Counting for Random Testing

Marie-Claude Gaudel

Université Paris-Sud 11, LRI, Orsay, F-91405,
and CNRS, Orsay, F-91405
mcg@lri.fr
http://www.lri.fr/~mcg

Abstract. The seminal works of Wilf and Nijenhuis in the late 70s have led to efficient algorithms for counting and generating uniformly at random a variety of combinatorial structures. In 1994, Flajolet, Zimmermann and Van Cutsem have widely generalised and systematised the approach. This extended abstract presents several applications of these powerful results to software random testing, and random model exploration.

Keywords: software testing, random walks, combinatorics

1 Introduction

In the area of analytic combinatorics, the seminal works of Wilf and Nijenhuis in the late 70s have led to efficient algorithms for counting and generating uniformly at random a variety of combinatorial structures [13, 19]. In 1994, Flajolet, Zimmermann and Van Cutsem have widely generalised and systematised the approach [7]. The recent book by Flajolet and Sedgewick [8] presents a detailed survey of this corpus of knowledge. These works constitute the basis of powerful tools for uniform random generation of complex entities, such as graphs, trees, words, paths, etc.

This extended abstract summarises several applications of these powerful results to software random testing, and random model exploration.

Random methods look attractive for testing large programs or checking large models. However, designing random methods that have a good and assessable fault detection power is not so easy: the underlying probability distribution of inputs or paths must be carefully designed if one wants to ensure a good coverage of the program or model, or of potential fault locations, and to quantify this coverage.

This extended abstract continues as follows: Section 2 recalls some basic facts on software random testing and random walks; Section 3 briefly presents methods for uniform generation of bounded paths that are completely described in [17] and [5]; Section 4 studies on how to take into account other coverage criteria, gives a definition of randomised coverage satisfaction [5] ; finally Section 5 gives some hints of the application of these methods to LTL model-checking [18].

2 Some preliminaries on random testing and random walks

There are three main categories of methods for software random testing: those based on the input domain of the system under test, those based on some knowledge of its environment, and those based on some model of its behaviour.

We focus on the third case, where some graphical description of the behaviour of the system under test is used. Random walks [1] are performed on the set of paths of this description. Classical random walk methods, sometimes called isotropic, progresses from one state by drawing among the successors uniformly at random. The big advantage of this method is that is easy to implement and only requires local knowledge of the model. A serious drawback is that in case of irregular topology of the underlying graph, uniform choice of the next state is far from being optimal from a coverage point of view: some examples are given in [4] and [5] . similarly, getting an estimation of the coverage obtained after one or several random walks would require some complex global analysis of the topology of the model.

The works presented in this extended abstract aim at improving the quality of random walks with respects of various coverage criteria: bounded paths coverage, transitions/branches coverage, states/statements coverage, lassos. There is a price to pay: some non-local knowledge of the models is required, based on counting the elements to be covered accessible from each successor of the current state. Thank to the powerful results mentioned above, and to sophisticated implementation methods, it is possible to get good compromises between memory requirement, efficiency of drawing, and quality of the achieved coverage.

All the works below rely on combinatorial algorithms, based on a representation of models or programs by some automaton or a by some product of several automata, synchronised or not. The basic algorithms are implemented and available in the RUKIA C++ library (http://rukia.lri.fr/en/index.html)

3 Improvements of recursive uniform path generation

3.1 The classical recursive method

This classical method was first presented in [19, 7] Let us consider a deterministic finite automaton of $\mathcal{A}$ with q states $\{1, 2, \ldots, q\}$ among which are distinguished an initial state and some final states. For each state s, let $l_s(n)$ be the number of paths of length n starting from s and ending at a terminal state. Such values can be computed with the following recurrences on n (where $\mathcal{F}$ denotes the set of final states in $\mathcal{A}$):

$$\begin{cases} l_s(0) = 1 & \text{if } s \in \mathcal{F} \\ l_s(0) = 0 & \text{if } s \notin \mathcal{F} \\ l_s(i) = \sum_{s \to s'} l_{s'}(i-1) & \forall i > 0 \end{cases} \tag{1}$$

If we note the vector $L_n = \langle l_1(n), l_2(n), \ldots, l_q(n) \rangle$, the principle of the recursive method is in two steps:

- Compute and store L_k for all $1 \leq k \leq n$. This calculation is done starting from L_0 and using euation (1).
- Generate a path of length n by choosing, when the current state is s and the path has already $n - m$ states, successor s_i with the probability:

$$\mathbb{P}(s_i) = \frac{l_{s_i}(m-1)}{l_s(m)}. \tag{2}$$

When using floating point arithmetic as in [6], the complexity of the algorithm is in $\mathcal{O}(qn)$ space and time for the preprocessing stage and $\mathcal{O}(n)$ for the generation, where n denotes the length of the path to be generated, and q denotes the number of states.

For big models and long paths, this method does not scale up well. This was the motivation for two pieces of work presented below.

3.2 A dichotomic algorithm for the uniform random generation of paths

In [17] Oudinet et al. have presented the so-called *dichopile* method, which is based on a divide-and-conquer approach, avoids numerical instability and offers an excellent compromise in terms of space and time requirements.

Note that to choose the successor of the initial state, we only need L_n and L_{n-1}. Then, L_{n-1} and L_{n-2} allow to choose the next state and so on. Thus, if we had a method that compute efficiently $L_n, L_{n-1}, \ldots, L_0$ in descending order, we could store the two last vectors only and reduce space complexity. This *inverse* approach constitutes the principle of Goldwurm's method [10]. However, in [15], Oudinet showed that this method is numerically instable, thus forbidding the use of floating-point arithmetics.

The idea of the *dichopile* algorithm is as follows. Compute the number of paths of length n from the number of paths of length 0 while saving in a stack a logarithmic number of intermediate steps: the number of paths of length $n/2$, of length $3n/4$, of length $7n/8$, etc. For computing the number of paths of length $n - i$, it is computed again from the intermediate stage that is at the top of the stack. Recall that L_j denotes the vector of q numbers of paths of length j, that is the $l_s(j)$'s for all states s.

Unlike the classical recursive method, there is no preprocessing phase. In [17] it is proved that using floating-point numbers with a mantissa of size $\mathcal{O}(\log n)$, bit complexities of drawing are $\mathcal{O}(q \log^2 n)$ in space and $\mathcal{O}(dqn \log^2 n)$ in time, where d stands for the maximal out-degree of the automaton.

The classical recursive method is much faster after the preprocessing stage, but it is unusable for long paths and large models due to its space requirement. *dichopile* is an excellent compromise when considering both space and time complexities. In our experiments with automata from the VLTS benchmark suite (Very Large Transition Systems, http://tinyurl.com/yuroxx), examples of

limits for the recursive method were 8879 states, 24411 transitions and paths of length 64000, or 10849 states 56156 transitions and paths of length 16000, where dichopile was able to generate paths of length 128000 and more. *dichopile* was able to deal with 12323703 states, 27667803 transitions and paths of length 8000. Both methods are implemented in the RUKIA library.

3.3 Uniform path exploration in very large composed models

Fortunately, huge models are rarely stated from scratch. They are obtained by composition of smaller ones, the main source of state number explosion being parallel compositions.

When there is no synchronisation, the parallel composition of r models $Mod_1, \ldots, Mod_r$ is the product of the underlying automata [2]. A brute force method to uniformly drawing paths is to build the product and to use the methods above. Since it is possible for moderate sizes only we have developed an alternative method that avoids the construction of the global model. This method is presented in detail in [4] [14] and [5]. We sketch it below.

- Given n the length of the global path to be drawn
- Choose some lengths $n_1, \ldots, n_r$ such that $\sum_{i=1,\ldots,r} n_i = n$, with adequate probabilities (see below)
- For each Mod_i, draw uniformly at random some path w_i of length n_i
- Interleave the r w_i in a randomised way that ensures uniformity among interleavings.

Let $\ell(n)$ be the number of global paths of length n, and $\ell_i(k), i = 1, \ldots, r$ the number of paths of length k in Mod_i. The choice of the $n_1, \ldots, n_r$ should be done with the probability below:

$$\Pr(n_1, \ldots, n_r) \;=\; \frac{\binom{n}{n_1,\ldots,n_r}\ell_1(n_1)\ldots\ell_r(n_r)}{\ell(n)} \tag{3}$$

where the numerator is the number of interleavings of length n that can be built with r local paths of lengths $n_1, \ldots, n_r$. Since computing the exact value of $\ell(n)$ would require the construction of the global model and of the corresponding tables, we use the following approximation from [8]:

$$\ell(n) \sim C\omega^n. \tag{4}$$

where C and ω are two constants. A sufficient, but not necessary, condition for this approximation to hold is aperiodicity and strong connectivity of the automaton, which is satisfied by any LTS with a reset. Details of weaker conditions can be found in [8]. This approximation is precise enough even for small values of n since $C\omega^n/\ell(n)$ converges to 1 at exponential rate.

Using the same approximations for the $\ell_i(n_i), i = 1, \ldots, r$, we get (see [4]):

$$\ell(n) \sim C_1 \ldots C_r(\omega_1 + \ldots + \omega_r)^n \tag{5}$$

and then

$$\Pr(n_1, \ldots, n_r) \sim \frac{\binom{n}{n_1,\ldots,n_r}\omega_1^{n_1}\omega_2^{n_2}\ldots\omega_r^{n_r}}{(\omega_1+\omega_2+\ldots+\omega_r)^n}. \quad (6)$$

This avoids the computation of $\ell(n)$, and the constants $\omega_i, i = 1, \ldots, r$ are computable in polynomial time with respect to the size of the Mod_i. It means that the complexity is dependent on the size of the components only, and not of the size of the global model.

In [4], we provide an algorithm for drawing $n_1, \ldots, n_r$ with this probability without computing it: draw a random sequence of n integers in $\{1, \ldots, r\}$, with the probability to choose i equal to $\Pr(i) = \frac{\omega_i}{\omega_1+\omega_2+\ldots+\omega_r}$; then take as n_i the number of occurrences of i in this sequence.

This concludes the issue of the choice of the $n_1, \ldots, n_r$. A classical randomised way of interleaving r w_i of lengths n_i that ensures uniformity is used.

This method is available in the RUKIA library. Experiments have been successfully led on models with 10^{34} states, with performances that show that the approach makes it possible to uniformly explore even larger models [14].

The generalisation one synchronisation is given in [9]. The case of several synchronisations is studied in Oudinet's Ph.D. thesis [16]
(available at http://www.lri.fr/ oudinet/en/research.html#publications). It turns out be practicable in the case of a small number of synchronisations only. The number of synchonisations can be increased by considering partial order reduction, i.e. by collapsing interleavings. Besides, in presence of many synchronisations, the synchronised product is smaller and a brute-force method, where it is constructed and used for uniform drawings, may become feasible. Actually, practical solutions are probably combinations of these two approaches depending on the architecture of the global system.

4 Randomised coverage of states, transitions, and other features

Path coverage is known to be too demanding, due to the path number explosion. Thus it is of interest to consider other coverage criteria. In [11] [3] [5], we have defined a notion of randomised coverage satisfaction for random testing methods.

What does it mean for a random exploration method to take into account a coverage criterion? Let $E_C(\mathcal{G})$ be the set of elements characterising a coverage criterion C for a given graph $\mathcal{G}$, for instance the vertices or the arcs of $\mathcal{G}$, or some subset of them.. The satisfaction of this coverage criterion C by some random exploration of the graph $\mathcal{G}$ can be characterised by the minimal probability $q_{C,N}(\mathcal{G})$ of covering any element of $E_C(\mathcal{G})$ when drawing N paths. $q_{C,N}(\mathcal{G})$ can be easily stated as soon as $q_{C,1}(\mathcal{G})$ is known: it is given by the distribution associated to the method. One has $q_{C,N}(\mathcal{G}) = 1 - (1 - q_{C,1}(\mathcal{G}))^N$.

Given a coverage criteria and some given random testing method, the elements to be covered have generally different probabilities to be reached by a test. Some of them are covered by all the tests. Some of them may have a very

weak probability, due to the structure of the graph or to some specificity of the testing method.

Let $E_C(\mathcal{G}) = \{e_1, e_2, ..., e_m\}$ and for any $i \in (1..m), p_i$ the probability for the element e_i to be exercised during the execution of a test generated by the considered random testing method. Then

$$q_{C,N}(\mathcal{G}) = 1 - (1 - p_{min})^N \tag{7}$$

where $p_{min} = min\{p_i | i \in (1..m)\}$. This definition corresponds to a notion of *randomised coverage satisfaction*. It makes it possible to assess and compare random exploration methods with respect to a coverage criterion.

Conversely, the number N of tests required to reach a given probability of satisfaction $q_C(\mathcal{G})$ is

$$N \geq \frac{log(1 - q_C(\mathcal{G}))}{log(1 - p_{min})} \tag{8}$$

By definition p_{min} gives $q_{C,1}(\mathcal{G})$. Thus, from the formula above one immediately deduces that for any given $\mathcal{G}$, for any given N, maximising the quality of a random testing method with respect to a coverage criteria C reduces to maximising $q_{C,1}(\mathcal{G})$, i. e. p_{min}. Note that uniform drawing of bounded paths, as presented in Section 3 maximises p_{min} on the set of paths to be covered.

A more developed discussion of these issues can be found in [5], together with the treatment of state coverage and transition coverage, i.e. some method for computing their probabilities for a given model . These methods were first developed, implemented and experimented for C programs in [11] [3].

5 Uniformly randomised LTL model-checking

The big challenge of model-checking is the enormous sizes of the models. Even when the best possible abstractions and restrictions methods have been applied, it may be the case that the remaining size is still significantly too large to perform exhaustive model explorations. Giving up the idea of exhaustivity for model-checking leads to the idea of using test selection methods for limiting the exploration of models.

One of these methods is randomisation of the search algorithm used for model exploration. A first introduction of randomisation into model-checking has been described and implemented in [12] as a Monte-Carlo algorithm for LTL model-checking. The underlying random exploration is based on a classical uniform drawing among the transitions starting from a given state. As said in Section 2 the drawback of such random explorations is that the resulting distribution of the exploration paths is dependent on the topology of the model, and some paths may have a very low probability to be traversed.

In [18], we have studied how to perform uniform random generation of lassos, which are the kind of paths of interest for LTL model-checking. This implies counting and drawing elementary circuits, which is known as a hard problem. However, efficient solutions exist for specific graphs, such as reducible data flow

graphs which correspond to well-strucured programs and control-command systems. An immediate perspective is to embed this method in an existing model-checker such as SPIN or CADP, with the aim of developing efficient randomised methods for LTL model-checking with as result a guaranted probability of satisfaction of the checked formula.

This approach maximises the minimal probability to reach a counter-example, and makes it possible to state a lower bound of this probability after N drawings, giving an assessment of the quality of the approximation.

6 Conclusion

In this set of works, we study the combination of coverage criteria with random walks. Namely, we develop some methods for selecting paths at random in a model. The selection is biased toward a coverage criterion. We have introduced a notion of randomised coverage satisfaction of elements of the model such as states, transitions, or lassos which are of interest for checking or testing LTL formulas.

We use methods for counting and generating combinatorial structures, presenting several original applications of this rich corpus of knowledge. They open numerous perspectives in the area of random testing, model checking, or simulation of protocols and systems.

Acknowledgments. The works reported here have been led in the RASTA working group (RAndom System Testing and Analysis), which involves members of various research teams of LRI (Algorithms and Complexity, Bioinformatics, Formal Testing and System Exploration) and of the Equipe de Logique Mathématique at University Paris 7 . The current and past members are: Alain Denise, Marie-Claude Gaudel, Sandrine-Dominique Gouraud, Richard Lassaigne, Johan Oudinet, and Sylvain Peyronnet.

References

1. Aldous, D.: An introduction to covering problems for random walks on graphs. J. Theoret Probab. 4, 197–211 (1991)
2. Arnold, A.: Finite Transition Systems. Prentice-Hall (1994)
3. A. Denise, M.-C. Gaudel, and S.-D. Gouraud. A generic method for statistical testing. In *IEEE Int. Symp. on Software Reliability Engineering (ISSRE)*, pages 25–34, 2004.
4. A. Denise, M.-C. Gaudel, S.-D. Gouraud, R. Lassaigne, and S. Peyronnet. Uniform random sampling of traces in very large models. In *1st International ACM Workshop on Random Testing*, pages 10 –19, July 2006.
5. Alain Denise, Marie-Claude Gaudel, Sandrine-Dominique Gouraud, Richard Lassaigne, Johan Oudinet, and Sylvain Peyronnet. Coverage-biased random exploration of large models and application to testing. *STTT, International Journal on Software Tools for Technology Transfer*, Online First, 2011. 26 pages.

6. Denise, A., Zimmermann, P.: Uniform random generation of decomposable structures using floating-point arithmetic. Theoretical Computer Science 218, 233–248 (1999)
7. Flajolet, P., Zimmermann, P., Cutsem, B.V.: A calculus for the random generation of labelled combinatorial structures. Theoretical Computer Science 132, 1–35 (1994)
8. Flajolet, P., Sedgewick, R.: Analytic Combinatorics. Cambridge University Press (2009)
9. Gaudel, M.C., Denise, A., Gouraud, S.D., Lassaigne, R., Oudinet, J., Peyronnet, S.: Coverage-biased random exploration of large models. In: 4th ETAPS Workshop on Model Based Testing. Electronic Notes in Theoretical Computer Science, vol. 220, Issue 1, 10, pp. 3–14 (2008), invited lecture
10. Goldwurm, M.: Random generation of words in an algebraic language in linear binary space. Information Processing Letters 54(4), 229–233 (1995)
11. S.-D. Gouraud, A. Denise, M.-C. Gaudel, and B. Marre. A new way of automating statistical testing methods. In *IEEE International Conference on Automated Software Engineering (ASE)*, pages 5–12, 2001.
12. Grosu, R., Smolka, S.A.: Monte-Carlo Model Checking. In: Proc. of Tools and Algorithms for Construction and Analysis of Systems (TACAS 2005). LNCS, vol. 3440, p. 271286. Springer-Verlag (2005)
13. Nijenhuis, A., Wilf, H.S.: The enumeration of connected graphs and linked diagrams. J. Comb. Theory, Ser. A 27(3), 356–359 (1979)
14. Johan Oudinet. Uniform random walks in very large models. In *RT '07: Proceedings of the 2nd international workshop on Random testing*, pages 26–29, Atlanta, GA, USA, November 2007. ACM Press.
15. Oudinet, J.: Random exploration of models. Tech. Rep. 1534, LRI, Université Paris-Sud XI (June 2010), 15 pages
16. Oudinet, J.: Approches combinatoires pour le test statistique à grande échelle. Tech. Rep. 1534, LRI, Université Paris-Sud 11, Université Paris-Sud XI (November 2010), 118 pages
17. Johan Oudinet, Alain Denise, and Marie-Claude Gaudel. A new dichotomic algorithm for the uniform random generation of words in regular languages. In *Conference on random and exhaustive generation of combinatorial objects (GASCom)*, Montreal, Canada, September 2010. To appear. 10 pages.
18. Oudinet, J., Denise, A., Gaudel, M.C., Lassaigne, R., Peyronnet, S.: Uniform Monte-Carlo model checking. In: FASE. LNCS, vol. 6603, pp. 127–140. Springer (2011)
19. Wilf, H.: A unified setting for sequencing, ranking, and selection algorithms for combinatorial objects. Advances in Mathematics 24, 281–291 (1977)