An Efficient Algorithm to Generate Random Uncorrelated Euclidean Distances: The Random Link Model

César Augusto Sangaletti Terçariol
Centro Universitário Barão de Mauá
Rua Ramos de Azevedo, 423
14090-180, Ribeirão Preto, SP, Brazil and
Faculdade de Filosofia, Ciências e Letras de Ribeirão Preto,
Universidade de São Paulo
Avenida Bandeirantes, 3900
14040-901, Ribeirão Preto, SP, Brazil

and Alexandre Souto Martinez
Faculdade de Filosofia, Ciências e Letras de Ribeirão Preto,
Universidade de São Paulo,
Avenida Bandeirantes, 3900,
14040-901, Ribeirão Preto, SP, Brazil

Received on 27 October, 2005. Revised version received 20 January, 2006

A disordered medium is often constructed by N random points independently and identically distributed in a d-dimensional hyperspace. Characteristics related to the statistics of this system are known as the random point problem. As $d \to \infty$, the distances between two points become independent random variables, leading to its mean field description: the random link model. While the numerical treatment of large random point problems poses no major difficulty, due to Euclidean restrictions the same is not true for large random link systems. Exploring the deterministic nature of the pseudo-random number generators, we present techniques which allow to consider models with memory consumption of $O(N)$, instead of $O(N^2)$ obtained by a naive implementation, but with the same time dependence $O(N^2)$.

Keywords: Random media; Random point problem; Random link model; Random-number generator

I. INTRODUCTION

The random point problem (RPP) is a classical approach to construct disordered media. The coordinates $x_i^{(k)}$ of N random points are independently and randomly generated following a given common probability density function (pdf). Uniform deviates along the unitary length edges of a d-dimensional hypercube are usually considerated. The distance between any pair of points i and j is obtained by the Euclidean metrics: $D_{ij}^2 = \sum_{k=1}^{d} \left[x_i^{(k)} - x_j^{(k)}\right]^2$. A possible computational implementation of the RPP mainly consists of the following steps:

1. randomly generate the coordinates $x_i^{(k)}$,
2. store them in a $N \times d$ matrix, called *coordinate matrix*,
3. use the Euclidean metrics to calculate the distance between all pairs of points, and
4. store them in a $N \times N$ matrix, called *distance matrix*.

In this naive algorithm, both time and memory consumptions are proportional to N^2. In numerical applications, this limits system sizes typically to $N = 15000$ when using advanced techniques of random access memory (RAM) allocation.

Due to boundary effects and Euclidean restrictions, the distances between any two points are not all independent random variables, in fact, there are correlations among them. For fixed N, as the system dimensionality increases, boundary effects become more and more pronounced (periodic boundary conditions is a technique frequently used to minimize these effects) and the distances become less and less correlated. In the limit $d \to \infty$, all the two-point distances are independently and identically distributed (i.i.d.) random variables and this model is known as the *random link (distance) model* [1] (RLM). Yet two Euclidean constraints remain in the RLM: (i) the distance from a point to itself is always null ($D_{ii} = 0$, for all i) and (ii) the forward and backward distances are equal ($D_{ij} = D_{ji}$, for all i, j). If these constraints are relaxed, this model becomes the *random map model* [2, 3], which is the mean field approximation for Kauffman automata [4].

Both, the RPP and its mean field description, the RLM, have been very fruitful in the determination of numerical and analytical results in several interesting systems, such as statistics on the optimal trajectories for the traveling salesman problem on a random set of cities [5–9], frustrated dimerization optimization modeled by the minimum matching problem [10, 11] (or equivalently spin-glasses [10]) and partial self-avoiding deterministic [12, 13] and stochastic [14, 15] walks. The high-dimensional case for these partial self-avoiding walks has been our main motivation to consider the RLM. In deterministic walks [16–19], one is mainly interested on the neighborhood ranking of random points. Euclidean distances are only a means to obtain this ranking, which is independent of a particular choice for the distance pdf [13]. Here, we will consider only uniform deviates, nevertheless the algo-

rithm can be easily adapted to treat any distance pdf, such as the one with the pseudo-dimension parameter [8].

In Sec. II we briefly review the numerical implementation of the conguential random number generation and discuss aspects related to RAM allocation for the dynamic and static variables. The RPP is considered and an improvement to its numerical implementation is suggested in Sec. III. In Sec. IV, we consider the RLM and show that the straightforward implementation permits the simulation of only small numerical systems in a computer. Thus, two alternative (one- and multiple-seed) algorithms are presented and explore the reproducibility of the pseudo-random number generators. Final remarks are addressed in Sec. V.

II. SOME REMARKS ABOUT RANDOM NUMBER GENERATORS

The algorithms presented here have in commom the use of a pseudo-random number generator. In its simplest implementation, it is a deterministic algorithm, which requires a single integer parameter, called *seed*. The algorithm works as follows: before the first call to the generator, the seed is initialized to an (usually) arbitrary value (S_1). In the subsequent generator calls, the seed values are deterministically modified and give rise to the uncorrelated long sequence of integers S_1, S_2, S_3, ..., uniformly distributed in the interval $[0, 2^m - 1]$, for some $m \geq 0$. After running the whole sequence, the seed reassumes its initial value S_1 and the m-cycle reinitiates. Each initializing value S_1 will typically return a different subsequent random sequence. However, the *same* initializing value of S_1 will always lead to the *same* random sequence S_2, S_3, ... It is this deterministic feature of the random number generator that is the basis of our algorithm. For each integer seed value, the generator commonly returns a real number ($S_i/2^m$), which is uniformly distributed in the interval $[0,1[$, while keeping track of the seed sequence. It is this real number which is commonly used in numerical simulations.

Attention must be paid to a particular numerical programming aspect. The algorithms proposed here assume that all random number generator local variables are set as dynamic. To understant this aspect, let us briefly recall how variables are generally handled by subroutines. When a numerical code enters a routine, a RAM address is allocated for each local dynamic variable and these addresses are freed when the code leaves this routine. The internal routine variable values are not preserved. More sophisticated random number generators make use of internal static variables to keep their variable values stored for future use. Unlike dynamic, static variables have their addresses and current values allocated in RAM even when the routine is left. In this kind of generator, two calls with the *same* seed do not yield the *same* result, because the values of static variables are not the same in those two calls. To obtain the desired reproducibility of this kind of random-number generator, these static variables must be treated separately: a vector of size N must be declared to each one of them to recover their values. Even in this case, the memory and time consumption orders of magnitude are not altered.

III. RANDOM POINT PROBLEM

An alternative and better procedure to the RPP algorithm described in the introduction is to replace the $N \times N$ distance matrix by a vector of size N, called *mask*, and calculate only the distances related to a given point at each time step (for instance, to determine its neighborhood ranking). The unique computational waste is to calculate the same distance ($D_{ij} = D_{ji}$) twice (it could have been calculated only once if one had the distance matrix). Nevertheless, the time dependence is kept proportional to N^2, while the memory consumption becomes proportional to N.

To minimize the boundary effect, it is important to consider periodic boundary conditions and to keep fixed the mean point separation ($\ell = L/N^{1/d} = \rho^{-1/d}$, where L is a typical system size and ρ the point density). For this reason one has to increase N as the system dimensionality increases. Since high-dimensional systems are to be considered, even the declaration of the coordinate matrix may consume huge computer RAM. This introduces additional computational difficulties once the system size has found a barrier imposed by the dimensionality.

Nevertheless, using periodic boundary conditions, the higher the system dimensionality is, the weaker the correlations among the distances (triangular inequality, for example) are. In the high-dimensionality limit ($d \to \infty$), the distances between all pairs of distinct points can be considered as $N(N-1)/2$ i.i.d. random variables and the RPP converges to the RLM.

IV. RANDOM LINK MODEL

To work numerically with the RLM, one must directly generate the i.i.d. random distances, rather than randomly generate the point coordinates and calculate the distances by applying the Euclidean metrics to the coordinate matrix. This solves the large computer RAM allocation problem due to high system dimensionality, since the coordinate matrix does not exist anymore.

A. Conventional Implementation

In a straightforward implementation of the i.i.d. random distances in RLM, one declares a $N \times N$ distance matrix and simply applies the two Euclidean constraints mentioned in the introduction: (i) set 0 to its whole main diagonal and (ii) randomly generate each distance D_{ij}, with $i = 1, 2, \ldots, N-1$ and $j = i+1, i+2, \ldots, N$ (right-hand side of the main diagonal), and set its value to D_{ji} (left-hand side of the main diagonal). Nevertheless, the symmetry restriction ($D_{ij} = D_{ji}$) imposes serious numerical difficulties, a $N \times N$ distance matrix must be declared now. The time dependence and memory consumption to run this conventional computer algorithm are both proportional to N^2 (see Table I). This limits the RLM use to computational small systems. For standard memory allocation at disposal, systems can have up to $N = 10^4$ points.

TABLE I: Memory and time consumptions for the conventional, one-seed and multiple-seed methods described. The multiple-seed method is the best one since it combines the low $O(N)$ memory consumption of the one-seed method and the low $O(N^2)$ time spent in the conventional implementation method.

Method	Conventional	One-Seed	Multiple-Seed
Time	N^2	N^3	N^2
Memory	N^2	N	N

We present below two methods, which replace the distance matrix by a mask, just as in the case of RPP. To re-obtain the distance D_{ij} (and obey the symmetry restriction), the deterministic feature of pseudo-random number generator is extensively explored.

B. One-Seed Method

The one-seed method uses two integer variables alternately as parameter to the random number generator, namely *seed* (used to generate new distances at right-hand side of the distance matrix main diagonal) and *dummy* (used to re-obtain the distances already generated and obey the metric symmetry). As a first step, initialize *seed* with an arbitrary value S_1 and save this value elsewhere. In the *i*th row, with *i* ranging from 1 to N:

1. set $D_{i,i} = 0$,
2. use the variable *seed* to sequentially generate new distances $D_{i,i+1}$, $D_{i,i+2}$, ..., $D_{i,N}$, and
3. to re-obtain each distance $D_{i,j}$, with j ranging from 1 to $i-1$, use the variable *dummy* in the following steps:
   (a) initialize it with S_1,
   (b) use it to make $i-j-1$ generator calls (these calls are made just to set *dummy* to the same *seed* value used to generate the distance $D_{j,i}$, their return values will be not used),
   (c) make a single call to re-obtain the desired distance $D_{i,j} = D_{j,i}$, and
   (d) make $N-i$ more calls (just to prepare *dummy* to the next j iteration, the return values are again not used, see Fig. 1).

This method enables us to numerically construct systems much larger than with the conventional method, but at the expense of a longer computational time. To generate all distances in a N-point map, the memory consumption is $O(N)$ (see Table I) while the required time is proportional to N^3, due to the expensive generator dummy calls (see Fig. 2).

C. Multiple-Seed Method

An improvement to the one-seed method is obtained replacing the integer variable *dummy* by an integer vector of size N, named *s_vec*. The initial seed of each row (S_1, S_N, S_{2N-2} ...) is stored in the correspondent entry of *s_vec*. Remark that only $N-1$ entries will be really used, but using N avoids undesired boundary tests. The integer variable *seed* is still used. Initialize it with some arbitrary value S_1. To understand the implementation, let us suppose the *i*-th row are been genereted, with *i* varying from 1 to N as before:

1. set $D_{i,i} = 0$,
2. store the current value of *seed* in *s_vec*(*i*) (*i*-th entry of *s_vec*) to be used further,
3. use *seed* to sequentially generate the distances $D_{i,i+1}$, $D_{i,i+2}$, ..., $D_{i,N}$, and
4. to re-obtain the distances $D_{i,j} = D_{j,i}$, with j varying from 1 to $i-1$, simply use *s_vec*(j) to make a single generator call, afterwards the current value of *s_vec*(j) will be modified for the calculation of the next iteraction (see Fig. 1).

The multiple-seed method keeps track of the seeds along the construction of all pair distances and may drastically reduce computation time to order N^2 just like the conventional implementation, but with memory consumption of order N, just as the one-seed method.

V. CONCLUSION

In this paper we have developed new algorithms to deal numerically with large random link systems. In particular, the multiple-seed algorithm is the best compromise between time and memory consumptions.

Acknowledgements

We thank O. Kinouchi and R. S. González for fruitful discussions. A.S.M. acknowledges the support from CNPq (305527/2004-5) and FAPESP (2005/02408-0).

[1] M. Mézard, G. Parisi, Europhys. Lett. **2**, 913 (1986).
[2] B. Harris, Ann. Math. Stat. **31**, 1045 (1960).
[3] B. Derrida, H. Flyvbjerg, J. Phys. (Paris) **48**, 971 (1987).
[4] S. A. Kauffman, J. Teor. Biol. **22**, 437 (1969).

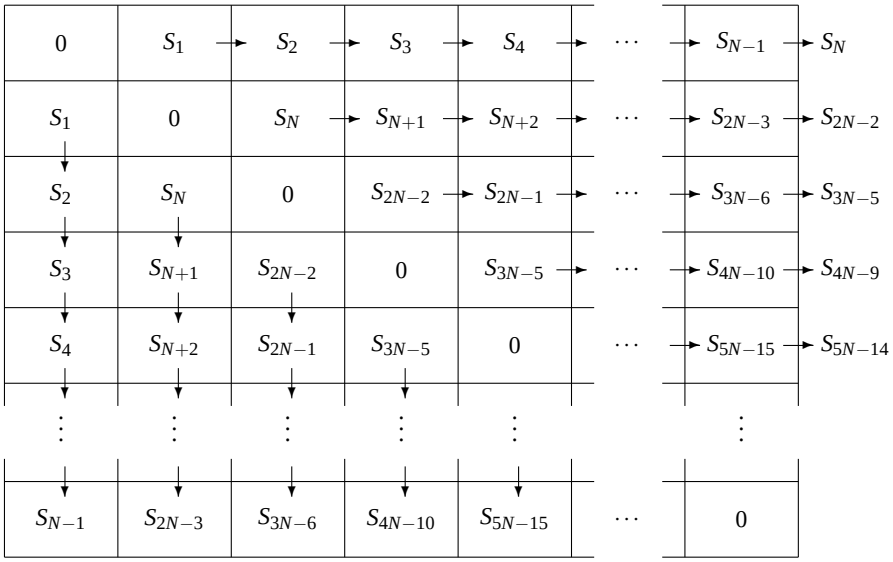

FIG. 1: Illustration of the seed evolution at each call to the numerical random number generator along the distance matrix in the conventional, one- and multiple-seed algorithms. Each seed is used to generate the distance placed in its correspondent position.

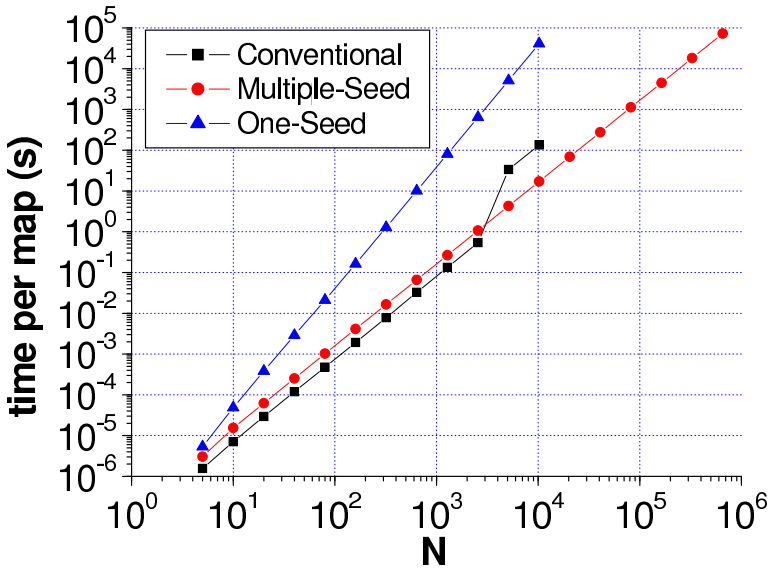

FIG. 2: Processing time as a function of system size. This dependence is well described by a power law with exponent 2 for the conventional and multiple-seed methods and exponent 3 for the one-seed method. While the conventional method can deal with systems of size around 10^4, the multiple-seed method can deal with systems more than 100 times greater. The discontinuity in the conventional method curve corresponds to the moment when swap to the disk started to be performed.

[5] A. G. Percus, O. C. Martin, Phys. Rev. Lett. **76**, 1188 (1996).
[6] N. J. Cerf, J. H. B. de Monvel, O. Bohigas, O. C. Martin, A. G. Percus, J. Phys. I (France) **7**, 117 (1997).
[7] A. G. Percus, O. C. Martin, J. Stat. Phys. **94**, 739 (1999).
[8] D. Aldous, A. G. Percus, Proc. Nat. Acad. Sci. USA **100**, 11211 (2003).
[9] D. J. Aldous, Proc. R. Soc. A **461**, 825 (2005).
[10] J. H. B. de Monvel, O. C. Martin, Phys. Rev. Lett. **79**, 167 (1997).
[11] J. Houdayer, J. H. B. de Monvel, O. C. Martin, Eur. Phys. J. B **6**, 383 (1998).
[12] O. Kinouchi, A. S. Martinez, G. F. Lima, G. M. Loureno, S. Risau-Gusman, Physica A **315**, 665 (2002).
[13] C. A. S. Terçariol, A. S. Martinez, Phys. Rev. E **72**, 021103 (2005).
[14] S. Risau-Gusman, A. S. Martinez, O. Kinouchi, Phys. Rev. E **68**, 016104 (2003).
[15] A. S. Martinez, O. Kinouchi, S. Risau-Gusman, Phys. Rev. E **69**, 017101 (2004).
[16] G. F. Lima, A. S. Martinez, O. Kinouchi, Phys. Rev. Lett. **87**,

010603 (2001).
[17] H. E. Stanley, S. V. Buldyrev, Nature (London) **413**, 373 (2001).
[18] D. Boyer, O. Miramontes, G. Ramos-Fernandez, J. L. Mateos, G. Cocho, Physica A **342**, 329 (2004).
[19] D. Boyer, H. Larralde, Complexity **10**, 52 (2005).