FlowSim Simulation Benchmarking Platform

Jonathan Bogdoll, Holger Hermanns, Lijun Zhang

Department of Computer Science, Saarland University, Germany
{bogdoll,hermanns,zhang}@cs.uni-sb.de

***Abstract*— *Probabilistic* model checking has emerged as a versatile system verification approach, but is frequently facing state-space explosion problems. One promising attack to this is to construct an abstract model which *simulates* the original model, and to perform model checking on that abstract model. Recently, efficient algorithms and optimizations for deciding simulation of probabilistic models have been proposed, which reduces the theoretical complexity bounds of existing algorithm drastically. In this paper we present a tool to compare the performance of these approaches for deciding the simulation preorder of probabilistic models.**

I. Introduction

System performance and dependability become more and more important with the ubiquity of computing systems. Discrete-time and continuous-time Markov chains (DTMCs and CTMCs) [1] are widely used to model and analyze performance and dependability of such probabilistic systems. Probabilistic automata (PAs) [2] extend DTMCs with nondeterminism, which constitute a natural model of concurrent computation involving random phenomena. Similarly, continuous-time probabilistic automata (CPAs) [3] are obtained by extending CTMCs with nondeterminism, just as PAs extend DTMCs. CPAs are a natural semantic model for various performance and dependability modelling formalisms including stochastic activity networks [4], generalised stochastic Petri nets [5].

Model checking has successfully been applied to automatically verify whether the probabilistic system satisfies a specification, which is expressed by PCTL [6] formulas for discrete-time models or CSL [7] formulas for continuous-time models. For instance the probabilistic reachability property "the probability to reach a set of bad states is at most 3%" can be expressed in PCTL. Model checkers, such as PRISM [8] and MRMC [9], are available for checking such properties over these probabilistic models. However, this approach is limited by the infamous state space explosion problem. Among other solutions, simulation relations [10], [2] have been proposed for these models, which, in correspondence to the non-probabilistic setting, preserve relevant fragments of the logics PCTL and CSL, respectively.

Let m and n denote the number of transitions and states respectively. For PAs, Baier *et al.* [11] introduced a polynomial decision algorithm with time complexity $\mathcal{O}((mn^6 + m^2n^3)/\log n)$ and space complexity $\mathcal{O}(n^2)$, by exploiting a network flow algorithm. Exploiting parametric maximum flow algorithms, the time complexity has been drastically improved in [12] to $\mathcal{O}(m^2n)$, while the space complexity is increased to $\mathcal{O}(m^2)$. Since space requirements often become the practical bottleneck, a space-efficient algorithm based on partition refinement has also been studied in [13] for probabilistic models.

In this paper we present the tool FlowSim to compare the performance of these approaches for deciding simulation of both discrete-time and continuous-time probabilistic models. FlowSim supports additional useful optimizations of the algorithms proposed in [14]. FlowSim is free software under the terms of the GNU General Public License as published by the Free Software Foundation.

This work is supported by the DFG as part of the Transregional Collaborative Research Center SFB/TR 14 AVACS, and by the European Community's Seventh Framework Programme under grant agreement n^o 214755.

II. Key Features

Primarily, FlowSim is used to measure the time and memory resources needed to find the simulation relation for a probabilistic model. Starting from the algorithm in [11], we consider different improvements and optimizations:

- *State partitioning* – In the first iteration of the refinement loop [11], states fulfilling certain similarity criteria are grouped into equivalence classes and simulation is decided between representative states instead of all states.
- *Parametric maximum flow* – An algorithm [12] which drastically improves the theoretical time complexity, but uses more space.
- *P-Invariant checking* – A heuristic which verifies the simple condition on maximum flow networks that source-side edge capacities must be less or equal to the sum of sink-side edge capacities connected to it. This can sometimes avoid more complex computations.
- *Significant arc detection* – A combination of parametric maximum flow and P-Invariant checking which immediately discards certain maximum flow networks. A network is discarded if an edge deletion causes the P-Invariant to be violated.
- *Quotient* – A space efficient algorithm [13] based on partition refinement.

The above improvement and optimisations can be applied for concrete input models (dtmc,ctmc,pa,cpa). Moreover, a key feature of FlowSim is the ability to work on random models (random) generated on demand. This allows the user to investigate the properties of individual algorithms with respect to certain model properties while abstracting to some degree from the statistical interdependencies inherent in "real world" models. The structure of these randomly generated models can be adjusted in a gradual fashion. For example, in *uniformly random models*, any two states are equally likely to have a transition going from one to the other, but it is possible to modify this likelihood in order to create clusters in the model where the transition density is higher within certain clusters and lower between these clusters. We refer to such a deviation from uniformity as a *bias*. The following biases can be specified:

- *probability bias*: $pb \in [0; 1]$, which describes whether the transition probabilities from one state to a set of successors are distributed uniformly across that set ($pb = 0$) or randomly ($pb = 1$).
- *fanout bias*: $fb \in [-1; 1]$, which defines if a state is more likely to have the minimum ($fb < 0$) or maximum ($fb > 0$) number of successors.
- *linearity bias*: $lb \in [0; 1]$, which defines how likely it is that the model contains cycles: At $lb = 0$, transitions are completely random and thus likely to form cycles, at $lb = 1$, the model is acyclic.

- *clustering bias*: $cb \in [0; 1]$ together with an integer number of clusters, *cb* specifies the degree of separation between randomly chosen clusters. At $cb = 0$, no clusters are formed whereas at $cb = 1$, there is no transition between any two states of two different clusters.
- *successor bias*: $sb \in [0; 1]$ specifies the likelihood for a certain, randomly chosen set of states to be chosen as successors more frequently than the remaining states. At $sb = 0$, no bias is present; at $sb = 1$, there is at least one common successor for all states.

Randomly generated models are specified as 16-tuples stored as space-separated sequences. Consider the following example:

```
200 4 16 1 $x 0 0 0 0 5 -1 1 0 0 10 2
```

This sequence is interpreted as a random model with 200 states, each of which has between 4 and 16 successors. Two different labels are distributed across the state space. Furthermore, the fanout bias will be set to ten different values between -1 and 1. The precise meaning of each element is explained further in the `FlowSim` manual. Biases can also be used as plot variables, i.e. it is possible to create a plot over the effect of a bias on a certain resource, such as time or memory.

Averaging the results over a number of different runs makes the results less prone to random fluctuations. Similarly, generating multiple instances of a random model with certain parameters helps isolate the effects of model properties from those of random fluctuations. These settings can all be adjusted by the user.

The user interface of our tool is currently via command-line. The results can be printed in plain text, in LaTeX table format ready to be included in an article or paper, or exported as a gnuplot script and corresponding data file.

III. Usage Examples

As an example, consider the Leader Election model from PRISM [8] with three leaders. In the leader election family of models only one state has a larger number (denoted by k) of successors while the remaining states have a unique successor. We would like to see a comparison between the resource usage (in this case, processing time and the number of times that a maximum flow is computed) of the straightforward, unoptimized algorithm [11], the parametric maximum flow algorithm [12] and the state partitioning optimization [14].

The resulting tables are shown as Table I and Table II for different k values. Since the required CPU time directly depends on the processing power of the underlying system, Table I only serves to investigate relative differences between the methods. The data displayed in Table II on the other hand, is of deterministic nature and independent from the platform. Given knowledge of the underlying algorithms, it sheds more light on the relative differences observed in Table I.

`FlowSim` offers data about other kinds of resources than the two used as examples above to be tabulated, offering more insight into the operation of the different algorithms and optimizations thereof on different models of interest.

By using randomly generated models, we can investigate the impact of certain model properties such as transition density and biases. For example, we can change the fanout bias and observe the effect on how frequently the maximum flow algorithm is invoked without optimization and with state partitioning [14]. See Table III.

IV. Implementation and Availability

`FlowSim` is written in C/C++ for Linux/Unix platforms and makes use of the Boost Graph Library (BGL). The sources of the tool are distributed under GPL, and are available under: `http://depend.cs.uni-sb.de/~zhang/flowsim/index.htm`.

	leader_3	leader_4	leader_5	leader_6
States	61	135	257	439
Transitions	87	198	381	654
No Optimization	0.007	0.102	1.253	10.583
State Partitioning	0.003	0.014	0.063	0.243
Parametric Max. Flow	0.006	0.100	1.249	10.557

TABLE I

Time (in seconds) to compute the simulation relation.

	leader_3	leader_4	leader_5	leader_6
No Optimization	1388	4634	18571	88580
State Partitioning	1103	4549	18323	72701
Parametric Max. Flow	1616	5788	22831	100358

TABLE II

Number of maximum flow computations.

Fanout Bias	No Optimization	State Partitioning
-1.000	17833	17833
-1.778	18170	780
-1.556	18592	367
-0.333	18779	202
-0.111	19369	165
0.111	19545	162
0.333	19681	215
1.556	19701	186
1.778	19761	348
1.000	19823	19823

TABLE III

Number of maximum flow computations on random model.

References

[1] W. J. Stewart, *Introduction to the Numerical Solution of Markov Chains*. Princeton University Press, 1994.

[2] R. Segala and N. A. Lynch, "Probabilistic simulations for probabilistic processes." *Nord. J. Comput.*, vol. 2, no. 2, pp. 250–273, 1995.

[3] M. Puterman, *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. John Wiley & Sons, 1994.

[4] W. H. Sanders and J. F. Meyer, "Reduced base model construction methods for stochastic activity networks." *IEEE Journal on Selected Areas in Communications*, vol. 9, no. 1, pp. 25–36, 1991.

[5] M. A. Marsan, G. Balbo, G. Conte, S. Donatelli, and G. Franceschinis, "Modelling with generalized stochastic petri nets." *SIGMETRICS Performance Evaluation Review*, vol. 26, no. 2, p. 2, 1998.

[6] H. Hansson and B. Jonsson, "A logic for reasoning about time and reliability." *Formal Asp. Comput.*, vol. 6, no. 5, pp. 512–535, 1994.

[7] A. Aziz, K. Sanwal, V. Singhal, and R. K. Brayton, "Verifying continuous time Markov chains." in *CAV*, 1996, pp. 269–276.

[8] A. Hinton, M. Z. Kwiatkowska, G. Norman, and D. Parker, "PRISM: A Tool for Automatic Verification of Probabilistic Systems," in *TACAS*, 2006, pp. 441–444.

[9] J.-P. Katoen, M. Khattri, and I. S. Zapreev, "A Markov reward model checker." in *QEST*, 2005, pp. 243–244.

[10] B. Jonsson and K. G. Larsen, "Specification and refinement of probabilistic processes," in *LICS*, 1991, pp. 266–277.

[11] C. Baier, B. Engelen, and M. E. Majster-Cederbaum, "Deciding bisimilarity and similarity for probabilistic processes." *J. Comput. Syst. Sci.*, vol. 60, no. 1, pp. 187–231, 2000.

[12] L. Zhang, H. Hermanns, F. Eisenbrand, and D. N. Jansen, "Flow faster: Efficient decision algorithms for probabilistic simulations," *Special Issue on TACAS 2007, Logical Method in Computer Science (LMCS)*, 2008.

[13] L. Zhang, "A space-efficient probabilistic simulation algorithm," in *CONCUR*, 2008, pp. 248–263.

[14] J. Bogdoll, H. Hermanns, and L. Zhang, "An experimental evaluation of probabilistic simulation," in *FORTE*, 2008, pp. 37–52.