Fast solution of NP-hard coloring problems on large random graphs

Andrea Bedini, Jesper Jacobsen

Fast solution of NP-hard coloring problems on large random graphs

Andrea Bedini[1] and Jesper Lykke Jacobsen[2]
[1] *Dipartimento di Fisica dell'Università degli Studi di Milano and INFN, Sezione di Milano, via Celoria 16, I-20133 Milano, Italy and*
[2] *LPTENS, 24 rue Lhomond, F-75231 Paris, France*
(Dated: March 23, 2010)

Combining tree decomposition and transfer matrix techniques provides a highly efficient and very general algorithm for computing exact partition functions of statistical models defined on large graphs. We illustrate this by considering the hard problem of computing the exact number of vertex colorings for randomly generated planar graphs with up to $N = 100$ vertices.

PACS numbers: 02.70.-c, 05.50+q, 89.70.Eg

Perhaps the most important outstanding question in theoretical computer science is whether the class P of decision problems that can be *solved* in polynomial time coincides with the class NP of problems for which a proposed solution can be *verified* in polynomial time. NP-complete problems are those to which any problem in NP can be reduced in polynomial time. At present, no polynomial-time algorithm has been found for any of the thousands of known NP-complete problems and it is hence widely believed that P $\neq$ NP. Likewise, one can define a counting analogue of NP, denoted by #P, as the class of enumeration problems in which the structures being counted are recognizable in polynomial time. Clearly, #P problems are as hard as problems in NP.

In parallel, in theoretical physics, there is a steady interest for problems related to graph theory and network design. In particular, the approach of statistical physics is to enclose the properties of a physical system in a partition function, which is a weighted sum over the states. Many interesting problems can therefore be restated as counting problems on graphs. Not surprisingly, these problems turn out to be themselves interesting problems in mathematics and theoretical computer science (for a review see, for example, [1]).

The Potts model is a central instance of this connection between statistical mechanics and enumerative combinatorics. Its partition function counts, for the zero-temperature antiferromagnet, the number of *vertex colorings*, which are assignations of any of Q different colors to each vertex, so that neighboring vertices are colored differently. This counting problem is in the #P class.

Finding efficient algorithms for solving those problems has therefore a two-fold interest: to study the models in their own right, and to explore the effective computational hardness of NP problems. Although it is very unlikely to find an efficient (with polynomial time requirements) algorithm, lowering the coefficient of the exponent can still make a huge difference.

Algorithmic progress has been made by several, usually widely separated, communities.

On one hand, statistical physicists have shown that the relevant partition functions can be constructed in analogy with the path integral formulation of quantum mechanics. To this end, the configuration of a partially elaborated graph are encoded as suitable quantum states, and the constant-time surface is swept over the graph by means of a time evolution operator known as the transfer matrix. Although rarely stated, this approach is valid not only for regular lattices but also for arbitrary graphs.

On the other hand, graph theorists have used that graphs can be divided into "weakly interacting" subgraphs through a so-called tree decomposition [2], and solutions obtained for the subgraphs can be recursively combined into a complete solution.

In this Letter we show how the tree decomposition and transfer matrix methods can be combined in a very natural way. The main idea is that the tree decomposition is compatible with a recursive generalization of the time evolution concept. Borrowing ideas from string theory, the combination of partial solutions is obtained by the fusion of suitable quantum spaces. The resulting algorithm works on any graph, and can readily be adapted to many other problems of statistical physics, by suitable modifications of the state spaces and the fusion procedure.

In particular, we will apply this technique to the problem of computing the chromatic polynomial on planar graphs, obtaining exact solutions, for graphs with $N \simeq 100$ vertices, in only a few seconds.

The Potts model and vertex coloring. We first recall the relation between the vertex coloring problem and the Potts model. Consider a graph $G = (V, E)$ with vertices V and edges E, and let $\sigma_i = 1, 2, \ldots, Q$ be the color of vertex $i \in V$. Then

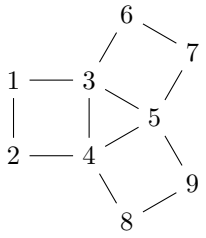
$$Z_G = \sum_{\sigma} \prod_{(ij)\in E} \mathrm{e}^{K\delta(\sigma_i,\sigma_j)} \tag{1}$$

is the partition function of the Potts model on G. The Kronecker delta $\delta(\sigma_i, \sigma_j) = 1$ if $\sigma_i = \sigma_j$, and 0 otherwise. Inserting the obvious identity $\mathrm{e}^{K\delta(\sigma_i,\sigma_j)} = 1 + v\delta(\sigma_i, \sigma_j)$, with $v = \mathrm{e}^K - 1$, and expanding out the product we obtain the Fortuin-Kasteleyn expansion [3]

$$Z_G(Q, v) = \sum_{A\subseteq E} v^{|A|} Q^{k(A)}\,, \tag{2}$$

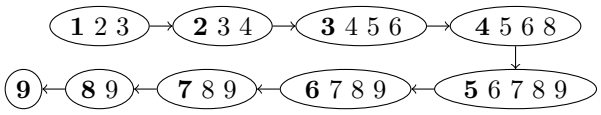
where $k(A)$ is the number of connected components in the subgraph $G' = (V, A)$. Obviously, in the antiferromagnetic limit $K \to -\infty$ (or equivalently $v \to -1$) the only surviving configurations are proper Q-colorings of the graph G. Indeed, the special case $\chi_G(Q) = Z_G(Q, -1)$ is a polynomial in Q, known as the *chromatic polynomial*, and equals the number of vertex colorings.

Transfer matrix. We first describe how to compute the partition function $Z_G(Q, v)$ in (2) by a traditional transfer matrix method. In short, the combined action of linear operators builds a superposition of all configurations appearing in the partition funciton with their correct Boltzmann weight entering as coefficients. To better illustrate this procedure, consider the following example graph G



(3)

We first have to define the order $\{v_t\}$ in which vertices will be processed. This order is the basis for the construction of a "time slicing" of the graph. With each *time step* is associated a *bag* (a vertex subset) of active vertices. A vertex becomes active as soon as one of its neighbors is processed and it stays active until it is processed itself. Taking the vertices in lexicographic order we obtain the following decomposition:



(4)

where we wrote in bold face the vertex being processed at each time step.

Each bag has its own set of basis states consisting of the *partitions* of the currently active vertices. For instance, in the first time step, the basis states are the five partitions of the three-element set $\{1, 2, 3\}$:

$$\big|\,\overset{\circ}{1}\ \overset{\circ}{2}\ \overset{\circ}{3}\,\big\rangle, \big|\,\overset{\frown}{1}\ \overset{}{2}\ \overset{\circ}{3}\,\big\rangle, \big|\,\overset{\circ}{1}\ \overset{\frown}{2}\ \overset{\circ}{3}\,\big\rangle, \big|\,\overset{\frown}{1}\ \overset{}{2}\ \overset{\frown}{3}\,\big\rangle, \big|\,\overset{\frown}{1}\ \overset{}{2}\ \overset{}{3}\,\big\rangle$$

These partitions describe how the active vertices are interconnected through $A \cap E_t$, where $E_t \subseteq E$ is the subset of edges having been processed at time t. A *state* is a linear superposition of basis states.

Processing a vertex consists in processing edges connecting it to unprocessed vertices and then deleting it. Since each edge $e \in E$ may or may not be present in A we process an edge (i, j) by acting on the state with an operator of the form $1 + v\mathsf{J}_{ij}$ where 1 is the identity operator and J_{ij} a *join operator*. A join operator acts on a basis state by amalgamating the blocks containing vertices i and j.

$$\mathsf{J}_{ij}\,\big|\,\overset{\circ}{i}\ \overset{\circ}{j}\,\big\rangle = \big|\,\overset{\frown}{i}\ \overset{}{j}\,\big\rangle, \qquad \mathsf{J}_{ij}^2 = \mathsf{J}_{ij} \tag{5}$$

Vertex deletion is defined in terms of a *deletion operator* D_i that removes i from the partiton and applies a factor Q (resp. 1) if i was (resp. was not) a singleton.

$$\mathsf{D}_i\,\big|\,\overset{\circ}{i}\ \overset{\circ}{j}\ \cdots\big\rangle = Q\,\big|\,\overset{\circ}{j}\ \cdots\big\rangle \tag{6a}$$

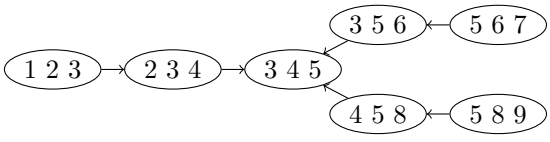
$$\mathsf{D}_i\,\big|\,\overset{\frown}{i}\ \overset{}{j}\ \cdots\big\rangle = \big|\,\overset{\circ}{j}\ \cdots\big\rangle \tag{6b}$$

For example, in (4), processing the first bag means computing the following composition $\mathsf{D}_1\,(1 + v\mathsf{J}_{12})\,(1 + v\mathsf{J}_{13})\,\big|\,\overset{\circ}{1}\ \overset{\circ}{2}\ \overset{\circ}{3}\,\big\rangle$ which gives $(Q + 2v)\,\big|\,\overset{\circ}{2}\ \overset{\circ}{3}\,\big\rangle + v^2\big|\,\overset{\frown}{2}\ \overset{}{3}\,\big\rangle$, concluding the first time step.

When a new active vertex is encountered it is inserted, as a singleton, in each partition composing the current state. After processing the last bag, the complete partition function (2) is obtained as the coefficient of the empty partition resulting from the deletion of the last active vertex.

At each step, the time and memory requirements are determined by the *bag size* n which is the number of vertices simultaneously active. If the graph is planar, the number of partitions to be considered is at most the Catalan number C_n, whose generating function is $C(z) = \sum_{n=0}^{\infty} C_n z^n = (1 - \sqrt{1 - 4z})/(2z)$. If the graph is not planar, the number of partitions is at most the Bell number B_n, with generating function $B(z) = \sum_{n=0}^{\infty} B_n \frac{z^n}{n!} = \exp(\mathrm{e}^z - 1)$. We have $C_n = 4^n n^{-3/2} \pi^{-1/2} [1 + O(1/n)]$, whereas B_n grows super-exponentially.

Tree decomposition. It turns out that the decomposition (4) of G is a special case of a more general construction. By definition, a *tree decomposition* [2] of a graph $G = (V, E)$ is a collection of bags, organized as a tree (a connected graph with no cycles), and satisfying the following requirements: i) For each $i \in V$, there exists a bag containing i; ii) For each $(ij) \in E$, there exists a bag containing both i and j; iii) For any $i \in V$, the set of bags containing i is connected in the tree. The previous decomposition (4) is just a special case of a tree decomposition (a *path decomposition*). As an example of the general construction, applied to (3), consider



(7)

where the arrows form the unique path that connects each bag to the central one (the *root* of the tree).

The transfer matrix approach can be adapted naturally to this new general setting: Properties i)–ii) guarantee that each edge and vertex are processed within a definite

bag. Property iii) implies that each vertex has a definite life time in the recursion, its insertion and deletion being separated by the processing of all edges incident on it.

In this new version the algorithm starts from the root of the tree, which can be chosen arbitrarily, and runs through the tree recursively. Going up from one daughter bag D to its parent P implies deleting vertices $D \setminus P$, inserting vertices $P \setminus D$ and finally processing edges in P. A tree decomposition does not specify when an edge $e = (ij)$ must be processed. A simple recipe would be to process e as soon as one encounters a bag containing both i and j. However we note that this freedom of choice can be exploited to optimize the algorithm (see below).

The advantage of working with tree instead of path decompositions relies on the fact that in the former case a decomposition with smaller bags can be obtained (the latter being just a special case). Therefore, the number of states one has to keep track of is exponentially smaller, and the gain is significant.

Fusion. We now discuss the case when a parent bag P has several daughters D_ℓ with $\ell = 1, 2, \ldots, d$. In this case, vertex deletions and insertions are followed by a special *fusion* procedure. Suppose first $d = 2$, and let $\mathcal{P}_1$ be a partition of $D_1 \cap P$ with weight w_1, and $\mathcal{P}_2$ a partition of $D_2 \cap P$ with weight w_2. We can define the set E_1 by writing $\mathcal{P}_1 = \left(\prod_{e\in E_1} \mathsf{J}_e\right) \mathcal{S}_1$, where $\mathcal{S}_1$ is the all-singleton partition of $D_1 \cap P$. Since $\mathsf{J}_e\mathsf{J}_{e'} = \mathsf{J}_{e'}\mathsf{J}_e$ the order in the above product is irrelevant. Similarly define E_2 from $\mathcal{P}_2$. The fused state is therefore

$$\mathcal{P}_1 \otimes \mathcal{P}_2 = \left(\prod_{e\in E_1\cup E_2} \mathsf{J}_e \right) \mathcal{S}_{12} \,,$$

and it occurs with weight $w_1 w_2$, where $\mathcal{S}_{12}$ is the all-singleton partition of $(D_1 \cup D_2) \cap P$. For $d > 2$ daughters, the complete fusion can be accomplished by fusing D_1 with D_2, then fusing the result with D_3, and so on.

Let us illustrate the fusion procedure for G with the tree decomposition (7). After processing the two left-most bags and deleting vertex 2, the propagating state is

$$\omega_1 \big|\, \overset{\circ}{3}\ \overset{\circ}{4} \,\big\rangle + \omega_2 \big|\, \overset{\frown}{\overset{\circ}{3}\ \overset{\circ}{4}} \,\big\rangle = (\omega_1 + \omega_2 \mathsf{J}_{34}) \big|\, \overset{\circ}{3}\ \overset{\circ}{4} \,\big\rangle, \qquad (8)$$

where $\omega_1 = Q^2 + 3v(Q+v)$ and $\omega_2 = Q^2 v + 3Qv^2 + 4v^3 + v^4$. By symmetry, the same result is obtained for the two top right bags (replacing 4 by 5) and for the two bottom right bags (replacing 3 by 5). The fused state arriving in the central bag is then

$$\begin{aligned}(\omega_1 + \omega_2\mathsf{J}_{34})(\omega_1 + \omega_2\mathsf{J}_{35})(\omega_1 + \omega_2\mathsf{J}_{45}) \big|\, \overset{\circ}{3}\ \overset{\circ}{4}\ \overset{\circ}{5} \,\big\rangle \\ = \omega_1^3 \big|\, \overset{\circ}{3}\ \overset{\circ}{4}\ \overset{\circ}{5} \,\big\rangle + (3\omega_1\omega_2^2 + \omega_2^3) \big|\, \overset{\frown}{\overset{\circ}{3}\ \overset{\circ}{4}}\ \overset{\circ}{5} \,\big\rangle \\ + \omega_1^2\omega_2 \Big(\big|\, \overset{\frown}{\overset{\circ}{3}\ \overset{\circ}{4}}\ \overset{\circ}{5} \,\big\rangle + \big|\, \overset{\circ}{3}\ \overset{\frown}{\overset{\circ}{4}\ \overset{\circ}{5}} \,\big\rangle + \big|\, \overset{\frown}{\overset{\circ}{3}\ \overset{\circ}{4}\ \overset{\circ}{5}} \,\big\rangle \Big), \end{aligned} \qquad (9)$$

from which the result $Z_G(Q, v)$ follows upon deleting vertices 3,4,5.

Pruning. Problem specific features can be exploited to reduce further the number of basis states to be considered. As an example of this, note that in the coloring case ($v = -1$) the operator $\mathsf{O}_{ij} = 1 + v\mathsf{J}_{ij}$ associated with an edge (ij) is a projector, $\mathsf{O}_{ij}^2 = \mathsf{O}_{ij}$, and it annihilates the subspace of partitions where i and j are connected. It follows that one can discard basis states in which two vertices are connected, as soon as one discovers that an edge between them is going to be processed in the following. Especially before fusions this simple trick reduces substantially the number of basis states and thus leads to a big speed up.

Performance. For a planar graph, the state of a bag of size n is spanned by C_n basis states. (For a non-planar graph, replace C_n by B_n.) The memory needed by the algorithm is therefore proportional to $C_{n_{\max}}$, where $n_{\max}$ is the size of the largest bag. The time needed to process one edge in a bag of size n is proportional to C_n.

However, most of the time is spent fusing states. For a parent P with d daughters D_ℓ, the number of basis state pairs to be fused is

$$\sum_{\ell=1}^{d} C_{|\mathcal{D}_{\ell-1}\cap P|} C_{|D_\ell \cap P|} \,, \qquad (10)$$

where we have set $\mathcal{D}_k = \cup_{\ell=1}^{k} D_\ell$. Each of these elementary fusions can be done in time linear in the number of participating vertices. Note that we can choose the order of successive fusions so as to minimize the quantity (10).

It is therefore essential for the algorithm that one knows how to obtain a good tree decomposition. The minimum of $n_{\max} - 1$ over all tree decompositions is known as the tree width k, but obtaining this is another NP-hard problem. However, the simple algorithm `GreedyFillIn` [4] gives an upper bound k_0 on k and a tree decomposition of width k_0 in time *linear* in the number of vertices N. For uniformly generated planar graphs we find that for $N = 40$—a value enabling comparison with algorithms that determine k exactly—that $k_0 = k$ nearly always ($k_0 = k + 1$ with probability $\simeq 10^{-3}$). When k_0 is small enough that all the partitions can fit into the computer's memory, the algorithm has proven to be very fast with execution time in the order of seconds.

We choose to test our algorithm against the one presented by Haggard et al. in [5]. We first generated a uniform sample of 100 planar graphs for each size $N = |V|$ between 20 and 100 using Fusy's algorithm [6]. We then ran four different algorithms over this sample: the algorithm of [5], our first path-based transfer matrix algorithm, the new tree-based version algorithm and a tree-based version using the above pruning optimization. Path decompositions were obtained with a variant of the `GreedyFillIn` algorithm in which the resulting tree decomposition was forced not to branch. Average running times are presented in Fig. 1.

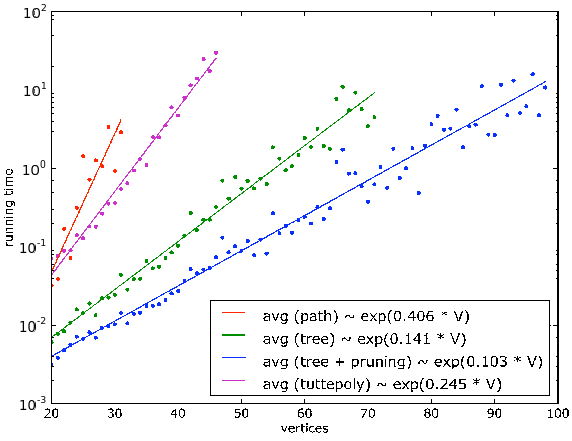


FIG. 1: Average running time in seconds on a random planar graph of fixed size. Each point is averaged over 100 graphs.

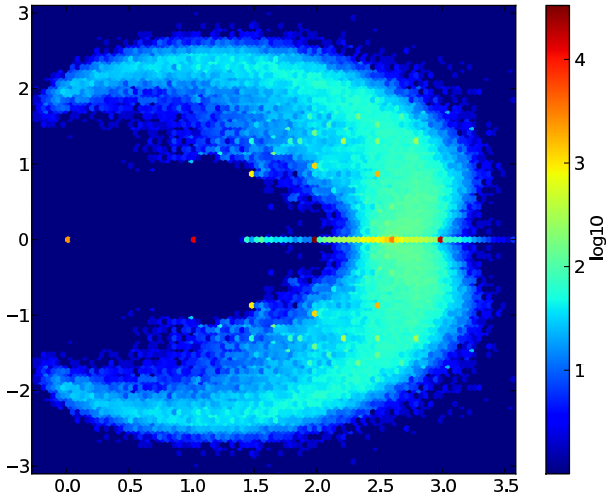


FIG. 2: Frequency histogram of $2.5 \cdot 10^5$ complex chromatic roots, obtained for a sample of 2500 planar graphs of size 100.

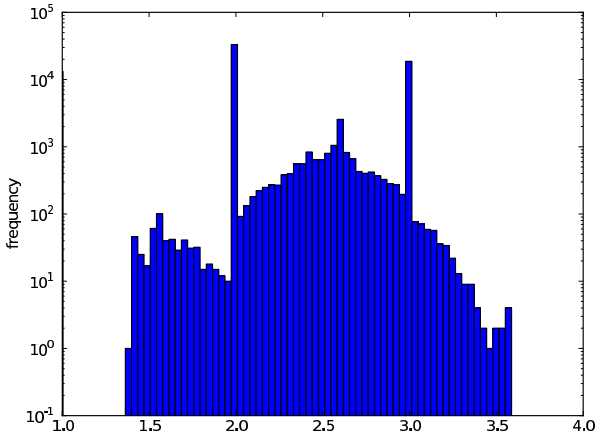


FIG. 3: Frequency histogram of the subset of chromatic roots in Fig. 2 that lie on the real axis.

Technical aspects. For completeness we give a few implementational details. Partitions relevant to the propagating state are kept in a hash table for fast (amortized constant) time access. The corresponding weights are polynomials in Q whose coefficients rapidly exceed the machine integer size $M = 2^{32}$ when running on big graphs. To solve this issue without requiring additional memory we used modular arithmetics: the algorithm was run many times with coefficients computed modulo primes $p < M$, and the original coefficients were reconstructed by the Chinese remainder theorem.

Results. As a simple application of our algorithm we obtained the distribution of the chromatic roots $\{Q \,|\, \chi_G(Q) = 0\}$ for large graphs in the complex plane (Fig. 2) and on the real axis (Fig. 3). This problem is interesting both in statistical mechanics and in graph theory [7]. As known from Lee-Yang theory, the roots signal phase transitions.

The absense of roots on the negative real axis, and the intervals $(0, 1)$ and $(1, 32/27]$ follow from a theorem [8]. It has been observed [7] that for *regular* planar graphs the Beraha numbers $B_k = \left(2\cos(\pi/k)\right)^2$, with $k \geq 2$ integer, are accumulation points of chromatic roots. While Fig. 3 shows clear peaks at $Q = B_k$ with $k = 2, 3, 4, 5, 6$ (we have $B_5 \simeq 2.61803$) these occur on top of a non-zero background density of roots. It seems likely that for $N \to \infty$ this background will extend to the interval $(32/27, 4)$ and peaks will occur at all Beraha numbers. We will discuss this further elsewhere [9].

Conclusion. We have presented an efficient algorithm for solving NP-hard enumeration problems on large graphs and illustrated it for vertex colorings. The algorithm can be extended to the counting versions of many other NP-hard problems: Hamiltonian walks and cycles, vertex cover, dominating set, feedback vertex set, minimum maximal independent set, *etc.* [9].

Acknowledgments. This work was supported by the European Community Network ENRAGE (grant MRTN-CT-2004-005616) and by the Agence Nationale de la Recherche (grant ANR-06-BLAN-0124-03).

[1] D.J.A. Welsh, *The computational complexity of some classical problems from statistical physics*, in G.R. Grimmett and D.J.A. Welsh (eds.), *Disorder in physical systems*, Clarendon Press, Oxford, 1990, pp. 307–321.

[2] N. Robertson and P.D. Seymour, J. Comb. Theory B **36**, 49-64 (1984).

[3] C.M. Fortuin and P.W. Kasteleyn, Physica **57**, 536–564 (1972).

[4] A.M.C.A. Koster, H.L. Bodlaender and S.P.M. van Hoesel, Elec. Notes in Disc. Math **8**, 54–57 (2001).

[5] G. Haggard, D.J. Pearce and G. Royle, *Computing Tutte polynomials.* Technical report NI09024-CSM, Isaac New-

ton Institute for Mathematical Sciences, 2009.
[6] E. Fusy, Random Struct. Alg. **35**, 464–522 (2009).
[7] J. Salas J and A.D. Sokal, J. Stat. Phys. **104**, 609–699 (2001).
[8] C. Thomassen, Comb. Prob. Comp. **6**, 497–506 (1997).
[9] A. Bedini and J.L. Jacobsen, in preparation.