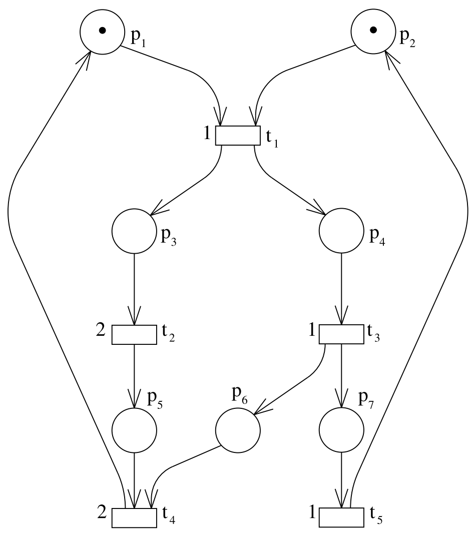

Fig. 1. Example MG. The number shown at the left of each transition represents its mean firing delay.

– a performance bound for the steady state throughput of a stochastic Marked Graph and
– a subnet representing the bottleneck of the stochastic Marked Graph.

As it will be explained, the method makes intensive use of linear programming techniques for which polynomial complexity algorithms exist. Given that the performance bound is refined in each iteration, the accuracy of the final bound depends on the number of iterations to be performed. The obtained results show that the proposed method offers a good trade-off between accuracy and computational complexity load.

Let us illustrate our approach through the Marked Graph (MG) shown in Figure 1. The initial marking is: $\mathbf{m}(p_1) = \mathbf{m}(p_2) = 1$ and the rest of places have marking equal to 0. We assume that the firing delay of each transition follows an exponential distribution with means $\delta(t_1) = \delta(t_3) = \delta(t_5) = 1, \delta(t_2) = \delta(t_4) = 2$. The net has three cycles: $\{p_1, p_3, p_5\}$, $\{p_1, p_4, p_6\}$ and $\{p_2, p_4, p_7\}$. The token/delay ratio of each cycle is $\frac{1}{5}$, $\frac{1}{4}$ and $\frac{1}{3}$, respectively. The critical or bottleneck cycle is the one with minimum token to delay ratio, thus in our case, the bottleneck cycle is the one composed of places $\{p_1, p_3, p_5\}$ whose throughput is equal to $\frac{1}{5}$. Hence, the method takes $\frac{1}{5}$ as the initial throughput bound and $P_{bn} = \{p_1, p_3, p_5\}$ as the initial bottleneck.

The method now adds to the initial bottleneck P_{bn} those sets of places that are likely to be less saturated than the ones contained in P_{bn}. In order to produce a strongly connected component two choices are feasible: adding to P_{bn} either cycle $\{p_1, p_4, p_6\}$ or cycle $\{p_2, p_4, p_7\}$. Intuitively it makes more sense to add the cycle $\{p_1, p_4, p_6\}$ since it has lower token to delay ratio. The resulting net is a strongly connected component, more precisely it is a subnet of the whole MG, and hence its throughput is an upper bound for the throughput of the whole MG. Indeed, the net composed by places $\{p_1, p_3, p_4, p_5, p_6\}$ has a throughput equal to 0.1875, which is a 6.25% lower than the throughput of the initial bottleneck P_{bn}.

In general, the number of cycles in a system is exponential in the size of the net. Hence, looking for the cycle with minimum token/delay ratio is in general a non trivial task. In order to face this problem, a special marking called *tight marking* will be introduced. The tight marking will enable us to easily detect those cycles that are more constraining *a priori*. The tight marking associates a *slack* to each place, the lower the slack the higher the probability that place will constraint the system throughput. The slack for the places of the net in Figure 1 are $\mu(p_2) = 0.4$, $\mu(p_6) = 0.2$ and 0 for the rest of places. Notice that all the places in P_{bn} have null slack, and that the places with positive slack belong to the potential cycles to be added to P_{bn}. The place with minimum slack is p_6, thus the most sensible choice is to add to P_{bn} the cycle $\{p_1, p_4, p_6\}$. Finally, if we keep regrowing the net, we can add the remaining cycle $\{p_2, p_4, p_7\}$ what yields the whole net whose throughput is equal to 0.181208 (9.396% lower than the throughput of the initial critical cycle).

As a running example throughout the paper we will consider the Marked Graph (MG) shown in Figure 2. In this MG, we are able to get, in a few iterations, a performance bound which is 12.9% lower than the initial one. Indeed, a better performance bound, just 0.3% greater than the real performance, can obtained if more iterations are considered.

The most restrictive cycle of the net, which is calculated by solving a *linear programming* problem (LPP), is the one composed of places and transitions $\{\{p_2, p_4\}, \{t_1, t_3\}\}$. The first iteration of the method adds the cycle composed of $\{\{p_1, p_3\}, \{t_1, t_2\}\}$ to the previous one. The decision of adding $\{\{p_1, p_3\}, \{t_1, t_2\}\}$ is based on the fact that p_1 is the place with minimum slack connected to the initial bottleneck. The throughput of the resulting net is 12.9% lower than the throughput of the initial critical cycle. The proposed method is efficient due to the use linear programming techniques and is accurate because the bound converges in few iterations.

The balance of this paper is as follows. Section 2 introduces the notation and concepts we use in the rest of the paper: definition of MG, upper bound computation and *tight marking*. Section 3 details the graph regrowing strategy. Section 4 presents the results obtained after applying the approach to some circuits of the ISCAS benchmarks [12]. The main conclusions are addressed in Section 5.

2 Marked Graphs and Tight Marking

In this section we introduce the basic concepts needed to follow the rest of the paper, such as marked graph and a special marking, called *tight marking*, which

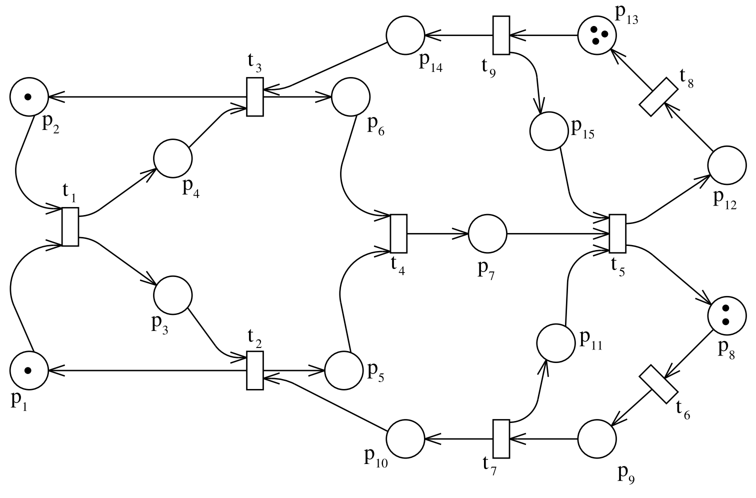

Fig. 2. Running example MG used through the paper.

allows us to compute easily slacks of places with respect to the critical cycle. It is assumed that the reader is familiar with Petri nets (see [13,14] for a gentle introduction).

2.1 Stochastic Marked Graph

Let us start by defining Petri nets in the untimed framework.

Definition 1. *A Petri net is a 4–tuple* $\mathcal{N} = \langle P, T, \mathbf{Pre}, \mathbf{Post}\rangle$*, where:*

- P *and* T *are disjoint non-empty sets of* places *and* transitions *(*$|P| = n$*,* $|T| = m$*) and*
- $\mathbf{Pre}$ *(*$\mathbf{Post}$*) are the pre–(post–)incidence non-negative integer matrices of size* $|P| \times |T|$*.*

Ordinary nets are Petri nets whose arcs have weight 1. The *pre-* and *post-set* of a node $v \in P \cup T$ are defined respectively as ${}^\bullet v = \{u \in P \cup T | (u, v) \in F\}$ and $v^\bullet = \{u \in P \cup T | (v, u) \in F\}$, where $F \subseteq (P \times T) \cup (T \times P)$ is the set of directed arcs. The *incidence matrix* of the Petri net is defined as $\mathbf{C} = \mathbf{Post} - \mathbf{Pre}$.

A vector $\mathbf{m} \in \{\mathbb{Z}^+\}^{|P|}$ which assigns a non-negative integer to each place is called *marking*.

Definition 2. *A* Petri net system*, or* marked Petri net $\mathcal{S} = \langle \mathcal{N}, \mathbf{m_0}\rangle$*, is a Petri net* $\mathcal{N}$ *with an* initial marking $\mathbf{m_0}$*.*

A transition $t \in T$ is *enabled* at marking $\mathbf{m}$ if $\mathbf{m} \geq \mathbf{Pre}[P, t]$. A transition t enabled at $\mathbf{m}$ can *fire* yielding a new marking $\mathbf{m'} = \mathbf{m} + \mathbf{C}[P, t]$ (*reached* marking). It is denoted by $\mathbf{m} \xrightarrow{t} \mathbf{m'}$. A sequence of transitions $\tau = \{t_i\}_{i=1}^n$

is a *firing sequence* in $\mathcal{S}$ if there exists a sequence of markings such that $\mathbf{m_0} \xrightarrow{t_1} \mathbf{m}_1 \xrightarrow{t_2} \mathbf{m}_2 \ldots \xrightarrow{t_n} \mathbf{m}_n$. In this case, marking $\mathbf{m}_n$ is said to be *reachable* from $\mathbf{m_0}$ by firing τ, and this is denoted by $\mathbf{m}_0 \xrightarrow{\tau} \mathbf{m}_n$. The *firing count vector* $\boldsymbol{\sigma}$ of the firable sequence τ is a vector such that $\boldsymbol{\sigma}(t)$ represents the number of occurrences of $t \in T$ in τ. If $\mathbf{m_0} \xrightarrow{\tau} \mathbf{m}$, then we can write in vector form $\mathbf{m} = \mathbf{m_0} + \mathbf{C} \cdot \boldsymbol{\sigma}$, which is referred to as the *linear (or fundamental) state equation* of the net.

The set $\mathrm{L}(\mathcal{S}) = \{\tau | \tau \text{ firable from } \mathbf{m_0}\}$ is the *language of firing sequences* of $\mathcal{S}$. $\mathrm{RS}(\mathcal{S})$ is the set of all reachable markings from $\mathbf{m_0}$. $\mathrm{RG}(\mathcal{S})$ is the reachability graph of $\mathcal{S}$ (a graph with $\mathrm{RS}(\mathcal{S})$ as set of vertices and whose set of edges are the firing sequences of length 1 between vertices).

A *p-semiflow* is a non-negative vector $Y : P \to \mathbb{N}$ such that is a left anuller of the net's flow matrix, $Y^T \cdot C = 0$. A *t-semiflow* is a non-negative vector $X : T \to \mathbb{N}$ such that is a right anuller of the net's flow matrix, $C \cdot X^T = 0$. A t-semiflow $\mathbf{v}$ is *minimal* when its support, $\|\mathbf{v}\| = \{i | \mathbf{v}[i] \neq 0\}$, is not a proper superset of the support of any other t-semiflow, and the greatest common divisor of its elements is one. A Petri Net is said to be *strongly connected* if there is a directed path joining any node A to any node B of the graph.

Marked graphs are a subclass of ordinary Petri nets that generalizes PERT charts and that is characterized by the fact that each place has exactly one input and exactly one output arc.

Definition 3. [14] *A marked graph (MG) is an ordinary Petri net such that* $\forall p \in P, |^{\bullet}p| = |p^{\bullet}| = 1$.

Since, we are primarily interested in bounded and repetitive behaviours, the MGs under consideration are assumed to be strongly connected. A Stochastic Marked Graph is defined as a Marked Graph to which exponential distribution functions are associated to the firing delays of transitions. More formally:

Definition 4. *A* Stochastic Marked Graph *(SMG) is a pair* $\langle \mathcal{S}, \delta \rangle$ *where* $\mathcal{S} = \langle P, T, \mathbf{Pre}, \mathbf{Post}, \mathbf{m_0} \rangle$ *is a marked graph and* $\delta : T \to \mathbb{R}^+$ *is a positive real function such that* $\delta(t)$ *is the mean of the exponential firing time distribution associated to each transition* $t \in T$.

Thus, when a transition t of an SMG becomes enabled, it fires u time units later, where u is a random value that follows an exponential distribution with mean $\delta(t)$. There exist different semantics for the firing of transitions, being *infinite* and *finite* server semantics the most frequently used. Given that *infinite* server semantics is more general (*finite* server semantics can be simulated by adding self-loop places), we will assume that the transitions of the SMGs work under *infinite* server semantics.

The average marking vector, $\overline{\mathbf{m}}$, in an ergodic Petri net system is defined as:

$$\overline{\mathbf{m}}(p) \underset{AS}{=} \lim_{T \to \infty} \frac{1}{T} \int_0^T \mathbf{m}(p)_\tau d\tau \qquad (1)$$

where $\mathbf{m}(p)_\tau$ is the marking of place p at time τ and the notation $\underset{AS}{=}$ means equal almost surely [15].

Similarly, the steady state throughput, χ, in an ergodic Petri net is defined as:

$$\chi(t) \underset{AS}{=} \lim_{T\to\infty} \frac{\boldsymbol{\sigma}(t)_T}{T} \tag{2}$$

where $\sigma(t)_T$ is the firing count of transition t at time T.

Notice that the reachability graph of a strongly connected SMG is finite and strongly connected. Therefore, its associated continuous time Markov chain is ergodic, what implies the ergodicity of $\mathbf{m}_\tau$ and σ_T, and the existence of the above limits.

2.2 Critical Cycle

The Little's formula [16] involves the average number of customers L in a queue, the input rate (throughput), λ, and the average time spent by a customer within the queue, W.

$$L = \lambda \cdot W \tag{3}$$

In an SMG, each pair $\{p,t\}$ where $p^\bullet = \{t\}, p \in P, t \in T$ can be seen as a simple queueing system for which Little's formula can be directly applied[1] [17]:

$$\overline{\mathbf{m}}(p) = \chi(p^\bullet) \cdot \overline{\mathbf{s}}(p) \tag{4}$$

where $\overline{\mathbf{s}}(p)$ is the average residence time at place p, i.e., the average time spent by a token in p. The average residence time is the sum of the average waiting time due to a possible synchronization and the average service time which in our case is $\delta(p^\bullet)$. Therefore, the service time $\delta(p^\bullet)$ is a lower bound for the average residence time. This leads to the inequality:

$$\overline{\mathbf{m}}(p) \geq \chi(p^\bullet) \cdot \delta(p^\bullet) \tag{5}$$

Let us notice that strongly connected MGs have a single minimal t-semiflow that is equal to $\mathbf{1}$. This implies that the steady state throughput is the same for every transition. Therefore, a single scalar variable Θ suffices to express the throughput bound to be computed for all transitions.

Proposition 1. *The solution Θ of the following LPP provides an upper bound for the steady state throughput of the transitions of an SMG [8]:*

$$Maximize\ \Theta :$$

$$\hat{\mathbf{m}}(p) \geq \delta(p^\bullet) \cdot \Theta \quad \forall p \in P \tag{6a}$$

$$\hat{\mathbf{m}} = \mathbf{m_0} + \mathbf{C} \cdot \sigma \tag{6b}$$

$$\sigma \geq 0 \tag{6c}$$

[1] In the sequel, for clarity we slightly abuse of notation and denote by $p^\bullet$ the only element of the set $p^\bullet = \{t\}$

The first constraint (6a) is obtained from (5), the second and third constraints (6b), (6c) establish that $\hat{\mathbf{m}}$ must be a solution of the state equation. The value of Θ is the exact throughput in the particular case of timed MG with deterministic distributions associated to the firing delays [1,18].

The LPP in (6) can be transformed in its dual, which after some manipulations becomes [19]:

$$\begin{aligned}\gamma = \ & \textit{maximum}\ \mathbf{y}\cdot\mathbf{D}\\ & \textit{subject to}\ \mathbf{y}\cdot\mathbf{C} = 0\\ & \qquad\qquad\ \mathbf{y}\cdot\mathbf{m_0} = 1\\ & \qquad\qquad\ \mathbf{y} \geq 0\end{aligned} \tag{7}$$

where $\mathbf{D}(p) = \delta(p^\bullet)$. It holds that $\Theta = \dfrac{1}{\gamma}$ where Θ is the solution of (6). The LPP (7) can be interpreted as a search for the most constraining p-semiflow, what in SMGs is equivalent to the most constraining cycle (or critical cycle), i.e., the one with lowest token/delay ratio. The support of $\mathbf{y}$ represents such a bottleneck cycle.

The LPP shown in Proposition 1 also allows us to calculate the critical cycle of an SMG. In fact, the critical cycle is the cycle whose places fulfill the equality, $\hat{\mathbf{m}}(p) = \delta(p^\bullet)\cdot\Theta$, in equation (6a). Notice that (6a) can be expressed as follows:

$$\mathbf{m}(p) = \delta(p^\bullet)\cdot\Theta + \mu(p)$$

where $\mu(p)$ is the *slack* of place p. For every place p in the critical cycle, it necessarily holds that $\mu(p) = 0$. For example, the slacks of the places of the SMG in Figure 1 are $\mu(p_1) = \mu(p_3) = \mu(p_5) = 0$, $\mu(p_2) = 0.16$, $\mu(p_4) = 0.08$, $\mu(p_6) = 0.12$ and $\mu(p_7) = 0.16$. In general, the same optimal value of the objective function can be achieved for different slack vectors, in fact, the particular value of vector μ will depend on the algorithm used by the LP solver.

2.3 Tight Marking

This section takes advantage of the degree of freedom of slacks in order to produce a marking, called *tight marking* and denoted $\tilde{\mathbf{m}}$, such that each transition has at least one input place with null slack. This marking will greatly ease the task of adding to the initial bottleneck cycle those cycles that have low ratio token/delay.

Definition 5. *A marking vector* $\tilde{\mathbf{m}} \in \mathbb{R}^{|P|}$ *is called a* tight *marking vector of an SMG if it satisfies:*

$$\tilde{\mathbf{m}} = \mathbf{m_0} + \mathbf{C}\cdot\sigma \tag{8a}$$

$$\forall\ p: \quad \tilde{\mathbf{m}}(p) \geq \delta(p^\bullet)\cdot\Theta \tag{8b}$$

$$\forall\ t\ \exists\ p \in {}^\bullet t: \quad \tilde{\mathbf{m}}(p) = \delta(p^\bullet)\cdot\Theta \tag{8c}$$

where $\tilde{\mathbf{m}} \in \mathbb{R}^{|P|}$*,* $\sigma \in \mathbb{R}^{|T|}$*, and* Θ *is the solution of* (6)*. A place* p *satisfying the condition* $\tilde{\mathbf{m}}(p) = \delta(p^\bullet) \cdot \Theta$ *is called* tight*.*

Since the places of the critical cycle do not have slack, they fulfill (8c) and hence are tight. On the other hand, non-critical places may have some positive slack. The tight marking exploits this flexibility by adjusting the marking in such a way that each transition has at least one input place that is tight.

It can be shown that a tight marking exists for each SMG [20]. Moreover it can be computed efficiently by solving an LPP.

Proposition 2. *A tight marking of an SMG can be computed by solving the following LPP:*

$$\begin{aligned} & Maximize\ \Sigma\sigma : \\ & \qquad \delta(p^\bullet) \cdot \Theta \leq \tilde{\mathbf{m}}(p) \quad \textit{for every } p \in P \\ & \qquad \tilde{\mathbf{m}} = \mathbf{m_0} + \mathbf{C} \cdot \sigma \\ & \qquad \sigma(t_p) = k \end{aligned} \tag{9}$$

where t_p *is a transition that belongs to a critical cycle and* k *is any real constant number.*

The proof of the Proposition 2 can be found in [20]. Since we are dealing with MGs, each row of the incidence matrix $\mathbf{C}$ contains a single positive (1) and a single negative (-1) value, while all other values are zeros. Therefore, the first two constraints of (9) can be transformed into a system of *difference constraints* and hence the LPP (9) can be efficiently solved by using the Bellman-Ford algorithm [21].

Recalling the SMG shown in Figure 1, if we calculate the tight marking we obtain $\tilde{\mathbf{m}}(p_1) = 0.2$, $\tilde{\mathbf{m}}(p_2) = 0.6$, $\tilde{\mathbf{m}}(p_3) = 0.4$, $\tilde{\mathbf{m}}(p_4) = 0.2$, $\tilde{\mathbf{m}}(p_5) = 0.4$, $\tilde{\mathbf{m}}(p_6) = 0.6$, $\tilde{\mathbf{m}}(p_7) = 0.2$.

To illustrate all the above mentioned concepts, we recall the example of SMG shown in Figure 2. The initial marking of the SMG is $\mathbf{m}(p_1) = \mathbf{m}(p_2) = 1$, $\mathbf{m}(p_8) = 2$, $\mathbf{m}(p_{13}) = 3$ and the rest of places have no initial tokens. Transitions have infinite server semantic and the delays are $\delta(t_1) = 1.2$, $\delta(t_2) = 1$, $\delta(t_3) = 1.5$, $\delta(t_4) = \delta(t_5) = 1$, $\delta(t_6) = 0.75$, $\delta(t_7) = 1$, $\delta(t_8) = 1.25$ and $\delta(t_9) = 0.5$.

Applying the LPP in Proposition 1, we obtain the maximum throughput of the SMG, which is, in this case, $\Theta = 0.3704$. The solution of LPP (9) yields the following *tight marking* vector: $\tilde{\mathbf{m}}(p_1) = 0.6296$, $\tilde{\mathbf{m}}(p_2) = 0.4444$, $\tilde{\mathbf{m}}(p_3) = 0.3704$, $\tilde{\mathbf{m}}(p_4) = \tilde{\mathbf{m}}(p_5) = 0.5556$, $\tilde{\mathbf{m}}(p_6) = \tilde{\mathbf{m}}(p_7) = 0.3704$, $\tilde{\mathbf{m}}(p_8) = 0.2778$, $\tilde{\mathbf{m}}(p_9) = 0.3704$, $\tilde{\mathbf{m}}(p_{10}) = 0.4259$, $\tilde{\mathbf{m}}(p_{11}) = 1.3519$, $\tilde{\mathbf{m}}(p_{12}) = 0.4630$, $\tilde{\mathbf{m}}(p_{13}) = 0.1852$, $\tilde{\mathbf{m}}(p_{14}) = 1.6111$ and $\tilde{\mathbf{m}}(p_{15}) = 2.3519$.

Interestingly, if we consider just the tight places and their input and output transitions, an SMG is obtained such that the only strongly connected component is critical cycle. The rest of tight places, and their input and output transitions, conform a kind of tree where the critical cycle is the root and all the transitions are reached.

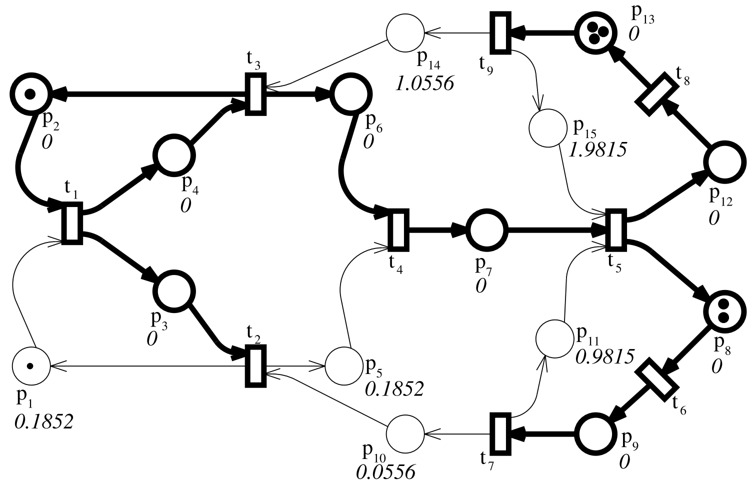

Fig. 3. Example SMG with highlighted tight places and values of *slacks*.

3 Graph Regrowing Strategy

This section presents an iterative strategy to grow the critical cycle and to compute an upper throughput bound. The idea of the strategy is to add in each iteration the cycle that is potentially more restrictive than the others and then calculate the throughput. Such a throughput cannot be higher than the one in the previous iteration, since more constraints have been added to the net. The iteration process will stop when no significant improvement of the bound is achieved.

Algorithm 1 represents the overall regrowing strategy used to compute throughput bounds. The algorithm needs as input data the SMG to be analysed, $\langle \mathcal{N}, \delta \rangle$, and the degree of precision ($\varepsilon > 0$) to be achieved. As output data, the upper throughput bound, Θ, and the bottleneck cycle of the SMG, $sccN'$, are obtained.

Firstly, an upper throughput bound of $\langle \mathcal{N}, \delta \rangle$ is calculated according to (6), which will be the initial upper bound. Then, the tight marking of the system is computed by using the LPP shown in (9). The vector of slacks μ is computed in step 3. The iteration process (steps 7–14) is repeated until no significant improvement is achieved with respect to the last iteration.

In steps 8–11, a new set of places and transitions is added to the current bottleneck. To achieve this, steps 8–9 look for the place q that is connected to the current bottleneck $sccN'$, i.e., $q^\bullet \in sccN'$, and has minimum slack. Then steps 10–11 build the new bottleneck by adding place q and the tight places that connect the current bottleneck to q. For brevity, in the algorithm we use $p \in \mathcal{N}$ ($p^\bullet \in \mathcal{N}$) to denote that a place p (transition $p^\bullet$) is contained in the set of places (transitions) of $\mathcal{N}$. When there exist several identical critical cycles, i.e, with the same token to delay ratio, steps 5 and 11 choose any of them.

```
Input: $\langle \mathcal{N}, \delta \rangle$, $\varepsilon$
Output: $\Theta$, $sccN'$
1  $\Theta$ = Upper throughput bound of $\mathcal{N}$ according to (6)
2  $\tilde{\mathbf{m}}$ = Tight marking according to (9)
3  $\mu(p) = \tilde{\mathbf{m}}(p) - \delta(p^\bullet) \cdot \Theta, \quad \forall p \in P$
4  $\mathcal{N}'$ = Graph resulting of removing from $\mathcal{N}$ every arc $\{p, p^\bullet\}$ such that $\mu(p) > 0$
5  $sccN'$= Strongly connected component of $\mathcal{N}'$
6  $\Theta' = 0$
7  while $\left(\frac{\Theta - \Theta'}{\Theta} \geq \varepsilon\right)$ do
8      $Q = \{q | q \in P, q \notin \mathcal{N}', q^\bullet \in sccN'\}$
9      $p_m = \{q | \mu(q) = \min_{p \in Q} \ \mu(p)\}$
10     $\mathcal{N}'$ = Graph resulting of adding arc $\{p_m, p_m^\bullet\}$ to $\mathcal{N}'$ where $\{p_m, p_m^\bullet\} \in \mathcal{N}$
11     $sccN'$= Strongly connected component of $\mathcal{N}'$
12     $\Theta' = \Theta$
13     $\Theta$ = Throughput of $sccN'$
14 end
```

Algorithm 1: The regrowing strategy algorithm.

In step 13, the throughput of the new bottleneck is taken as the new upper bound. In the next iteration, this new upper bound will be compared with the previous one in order to, depending on the degree of improvement achieved, either continue or finish the iteration process.

In the given example shown in Figure 2, whose delays are $\delta(t_1) = 1.2$, $\delta(t_2) = 1$, $\delta(t_3) = 1.5$, $\delta(t_4) = \delta(t_5) = 1$, $\delta(t_6) = 0.75$, $\delta(t_7) = 1$, $\delta(t_8) = 1.25$ and $\delta(t_9) = 0.5$, the critical cycle is composed by $\{P_{cb}, T_{cb}\} = \{\{p_2, p_4\}, \{t_1, t_3\}\}$. The throughput bound of the critical cycle is $\Theta_{cb} = 0.370370$ and the places which are connected (through a transition $t \in T$) to the critical cycle are p_1 and p_{14}, having slacks $\mu(p_1) = 0.1852$ and $\mu(p_{14}) = 1.0556$. Hence, the place with minimum slack is p_1. By regrowing the current bottleneck the new one is obtained, composed by $\{P_{cb'}, T_{cb'}\} = \{\{p_1, p_2, p_3, p_4\}, \{t_1, t_2, t_3\}\}$, which has a throughput of $\Theta_{cb'} = 0.322581$, which is 12.9% lower than the throughput of the previously bottleneck $\{P_{cb}, T_{cb}\}$.

Let us assume that $\varepsilon = 0.001$. As the relative difference between Θ_{cb} and $\Theta_{cb'}$ is 0.12903 (as commented previously), the iteration process carries on. At this moment, the places connected to the current bottleneck are p_{10} and p_{14}. The addition of the place p_{10} which has minimum slack produces a new bottleneck compounded of $\{\{p_1, p_2, p_3, p_4, p_6, p_7, p_8, p_9, p_{10}\}, \{t_1, t_2, t_3, t_4, t_5, t_6, t_7\}\}$, being the new throughput $\Theta = 0.297914$, which is an improvement of 7.647% with respect to the previous bottleneck $\{P_{cb'}, T_{cb'}\}$ and 19.563% with respect to the original bottleneck $\{P_{cb}, T_{cb}\}$.

Again, a new regrowing is possible because the relative difference is greater than ε. In this case, the candidate places to be chosen are p_5, p_{11} and p_{14}, which have slacks $\mu(p_5) = 0.0556$, $\mu(p_{11}) = 0.9815$ and $\mu(p_{14}) = 1.0556$. The addition of p_5 produces a new bottleneck with $\Theta = 0.297914$, which is an improvement of 3.193% with respect to the previous bottleneck. For the next regrowing, the candidate places are p_{11}, p_{14} and p_{15}. By adding the place p_{11} ($\mu(p_{11}) = 0.9815$) we obtain a bottleneck whose relative throughput is lower than ε with respect to the previous bottleneck, thus, the algorithm finishes. In summary, after four iterations, the throughput bound obtained is 22.132% lower than the original Θ calculated by LPP in (6).

4 Experiments and Results

In this Section we test our approach on a set of SMGs of the ISCAS benchmarking [12]. After applying the regrowing strategy, the obtained results are discussed.

The structure of the SMGs to be analysed is obtained from the strongly connected components of the ISCAS graphs. The initial marking of each place is a natural number which is randomly selected in the interval $[1 \ldots 10]$. The value of the $\delta(t)$ of each transition t is a real number randomly selected from the interval $[0.1 \ldots 1]$. The overall strategy has been implemented on MATLAB [22], while simulations of SMGs have been performed by the GreatSPN [23] simulation tool using a confidence level of 99% and an accuracy of 1%. The simulations have been run in a machine with a Pentium IV 3.6GHz processor and 2GB DDR2 533MHz RAM.

Graph	Size		% Size		Regrowing steps	Initial thr. bound	Θ
	$\|P\|$	$\|T\|$	$\|P'\|$ (%)	$\|T'\|$ (%)			
s1423	1107	792	79 (7.13%)	76 (9.59%)	3	0.236010	0.235213 (0.34%)
s1488	1567	1128	91 (5.8%)	86 (7.62%)	6	0.201300	0.173127 (13.99%)
s208	27	24	27 (100%)	24 (100%)	3	0.409390	0.377683 (7.75%)
s27	54	44	19 (35.18%)	18 (40.9%)	1	0.305960	0.304987 (0.31%)
s349	187	146	26 (13.9%)	24 (16.44%)	2	0.340320	0.327867 (3.66%)
s444	92	68	14 (15.21%)	12 (17.64%)	2	0.181670	0.181260 (0.22%)
s510	1038	734	45 (4.33%)	40 (5.45%)	5	0.133030	0.117819 (11.43%)
s526	113	92	18 (15.93%)	16 (17.39%)	2	0.313490	0.305860 (2.43%)
s713	271	208	11 (4.06%)	10 (4.8%)	1	0.428720	0.427840 (0.2%)
s820	1162	848	40 (3.44%)	38 (4.48%)	2	0.161060	0.147483 (8.43%)
s832	1293	948	84 (6.5%)	78 (12.04%)	5	0.239429	0.208798 (12.79%)
s953	415	312	88 (11.36%)	82 (26.28%)	6	0.369214	0.337811 (8.50%)

Table 1. Experiment results showing improvement of upper bound.

Table 1 shows the obtained results by our approach. The degree of accuracy for Algorithm 1 has been set to $\varepsilon = 0.005$. The first column is the graph name, followed by its size (number of places, $|P|$, and transitions, $|T|$). In the next column, it is shown the size of the net $sccN'$ ($|P'|$,$|T'|$) produced by the algorithm.

The column *Regrowing steps* shows the number of regrowing steps needed by the algorithm. The last columns of Table 1 show the initial upper throughput bound calculated by using the LPP (6), and the improved upper throughput bound, Θ, computed by the algorithm. Such a bound is computed by solving the Markov Chain associated to $sccN'$ when it is handleable by the computer, and by simulation otherwise (see [24] for an example of Markov Chain analysis). The last column shows the percentage of improvement with respect to the original upper throughput bound.

As it can be seen, our method is able to get a sharper upper bound than the original bound in a few regrowing steps, and the improvement varies from 0.2% (which indicates that the original upper bound is already very tight) up to 14%. We conjecture that the improvement depends on the structure of the graph. It is also worth mentioning that our approach uses a very low percentage of the size of the original graph, in most of cases this percentage is lower than 10%.

Graph	Original graph thr. CPU time (s)	Θ CPU time (s)	Original graph thr.	Θ	% thr.
s1423	59948.980	8.283	0.222720	0.235270	5.63%
s1488	36717.156	7.165	0.168760	0.172154	2.01%
s208	0.492	0.492	0.376892	0.376892	0%
s27	2166.002	0.954	0.305082	0.306166	0.35%
s349	141.210	0.441	0.328340	0.327398	−0.28%
s444	2278.231	0.205	0.181069	0.181260	0.11%
s510	13669.814	1.358	0.117500	0.118040	0.46%
s526	129.181	0.344	0.270010	0.305860	13.27%
s713	628.503	0.405	0.411630	0.427840	3.94%
s820	20775.811	0.788	0.144770	0.147699	2.02%
s832	16165.863	1.914	0.196920	0.208873	6.07%
s953	453.850	19.155	0.327910	0.338644	3.27%

Table 2. Graph throughput and CPU time comparative.

Table 2 summarises a comparative between the original throughput bound and the improved upper throughput bound and between the CPU time needed for both computations. The first column is the graph name, followed by the CPU time consumed to calculate the original throughput and to calculate the improved upper throughput bound Θ. The next columns are its original throughput and the improved upper throughput bound, Θ. The last column shows the relative error of Θ with respect to the original throughput. Due to the size of original graphs, the task of calculating their throughput is an unfeasible task in reasonable time. For this reason, the simulation parameters have been set to a confidence level of 95% and an accuracy of 4%. Owing to this reason, the values of Θ in Table 1 and in Table 2 can slightly vary. The negative relative errors are caused by such confidence level and accuracy degree. As it can be observed in the results shown in Table 2, the improved throughput bound varies from a value really close to the real throughput, to a value which is 13% over the real throughput. The latter case, which deserves further analysis, might be due to the existence of slow cycles far away from the critical cycle.

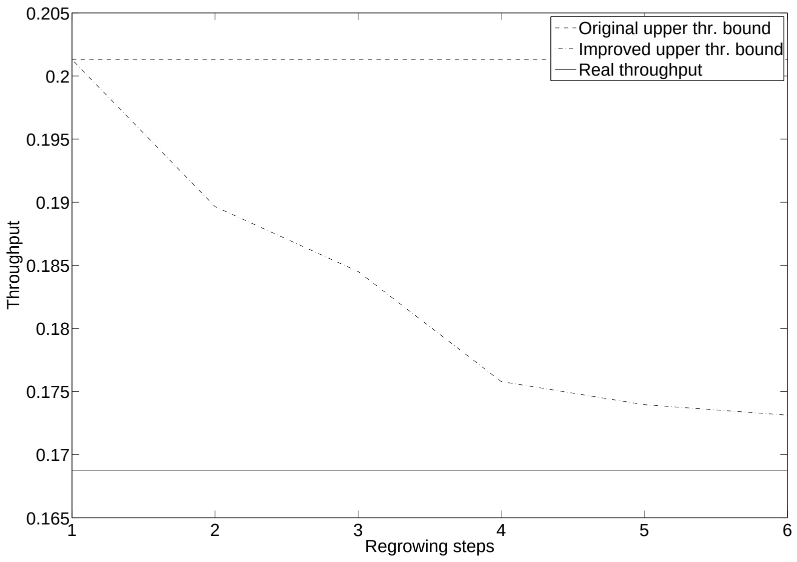

Fig. 4. Throughput of graph `s1488`.

Finally, Figure 4 shows the real throughput of the graph `s1488` (solid line), the original upper throughput bound (dashed line, result of LPP (6)) and the improved upper throughput bound (dot-dashed line) in each step of the strategy. As it can be observed, the improved bound gets close to the real throughput after few steps.

The main results that can be extracted from both tables can be summarised as follows:

– a sharp upper bound is obtained after few regrowing steps;
– the size of the obtained bottleneck is very low compared to the size of the original graph and
– the obtained bottleneck represents the actual constraint for the system throughput, and therefore it can be considered as a potential target to carry out performance optimization.

5 Conclusions

Current system requirements often impose tight constraints on time properties such as system performance. In order to check such requirements, it is necessary to have methods that accurately evaluate the system performance. Moreover, those methods must, not only be accurate, but also efficient in order to be applicable to the increasingly complex systems existing in practice.

The proposed approach is based on an iterative algorithm that takes an initial throughput bound and refines it in each iteration. The initial bound is given by the most constraining (or bottleneck) cycle, i.e., the one with minimum token to delay ratio. The refinements are achieved by adding to the bottleneck cycle places and transitions with low token to delay ratio. The bound is refined until no significant improvement is obtained. Given that most of the steps in the procedure are based on linear programming, the proposed approach exhibits a good trade-off between accuracy and efficiency.

The outputs of the method are an accurate estimate for the steady state throughput, and as a by-product, a subnet representing the bottleneck of the system. The method has been applied to a set of Stochastic Marked Graphs of different sizes. The results show that few iterations suffice to obtain accurate bounds and that, in general, such bounds are due to relatively small subnet bottlenecks of the system. Such system bottlenecks represent the targets on which potential methods for performance optimization might focus.

Acknowledgements

The authors would like to thank Javier Campos for his insightful comments during the elaboration of this paper.

References

1. Ramamoorthy, C.V., Ho, G.S.: Performance Evaluation of Asynchronous Concurrent Systems Using Petri Nets. IEEE Trans. Softw. Eng. **6**(5) (1980) 440–449
2. Burns, S.M.: Performance Analysis and Optimization of Asynchronous Circuits. PhD thesis, Pasadena, CA, USA (1991)
3. Kudva, P., Gopalakrishnan, G., Brunvand, E., Akella, V.: Performance analysis and optimization of asynchronous circuits. In: ICCD, IEEE Computer Society (1994) 221–224
4. Xie, A., Beerel, P.A.: Symbolic Techniques for Performance Analysis of Timed Systems Based on Average Time Separation of Events. In: ASYNC, IEEE Computer Society (1997) 64–75
5. Xie, A., Beerel, P.A.: Accelerating Markovian Analysis of Asynchronous Systems using String- based State Compression. In: ASYNC, IEEE Computer Society (1998) 247–260
6. Greenstreet, M.R., Steiglitz, K.: Bubbles Can Make Self-Timed Pipelines Fast. VLSI Signal Processing **2**(3) (1990) 139–148

7. Campos, J., Chiola, G., Colom, J., Silva, M.: Properties and Performance Bounds for Timed Marked Graphs. IEEE Transactions on Circuits and Systems I: Fundamental Theory and Applications **39**(5) (May 1992) 386–401
8. Chiola, G., Anglano, C., Campos, J., Colom, J., Silva, M.: Operational Analysis of Timed Petri Nets and Application to the Computation of Performance Bounds. In: Proceedings of the 5th International Workshop on Petri Nets and Performance Models, Toulouse, France, IEEE Computer Society Press (October 1993) 128–137
9. Liu, Z.: Performance Bounds for Stochastic Timed Petri Nets. In: Proceedings of the 16th International Conference on Application and Theory of Petri Nets, London, UK, Springer-Verlag (1995) 316–334
10. Xie, A., Kim, S., Beerel, P.A.: Bounding Average Time Separations of Events in Stochastic Timed Petri Nets with Choice. In: ASYNC '99: Proceedings of the 5th International Symposium on Advanced Research in Asynchronous Circuits and Systems, Washington, DC, USA, IEEE Computer Society (1999) 94–107
11. Bernardi, S., Campos, J.: Computation of Performance Bounds for Real-Time Systems Using Time Petri Nets. IEEE Transactions on Industrial Informatics **5**(2) (May 2009) 168–180
12. Brglez, F., Bryan, D., Kozminski, K.: Combinational Profiles of Sequential Benchmark Circuits. Circuits and Systems, 1989., IEEE International Symposium on **3** (May 1989) 1929–1934
13. Silva, M.: Introducing Petri Nets. In: Practice of Petri Nets in Manufacturing. Chapman & Hall (1993) 1–62
14. Murata, T.: Petri Nets: Properties, Analysis and Applications. In: Proceedings of the IEEE. Volume 77. (April 1989) 541–580
15. Florin, G., Natkin, S.: Necessary and Sufficient Ergodicity Condition for Open Synchronized Queueing Networks. IEEE Trans. Softw. Eng. **15**(4) (1989) 367–380
16. Little, J.D.C.: A Proof for the Queuing Formula: L= λ W. Operations Research **9**(3) (1961) 383–387
17. Campos, J., Silva, M.: Structural Techniques and Performance Bounds of Stochastic Petri Net Models. Lecture Notes in Computer Science **609** (1992) 352–391
18. Ramchandani, C.: Analysis of Asynchronous Concurrent Systems by Petri Nets. PhD thesis, Cambridge, MA, USA (1974)
19. Campos, J.: Performance Bounds. In Balbo, G., Silva, M., eds.: Performance Models for Discrete Event Systems with Synchronizations: Formalisms and Analysis Techniques. Editorial KRONOS, Zaragoza, Spain (September 1998) 587–635
20. Carmona, J., Júlvez, J., Cortadella, J., Kishinevsky, M.: Scheduling Synchronous Elastic Designs. In: Proceedings of the 2009 Application of Concurrency to System Design conference (ACSD 2009), Augsburg, Germany (July 2009)
21. Cormen, T.H., Stein, C., Rivest, R.L., Leiserson, C.E.: Introduction to Algorithms. McGraw-Hill Higher Education (2001)
22. The MathWorks: Matlab
`http://www.mathworks.com/` (February 2008) version R2008a.
23. University of Torino: The GreatSPN tool
`http://www.di.unitorino.it/~greatspn` (2002)
24. Ajmone Marsan, M., Balbo, G., Conte, G., Donatelli, S., Franceschinis, G.: Modelling with Generalized Stochastic Petri Nets. Wiley Series in Parallel Computing. John Wiley and Sons (1995)