• 1400191284
(cpic)

On RNC Approximate Counting

J. Díaz
M.J. Serna
P. Spirakis

Report LSI–94–45–R

On RNC Approximate Counting (Extended Abstract) [*]

J.Diaz[†] M.Serna[†] P.Spirakis[‡]

Abstract

In this work we look into the parallelization (in the NC sense) of the Markov Chain approach to almost random generation (as described in [Sin93]). We prove that for several problems rapid mixing is "equivalent" to RNC, but there are rapid mixing chains for which the method can not be parallelized (unless NC=P). In particular, the rapid mixing chain used by Jerrum and Sinclair to approximate counting the number of perfect matchings in a dense bipartite graph, is proved to be P-complete.

1 Introduction

Many problems involving the counting of the number of solutions of combinatorial structures, are well known to be difficult. Valiant defined the class $\#P$ of computationally equivalent counting problems [Val79b]. For many of the problems in this class, their decision counterpart is P. It is well known that unless the polynomial hierarchy collapses, $P \neq \#P$ [Pap94]. This fact implies that for any $\#P$-complete problem, exact counting is apparently intractable for many interesting problems that are complete for $\#P$ [Pap94]. The most paradigmatic of these problems is to compute the permanent of a matrix, that turns out to be equivalent to count the number of perfect matchings in a bipartite graph [Val79a]. The hardness of these counting problems has led to approximate the counting. Pioneer work in this line was the paper

[*]This research was supported by the ESPRIT BRA Program of the EC under contract no. 7141 ,project ALCOMII.

[†]Departament de Llenguatges i Sistemes, Universitat Politècnica Catalunya, Pau Gargallo 5, 08028-Barcelona {diaz,mjserna}@lsi.upc.es

[‡]CTI,Patras, Greece, spirakis@cti.gr

[KLM89] where they construct a Randomized Fully Approximation Scheme for some difficult counting problems. Later, it was discovered that for the problems which are self-reducibles, approximate counting is equivalent to almost uniform generation [JVV86]. The almost uniform generation problem consist in picking at random an element of a finite set characterized by some property, with a relative error of at most ϵ with respect to the probability that a given element is chosen.

Since most of the interesting problems are self-reducible, the approximation of counting is reduced to the almost uniform generation. A technique that has proved to be very useful to solve the almost uniform generation problem, is the Markov Chain technique. Given a problem, define a Markov chain where the states are all possible solution, plus possibly a small fraction of quasi-solutions, and the transitions are certain probabilistics rules that allow us to remain in the same state or to pass to a new state. Under certain properties of the underlying graph, it can be proved that a polynomial (in the input size) random walk on the states give us an almost random generated element from the stationary distribution of the Chain. The difficulty of this method is to prove rapid convergence to the stationary distribution, what is called "rapid mixing. Over the past years, a large body of literature has been devoted to the subject of almost random generation through Markov Chain, and methods of proving rapid mixing. Two excellent surveys are [Sin93, Vaz91].

A question of interest is the possibility of parallelizing the generation. In other words, finding a parallel simulation of the random walk, in polylog steps and using a polynomial number of processors, without changing the rapid mixing property.

In order to be able to parallelize the procedure, some assumptions are needed. First each transition must be done in RNC. Furthermore assume that the random choices along the walk can be done independently for each state. That means that all random choices can be done in $\Theta(1)$ parallel steps, at the beginning. In the case of Markov Chains for with some constant threshold probability indicating whether to proceed or not, we get shorter sequences of steps. Second, at least one state of the Markov Chain can be obtained in NC (or RNC).

In general we can model the problem as follows:

Associated problem: Given a state A of the Markov chain, together with a polynomial (in n) sequence of basic transitions (operations that change the state A). The goal is to obtain the state A^* obtained after processing a sequence, and represent by $[A, <$ sequence $>]$.

Let us look at some well known Markov chains. **The Hypercube** of dimension n can be used to generate uniformly at random a binary string of length n. In this case the sequence of operations will be a collection of bit switches, each transition is defined by the Hamming distance 1 between states. In Section 3.1 of [Sin93] it is shown that the chain is rapid mixing.

This Markov chain verifies the requirements stated before. In parallel, choose randomly l numbers in the range $1, \ldots, n$ that corresponds to the bits to be changed. As initial n-bit vector, take the 0 vector.

Thus the associated problem is: Given a n-bit vector A together with a sequence of l integers $a_1, \ldots, a_l \in \{1, \ldots n\}$. Compute the n-bit vector $A^* = [A, < a_1, \ldots, a_l >]$. Where the basic operation is to switch the a_i-th bit.

In order to perform the computation in parallel, we change the representation, each integer a_i is represented by a n-bit vector A_i where all components are 0 except the i-th, that is 1. Notice the switch of bit i in vector B can be computed by $B \oplus A_i$ where $\oplus$ denotes the modulo two addition. The problem can be reformulated as $A^* = (\ldots(A \oplus A_1)\ldots) \oplus A_l$. Notice that $\oplus$ is an associative operation, thus we can apply the *tree contraction technique* and compute A^* in NC [JaJ92]. Therefore we can have a RNC almost uniform generator for element of $\{0,1\}^n$.

2 The Spanning Tree

Let us look at a more involved example. The almost uniform generation of random spanning trees of a given graph $G = (V, E)$. This problem has an exact solution in NC. It is easy to make a parallel implementation of the sequential algorithm given in [Gue83]. However that exact NC generation has a large cost in resources, the work is $W = O(n^6 \log^3 n)$. A Markov Chain approach to the generation (and counting) of spanning trees, is given in example 3.20 of [Sin93]. Other approach is in the paper by [Bro89], where he generates a random spanning tree, by doing a random walk in the graph. It is not difficult, to parallelize his algorithm to get an NC random generator for spanning tree, with expected work $W = O(n^4 \log^2 n)$. Recall that we wish to pursuit the possibility of a systematic parallelization of the Markov Chain approach to counting and generation, therefore we shall concentrate in the Jerrum–Sinclair approach, as described in [Sin93]. Given an undirected graph $G = (V, E)$, $|V| = n$, the Sinclair Chain has as states all the spanning trees of G and the transitions are defined in the following way: In state T_i select $e \in E$ uniformly at random. If $e \in T_i$ then remain in T_i, else form $T_i \cup \{e\}$ and randomly break the cycle. This Markov Chain is rapidly mixing.

Given a polynomial random sequence of edges $l =< e_1, \cdots, e_{p(n)} >$ starting at state T do the following NC simulation:

Algorithm 1

1. In parallel split l into k subsequences, $l = l_1, \ldots, l_k$ each without repeated elements.
2. In parallel
   For each l_i, $1 \leq i \leq k$ obtain a (random) spanning forest F_i
3. In parallel merge the k spanning forests to get F
4. In parallel, merge T with F.

To merge in parallel two spanning forests F_i and F_j of a given graph $G = (V, E)$, do the following procedure,

Algorithm 2

1. $F' = \emptyset, G' = (V, F_i \cup F_j)$
2. In parallel, compute connected components in the graph with edge set F'
3. In parallel and independently,
   each connected component chooses at random a label from $\{0, 1\}$.
4. In parallel, each component connected to only one other component chooses its neighbors. After, each of the remaining components labelled 0 chooses a neighbor labelled 1.
5. In parallel for each pair of neighbors, choose one edge connecting them and place it in F'. Remove all edges connecting them from G'

To sketch the correctness of Algorithm 1, let $F_1 \mu F_2$ denote the set of possible outputs when merging F_1 and F_2.

Lemma 1 *Given two spanning forests F_1, F_2 we have*

1. *$F_1 \mu F_2$ only contains (maximal) spanning forests in the graph with edge set the union of edges in F_1 and F_2.*

2. *Algorithm 2 can be implemented in $O(\log^3 n)$ parallel steps, using $O(n^2)$ processors with probability of error less than $1/4$.*

The following lemma can be proved by induction, (the proof will be given in the full length version)

Lemma 2 *Given three spanning forests F_1, F_2, F_3 and every spanning forest F, we have*

1. $F_1\mu(F_2\mu F_3) = (F_1\mu F_2)\mu F_3$.

2. $\Pr\{F \in F_1\mu(F_2\mu F_3)\} = \Pr\{F \in (F_1\mu F_2)\mu F_3\}$.

Notice that in step 4 of Algorithm 1, we always merge a spanning tree with a spanning forest, therefore we always obtain a spanning tree. To get a correct simulation we have only to prove that the probability of getting an output is the same as in the sequential case.

Given a set of k edges B, let $S(B)$ be the set of permutations of the elements in B. For a given $S \in \sigma(B)$ and spanning tree T let $T\sigma S$ the set of trees that can be reached following the sequential random walk S, then we have

Lemma 3 *For every set of edges B, and spanning trees T, T_1 the following holds*

1. $T\mu B = \cup_{S\in S(B)} T\sigma S$

2. $\Pr\{T_1 \in T\mu B\} = \sum_{S\in S(B)} \Pr\{T_1 \in T\sigma S\} \Pr\{S\}$

Now, we can state our first Theorem,

Theorem 1 *Given a state T of the Markov chain, Algorithm 1 simulates a random walk directed by l in $O(\log^3 n)$ parallel steps, using $O(n^5)$ processors with probability of error less than $1/4$.*

Notice, our parallel procedure simulates a random walk in the original Markov chain corresponding to some permutation of the initially chosen edges as far as the random procedure ends. When we simulate the procedure in RNC we can get that some of the mergings do not arrive to the end, in such a case we just repeat it.

3 P-completeness

The previous examples seem to indicate that *rapid mixing* is synonym of NC simulation. Unfortunatelly things are not so simple.

Let us examine the following Markov chain defined to generate a random subset of size k or $k+1$, from a given set S with $|S| = n$. This Markov Chain has as states all subsets of S with sizes k or $k+1$. A transition from state A is defined by the following procedure (the numbers at the end of the line will be used later as references to the type of operation):

```
pick random $x, y \in S$
If $|A| = k + 1$ then
  if $x = y$ and they are in $S$, $A$ goes to $A - \{x\}$        [2]
  else $A$ stays in $A$
If $|A| = k$ then
  if $x \in A$ and $y \notin A$
    with probability 1/2 $A$ goes to $A - \{x\} + \{y\}$  [1]
    with probability 1/2 $A$ goes to $A + \{y\}$          [3]
  else $A$ stays in $A$
```

Lemma 4 *The above Markov chain is ergodic. Moreover it has the rapid mixing property.*

Therefore, any random walk of polynomial length, generates with distribution near the stationary, a subset of size k or $k+1$.

Consider the associated problem given a subset A of k elements, together with a sequence of operations. Compute $A^* = [A, <\text{sequence}>]$.

To prove that the associate problem is P-complete, we transform it into a decision problem, add an extra element s of S to the instance, and rather than asking for A^* ask whether $s \in A^*$.

We present a reduction from the monotone alternating fan out two Circuit Value Problem (from now on CVP), assuming that inputs have fan-out one. Given such a circuit $\alpha = < g_1, \ldots, g_r >$ let us do some preprocessing:

1. Enumerate all gates preserving the layered structure, that means inputs, followed by level 1 gates, followed by level 2 gates, ..., followed by the gates at the latest level.

2. Each gate computes for both inputs, whether it is the first of the second gate (in the above enumeration) that uses the input.

The two steps can be implemented in NC. Further as after step 2 each gate knows whether it is the first or the second gate that uses the inputs, we can assume that each gate is defined by the equation $g_k = g_i^\alpha * g_j^\beta$ where $\alpha, \beta \in \{1, 2\}$ and $*$ represents AND or OR.

The set S will have $3r$ elements, $a_1, \dots, a_r$, $b_1^1, \dots, b_r^1$ and $b_1^2, \dots, b_r^2$. k will be r and the set $A = \{a_1, \dots, a_r\}$. The extra element will be b_n^1. Each gate will produce a subsequence of operations that will follow the same gate's order. The construction is the following:

- When we have the equation $g_k = 1$ add operation $(a_k, b_k^1, 1)$
- When $g_k = g_i^\alpha \vee g_j^\beta$ add operations

$$(b_i^\alpha, b_k^1, 1), (b_k^1, b_j^\beta, 1), (b_k^1, b_k^2, 3), (a_k, a_k, 2)$$

- When $g_k = g_i^\alpha \wedge g_j^\beta$ add operations

$$(b_i^\alpha, b_k^1, 1), (b_j^\beta, b_k^1, 1), (b_k^1, b_k^2, 3), (a_k, a_k, 2)$$

where 1,2 and 3 corresponds to the type of transition according to the previous definition.

Theorem 2 *CVP is NC reducible to the associated problem to the Markov Chain for the problem under consideration.*

(Sketch of Proof)

Each step of the reduction can be performed in NC, furthermore the correctness of the reduction comes from the following considerations:

Claim 1 Let $B = [A, <(x, z), (z, y)>]$ assuming that $z \notin A$ we get

$$z \in B \text{ if and only if } x \in A \text{ and } y \in A$$

Claim 2 Let $B = [A, <(x, z), (y, z)>]$ assuming that $z \notin A$ we get

$$z \in B \text{ if and only if } x \in A \text{ or } y \in A$$

Claim 3 a_k remains in the set until gate k sequence is processed and gets out only when the output of gate k is true. In such a case after processing gate k sequence both b_k^1 and b_k^2 belong to the computed set. When the output of gate k is false, neither b_k^1 nor b_k^2 are in the set after processing the corresponding sequence.

Thus we get that α outputs true if and only if $b_r^1 \in A^*$. ■

One must be careful to understand the last result. It says that unless $NC = P$, we can not simulate in RNC a random walk on the described rapid mixing Markov Chain. It does not say anything about the RNC generation of the subsets. As a matter of fact, it is easy to obtain an exact generation of the subsets in the problem, by flipping coins. Therefore rapid mixing is not a sufficient condition to assure us RNC generation.

Surprisingly, the same kind of result applies to the generation of perfect matchings for bipartite graphs. The theorem below just says that the Markov Chain approach of Jerrum and Sinclair is hard to parallelize, we do not claim anything about the generic problem of almost generating (approximate counting) perfect matchings in bipartite graphs. The problem of using Markov chains to approximate the number of perfect matching in dense bipartite graphs was introduced by [Bro86]. The proof of his theorem had a mistake, and Jerrum and Sinclair used the conductance to show the correctness of the algorithm [Sin93]. Since then, other people have obtained results, [DL92] using Markov chains and [Ras94] using a direct approach. Our result applies to the Jerrum-Sinclair approach. Again we just give a hint of the proof.

Theorem 3 *The associated problem for almost random generation of perfect matchings on dense bipartite graphs is P-complete.*

(Sketch of proof)

Assume that we have a circuit and we do the same preprocessing as in the above reduction to get a representation $\alpha =< g_1, \ldots, g_r >$. We consider the following reduction. Each gate will have associated four vertices v_i, x_i, w_i, y_i two corresponding to the first output (v and w) and two to the second (x and y). And two additional nodes v_0 and w_0. We will assume that the graph is the complete bipartite graph, all v and x nodes are in one set and all w and y nodes are in the other.

The initial matching is as follows:

1-inputs and OR gates: each node is matched with it's twin, that means we have edges (v_i, w_i) and (x_i, y_i).

0-inputs and AND gates : each node is matched to the other node that is not its twin, that means we have edges (v_i, y_i) and (x_i, w_i)..

The additional vertices are matched by the edge (v_0, w_0).

The sequence is the following, the first edge is (v_0, w_0) then depending on the type of gate each gate will produce a sequence of edges and finally we will add (v_0, w_0).

For gate $g_k = g_i^\alpha \vee g_j^\beta$ (assume that $\alpha = \beta = 1$ otherwise substitute v/x and w/y). We define the sequence in three blocks:

1. $(w_i, v_0), (w_0, v_j), (w_j, v_0), (w_0, v_i), (w_0, v_j), (w_i, v_0)$
2. $(v_i, w_k), (x_k, y_k), (v_k, w_k), (w_j, v_k), (x_k, y_k), (v_k, w_k)$
3. $(v_i, w_i), (v_j, w_j), (x_i, w_i), (v_j, y_j)$

For a gate $g_k = g_i^\alpha \wedge g_j^\beta$ (assume that $\alpha = \beta = 1$ otherwise substitute v/x and w/y) we add:

1. $(w_i, v_0), (w_0, v_j), (w_j, v_0), (w_0, v_i), (w_0, v_j), (w_i, v_0)$
2. $(x_i, w_k), (v_k, y_k), (x_k, w_k), (y_j, v_k), (x_k, w_k), (v_k, y_k)$
3. $(v_i, w_i), (v_j, w_j), (x_i, w_i), (v_j, y_j)$

It is easy to see that the first block just changes the matching to a special configuration in which we can isolate the case in which the codification of g_k must be changed, the second block just changes g_k when necessary, and the third block just restores gates i and j. ■

References

[Bro86] A. Broder. How hard is to marry at random. In *18th ACM Symposium on Theory of Computing*, pages 50–58, 1986.

[Bro89] A. Broder. Generating random spanning trees. In *30th IEEE Symposium on Foundations of Computer Science*, pages 442–447, 1989.

[DL92] P. Dagum and M. Luby. Approximating the permanent of graphs with large factors. *Theoretical Computer Science*, 102:283–305, 1992.

[Gue83] A. Guenoche. Random spanning tree. *Journal of Algorithms*, 4:214–220, 1983.

[JaJ92] J. JaJa. *An introduction to parallel algorithms*. Addison-Wesley, 1992.

[JVV86] M. Jerrum, L. Valiant, and V. Vazirani. Random generation of combinatorial structures from a uniform distribution. *Theoretical Computer Science*, 43:169–188, 1986.

[KLM89] R.M.. Karp, M. Luby, and N. Madras. Monte-carlo approximation algorithms for enumeration problems. *Journal of Algorithms*, 18:429–448, 1989.

[Pap94] C. Papadimitriu. *Computational Complexity*. Addison-Wesley, 1994.

[Ras94] L.E. Rasmussen. Approximating the permanent: A simple approach. *Journal of Random Structures and Algorithms*, 5:349–361, 1994.

[Sin93] A. Sinclair. *Algorithm for random generation and counting: A Markov chain approach*. Birkhäuser, Boston, 1993.

[Val79a] L.G. Valiant. The complexity of computing the permanent. *Theoretical Computer Science*, 8:189–201, 1979.

[Val79b] L.G. Valiant. The complexity of enumeration and reliability problems. *SIAM J on Computing*, 8:410–421, 1979.

[Vaz91] V. Vazirani. Rapidly mixing Markov chains. In B. Bollobas, editor, *Probabilistic combinatorics and its applications*, pages 99–121. American Mathematical Society, 1991.

Departament de Llenguatges i Sistemes Informàtics
Universitat Politècnica de Catalunya

Research Reports – 1994

LSI-94-1-R	"Logspace and logtime leaf languages", Birgit Jenner, Pierre McKenzie, and Denis Thérien.
LSI-94-2-R	"Degrees and reducibilities of easy tally sets", Montserrat Hermo.
LSI-94-3-R	"Isothetic polyhedra and monotone boolean formulae", Robert Juan-Arinyo.
LSI-94-4-R	"Una modelización de la incompletitud en los programas" (written in Spanish), Javier Pérez Campo.
LSI-94-5-R	"A multiple shooting vectorial algorithm for progressive radiosity", Blanca Garcia and Xavier Pueyo.
LSI-94-6-R	"Construction of the Face Octree model", Núria Pla-Garcia.
LSI-94-7-R	"On the expected depth of boolean circuits", Josep Díaz, María J. Serna, Paul Spirakis, and Jacobo Torán.
LSI-94-8-R	"A transformation scheme for double recursion", José L. Balcázar.
LSI-94-9-R	"On architectures for federated DB systems", Fèlix Saltor, Benet Campderrich, and Manuel García-Solaco.
LSI-94-10-R	"Relative knowledge and belief: SKL preferred model frames", Matías Alvarado.
LSI-94-11-R	"A top-down design of a parallel dictionary using skip lists", Joaquim Gabarró, Conrado Martínez, and Xavier Messeguer.
LSI-94-12-R	"Analysis of an optimized search algorithm for skip lists", Peter Kirschenhofer, Conrado Martínez, and Helmut Prodinger.
LSI-94-13-R	"Bases de dades bitemporals" (written in Catalan), Carme Martín and Jaume Sistac.
LSI-94-14-R	"A volume visualization algorithm using a coherent extended weight matrix", Daniela Tost, Anna Puig, and Isabel Navazo.
LSI-94-15-R	"Deriving transaction specifications from deductive conceptual models of information systems", María Ribera Sancho and Antoni Olivé.
LSI-94-16-R	"Some remarks on the approximability of graph layout problems", Josep Díaz, María J. Serna, and Paul Spirakis.

LSI-94-17-R "SAREL: An assistance system for writing software specifications in natural language", Núria Castell and Àngels Hernández.

LSI-94-18-R "Medición del factor modificabilidad en el proyecto LESD" (written in Spanish), Núria Castell and Olga Slávkova

LSI-94-19-R "Algorismes paral·lels SIMD d'extracció de models de fronteres a partir d'arbres octals no exactes" (written in Catalan), Jaume Solé and Robert Juan-Arinyo.

LSI-94-20-R "Una paral·lelització SIMD de la conversió d'objectes codificats segons el model de fronteres al mdel d'octrees clàssics" (written in Catalan), Jaume Solé and Robert Juan-Arinyo.

LSI-94-21-R "Clausal proof nets and discontinuity", Glyn Morrill.

LSI-94-22-R "A formal method for the synthesis of update transactions in deductive databases without existential rules", Joan A. Pastor.

LSI-94-23-R "On the sparse set conjecture for sets with low density", Harry Buhrman and Montserrat Hermo.

LSI-94-24-R "On infinite sequences (almost) as easy as π", José L. Balcázar, Ricard Gavaldà, and Montserrat Hermo.

LSI-94-25-R "Updating knowledge bases while maintaining their consistency", Ernest Teniente and Antoni Olivé.

LSI-94-26-R "Structural facilitation and structural inhibition", Glyn Morrill.

LSI-94-27-R "Formalising existential rule treatment in the automatic synthesis of update transactions in deductive databases", Joan A. Pastor.

LSI-94-28-R "Proceedings of the Fifth International Workshop on the Deductive Approach to Information Systems and Databases" (Aiguablava, 1994), Antoni Olivé (editor).

LSI-94-29-R "Towards a VRQS representation", Mariona Taulé Delor.

LSI-94-30-R "MACO: Morphological Analyzer Corpus-Oriented", Sofía Acebo, Alicia Ageno, Salvador Climent, Javier Farreres, Lluís Padró, Francesc Ribas, Horacio Rodríguez, and Oscar Soler.

LSI-94-31-R "Lexical mismatches: a semantic representation of adjectives in the LKB", Salvador Climent and Carme Soler.

LSI-94-32-R "LDB/LKB integration", German Rigau, Horacio Rodríguez, and Jordi Turmo.

LSI-94-33-R "Learning more appropriate selectional restrictions", Francesc Ribas.

LSI-94-34-R "Oracles and queries that are sufficient for exact learning", Nader H. Bshouty, Richard Cleve, Ricard Gavaldà, Sampath Kannan, and Christino Tamon.

Copies of reports can be ordered from:

Nuria Sánchez
Departament de Llenguatges i Sistemes Informàtics
Universitat Politècnica de Catalunya
Pau Gargallo, 5
08028 Barcelona, Spain
secrelsi@lsi.upc.es