Testing spnps perfect sampling tool on fork-join queueing networks (tool paper)

Simonetta Balsamo
DAIS - Università Ca' Foscari Venezia
via Torino, 155
Venice, Italy
balsamo@dais.unive.it

Andrea Marin
DAIS - Università Ca' Foscari Venezia
via Torino, 155
Venice, Italy
marin@unive.it

Ivan Stojic
DAIS - Università Ca' Foscari Venezia
via Torino, 155
Venice, Italy
ivan.stojic@unive.it

ABSTRACT

Stochastic Petri nets (SPNs) are widely used for the performance evaluation of computer and telecommunication systems. They inherit from their untimed version the capability of modeling parallel computations in a simple, graphical way. Simulation of SPNs is an important way to assess the performance of a system measured as throughput, response time or expected number of customers/resources in some places. Perfect sampling allows for the selection of the initial state of a simulation with a probability which corresponds to its stationary probability and hence the warm-up period is not required any more. In this paper we present performance tests of spnps, a tool based on some previous works that implements perfect sampling algorithm for SPNs by using decision diagrams. We test performance of the tool on a class of stochastic models of great importance in the quantitative evaluation of distributed systems and communication networks: the fork-join queueing networks.

Categories and Subject Descriptors

C.4 [**PERFORMANCE OF SYSTEMS**]: Modeling techniques

General Terms

Performance

Keywords

Stochastic Petri nets, Simulation, Perfect sampling

Permission to make digital or hard copies of all or part of this work for
VALUETOOLS 2016, October 25-28, Taormina, Italy

DOI 10.4108/eai.25-10-2016.2266454

1. INTRODUCTION

Performance evaluation of parallel and distributed systems requires the capability of modeling the evolution of different tasks and their competition or their synchronization on sets of finite resources. In the literature, Timed Petri nets have been used for the performance evaluation of microprocessor systems [16], communication networks [14, 20] and other fields such as biological systems [1]. Stochastic Petri nets (SPNs) [18] are a subset of timed Petri nets for which firing delays associated with Petri net transitions are independent of each other (but they may depend on the model's state) and are exponentially distributed. It can be shown that the stochastic process underlying an SPN is a continuous-time Markov chain (CTMC). Unfortunately, in the literature, very few analytical results are available for the solution of SPNs and in general the exact computation of the stationary performance indices is extremely inefficient. For this reason, SPN models are usually studied by means of simulation [11]. Several tools are available for simulating SPNs among which we recall Timenet [22], GreatSPN [8], SPNP [13]. One of the problems in the stationary simulation of stochastic models (including SPNs) is the detection of the so called *warm-up* period, i.e., the period required by the model to move from its initial condition to the equilibrium (if it exists). There are several statistical approaches for the detection of the length of transient phase in a simulation which can be formulated for SPNs (see e.g., [15]); however, they usually depend on the users' skills (such as the Welch's graphical method) and hence are prone to errors. For ergodic models with finite state spaces, perfect sampling is a method to sample a reachable state of the model with a probability that corresponds to its stationary probability, i.e., we sample a reachable state according to the steady-state distribution of the SPN.

In this paper we present performance tests of a tool called Stochastic Petri Nets Perfect Sampling (spnps) that implements *coupling from the past* [19] algorithm for SPNs with finite state spaces. Spnps uses decision diagrams to contain the problem of the state space explosion and its strength lies in the generality of the models that it can handle. Although in the worst case the time and space complexities of spnps are more than exponential with respect to the net structure (since the whole net state space must be built), we observed that in many practical cases the structure of the model's underlying reachability graph has regularities that can be efficiently exploited by decision diagrams. For this reason, we test spnps on a class of models which are

known to play an important role in the analysis of distributed systems and telecommunication systems: fork-join queueing networks [12, 7]. For these models very few exact results are known [2] mainly relying on product-form theory. Therefore, the definition of techniques for approximation and simulation are the principal approaches for the estimation of their stationary performance indices. In [9, 5] the authors present algorithms for perfect sampling on restricted classes of fork-join queueing networks (in the former case consisting of one node, in the latter with limitation on the probabilistic routing). They take advantage of the reachability graph properties in order to improve the efficiency of the algorithms. Similarly, another approach [6] is applicable to a different class of fork-join queueing networks; however, in the queueing networks considered there, customer routing cannot depend on the total number of customers in a set of queues, while in the models that we consider, queues can have a shared capacity (i.e., restriction on the total number of customers in a set of queues) and a customer that cannot enter a destination queue due to shared capacity being full is blocked. Further research is needed in order to assess the possibility of applying to the models considered here the general method proposed in [6]. While spnps is less efficient than these approaches for particular classes of models, its generality allows for studying SPNs (and hence fork-join queueing networks that are representable as SPNs) with a more general structure. In particular, we show that for fork-join queueing networks, the time required for performing a perfect sampling is acceptable even for relatively large nets.

The paper is structured as follows. In Section 2 we briefly describe the tool interface and the supported class of SPNs and we outline the algorithm implemented by the tool. Next, in section 3 we introduce models used in testing, describe the testing procedure and discuss the performance of the tool. Finally, Section 4 concludes the paper.

2. SPNPS: TOOL DESCRIPTION

Spnps is a command-line tool that takes in input an SPN model specified in PNML [21], place marking bounds, i.e., the maximum number of tokens that a place can contain, and an ordering of places for the construction of the decision diagram.

When used for its main purpose, perfect sampling, the output of the tool is a specified number of perfect samples from the reachability set of the input SPN. In addition, the user can specify as a stopping criterion the maximum cardinality of the final set $\mathcal{A}_0$ in Algorithm 1 (1 is the default value) and in this case the sampling runs are stopped before completion and the entire set $\mathcal{A}_0$ is returned as the result. Performing such incomplete perfect sampling runs can be useful in the analysis of models that exhibit multimodal behaviour.

Spnps is a stand-alone tool and as such can be used in a preprocessing step to provide initial markings for stationary simulation runs that can then be performed by other tools. C++ source code of the tool can be obtained at the web page of one of the authors[1]. For a more detailed description of the tool, the interested reader can refer to [4].

2.1 Assumptions on the SPN

The class of SPNs that we consider are those introduced in [18] that are standard Petri nets in which each transition is associated with a firing delay that may depend on the model's state. The firing delays are exponentially distributed random variables. It can be shown that under these assumptions the stochastic process underlying the SPN is a CTMC and that the reachability set of the untimed Petri net is identical to the state space of the CTMC underlying the model. Spnps works on SPNs with finite state spaces and whose transitions implement one of the following firing semantics. *Single server (SS) semantics*, i.e., the firing delay is set when the transition is first enabled and a new delay is sampled in case the same transition is enabled after the firing. In other words, the firing rate of an enabled transition is state independent. *Infinite server (IS) semantics*, i.e., every enabling set of tokens is processed in parallel as soon as they arrive at the input places. Each of these concurrent delays associated with a transition are i.i.d. exponentially distributed random variables. According to the race policy, this corresponds to a single server semantics in which the firing rate depends on the marking of the transition's input places.

2.2 The algorithm behind spnps

This subsection contains a high-level overview of the perfect sampling algorithm as implemented in the present tool, while the details and a version of the algorithm that supports more general transition firing semantics can be found in a previous paper by the same authors [3]. The algorithm is based on coupling from the past [19], which is an algorithm for sampling from the stationary distributions of discrete time Markov chains (DTMC)—the tool simulates the DTMC obtained by uniformization of the original CTMC starting from all states and performs the simulation until all states couple into a single state. Algorithm 1 outlines main steps of the algorithm implemented in the present tool.

Algorithm 1: PERFECT_SAMPLING

Data:
- SPN $(\mathcal{P}, \mathcal{T}, I, O, W, \boldsymbol{m}_0)$,
- function $S : \mathcal{T} \to \{\text{SS}, \text{IS}\}$ defining transitions' firing semantics,
- vector of place marking bounds $\boldsymbol{B} \in \mathbb{N}^{|\mathcal{P}|}$,
- function $V : \mathcal{P} \to \{0, 1, \ldots, |\mathcal{P}| - 1\}$ defining ordering of places,
- i.i.d. uniform on $[0, 1]$ random variables $U_0, U_{-1}, \ldots$

Result: Sample from the stationary distribution of the SPN

```
1  begin
2      RS ← GENERATE_STATE_SPACE(SPN, B, V);
3      φ ← GENERATE_UPDATE_RULE(SPN, S, B, V);
4      m ← 1;
5      repeat
6          A_{-m} ← RS;
7          for i = −m + 1 to 0 do
8              A_i ← φ(A_{i−1}, U_i);
9          m ← 2m;
10     until |A_0| = 1;
11     return s ∈ A_0;
```

In line 2 state space $\mathcal{RS}$ of the CTMC underlying the SPN is generated and encoded by a multi-way decision diagram (MDD). The generation of the state space is performed by saturation algorithm [17] which can efficiently generate

[1] `http://www.dais.unive.it/~stojic/soft.html`

MDDs encoding very large state spaces for many types of practical models. Then, in line 3, several MDDs that together encode a simulation update rule $\phi : 2^{\mathcal{RS}} \times [0,1] \to 2^{\mathcal{RS}}$ are generated. Simulation update rule ϕ respects step probabilities of a DTMC that is obtained by uniformization from the CTMC underlying the SPN model: i.e., if $s_i, s_j \in \mathcal{RS}$ are two states of the DTMC with the corresponding single step probability p_{ij} and U is uniformly on $[0,1]$ distributed random variable then the following holds:

$$\Pr\{\phi(\{s_i\}, U) = \{s_j\}\} = p_{ij}.$$

To generate MDDs that encode ϕ we exploit the specific structure of a CTMC underlying an SPN, namely that there is only a limited number of ways that the marking can change (one for each transition) and that there is only a limited number of different transition rates in the CTMC (one for each combination of a transition and its enabling degree). Given an MDD that encodes a set of states $\mathcal{A}_{i-1} \subseteq \mathcal{RS}$ and a sample from U, the update rule ϕ performs one simulation step (of the DTMC obtained by uniformization of the CTMC) from all states in the set $\mathcal{A}_{i-1}$, producing an MDD that encodes set $\mathcal{A}_i$ of states that are reached from some state in $\mathcal{A}_{i-1}$ by the simulation step. Since the simulation step can couple some of the states in $\mathcal{A}_{i-1}$ (i.e., performing the simulation step from different states can produce the same state), the resulting set has equal or smaller cardinality, $|\mathcal{A}_i| \leq |\mathcal{A}_{i-1}|$. This process can then be continued until a set containing only a single state is obtained. Lines 4 to 10 perform such simulations for increasing number m of steps until the resulting set $\mathcal{A}_0$ contains only a single state $s \in \mathcal{RS}$ which is returned as the sample. Update rule ϕ and the algorithm are constructed so that the probabilities of sampling each of the states in $\mathcal{RS}$ are equal to their stationary probabilities in the CTMC underlying the SPN. The number m of steps that produces the coupling depends on the properties of the model and on the update rule ϕ, and its expected value is finite for all SPNs with finite state spaces [3]. Thus the algorithm terminates in a finite expected time.

3. TESTING THE TOOL ON FORK-JOIN QUEUEING NETWORKS

Fork-join queueing networks allow the jobs to be split into several tasks that are processed in parallel. Once all the tasks have been served, a join operation is performed and we consider the original job served. These models are very useful for the performance evaluation of distributed systems with task concurrency and synchronization. In this section we test spnps on a fork-join queueing network and discuss the performance of the perfect sampling algorithm that it implements.

3.1 The testing model

We consider the fork-join queueing network depicted in Figure 1. The model is described by parameters: n, m, r. Figure 1 shows the leftmost service station Q_0 that initially contains m jobs. Once a job is served, it is forked into n sub-tasks which are enqueued in $Q_1, \ldots, Q_n$, respectively. After being served by station Q_i, the sub-task i is routed to one of the stations $Q_{i1f}, \ldots, Q_{inf}$ according to a uniform probabilistic choice: this is the first phase of service. The second phase is performed by stations $Q_{i1s}, \ldots Q_{ins}$ with a constraint on the available resources: at most r sub-tasks can simultaneously be in the queues $Q_{i1s}, \ldots, Q_{ins}$ for each $i = 1, \ldots, n$. All the service times are independent and exponentially distributed and the stations are equipped with a single server. We assume a processor sharing queueing discipline.

The queueing network for parameters $n = 2$, $m = 3$, $r = 1$ is modeled by the Petri net in Figure 2. There are $m = 3$ tokens in place P_0, representing jobs. Transition T_1 models the fork and T_{14} the join. The resources are modelled as tokens in places P_{r1} and P_{r2}. Places P_{1j} and P_{2j} model the waiting room for sub-tasks which have been served and are waiting to be joined. The remaining places model the waiting/service rooms of the corresponding queueing stations. The resulting Petri net has $2n^2 + 3n + 1$ places and $3n^2 + 2$ transitions.

We make two important observations on the model of Figure 2: the first is that the SPN for $n \geq 2$ is *not* a free choice SPN. In fact, in a free choice Petri net if there is an arc from a place s to a transition t, then there must be an arc from any input place of t to any output transition of s [10]. Now, consider the net of Figure 2 and let s be for example P_{r1} and t be transition T_4. Consider the input place P_{11f} of T_4 and the output transition T_5 of P_{r1}: since there is no arc from P_{11f} to T_5 the net is not free choice. Therefore, the optimised algorithms for perfect sampling of free choice SPNs cannot be applied (see [5] and the references therein). The second observation is on the semantics of the join specified in terms of SPN. In fact T_{14} does not guarantee that the join occurs among the sibling sub-tasks since a sub-task may overtake another sub-task that was previously forked. The more detailed representation of the join operation requires a much more complicated stochastic process and is out of the scope of this paper.

3.2 Performance evaluation

To test the performance of spnps on the fork-join model we performed two groups of tests.

In the first group of tests, we study how the performance scales with number of initial jobs. We fixed the size of the SPN by setting $n = 2$ and run the tests for parameter m ranging from 2 to 40 with step 2 and with number of resources per fork equal to half the number of jobs, $r = m/2$.

In the second group, we examine how the performance scales with the size of the SPN by fixing the number of jobs to $m = 2$ and number of resources per fork to $r = 1$ and running the tests for parameter n ranging from 2 to 16.

For each of these $20 + 15 = 35$ sets of parameters, 50 independent tests were run by performing the perfect sampling procedure with different random seeds. Rates of all transitions of the tested SPN models were set to 1. We report the results as 95% confidence intervals for the means of measured values.

Testing was done on a GNU/Linux system with Intel(R) Core(TM) i5-2310 CPU with maximum clock of 3.2 GHz and with 8GB of main memory.

The results of the testing are shown in Table 1 and 2 for the first and second batch of tests, respectively. The meaning of the columns is the following:

- n_P: number of places of the SPN.
- n_T: number of transitions of the SPN.
- $|\mathcal{RS}|$: cardinality of reachability set of the SPN.

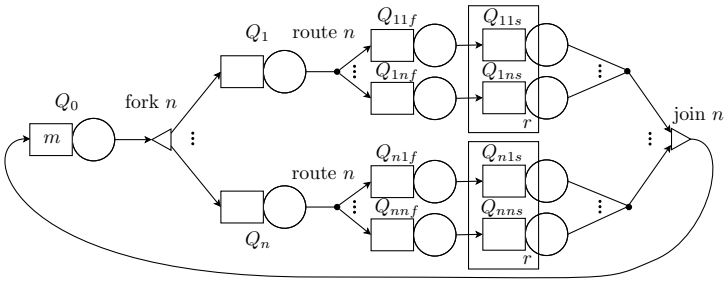


Figure 1: Example of fork-join queueing network.

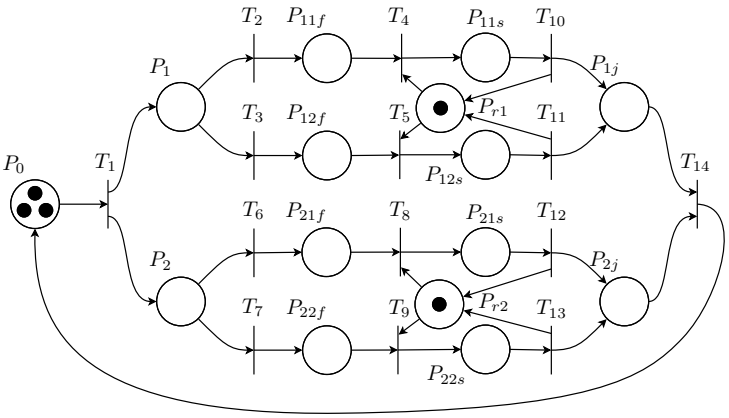


Figure 2: SPN associated with the fork-join queueing network of Figure 1 for $n = 2$, $m = 3$ and $r = 1$.

- init time (s): time in seconds spent in generating decision diagrams that encode the reachability set and simulation update rule used in perfect sampling. If multiple samples are generated, this needs to be done only once and these decision diagrams can be reused in different sampling runs (i.e., this time can be amortized over the sampling runs). The init times are independent of the random seed, resulting in very tight confidence intervals; we therefore report only the means.

- sampling time (s): 95% confidence interval for mean time in seconds spent in a perfect sampling run.

- iterations: 95% confidence interval for mean number of simulation steps in a perfect sampling run.

- peak memory (kB): 95% confidence interval for mean peak memory use in kilobytes for the entire program. The program loads the SPN from a PNML file, parses it and runs the perfect sampling. The non-sampling parts use about 5MB of memory.

Common magnitudes of intervals' boundaries are shown outside the intervals.

We observe that the tool scales very well with the size $|\mathcal{RS}|$ of the reachability set when we vary the size of the net (second group of tests), but not as well when we vary the size of the initial marking (first group of tests). Figure 3 shows log-log plots of memory use on the left and sampling time on the right as functions of the reachability set size. For the model from the first group of tests with largest tested reachability set size $|\mathcal{RS}| \approx 4.857 \times 10^{12}$, the memory consumption is comparable to the model from the second group of tests for which $|\mathcal{RS}| \approx 1.152 \times 10^{26}$, and the sampling time is comparable to the model from the second group of tests with the largest tested $|\mathcal{RS}| \approx 3.882 \times 10^{42}$.

As performance of the algorithm depends in large part on efficiency of MDDs that are used to encode subsets of the reachability set and the simulation update rule, and efficiency of MDDs depends on the properties of the sets or functions that they encode, the observed difference in performance between the two groups of tests is likely in part due to the structure of the reachability sets. If we consider the reachability sets of the n SPN components between fork and join transitions, then for $m = 1$ the reachability set of the entire SPN is equal to the union of a singleton set containing initial marking (where the single token is in the initial place) and the Cartesian product of the n components' reachability sets restricted to the case when the initial place is empty. This is highly structured and is well suited to representation using decision diagrams (but note that the high symmetry of the reachability sets is not exploited in the decision diagram representation, only the high decomposability; i.e., the important property is that the reachability set is similar to a Cartesian product and not that the sets in the Cartesian product are equal). Similar considerations apply for other small values of m. In contrast, for small n and large m, the reachability set is not as well structured (we obtain a large union of $m + 1$ Cartesian products over components).

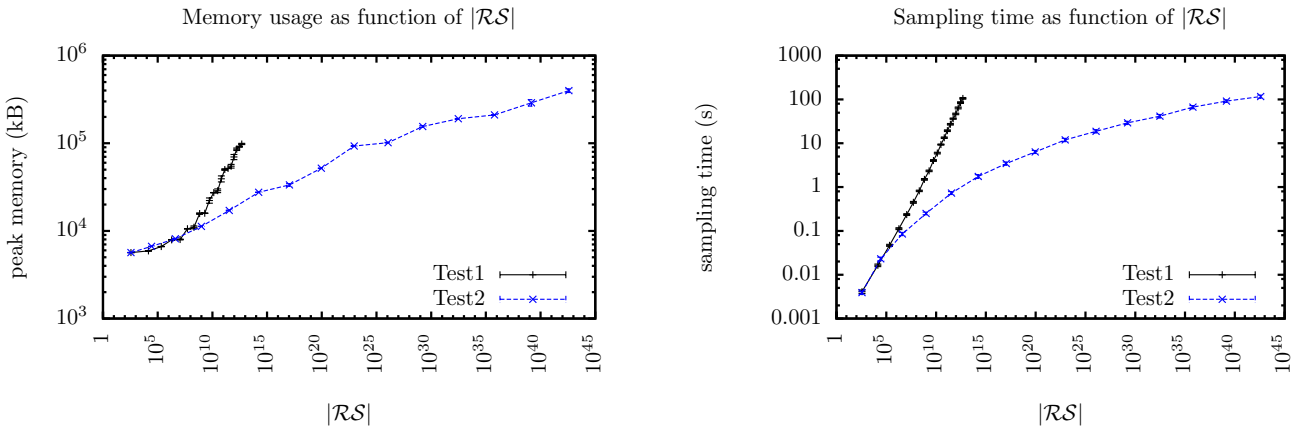
In addition, because the implementation encodes reachability sets and their subsets by decision diagrams in which

m	r	$\lvert\mathcal{RS}\rvert$	init time (s)	sampling time (s)	iterations	peak memory (kB)
2	1	361	3.204×10^{-3}	$[3.735, 4.489] \times 10^{-3}$	$[2.386, 3.169] \times 10^{2}$	$[5.638, 5.682] \times 10^{3}$
4	2	14207	5.294×10^{-3}	$[1.530, 1.754] \times 10^{-2}$	$[3.124, 4.095] \times 10^{2}$	$[5.900, 5.945] \times 10^{3}$
6	3	214492	8.835×10^{-3}	$[4.449, 4.929] \times 10^{-2}$	$[4.134, 5.286] \times 10^{2}$	$[6.618, 6.659] \times 10^{3}$
8	4	1.865×10^{6}	1.372×10^{-2}	$[1.068, 1.181] \times 10^{-1}$	$[5.372, 6.711] \times 10^{2}$	$[7.836, 7.981] \times 10^{3}$
10	5	1.130×10^{7}	2.134×10^{-2}	$[2.250, 2.472] \times 10^{-1}$	$[5.998, 7.723] \times 10^{2}$	$[7.897, 8.074] \times 10^{3}$
12	6	5.290×10^{7}	3.101×10^{-2}	$[4.277, 4.707] \times 10^{-1}$	$[6.945, 8.415] \times 10^{2}$	$[1.053, 1.062] \times 10^{4}$
14	7	2.045×10^{8}	4.568×10^{-2}	$[7.911, 8.528] \times 10^{-1}$	$[0.888, 1.057] \times 10^{3}$	$[1.057, 1.141] \times 10^{4}$
16	8	6.809×10^{8}	6.338×10^{-2}	$[1.434, 1.568]$	$[1.101, 1.357] \times 10^{3}$	$[1.546, 1.606] \times 10^{4}$
18	9	2.012×10^{9}	8.897×10^{-2}	$[2.255, 2.441]$	$[1.219, 1.505] \times 10^{3}$	$[1.583, 1.597] \times 10^{4}$
20	10	5.392×10^{9}	1.185×10^{-1}	$[3.861, 4.229]$	$[1.728, 2.246] \times 10^{3}$	$[2.072, 2.379] \times 10^{4}$
22	11	1.332×10^{10}	1.554×10^{-1}	$[5.724, 6.221]$	$[1.563, 2.000] \times 10^{3}$	$[2.721, 2.747] \times 10^{4}$
24	12	3.071×10^{10}	1.998×10^{-1}	$[9.056, 9.640]$	$[2.242, 2.796] \times 10^{3}$	$[2.739, 3.011] \times 10^{4}$
26	13	6.673×10^{10}	2.606×10^{-1}	$[1.286, 1.389] \times 10^{1}$	$[2.359, 3.089] \times 10^{3}$	$[3.620, 4.244] \times 10^{4}$
28	14	1.378×10^{11}	3.263×10^{-1}	$[1.854, 2.021] \times 10^{1}$	$[2.896, 3.822] \times 10^{3}$	$[4.916, 5.118] \times 10^{4}$
30	15	2.719×10^{11}	4.086×10^{-1}	$[2.607, 2.824] \times 10^{1}$	$[3.399, 4.466] \times 10^{3}$	$[5.150, 5.164] \times 10^{4}$
32	16	5.158×10^{11}	5.004×10^{-1}	$[3.521, 3.757] \times 10^{1}$	$[3.759, 4.843] \times 10^{3}$	$[5.207, 5.737] \times 10^{4}$
34	17	9.446×10^{11}	6.199×10^{-1}	$[4.481, 4.871] \times 10^{1}$	$[4.043, 5.378] \times 10^{3}$	$[6.514, 7.456] \times 10^{4}$
36	18	1.676×10^{12}	7.517×10^{-1}	$[6.095, 6.723] \times 10^{1}$	$[4.641, 6.869] \times 10^{3}$	$[8.265, 8.833] \times 10^{4}$
38	19	2.890×10^{12}	9.036×10^{-1}	$[8.086, 8.860] \times 10^{1}$	$[4.897, 6.694] \times 10^{3}$	$[9.042, 9.256] \times 10^{4}$
40	20	4.857×10^{12}	1.080	$[1.011, 1.106] \times 10^{2}$	$[5.392, 7.552] \times 10^{3}$	$[9.651, 9.976] \times 10^{4}$

Table 1: First group of tests with $n = 2$, m ranging from 2 to 40 and $r = m/2$. The net has 15 places and 14 transitions.

n	n_P	n_T	$\lvert\mathcal{RS}\rvert$	init time (s)	sampling time (s)	iterations	peak memory (kB)
2	15	14	361	3.204×10^{-3}	$[3.589, 4.216] \times 10^{-3}$	$[2.189, 2.829] \times 10^{2}$	$[5.645, 5.679] \times 10^{3}$
3	28	29	27513	9.316×10^{-3}	$[2.081, 2.523] \times 10^{-2}$	$[5.947, 8.389] \times 10^{2}$	$[6.654, 6.691] \times 10^{3}$
4	45	50	4.111×10^{6}	2.401×10^{-2}	$[7.792, 9.251] \times 10^{-2}$	$[1.342, 1.812] \times 10^{3}$	$[8.101, 8.181] \times 10^{3}$
5	66	77	9.927×10^{8}	5.041×10^{-2}	$[2.329, 2.687] \times 10^{-1}$	$[2.295, 3.030] \times 10^{3}$	$[1.123, 1.129] \times 10^{4}$
6	91	110	3.513×10^{11}	9.699×10^{-2}	$[6.775, 7.887] \times 10^{-1}$	$[3.848, 5.245] \times 10^{3}$	$[1.698, 1.726] \times 10^{4}$
7	120	149	1.714×10^{14}	1.656×10^{-1}	$[1.572, 1.925]$	$[5.368, 7.657] \times 10^{3}$	$[2.754, 2.771] \times 10^{4}$
8	153	194	1.103×10^{17}	2.890×10^{-1}	$[3.153, 3.755]$	$[7.363, 9.594] \times 10^{3}$	$[3.229, 3.460] \times 10^{4}$
9	190	245	9.065×10^{19}	5.198×10^{-1}	$[5.843, 6.849]$	$[0.889, 1.176] \times 10^{4}$	$[5.155, 5.226] \times 10^{4}$
10	231	302	9.261×10^{22}	7.359×10^{-1}	$[1.087, 1.298] \times 10^{1}$	$[1.432, 1.911] \times 10^{4}$	$[9.214, 9.395] \times 10^{4}$
11	276	365	1.152×10^{26}	1.223	$[1.715, 2.031] \times 10^{1}$	$[1.697, 2.203] \times 10^{4}$	$[1.009, 1.025] \times 10^{5}$
12	325	434	1.714×10^{29}	1.616	$[2.683, 3.184] \times 10^{1}$	$[1.991, 2.793] \times 10^{4}$	$[1.522, 1.585] \times 10^{5}$
13	378	509	3.006×10^{32}	2.558	$[3.772, 4.537] \times 10^{1}$	$[2.097, 2.785] \times 10^{4}$	$[1.888, 1.920] \times 10^{5}$
14	435	590	6.141×10^{35}	3.175	$[6.133, 7.234] \times 10^{1}$	$[3.093, 4.247] \times 10^{4}$	$[2.101, 2.114] \times 10^{5}$
15	496	677	1.445×10^{39}	4.103	$[8.546, 9.790] \times 10^{1}$	$[3.386, 4.216] \times 10^{4}$	$[2.653, 3.148] \times 10^{5}$
16	561	770	3.882×10^{42}	6.012	$[1.067, 1.259] \times 10^{2}$	$[3.452, 4.675] \times 10^{4}$	$[3.798, 4.173] \times 10^{5}$

Table 2: Second group of tests with n ranging from 2 to 16, $m = 2$ and $r = 1$.



(a) Memory consumption

(b) Time consumption

Figure 3: Comparison of the memory usage and the sampling time for the first and the second batch of tests.

levels correspond to Petri net places, enlarging the net makes the decision diagrams taller (more levels), while enlarging the marking makes them wider (more pointers per node). Since the decision diagrams gain efficiency, in comparison to a tree representation of the reachability set, by reusing nodes on lower levels, this makes them likely to be more efficient in the former case—in taller diagrams there are more levels and reusing nodes deeper in the diagram is often more profitable for efficiency.

4. CONCLUSION

We have presented the tool spnps which implements a variant of the coupling from the past algorithm for perfect sampling in bounded SPNs. Spnps uses decision diagrams as a data structure to perform the perfect sampling and we have shown that when the SPN has a certain regularity in its reachability set, the sampling can be done efficiently even in nets with very large state spaces.

We have evaluated the performance of the tool on SPN models of fork-join queueing networks with some resource competition. The results of our tests suggest that the algorithm as implemented is likely to be more efficient for sampling in models that are composed of many loosely coupled components. Future work will explore the possibility of improving the tool performance by using different encodings. In fact, as mentioned before, currently each SPN place is encoded by a single level in the decision diagram, while multiple levels per place or multiple places per level could be used.

Acknowledgements

Work partially supported by MIUR fund Fondo per il sostegno dei giovani "Programma strategico: ICT e componentistica elettronica".

5. REFERENCES

[1] Baldan, P., Cocco, N., Marin, A., and Simeoni, M. Petri nets for modelling metabolic pathways: a survey. *Natural Computing 9*, 4 (2010), 955–989.

[2] Balsamo, S., Harrison, P. G., and Marin, A. Methodological Construction of Product-form Stochastic Petri-Nets for Performance Evaluation. *J. of System and Software 85*, 7 (2012), 1520–1539.

[3] Balsamo, S., Marin, A., and Stojic, I. Perfect sampling in stochastic Petri nets using decision diagrams. In *Proc. of 23rd Int. Symp. on Modeling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS)* (2015), pp. 126–135.

[4] Balsamo, S., Marin, A., and Stojic, I. *Spnps: A Tool for Perfect Sampling in Stochastic Petri Nets.* Springer International Publishing, Cham, 2016, pp. 163–166.

[5] Bouillard, A., and Gaujal, B. Backward coupling in bounded free-choice nets under Markovian and non-Markovian assumptions. *Discrete Event Dynamic Systems 18*, 4 (2008), 473–498.

[6] Bušić, A., Gaujal, B., and Vincent, J.-M. Perfect simulation and non-monotone Markovian systems. In *Proceedings of the 3rd International Conference on Performance Evaluation Methodologies and Tools* (ICST, Brussels, Belgium, 2008), ValueTools '08, ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), pp. 27:1–27:10.

[7] Chen, R. An upper bound solution for homogeneous fork/join queueing systems. *IEEE Trans. on Parallel and Distributed Systems 22*, 5 (2011), 874–878.

[8] Chiola, G., Franceschinis, G., Gaeta, R., and Ribaudo, M. GreatSPN 1.7: Graphical editor and analyzer for timed and stochastic Petri nets. *Perf. Eval. 24* (1995), 47–68.

[9] Dai, H. Exact simulation for fork-join networks with heterogeneous service. *Int. J. of Statistics and Probability 4*, 1 (2015).

[10] Desel, J., and Esparza, J. *Free choice Petri nets.* Cambridge Tracts in Theoretical Computer Science 40. Cambridge University Press, 1995.

[11] Haas, P. *Stochastic Petri Nets: Modelling, Stability, Simulation.* Springer Series in Operations Research. Springer, 2002.

[12] Heidelberger, P., and Trivedi, K. Analytic queueing models for programs with internal concurrency. *IEEE Trans. Comput. C-32* (1983), 73–82.

[13] Hirel, C., Tuffin, B., and Trivedi, K. S. SPNP: stochastic Petri nets. Version 6.0. In *Computer Performance Evaluation: Modelling Techniques and Tools, 11th International Conference, TOOLS 2000, Schaumburg, IL, USA, March 27-31, 2000, Proceedings* (2000), pp. 354–357.

[14] Kant, K. *Introduction to Computer System Performance Evaluation.* McGraw-Hill, 1992.

[15] Law, A. M., and Kelton, W. D. *Simulation modeling and analysis*, 3rd ed. McGraw-Hill, 2000.

[16] Marsan, M. A., Conte, G., and Balbo, G. A class of generalized stochastic Petri nets for the performance evaluation of multiprocessor systems. *ACM Trans. Comput. Syst. 2*, 2 (1984), 93–122.

[17] Miner, A. Saturation for a general class of models. In *Proc. of Quantitative Evaluation of Systems (QEST)* (2004), pp. 282–291.

[18] Molloy, M. K. Performance analysis using stochastic Petri nets. *IEEE Trans. on Comput. 31*, 9 (1982), 913–917.

[19] Propp, J. G., and Wilson, D. B. Exact sampling with coupled Markov chains and applications to statistical mechanics. *Random Struct. Algorithms 9*, 1-2 (1996), 223–252.

[20] Tutsch, D. *Performance analysis of network architectures.* Springer, 2006.

[21] Weber, M., and Kindler, E. The Petri net markup language. In *Petri Net Technology for Communication-Based Systems*, vol. 2472 of *Lecture Notes in Computer Science.* Springer Berlin Heidelberg, 2003, pp. 124–144.

[22] Zimmermann, A., Freiheit, J., German, R., and Hommel, G. Petri net modelling and performability evaluation with TimeNET 3.0. In *TOOLS '00: Proc. of the 11th Int. Conf. on Computer Performance Evaluation: Modelling Techniques and Tools* (London, UK, 2000), Springer-Verlag, pp. 188–202.