Uniform Random Walks in Very Large Models

Johan Oudinet
LRI
Bât 490 Université Paris-Sud
91405 Orsay Cedex, France
oudinet@lri.fr

ABSTRACT

This paper presents first experimental results on uniform random generation of paths in very large graphs that model concurrent systems, as it was presented in [3]. The approach is based on techniques and tools for counting and drawing uniformly at random in combinatorial structures. It exploits the fact that in a system made of several concurrent components, local uniform drawings of component paths can be combined into a very good approximation of uniform drawing of paths in the global system, without constructing the global model. The paper describes some implementation of the methods presented in [3], reports results on a first suite of benchmarks, exploring the limits and the possibility of this new approach to uniform random walks.

Categories and Subject Descriptors

D.2.4 [**Software Engineering**]: Software/Program Verification; D.2.5 [**Software Engineering**]: Testing and Debugging

Keywords

random walk, uniform generation, modular models, model-checking, model-based testing

1. INTRODUCTION

Random walks are widely used for software simulation, testing (structural testing or model-based testing), and more recently, model checking. In general, classical random walks cannot guarantee good coverage of the underlying graph.

In [2, 8], Denise, Gaudel, and Gouraud show how to use techniques for counting and drawing uniformly at random in combinatorial structures for drawing paths either uniformly, or biased toward a given coverage criteria. The AuGuSTe tool [7] was a first application of this approach to statistical structural testing. It was experienced on several C programs, with control graphs up to hundred nodes.

RT '07, November 6, 2007, Atlanta, GA, USA

To deal with larger graphs, such as those modelling concurrent systems, Denise, Gaudel, Gouraud, Lassaigne, and Peyronnet had the idea of using the set of concurrent, smaller models that compose the very large global model. In [3], they approximate the number of (fixed-length) paths without constructing the global model using techniques from analytic combinatorics [4]. This results in a very good approximation (see details in [3]) of uniform paths drawing in very large graphs.

In this paper, we first explore the limit of the method used in AuGuSTe for drawing paths uniformly at random in a single graph. Section 2 describes our implementation and reports experimental results on medium-sized graphs, up to 18746 nodes for paths of length 2000. Then, Section 3 presents the implementation of approximate uniform drawing in very large graphs that are asynchronous product of concurrent, medium-sized models, as those considered in Section 2. The biggest considered graph is of size 4.10^{24} for paths of length 8000, but it is by no means a limit of what can be dealt with. Finally, Section 4 makes use of the above results to approximate uniform drawing in a system of concurrent models in presence of synchronization. The presented results are limited to one synchronization per model. The biggest considered graph is of the same size as above, for paths of length 500. Improvements of the implementation to take into account several synchronizations and longer paths are in progress. Part of them can be found in [9].

2. EXACT UNIFORM DRAWING IN ONE GRAPH

This section describes the implementation for drawing uniformly paths of a given length from a single graph described in the BCG format (Binary Coded Graphs [6]).

2.1 Principle

The first step is to build a combinatorial structure from the BCG file that describes the graph. A C++ program, which uses the CADP toolbox, creates a MuPAD-Combinat [10] script that contains the combinatorial structure related to the graph. The MuPAD-Combinat library, based on the computer algebra system MuPAD [1], can handle combinatorial objects and generate C programs for drawing uniformly paths among those of length n in the combinatorial structure. This program takes as input the length and the number of paths that must be generated.

2.2 Results and Benchmarks

Benchmarks have been realized on an Intel 2.8GHz with

Table 1: Elapsed time for the MuPAD-Combinat script to generate a C program drawing uniformly paths

	# states	# transitions	MuPAD → C
vasy_0_1	289	1224	2s
vasy_1_4	1183	4464	20s
vasy_5_9	5486	9676	10m
vasy_8_24	8879	24411	43m
vasy_8_38	8921	38424	17m
vasy_10_56	10849	56156	34m
vasy_18_73	18746	73043	4h

Table 2: Elapsed time to build the counting table.

	200	1000	2000	3000	5000	8000
vasy_0_1	0.1s	0.3s	0.9s	1.5s	3.7s	8.8s
vasy_1_4	0.2s	1.1s	2.8s	5.1s	11.9s	∞
vasy_5_9	0.3s	4.9s	12.2s	22.0s	∞	∞
vasy_8_24	1.0s	9.0s	22.0s	∞	∞	∞
vasy_8_38	0.6s	4.4s	11.4s	22.9s	∞	∞
vasy_10_56	3.2s	22.5s	∞	∞	∞	∞
vasy_18_73	2.7s	20.0s	50.2s	∞	∞	∞

1GB of RAM. Each BCG graph comes from the VLTS (Very Large Transition Systems [5]) benchmark suite. These models are from the industrial world.

Table 1 shows execution time for the MuPAD-Combinat script that generates the C program for drawing paths uniformly. This execution time is at worst quadratic in the graph's size (it is linear for graphs with a bounded edge degree), but it must be done only once before drawing as many paths as needed.

The generated C program can draw paths of various lengths. So, it has to build the counting table (i.e., the numbers of paths of length i for all $0 \leq i \leq n$ starting from each state) before drawing one or several paths. The number of arithmetic operations needed to compute this table is $O(n \times |G|)$ where $|G|$ is the graph's size [4]. Table 2 shows the needed time to build these counting tables. The ∞ symbol means there is not enough memory for the table.

Finally, after building the counting table, drawing several paths of length n can be done in $O(n)$ arithmetic operations. Table 3 shows the elapsed time for generating 100 paths of length n (n varies from 200 to 8000) minus the preprocessing time for building counting tables.

Those tables show the efficiency of our approach: the *preprocessing* stage can be slow (4h for the biggest tested model that has 18746 states) but is done once, and the generation step is extremely fast. Hence, a lot of paths can be generated and the graph can be intensively explored. However, those tables also demonstrate the limits of this approach: it is necessary to hold huge counting tables in memory. And even if a large memory is available, generation time is linear in the graph's size, so dealing with very large graphs will still be impracticable.

Table 3: Elapsed time to generate 100 paths.

	200	1000	2000	3000	5000	8000
vasy_0_1	0.0s	0.9s	2.9s	6.3s	15.9s	40.1s
vasy_1_4	0.1s	1.0s	3.2s	6.7s	18.2s	∞
vasy_5_9	0.0s	0.9s	2.4s	5.2s	∞	∞
vasy_8_24	0.2s	0.8s	2.4s	∞	∞	∞
vasy_8_38	0.1s	1.6s	5.5s	10.8s	∞	∞
vasy_10_56	0.0s	1.3s	∞	∞	∞	∞
vasy_18_73	0.2s	0.9s	3.3s	∞	∞	∞

3. APPROXIMATE UNIFORM DRAWING IN A SYSTEM OF MODULES WITHOUT SYNCHRONIZATION

In the previous section, we draw uniformly at random paths in medium-sized graphs. In this section, we deal with larger graphs, that are the result of asynchronous composition of r modules $(M_1, \ldots, M_r)$. To each module corresponds a medium-sized graph. The result of asynchronous composition is the product of r models. A path in the global model is the shuffle of paths drawn in each of the r models. We have implemented the solution provided by [3] in order to handle very large models from concurrent systems. The main idea is to avoid the construction of the global model by suitably combining local uniform drawings of module paths to get a very good approximation of uniform drawing of paths in the global system.

Considering asynchronous composition is a first step: the algorithm presented in this section will also be useful for the synchronization case.

3.1 Principle

The uniform generation from asynchronous composition of r modules can be summarized in 4 steps:

1. set the length n of the path,
2. choose the lengths $(n_1, \ldots, n_r)$ of the r paths that are drawn in each module with a suitable probability,
3. for each module M_i draw a path of length n_i (using the algorithm of the previous section),
4. shuffle the r paths for getting a path of length n.

We show below a simple algorithm that shuffles the r paths in such a way that uniformity is ensured [3], thus Item 2 is the most difficult one.

A C++ program was created for drawing a r-uple $(n_1, \ldots, n_r)$ such that $n_1 + \cdots + n_r = n$. This r-uple corresponds to the lengths of paths that are drawn in each module.

The r-uple $(n_1, \ldots, n_r)$ has to be chosen with the probability:

$$P(n_1, \ldots, n_r) = \frac{\binom{n}{n_1 \ \cdots \ n_r} l_1(n_1) \cdots l_r(n_r)}{l(n)}$$

where $l_i(k)$ is the number of paths of length k in the module M_i and $l(n)$ is the number of paths of lengths n in the global model. But, computing $l(n)$ is too time consuming as it needs the global model. So we should approximate it using a theorem in [4] and the fact that the global model is the result of the composition of r modules:

$$l(n) \sim (C_1 \ldots C_r)(\omega_1 + \cdots + \omega_r)^n$$

where C_i and ω_i are computed from the generating series of the module M_i.

Table 4: Describing for each test the global model size and the number of modules. Here, only the vasy_0_1 model was used as module.

name	# states	number of modules
test 2	$8 \cdot 10^4$	2
test 3	$2.4 \cdot 10^7$	3
test 4	$7 \cdot 10^9$	4
test 5	$2 \cdot 10^{12}$	5
test 6	$5.8 \cdot 10^{14}$	6
test 7	$1.7 \cdot 10^{17}$	7
test 10	$4 \cdot 10^{24}$	10

Table 5: Elapsed time for the *preprocessing* step (i.e., all steps that need to be done once whenever the number of paths).

	200	1000	2000	3000	5000	8000
test 2	33.5s	34.3s	34.6s	35.3s	36.5s	39.4s
test 3	50.9s	51.3s	51.4s	52.1s	53.0s	55.3s
test 4	68.1s	68.0s	68.7s	69.3s	69.9s	71.7s
test 5	85.3s	85.2s	85.2s	86.8s	86.8s	88.6s
test 6	94.1s	98.3s	98.3s	99.4s	1m40	1m40
test 7	1m54	1m56	1m54	1m56	1m57	1m57
test 10	2m36	2m36	2m36	2m36	2m37	2m39

Hence, $P(n_1, \ldots, n_r)$ is approximated as follows:

$$P(n_1, \ldots, n_r) \sim \frac{\binom{n}{n_1 \ \cdots \ n_r} \omega_1^{n_1} \cdots \omega_r^{n_r}}{(\omega_1 + \cdots + \omega_r)^n}$$

After computing those values, the algorithm draws an index i from $[1..r]$ with probability $Pr(i) = \frac{\omega_i}{\sum_{j=1}^{r} \omega_j}$. The length n_i is incremented. This algorithm is repeated n times. Hence, the final r-uple $(n_1, \ldots, n_r)$ verifies $n_1 + \cdots + n_r = n$ and has probability $\frac{\binom{n}{n_1 \ \cdots \ n_r} \omega_1^{n_1} \cdots \omega_r^{n_r}}{(\omega_1 + \cdots + \omega_r)^n}$

Finally, after drawing r paths w_i of length n_i with the algorithm of the previous section, a C++ program shuffles uniformly those r paths. This program takes r paths and returns a path of length n (because the sum of all path lengths is equal to n). The algorithm is as follows:

1. $w = \epsilon$, $n_i = |w_i| \ \forall i \in [1..r]$, $n = \sum_{i=1}^{r} n_i$
2. choose i with probability $\frac{n_i}{n}$,
3. $w = w.f(w_i)$, where $f(w_i)$ gets the first letter of w_i,
4. $n = n - 1$ and $n_i = n_i - 1$
5. repeat the last 3 operations n times. The result is w.

3.2 Results and benchmarks

Table 4 describes the models that are used to generate paths from an asynchronous composition of modules.

Then, table 5 summarizes the *preprocessing* times.

Finally, table 6 shows the elapsed time to draw 100 paths, minus the *preprocessing* step. Those values show that the generation is fast even if the model has more than 10^{20} states: only module sizes matter (here, 289 states). Note that the generation is faster when the number of modules grows, because when there are more modules then the average length of paths to be drawn in each module decreases.

Table 6: Elapsed time to draw 100 paths, minus the *preprocessing* step. This value is the result of timing the generation of 101 paths and minus it by the timing for drawing one path.

	200	1000	2000	3000	5000	8000
test 2	0.4s	0.9s	2.4s	4.3s	10.2s	23.5s
test 3	0.3s	0.8s	2.3s	3.3s	7.7s	12.9s
test 4	0.0s	1.1s	1.6s	2.8s	6.2s	13.8s
test 5	0.6s	0.2s	1.9s	1.8s	5.5s	12.3s
test 6	0.0s	0.6s	1.6s	1.9s	5.1s	12.0s
test 7	0.6s	0.0s	2.9s	2.7s	5.4s	10.7s
test 10	0.3s	0.3s	1.3s	2.8s	4.6s	9.7s

Finally, those experiments show that drawing paths in the concurrent modules that describe the global model, makes possible the exploration of very large models.

4. APPROXIMATE UNIFORM DRAWING IN PRESENCE OF SYNCHRONIZATION

In the previous section, all modules are completely independent. In practice, this assumption is rarely true. Thereby, modules may wait for one another before taking a transition; those special transitions are called synchronized transitions. Now, we suppose that each module contains exactly one synchronized transition, denoted α. Thus, in the global system all modules must take the α transition at the same time. This case has been studied in [3]. We explain the algorithm that we have implemented and analyze the benchmarks.

4.1 Principle

Let α be the synchronized symbol and w be a path from the global system. If w contains m synchronizations then $w = w_0 \alpha w_1 \alpha \ldots \alpha w_m$ and there is no α in any of the w_i. So, we use the algorithm from the previous section to draw the $m+1$ sub-paths w_i, subject to break each module up into 4 asynchronous modules (Figure 1) because w_0 must start from the initial states of all modules and finish to states that precede an α transition.

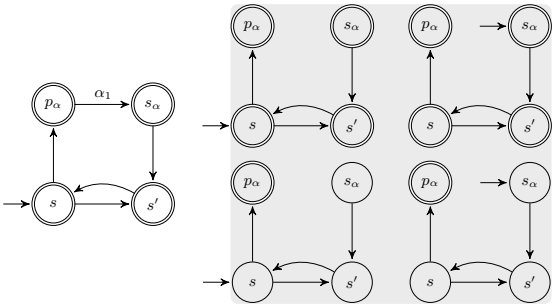

Figure 1: A module M_i that contains one synchronized transition denoted α and its four associated, asynchronous modules.

Finally, if we compute $s(n)$ (resp. $s(n, m)$), the number of paths of length n (resp. that contain m synchronizations) then we can sketch an algorithm for generating a path of length n:

1. choose m with probability $P(m) = s(n, m)/s(n)$,

Table 7: Elapsed time for the *preprocessing* step (i.e., all steps that need to be done once whenever the number of paths).

	100	200	300	400	500	600
test 2	3m10	3m25	4m12	5m55	9m06	∞
test 3	4m45	5m02	5m54	7m46	11m11	∞
test 4	6m19	6m39	7m34	9m34	13m13	∞
test 5	7m51	8m16	9m15	11m23	15m18	∞
test 6	9m24	9m50	10m56	13m12	17m21	∞
test 7	11m00	11m27	12m35	15m03	19m24	∞
test 10	12m28	12m31	12m33	12m32	20m26	∞

Table 8: Elapsed time to draw 100 paths, minus the *preprocessing* step. This value is the result of timing the generation of 101 paths and minus it by the timing for drawing one path.

	100	200	300	400	500	600
test 2	1.9s	2.1s	4.3s	4.7s	4.4s	∞
test 3	0.9s	1.2s	1.2s	2.6s	3.8s	∞
test 4	1.5s	1.5s	0.9s	3.2s	2.5s	∞
test 5	3.0s	0.4s	0.9s	3.1s	2.5s	∞
test 6	4.1s	0.7s	0.9s	3.8s	2.5s	∞
test 7	3.7s	2.9s	1.9s	1.6s	2.4s	∞
test 10	4.6s	0.6s	0.8s	1.5s	2.2s	∞

2. choose the lengths of $w_0, w_1, \ldots, w_m$ with a suitable probability,

3. generate each w_k of length i_k in the correct modules.

Computing Item 1 requires $O(n)$ arithmetic operations in the worst case (with $O(rn^4)$ arithmetic operations to precompute all of the $s(n,m)$ and $s(n)$) [3]. And computing Item 2 requires $O(n^2)$ arithmetic operations.

4.2 Results and Benchmarks

The same models as in Table 4 have been used but one transition in each module was labeled α.

Table 7 shows the elapsed time for the *preprocessing* steps and Table 8 shows the elapsed time to draw 100 paths. The ∞ symbol means there is not enough memory to compute all of the $s(n,m)$ and $s(n)$.

Finally, those experiments show that drawing paths in very large models is possible even if synchronization exists between the concurrent modules that describe the global model. The case where there are several synchronizations labeled by different symbols is more complex but the algorithm is already presented in [9] and the benchmarks are in progress.

5. CONCLUSION AND PERSPECTIVES

This paper reports first experimental results on how to perform globally uniform random walks in very large systems described as sets of concurrent, smaller components. By globally uniform random walk, we mean that the choice of the successor at every step is biased such that all paths of the global model have equal probability to occur.

Section 2 shows that brute-force method (namely counting the number of paths of the desired length starting from each successor and adjusting its probability accordingly) is feasible for medium-sized models only: up to approximately 10^4 states. For larger systems, experiments in Section 3 and 4 demonstrate how local uniform drawing and estimation of the number of global paths make possible to uniformly explore very large models described as sets of concurrent, smaller models, the complexity being limited to the size of the biggest component.

More development and experiments are in progress: The implementation described in Section 4 is under revision for accepting longer paths; The generalization to several synchronizations is described in [9] and its implementation is on-going.

This method can be useful for random testing, model-checking, or simulation of protocols that involve many distributed entities, as it is often the case in practice.

6. ACKNOWLEDGMENTS

I am indebted to my master thesis directors Alain Denise and Marie-Claude Gaudel and to the RaSTa group for their help and advice.

7. REFERENCES

[1] C. Creutzig and W. Oevel. *MuPAD Tutorial*. Springer, second edition, 2004.

[2] A. Denise, M.-C. Gaudel, and S.-D. Gouraud. A generic method for statistical testing. In *ISSRE '04: Proceedings of the 15th International Symposium on Software Reliability Engineering (ISSRE'04)*, pages 25–34, Washington, DC, USA, 2004. IEEE Computer Society.

[3] A. Denise, M.-C. Gaudel, S.-D. Gouraud, R. Lassaigne, and S. Peyronnet. Uniform random sampling of traces in very large models. In *RT '06: Proceedings of the 1st international workshop on Random testing*, pages 10–19, New York, NY, USA, 2006. ACM Press.

[4] P. Flajolet and R. Sedgewick. Analytic combinatorics: Functional equations, rational and algebraic functions. Research Report 4103, INRIA, 2001. 98 pages.

[5] H. Garavel and N. Descoubes. Very large transition systems. http://tinyurl.com/yuroxx.

[6] H. Garavel and R. Ruffiot. Binary Coded Graphs. http://www.inrialpes.fr/vasy/cadp/man/bcg.html.

[7] S.-D. Gouraud. *Utilisation des Structures Combinatoires pour le Test Statistique*. PhD thesis, Université Paris-Sud 11, LRI, june 2004.

[8] S.-D. Gouraud, A. Denise, M.-C. Gaudel, and B. Marre. A new way of automating statistical testing methods. In *ASE '01: Proceedings of the 16th IEEE international conference on Automated software engineering*, page 5, Washington, DC, USA, 2001. IEEE Computer Society.

[9] J. Oudinet. Uniform random walks in concurrent models. Master's thesis, Université Paris-Sud 11, LRI, http://www.lri.fr/∼oudinet/publications/07/mastersthesis.pdf, September 2007.

[10] N. M. Thiéry. Mupad-combinat algebraic combinatorics package for mupad. http://mupad-combinat.sourceforge.net/.