A new algorithm for the MAX-CUT problem

Anna Galluccio
Istituto di Analisi dei Sistemi ed
Informatica - CNR
viale Manzoni 30, 00185 Roma, Italy
galluccio@iasi.rm.cnr.it

[*]**Martin Loebl**
Department of Applied Mathematics
Charles University
Malostrànske nàm. 25, 11800 Praha, Czech Republic
loebl@kam.ms.mff.cuni.cz

Jan Vondrák
Department of Applied Mathematics
Charles University
Malostrànske nàm. 25, 11800 Praha, Czech Republic
jvon@kam.ms.mff.cuni.cz

Abstract

We present a polynomial time algorithm to find the maximum weight of an edge-cut in graphs embeddable on an arbitrary orientable surface, with integral weights bounded in the absolute value by a polynomial of the size of the graph.

The algorithm has been implemented for toroidal grids using modular arithmetics and the generalized nested dissection method. The applications in statistical physics are discussed.

1 Introduction

The *MAX-CUT problem* is a combinatorial optimization problem which is easy to define, but surprisingly hard to solve:

Given a graph, divide its vertices into two parts so that the number of edges between them is as large as possible. More generally, the edges may have arbitrary weights, and we need to maximize the sum of weights over all the edges between the two sets of vertices.

This combinatorial problem has a history on its own, but what makes it so widely studied is the enormous number of applications it finds in different fields. One of these relevant application comes from the study of the *Ising model*: a theoretical physics model of the nearest-neighbor interactions in a crystal structure. This application was the main motivation of this work and it will be considered more closely in this paper.

The complete description of a specific Ising model is embodied in the *partition function* from which all fundamental physical quantities can be derived. It turns out that this function is very close to the *generating function of cuts* which is a standard concept in enumerative

[*]Supported by FONDAP on applied mathematics, GACR 201/99/0242 and GAUK 158/99

combinatorics. The value of the MAX-CUT can be obtained as a by-product from the generating function of cuts; thus, computing the generating function is usually a more difficult task. Rather than the MAX-CUT problem, this work is actually concerned with the generating function of cuts.

The general MAX-CUT problem has defied efficient solution so far, and indeed, it was proved to be *NP-hard* [9], even in the case when all edge weights are equal to 1. In spite of that, many attempts have been made to tackle the problem with approximation and randomized algorithms. Delorme and Poljak [6] and Poljak and Rendl [18] solved a relaxation of the problem using eigenvalues. A similar approach based on semidefinite programming was developed by Goemans and Williamson who presented a randomized algorithm in [11] with a performance guarantee of 0.878.

Polynomial time methods to find the exact solution of the Ising model on toroidal square lattices were suggested in the early 60's by Kasteleyn [16, 15], and Kac and Ward [13]. Kac and Ward tried to calculate the partition function as a determinant of a $4n \times 4n$ matrix over complex numbers and even though their original derivation was not quite exact, it showed that such an approach was indeed possible. A similar method was used by the physicists Kardar and Saul [14] to obtain the partition function of a toroidal spin glass in estimated time $O(n^{3+\epsilon})$, $\epsilon < 1$. They have asked if an efficient method exists for general toroidal graphs. This paper answers the question affirmatively.

Another polynomial time algorithm to solve the Max Cut Problem in toroidal square lattices was proposed by Barahona in [1] and in an unpublished manuscript [2].

There are also branch-and-cut algorithms which can be applied effectively to the case of toroidal square lattices. Barahona et al. [3] and De Simone et al. [4], [5] used integer programming to solve large instances of the Max Cut Problem for toroidal square lattices, with general weights or with weights $+1$ and -1 only. The method of De Simone et al. [5], which has been so far the most successful one to find exact spinglass ground state, works in estimated time n^3 for toroidal square lattices of n vertices, for $n \leq 50$. The method is of course non-polynomial for general n.

Several algorithms for solving the Max Cut Problem for toroidal grids have been proposed and implemented by physicists (see [4], [5] for the references). A successful Dagstuhl seminar on this subject was held in the end of 1997 (see [12]).

However, there has been no deterministic algorithm which would produce an exact result in polynomial time for any toroidal graph, for example. Due to a recent result proved by two of the authors [8], though, it has become possible to solve MAX-CUT in polynomial time for any graph of genus bounded by a constant. The method produces actually the generating function of cuts which contains information about all possible cuts in the graph. In terms of statistical physics, this means the distribution of physical states over all possible energy levels of the system. This can be of tremendous interest for anyone studying a particular Ising model.

There are two main goals of this paper: the first one is to describe the technical details of the polynomial algorithm, and the second one is to develop the algorithm to functional software.

The latter goal was achieved mainly by the third author [20], who solved all the difficulties encountered on the way from the theoretical sketch of the algorithm in [8] to its implementation. Many of these implementation tricks are nontrivial and elegant and the result is a new type of algorithm to solve combinatorial optimization problems via the use of generating functions.

2 MAX-CUT and the Ising model

2.1 Basic definitions

A graph is a pair $G = (V, E)$ where V is a finite set of *vertices* and E is a set of unordered pairs $\{u, v\} \subset V$, called *edges*. In addition, w_e will denote a rational weight associated with the edge $e \in E$. For a subset of edges $\alpha \subset E$, $w(\alpha)$ means the sum of the weights associated with the edges in α.

Let S be a set of subsets of edges of G and let us assign to each element of α of S a function $l(\alpha)$. Then the *generating function* $\mathcal{S}(G, l(\alpha))$ of S (with respect to the function $l(\alpha)$) is

$$\mathcal{S}(G, l(\alpha)) = \sum_{\alpha \in S} l(\alpha)$$

The function $l(\alpha)$ may take values in any abelian group, but more usually consists of monomials in the ring of polynomials. Typically, each element α of S will have *weight* $w(\alpha)$ and we take the function $l(\alpha)$ to be $x^{w(\alpha)}$, where x is an indeterminate. Then knowing the generating function $\mathcal{S}(G, l(\alpha))$ of a set S is equivalent to knowing the elements of S of each weight. Thus the concept of generating function of a set is a generalization of the concept of cardinality.

In this paper, we show how the generating functions can be used successfully also to solve combinatorial optimization problems.

Definition 2.1 *A cut of a graph $G = (V, E)$ is a partition of its vertices into two disjoint subsets $V_1, V_2 \subset V$, and the implied set of edges between the two parts: $C(V_1, V_2) = \{\{u, v\} \in E : u \in V_1, v \in V_2\}$*

The generating function of cuts is a polynomial $\mathcal{C}(G, x)$ which equals the sum of $x^{w(C)}$ over all cuts C of G.

Definition 2.2 *An eulerian subgraph of a graph $G = (V, E)$ is a set of edges $U \subset E$ such that each vertex of V is incident with an even number of edges from U.*

The generating function of eulerian subgraphs is a polynomial $\mathcal{E}(G, x)$ which equals the sum of $x^{w(U)}$ over all eulerian subgraphs U of G.

Definition 2.3 *A perfect matching of a graph $G = (V, E)$ is a set of edges $P \subset E$ such that each vertex of V is incident with exactly one edge from P.*

The generating function of perfect matchings is a polynomial $\mathcal{P}(G, x)$ which equals the sum of $x^{w(P)}$ over all perfect matchings P of G.

These generating functions are considered in more general form in [8], where a variable is associated with each edge instead of a weight. However for the purpose of this paper the present definition is more appropriate.

Definition 2.4 The MAX-CUT Problem

Input: *Graph* $G = (V, E)$ *and Weight function* $w : E \to \mathbf{Z}$

Question: *What is the maximum possible weight of a cut of G?*

$$MAXCUT(G) = \max_{C\,cut} \sum_{e \in C} w_e$$

2.2 The Ising model

Let us consider the *Ising model* of a physical system: the vertices of a graph represent particles and the edges describe interactions between pairs of particles. The most common example is a planar lattice where each particle interacts only with its neighbors. Often, one adds edges connecting the first and last vertex in each row and column, which represent *periodic boundary conditions* in the model. This makes the graph a *toroidal lattice.*

Now, we assign a factor J_{ij} to each edge $\{i, j\}$; this factor describes the nature of the interaction between particles i and j. A physical state of the system is an assignment of $\sigma_i \in \{+1, -1\}$ to each vertex i. This describes the two possible spin orientations the particle can take. The *Hamiltonian* (or *energy function*) of the system is then defined as

$$H(\sigma) = -\sum_{\{i,j\} \in E} J_{ij} \sigma_i \sigma_j$$

One of the key questions we may ask about a specific system is:

"What is the lowest possible energy (the *ground state*) of the system?"

Before we seek an answer to this question, we should realize that the physical states (spin assignments) correspond exactly to the cuts of the underlying graph. Let us define:

$$V_1 = \{i \in V; \sigma_i = +1\}$$
$$V_2 = \{i \in V; \sigma_i = -1\}$$

Then this partition of vertices encodes uniquely the assignment of spins to particles. The edges contained in the cut $C(V_1, V_2)$ are those connecting a pair of particles with different spins, and those outside the cut connect pairs with equal spins. This allows us to rewrite the Hamiltonian in the following way:

$$H(\sigma) = \sum_{\{i,j\} \in C} J_{ij} - \sum_{\{i,j\} \in E \setminus C} J_{ij} = 2w(C) - W,$$

where $w(C) = \sum_{\{i,j\} \in C} J_{ij}$ denotes the weight of a cut, and $W = \sum_{\{i,j\} \in E} J_{ij}$ is the sum of all edge weights in the graph.

Clearly, if we find the value of MAX-CUT, we have found the maximum energy of the physical system. Similarly, MIN-CUT (the cut with minimum possible weight) corresponds to the minimum energy of the system. Note that if we allow negative edge weights, we can transform MAX-CUT into MIN-CUT (and vice versa) simply by reversing the value of each edge weight. So the two problems are equivalent (unlike the MIN-CUT problem with positive edge weights arising in the study of network flows, which is significantly easier).

However, the analogy between cuts and physical states goes even deeper than that.

If we assign weight $w_e = J_{ij}$ to each edge $e = \{i, j\}$, the generating function becomes

$$\mathcal{C}(G, x) = \sum_{cut C} x^{w(C)} = \sum_k c_k x^k$$

where c_k denotes the number of cuts of weight k. But this is also equal to the number of states with energy $2k - W$. Thus, if we knew the complete generating function, we would also know the distribution of physical states over all possible energy levels. In the language of physics, this information is encapsulated in the *partition function*:

$$Z(\beta) = \sum_\sigma e^{-\beta H(\sigma)}$$

Substituting the equalities above and counting over cuts instead of σ assignments yields:

$$Z(\beta) = 2 \sum_{cut C} e^{-\beta(2w(C)-W)} = 2e^{\beta W} \sum_{cut C} e^{-2\beta w(C)} = 2e^{\beta W} \mathcal{C}(G, e^{-2\beta})$$

So we can obtain the value of the partition function simply by substituting $e^{-2\beta}$ into the generating function of cuts. The other way round, multiplying the partition function by $\frac{1}{2}e^{-\beta W}$ and expressing it as a polynomial in $e^{-2\beta}$ yields the generating function of cuts. The difference between the two functions is merely formal; both of them encode the state/energy distribution in a similar way.

Finding the generating function of cuts is generally a harder problem than MAX-CUT, because knowing all the coefficients of the polynomial, we can simply find the highest power $x^{k_{max}}$ with a non-zero coefficient; k_{max} is then the maximum cut value. Similarly, the first non-zero term of the partition function corresponds to the ground state energy, but there is much more information contained in it.

3 Theory of Pfaffian orientations

3.1 From cuts to eulerian subgraphs

In planar graphs, there is a well-known duality between cuts and eulerian subgraphs. Because the edges incident with a vertex correspond to a dual circle (encircling the vertex and connecting all the adjacent faces), it is easy to see that a cut becomes an eulerian subgraph of the dual graph.

However, this does not work in the general case. Yet there is a duality between the generating function of cuts and eulerian subgraphs (of the same graph!), which was discovered by van der Waerden [19]. We will derive the relation for the generating function of cuts.

The basic idea of van der Waerden was to substitute hyperbolic functions for the exponential terms:

$$x^y = \cosh(x,y) + \sinh(x,y) = \cosh(x,y)(1 + \tanh(x,y))$$

where

$$\cosh(x,y) = \frac{x^y + x^{-y}}{2}, \qquad \sinh(x,y) = \frac{x^y - x^{-y}}{2}, \qquad \tanh(x,y) = \frac{\sinh(x,y)}{\cosh(x,y)}$$

Thus we obtain the generating function in the following form: let $w = (w_{ij}; \{i,j\} \in E(G))$ and $W = \sum_{\{i,j\} \in E(G)} w_{ij}$. Then

$$\mathcal{C}(G,x) = \frac{1}{2} x^{\frac{W}{2}} \sum_{\sigma} \prod_{\{i,j\} \in E} x^{-\frac{1}{2} w_{ij} \sigma_i \sigma_j} =$$

$$= \frac{1}{2} x^{\frac{W}{2}} \sum_{\sigma} \prod_{\{i,j\} \in E} \cosh(x, -\frac{1}{2} w_{ij} \sigma_i \sigma_j)(1 + \tanh(x, -\frac{1}{2} w_{ij} \sigma_i \sigma_j) =$$

$$= \frac{1}{2} x^{\frac{W}{2}} \prod_{\{i,j\} \in E} \cosh(x, -\frac{1}{2} w_{ij}) \sum_{\sigma} \sum_{U \subset E} \prod_{\{i,j\} \in U} \sigma_i \sigma_j \tanh(x, -\frac{1}{2} w_{ij}) =$$

$$= \frac{1}{2} x^{\frac{W}{2}} \prod_{\{i,j\} \in E} \cosh(x, -\frac{1}{2} w_{ij}) \sum_{U \subset E} (\prod_{\{i,j\} \in U} \tanh(x, -\frac{1}{2} w_{ij}) \sum_{\sigma} \prod_{i \in V} \sigma_i^{d_U(i)})$$

where $d_U(i)$ means the number of edges in U incident with the vertex i. If we consider the sum over all assigments σ for a given $U \subset E$, we can see that whenever there exists a vertex i incident with an odd number of edges from U, the resulting sum will be zero. This is because the terms arising from assigments where $\sigma_i = +1$ will exactly cancel out the corresponding terms with $\sigma_i = -1$. On the other hand, if all vertices have even degrees in U, the sign of all contributing terms will be positive. So if n is the total number of vertices:

$$\sum_{\sigma \in \{\pm 1\}^n} \prod_{i \in V} \sigma_i^{d_U(i)} = 2^n$$

whenever U is eulerian, and zero otherwise. Finally, we get

$$\mathcal{C}(G,x) = 2^{n-1} x^{\frac{W}{2}} \prod_{\{i,j\} \in E} \cosh(x, -\frac{1}{2} w_{ij}) \sum_{U eulerian} \prod_{\{i,j\} \in U} \tanh(x, -\frac{1}{2} w_{ij}) =$$

$$= 2^{n-1} x^{\frac{W}{2}} \prod_{\{i,j\} \in E} \cosh(x, -\frac{1}{2} w_{ij}) \mathcal{E}(G', \tanh(x, -\frac{1}{2} w_{ij})).$$

So the generating function of cuts can be expressed as the generating function of eulerian subgraphs of the same graph, with $x^{w_{ij}}$ replaced by $\tanh(x, -\frac{1}{2} w_{ij})$.

3.2 From eulerian subgraphs to perfect matchings

Now we need a graph transformation converting eulerian subgraphs into perfect matchings. The transformation given here is based on Fisher's construction described in [7]. It is local in the sense that it only modifies each vertex in a way dependent on its degree. It can also be seen that it preserves the genus of the graph.

Definition 3.1 *Let $G = (V, E)$ be a graph embedded in an orientable surface of genus g, and $v \in V$ a vertex. Let $e_1, e_2, ..., e_d \in E$ denote the edges incident with v, ordered clockwise as they spread out from v in the embedding. Then the even splitting of v is a graph $G' = (V', E')$ where*

- $V' = V \setminus \{v\} \cup \{v_1, ..., v_d, v'_1, ..., v'_d\}$
- $E' = E \setminus \{e_1, e_2, ..., e_d\} \cup \{e'_1, e'_2, ..., e'_d\} \cup E^A$
- $E^A = \{\{v_i, v'_i\}; i = 1, ..., d\} \cup \{\{v_i, v'_{i-1}\}; i = 2, ..., d\} \cup \{\{v'_i, v'_{i+1}\}; i = 1, ..., d-1\}$

*The edges $e'_i \in E'$ (*image edges*) are obtained from $e_i \in E$ by replacing the vertex v by v_i. The edges E^A will be called* auxiliary*.*

Lemma 3.2 *The graph obtained by even splitting can be again embedded in the same surface.*

Proof. The transformation replaces a vertex $v \in V$ by a cluster of $2d$ vertices and $3d - 2$ edges. The cluster itself is a planar graph which can be embedded in a small neighborhood of the original location of the vertex v. The images of the edges incident with v can be embedded in the same way as they were in the original graph. □

Definition 3.3 *Let $G = (V, E)$ be a graph and $G_s = (V_s, E_s)$ the graph obtained by successive even splitting of all vertices in V. If there are weights w_e assigned to edges $e \in E$, we assign the same weights to their images in E_s: $w_{e'} = w_e$. The auxiliary edges $f \in E_s$ get assigned $w_f = 0$.*

Theorem 3.4 *If G is a graph of genus g, G_s has genus g as well. With the assigment of weights described above, the generating function of perfect matchings of G_s is equal to the generating function of eulerian subgraphs of G:*

$$\mathcal{P}(G_s, x) = \mathcal{E}(G, x).$$

Proof. From the definition of even splitting and Lemma 3.2, it follows that the resulting graph G_s can be embedded again in the same surface.

If M is a perfect matching in G_s, it must cover each of its vertices exactly once. Because the cluster replacing every vertex has an even number of vertices, and any of the auxiliary edges which is in M covers a pair of vertices of the cluster, there remain an even number of vertices to be covered by the image edges incident with the cluster. Therefore, every cluster coincides with an even number of image edges which are in M; in other words, these edges form the image of an eulerian subgraph of G.

Vice versa, the image of any eulerian subgraph of G can be extended (uniquely) by adding some of the auxiliary edges in G_s to make a perfect matching in G_s. Thus, there is a one-to-one correspondence between the perfect matchings of G_s and the eulerian subgraphs of G. As all the auxiliary edges have weights equal to 0, the corresponding terms contributing to either of the generating functions are equal. Consequently, the two generating functions are equal. □

3.3 Perfect matchings and Pfaffians

This section deals with the generating function of perfect matchings and an efficient way to compute it. We will start out with planar graphs for which the solution was found by Kasteleyn [16]. We shall see that the generating function can be expressed in an algebraic form very similar to a determinant.

Definition 3.5 *Let $G = (V, E)$ be a graph with $2n$ vertices, $(w_e; e \in E)$ the weights assigned to edges and D an orientation (a fixed ordering of the two vertices of each edge). Let $A(D)$ denote the antisymmetric adjacency matrix where*

- $a_{ij} = x^{w_{i,j}}$, *if (i,j) is a directed edge*
- $a_{ij} = -x^{w_{i,j}}$, *if (j,i) is a directed edge and*
- $a_{ij} = 0$, *if {i,j} is no edge*

The Pfaffian of this matrix is defined as

$$Pf(A(D)) = \sum_P sgn(P) a_{i_1 j_1} a_{i_2 j_2} ... a_{i_n j_n}$$

where the sum is taken over all partitionings of the index set $\{1, 2, ..., 2n\}$ into pairs $i_1 < j_1$, $i_2 < j_2$,...,$i_n < j_n$ and $sgn(P)$ is the sign of the permutation $(i_1, j_1, i_2, j_2, ..., i_n, j_n)$.

Note: The ordering of the pairs is specified here only to remove multiple occurrences of the same partitioning. The resulting contribution to the Pfaffian does not depend on it, as swapping a pair of indices $\{i_k, j_k\}$ reverses both the sign of the permutation and the sign of the factor $a_{i_k j_k}$.

It can be observed that the non-zero terms contributing to the Pfaffian are exactly those corresponding to perfect matchings of G (partitionings of the set of vertices into pairs where there is an edge between each pair). However, each of them comes with a positive or negative sign, depending on the orientation D. What we would like to find is a special orientation which produces positive signs for all perfect matchings. Then the

Pfaffian would be exactly equal to the generating function of perfect matchings.

Definition 3.6 *Let M and P be two perfect matchings. An alternating cycle is a cycle of even length which contains edges from M and P alternately. In a directed graph, if the number of edges oriented in either of the two directions around the cycle is even, the cycle is called clockwise even, otherwise it is called clockwise odd.*

Lemma 3.7 *Let M be a fixed perfect matching of graph G and D its orientation. Let $sgn(M)$ be the sign of the contribution of M to $Pf(A(D))$ and $sgn(D)$ be the sign of another perfect matching D contributing to the Pfaffian. The edges in $M\Delta P$ (symmetric difference) form disjoint alternating cycles. Let $\delta(M,P)$ denote the number of clockwise even cycles in $M\Delta P$. Then*

$$sgn(P) = (-1)^{\delta(M,P)} sgn(M)$$

This observation leads to the following definition and theorem (due to Kasteleyn).

Definition 3.8 *An orientation D is called Pfaffian, if the alternating cycles of any fixed perfect matching M are all clockwise odd.*

Theorem 3.9 *Let G be a graph with a Pfaffian orientation D. Then*

$$Pf(A(D)) = (\pm)\mathcal{P}(G,x)$$

Proof. Because the orientation is Pfaffian, the alternating cycles formed by any two perfect matchings are clockwise odd. By the lemma, all the terms constituting the Pfaffian have the same sign, which is either positive or negative. Which of the two alternatives occurs depends only on the numbering of vertices in the matrix $A(D)$. If we want to make sure that the sign will be positive, we can pick an arbitrary perfect matching, calculate the sign of the corresponding permutation, and if it comes out negative, just exchange any two vertices in the numbering. $\square$

Theorem 3.10 *Every planar graph has a Pfaffian orientation.*

Proof. First, we shall suppose that the graph is 2-connected. We will try to find such an orientation that every inner face is clockwise odd. This can be accomplished by starting out from any inner face and adding adjacent faces with suitable edge orientations. When adding a new face, there is always at least one new edge whose orientation can be chosen so that the face is clockwise odd. This process continues until we cover the whole graph. (It is not necessary that the outer face comes out clockwise odd.)

If the graph is not 2-connected, we can proceed in almost the same way, effectively ignoring the vertices and edges which do not belong to any cycle (their orientation does not matter here). The only difference will be that the faces containing an odd number of such vertices must be clockwise even, whereas those containing an even number of them must be clockwise odd.

This scheme implies that every cycle encircling an even number of vertices is clockwise odd, whereas any cycle encircling an odd number of vertices is clockwise even. As any alternating cycle of M must contain an even number of vertices (there must a partial matching inside the circle, and no edges may cross it), all the alternating cycles are clockwise odd. $\square$

When we find the Pfaffian orientation, all we need is to calculate the Pfaffian of the resulting adjacency matrix. Supposing this can be done efficiently, we are now able to solve the MAX-CUT problem, or produce the entire partition function for any planar graph.

4 Graphs of bounded genus

For general graphs we cannot rely on the Pfaffian orientation, as it may not always exist. However, the situation is solved by the theorem proved in [8] which states that a graph G of genus g has 4^g *relevant orientations* such that a suitable linear combination of the corresponding Pfaffians yields the generating function of perfect matchings of G.

Let us define surfaces and graphs of genus g in a slightly different form than usual:

Definition 4.1 *A surface of genus g consists of a base B_0 and $2g$ bridges B_j^i, $i = 0, ..., g-1$ and $j = 0, 1$.*

1. *The base B_0 is a convex $4g$-gon with vertices $a_0, ...a_{4g-1}$.*
2. *The bridge B_j^i is a quadrangle with vertices k, l, m, n where the edge $[k, l]$ is identified with $[a_{4i+j}, a_{4i+j+1}]$ and $[m, n]$ is identified with $[a_{4i+j+2}, a_{4i+j+3}]$.*

Definition 4.2 *G is a g-graph if it can be embedded in a surface of genus g, so that*

1. *All the vertices are contained in the base B_0.*
2. *Every edge uses at most one bridge B_j^i.*

Let E_0 denote the edges embedded entirely inside the base B_0 and E_j^i the edges using bridge B_j^i. Let $G_0 = (V, E_0)$ and $G_j^i = (V, E_0 \cup E_j^i)$.

Definition 4.3 *G is a proper g-graph if it is a g-graph satisfying the following conditions:*

1. *The outer face of G_0 is a cycle C_0 embedded on the boundary of B_0.*
2. *The edges in E_j^i are embedded entirely inside the bridge B_j^i and their endpoints are located on the boundary of B_0.*
3. *Every vertex is incident with at most one edge outside of E_0.*
4. *G_0 has a perfect matching M_0.*

For the purpose of the theorem, we shall require that the graph G is a proper g-graph. However, any graph of genus g can be transformed into a proper g-graph by splitting the edges traversing several bridges into paths and adding vertices and edges on the boundary of B_0 if necessary. The details may be found in [8].

Definition 4.4 *Let G be a proper g-graph. G_0 is a planar subgraph of G, therefore it has a Pfaffian orientation D_0. In addition, we can choose D_0 so that M_0 has a positive sign in the Pfaffian of the matrix $A(D_0)$.*

Furthermore, G^i_j can be embedded in the plane in the following way: Take the planar embedding of G_0 and add the bridge B^i_j so that it is contained in the outer face of G_0. Then the orientation D_0 can be supplemented with an orientation D^i_j of the edges in E^i_j so that the orientation $D_0 \cup D^i_j$ is still Pfaffian.

Finally, we define the 4^g relevant orientations of the entire graph G: The relevant orientation D_r (where $r \in \{+1, -1\}^{2g}$) is composed of D_0 and $r_{2i+j} D^i_j$ for $i = 0, ..., g-1$ and $j = 0, 1$ (where $-D^i_j$ means the reversed orientation D^i_j).

Theorem 4.5 (Galluccio, Loebl) *Let G be a proper g-graph and $D_r, r \in \{+1, -1\}^{2g}$ its relevant orientations. Let us define*

$$c_r = \prod_{i=0}^{g-1} c(r_{2i}, r_{2i+1})$$

where

$$c(+1,+1) = \frac{1}{2}, \quad c(+1,-1) = \frac{1}{2}, \quad c(-1,+1) = \frac{1}{2}, \quad c(-1,-1) = -\frac{1}{2}.$$

Then the generating function of perfect matchings of G is equal to a linear combination of 4^g Pfaffians:

$$\mathcal{P}(G, x) = \sum_{r \in \{+1,-1\}^{2g}} c_r Pf(A(D_r))$$

where $A(D_r)$ is the antisymmetric adjacency matrix corresponding to the orientation D_r.

The proof of this theorem is difficult and it can be found in [8].

As a consequence, the generating function of perfect matchings can be computed efficiently for any graph of bounded genus. Needless to say, this approach is practical only for very small values of g. For example, toroidal graphs (i.e., 1-graphs) can be treated in the same way as planar graphs, but there will be four relevant orientations and four Pfaffians instead of one.

5 Modular Arithmetic

The approach described in section 3 leads to an efficient algorithm indeed, but it still leaves a number of questions open:

- How to treat the matrix elements? (They are symbolic functions.)
- How to calculate the Pfaffian in a polynomial number of steps?

5.1 Data structures

The first question involves a decision on the data structures, as well as the type of programming language to use. Consider the data flow of the program:

> **Input:** Graph G, edge weights w_{ij}
>
> **Intermediate data:** Matrices whose elements are symbolic functions of a single variable.
>
> **Output:** Coefficients of the generating function.

At first glance, the obvious approach seems to be: treat the matrix elements as polynomials, perform all arithmetic operations symbolically, and obtain the result in a symbolic form. For this purpose, one would most probably use a symbolic computation package like *Mathematica*. However, using a symbolic computation package (instead of a general purpose programming language) would inevitably bring a substantial loss of speed and increase the amount of required memory. Also, it is not apparent how this symbolic computation could be parallelized.

Our solution is this: we know the generating function of cuts is a polynomial in the variable x with integer coefficients. Instead of computing the entire polynomial at once, we can evaluate it at sufficiently many distinct points, and then we can obtain the polynomial coefficients by interpolation. This turns the problem of treating symbolic functions into ordinary numeric calculation, albeit still cumbersome in a sense. As the expected results are exponentially large (the total number of cuts in a graph with n vertices is 2^n), a way how to handle these huge numbers had to be resolved. Instead of using floating point (which is imprecise) or long integer arithmetic (which is possible, but slow and consuming lots of memory), we decided to split the task once more. The idea is to use only elementary operations which can be performed in constant time - the basic operations in a finite field. Of course, the results are elements of the finite field as well, but this loss of information can be amended, as will be shown. The major advantage is that the algorithm employs only simple integer operations and it can be implemented easily in any general purpose programming language, like C++.

The following lemma justifies this approach in a formal way:

Lemma 5.1 *Let $P(x)$ be a polynomial of degree n with integer coefficients, $GF[m]$ a finite field of size $m > n$ and $x_1, x_2, ..., x_n$ distinct elements of $GF[m]$. Then there exists a unique polynomial $Q(x)$ of degree n over $GF[m]$ such that*

$$Q(x_i) = P(x_i) \bmod m, i = 0...n$$

Moreover, the coefficients of $Q(x)$ are equal to the coefficients of $P(x) \bmod m$.

Proof. Let $P(x_i) = y_i$. Define $Q(x)$ by Langrangian interpolation:

$$Q(x) = \sum_{i=0}^{n} \frac{\prod_{j \neq i}(x - x_j)}{\prod_{j \neq i}(x_i - x_j)} (y_i \bmod m)$$

where all operations are performed in $GF[m]$. The expression is well defined because

$$(i \neq j \Longrightarrow x_i \neq x_j) \Longrightarrow \prod_{j \neq i} (x_i - x_j) \neq 0$$

and any non-zero element is invertible in a field.

$Q(x)$ satisfies the conditions $Q(x_i) = y_i \bmod m$. Now let us suppose there is another polynomial $R(x)$ over $GF[m]$, such that $R(x_i) = y_i \bmod m, i = 0...n$. Then we define the difference polynomial $Z(x) = R(x) - Q(x)$, and it follows that

$$Z(x_i) = R(x_i) - Q(x_i) = 0 \pmod m, i = 0...n$$

However, a non-zero polynomial of degree n in any field has at most n roots, therefore $Z(x)$ is a zero polynomial and $R(x) = Q(x)$.

The polynomial $P'(x)$ whose coefficients are the coefficients of $P(x) \bmod m$, satisfies the same conditions, so by the same argument $P'(x) = Q(x)$. □

Note: Our polynomial can actually contain negative powers which could cause trouble in the interpolation formula. Therefore it is necessary to make the exponents positive before interpolation by multiplying the function by x^k for a suitable k. The degree of the polynomial is then confined by the difference of maximum and minimum cut weights. As we do not know the values of MAX-CUT and MIN-CUT in advance, we can only estimate them. For now, it will suffice to know that if the edge weights are bounded by a constant, the degree is bounded by $O(m)$, where m is the number of edges. In graphs of bounded genus, this is the same as $O(n)$, where n is the number of vertices.

To summarize the lemma, if we can supply the values of our polynomial modulo some prime p, we can also extract its coefficients modulo p. For now, we will leave aside the question of how to evaluate the polynomial, and we will suppose we have acquired the partial results (modulo p). What good can they be? Instead of a number a, we only get the remainder $a \bmod p$. There is certainly less information in it than we need, even if we only want to detect whether a given coefficient is zero. (As you may recall, finding the MAX-CUT amounts to detecting the first non-zero coefficient in the generating function.) However, we can still make a guess. If $a \bmod p \neq 0$, then surely $a \neq 0$. But if $a \bmod p = 0$, we cannot be sure that $a = 0$. To estimate our chance of success, we give the following lemma.

Lemma 5.2 *Suppose $a < N$, and we pick a prime $p < M$ at random. Then the probability that a is divisible by p is at most $\mathcal{P} = O(\frac{\log N \log M}{M})$.*

Therefore, knowing an upper bound on our polynomial coefficients (which is $N = 2^n$, where n is the number of vertices), we can make our chance of error small enough by allowing ourselves to select from a wide enough range of primes. For example, a choice of $M = n^2$ gives $\mathcal{P} = O(\frac{\log n}{n})$.

5.2 The Chinese Remainder Theorem

Of course, sometimes we may require the complete exact results, which are integer numbers ranging from 0 to 2^n. Even then, modular arithmetic can be useful. Consider the *Chinese Remainder Thorem*:

Theorem 5.3 *Let us select k different prime numbers $p_1, p_2, ..., p_k$. Then for any $a_1, a_2, ..., a_k$, $0 \leq a_i < p_i$, there is exactly one x, $0 \leq x < \prod_{i=1}^{k} p_i$, such that:*

$$x \bmod p_i = a_i, i = 1...k$$

This is a well-known fact, and the number x can also be efficiently constructed from its "modulo coordinates". Let us define

$$r_i = \prod_{j \neq i} p_j, \quad q_i = r_i^{-1} \pmod{p_i}, \quad x = \sum_{j=1}^{n} a_j q_j r_j.$$

Then x is the desired solution, because every $r_j, j \neq i$ is divisible by p_i, and

$$x \bmod p_i = (a_i q_i r_i) \bmod p_i = (a_i r_i r_i^{-1}) \bmod p_i = a_i$$

□

In other words, we can obtain *exact* results, if we choose a sufficient number of primes $p_1, p_2, ..., p_k$, so that

$$\prod_{i=1}^{k} p_i > 2^n$$

Then if all results are confined to the interval $[0; 2^n]$, the partial results modulo $p_1, p_2, ..., p_k$ determine the complete result uniquely. Furthermore, the complete result can be calculated from the modulo results efficiently, using long integer addition and multiplication. Of course, this will cost us several repetitions of the computation - in different finite fields. Suppose we can use primes from the interval $[\frac{M}{2}; M]$. Then (assuming there are enough primes to choose from), we need

$$k = O(\frac{n}{\log M})$$

primes $p_1, p_2, ..., p_k$ to ensure that their product is greater than 2^n.

However, these computations are absolutely independent - they can be run separately on different computers.

6 Computing Pfaffians

Being armed with the results of section 5, we can assume that we reside in a finite field $GF[m]$, and our sole purpose is to calculate a single value of the generating function of cuts at a specific point. As we know, it can be transformed into the generating function of perfect matchings of a modified graph:

$$\mathcal{C}(G,x) = 2^{n-1}x^{\frac{W}{2}} \prod_{\{i,j\}\in E} \cosh(-\frac{1}{2}w_{ij})\mathcal{P}(G_s, \tanh(x, -\frac{1}{2}w_{ij})$$

Moreover, the generating function of perfect matchings can be expressed as a linear combination of Pfaffians:

$$\mathcal{P}(G,x) = \sum_{r\in\{+1,-1\}^{2g}} c_r Pf(A(D_r))$$

where the Pfaffian of a matrix is defined as a sum over all partitionings of the index set into pairs:

$$Pf(A(D)) = \sum_P sgn(P) a_{i_1 j_1} a_{i_2 j_2} \dots a_{i_n j_n}$$

The sum consists of an exponential number of terms, but (just like a determinant) it can be calculated very efficiently.

The first step is to define a basic operation which can be used to transform the matrix into a special form, while the value of the Pfaffian is preserved. For instance, determinants are invariant under elementary row/column operations and these can be used in Gaussian elimination to calculate a determinant or to solve a system of linear equations. Pfaffians must be treated somewhat differently, though.

Definition 6.1 *Let A be an antisymmetric matrix of size $2n \times 2n$ over a field $\mathcal{F}$. A cross of the matrix is the union of a row and a column of the same index. I.e., the k-th cross is the following set of elements:*

$$A_k = \{a_{ik}; 1 \leq i \leq 2n\} \cup \{a_{kj}; 1 \leq j \leq 2n\}$$

Definition 6.2 Cross operations:

Multiplying a cross A_k by a scalar $\alpha \in \mathcal{F}$ means simply multliplying each element of A_k by α.

Swapping crosses A_k and A_l of an antisymmetric matrix A means exchanging both the respective rows and columns. Another way of regarding the swap operation is that it exchanges the values of k and l in both of the index positions. The resulting matrix B is antisymmetric again;

Adding cross A_k to cross A_l means adding first the k-th row to the l-th one, and then adding the respective columns. The matrix remains antisymmetric.

These operations may be used to transform our matrix into a form where the Pfaffian can be determined trivially.

Such algorithm needs at most $O(n^2)$ cross operations, each of which consists of $O(n)$ finite field operations which can be carried out in constant time. (Addition and multiplication are trivial, and division can be reduced to multiplication with a precalculated table of inverse elements.) The total running time is therefore $O(n^3)$. However, this crude estimate does not take into account the special structure of our matrix. We achieve a better running time by employing some of the techniques developed for fast Gaussian elimination.

We consider a method which was developed by Alan George [10] and later refined by Lipton, Rose and Tarjan [17]. Their approach, though intended for Gaussian elimation, can be geared to Pfaffian elimination. The details may be found in [20].

Theorem 6.3 *For the graphs G of bounded genus, Pfaffian elimination on matrix $A(D)$ where D is an orientation of G and G has n vertices can be performed in $O(n \log n)$ space and $O(n^{\frac{3}{2}})$ time.*

7 Algorithm overview

Let us summarize the algorithm:

1. `Find prime numbers` $p_1, p_2, ..., p_k < 2^{16}$ `so that` $\prod_{i=1}^{k} p_i > 2^{|V(G)|}$`.`

   `For each of them, repeat the remaining steps of the algorithm, performing all operations in` $GF[p_i]$`.`

2. `Choose` $m+1$ `distinct elements` $x_0, ..., x_m \in GF[p_i]$`, where` m `is the maximum possible degree of the generating function. (Avoid` $x_j = 0$ `because the elements must be invertible) For each of them, repeat the following step.`

3. `Construct the matrices encoding the relevant orientations of the modified graph` G_s`, with hyperbolic functions` $\tanh(x, -1/2w_{ij})$ `substituted for edge weights. For every occurrence of` $x^{-1/2}$ `substitute` $x^{-1/2} = x_i$ `and calculate the four Pfaffians. From these values, calculate the value of` $\mathcal{C}(G, x_i^{-2})$`.`

4. `Obtain the coefficients of` $C(G,x)$ `(modulo` p_i`) by interpolation in` $GF[p_i]$`.`

5. `Apply the Chinese remainder theorem and compose the results from all the finite fields to obtain the complete partition function.`

Notes on the complexity:

- The number of finite fields is $O(n)$, assuming there is a sufficient number of primes in the range where our hardware is able to perform modular arithmetics in constant time. Due to this constraint, the algorithm is actually unable to work properly for an arbitrarily large input, and any complexity analysis in the sense of asymptotic behavior is meaningless. On the other hand, the limit on the size of lattices we are able to process is well beyond our practical possibilities. The product of all primes below 2^{16} is approximately $2^{2^{16}}$, which means we can process lattices with at most 2^{16} vertices. Such a computation would last $10,000$ years on a typical PC. If this should prove insufficient, we could still move to 32-bit arithmetics and the limit of 2^{32} vertices which would be enough to keep all our computing resources busy for more than the lifetime of our universe.

- If the edge weights are bounded by a constant, the number of evaluations in each of the fields is $O(n)$. A single evaluation of the polynomial takes $O(n^{\frac{3}{2}})$, so the computation in each finite field takes $O(n^{\frac{5}{2}})$ time. The total time complexity of Step `3.` (under the restrictions mentioned above) is therefore $O(n^{\frac{7}{2}})$.

- The interpolation in Step `4.` and the final composition in Step `5.` take $O(n^3)$ time in total, so they are faster than the Pfaffian evaluation in Step `3.`.

- Steps `2.`, `3.` and `4.` can be parallelized easily. The computation in each of the finite fields can be performed separately, and the communication is trivial - in Step `5.`, we only have to send the results modulo each of the primes; i.e., data of size $O(n^2)$. With this degree of parallelization ($O(n)$ processors are needed), Steps `2.`, `3.` and `4.` take $O(n^{\frac{5}{2}})$ time, whereas Step `5.` takes $O(n^3)$. To remove this obstacle, we could parallelize it as well; every processor would produce one of the $O(n)$ coefficients in $O(n^2)$ time. Then the total (parallel) time complexity would be $O(n^{\frac{5}{2}})$.

- As mentioned above, the computation in each of the finite fields takes $O(n^{\frac{5}{2}})$ time, and one such computation suffices to know the value of Max Cut with very high probability. This time complexity compares favourably with the estimated complexity $O(n^3)$ (for toroidal grids of sizes up to (50×50) only) of the most successful method using integer programming technics of [5], mentioned in the introduction.

- The complexity of our implementation is comparable with the estimated complexity of the implementation of Kardar and Saul [14]. Apart of our method, this is the only method for the exact calculation of the generating function of cuts (for toroidal square lattices) we know about. In experiments our implementation behaves better: for instance the calculations for 10×10 $\pm J$ spin glass lattice take 30 sec. instead of the 110 sec. reported in [14]. However, the main advantages are in the structure of the algorithm: our implementation is selfcontained, short, easy to parallelize and flexible. Moreover, computations in a few fields give the value of some coefficients with a very high probability.

8 Implementation for toroidal lattices

One of the main goals of this work was to develop a functional implementation of the algorithm described here for the calculations of the partition function of specific Ising models. After consulting literature on computation in statistical physics so far, it seemed natural to focus on random Ising model of toroidal lattices. In this case, the implementation may be improved by several tricks described in [20]. Moreover, it seems that the following quantities are most interesting for the physics applications: the ground state energy, the number of ground states, the second lowest energy of a state and the number of states of that energy. To obtain these values with a very high probability, a smaller number of finite field computations needs to be performed. At present we started to calculate some experiments. The details of this research will appear elsewhere.

For illustration, we give here a table of the running time required for various parts of the computation (sequentially on a Pentium/200 MHz machine):

lattice size	*1 Pfaffian*	*1 Finite Field*	*Complete partition function*
10×10	0.03 s	5 s	40 s
14×14	0.1 s	30 s	7 min
20×20	0.25 s	150 s	1 hour
30×30	1 s	20 min	20 hours
40×40	3 s	2 hours	8 days
50×50	6 s	6 hours	40 days
100×100	90 s	14 days	24 years

Notes:

- *1 Pfaffian* - this is the basic building block, the Pfaffian elimination of a single matrix; in principle, each of these could be executed in parallel, if $O(n^2)$ processors were available. However, then the communication and composition of results would become the bottleneck of the entire computation.
- *1 Finite Field* - this means the computation of the partition function in a single finite field; it is actually the parallel complexity of the algorithm using $O(n)$ processors, or it can be understood as the complexity of a sequential randomized algorithm to find the MAX-CUT value (see Section 5).
- *Complete partition function* - this is the total sequential running time of the computation in all the finite fields plus the Chinese composition step which produces the exact coefficients of the partition function.

Acknowledgement. We would like to thank Jirka Matoušek for helpful discussions and in particular for drawing our attention to the modular arithmetics method. We would also like to thank Giovanni Rinaldi for enlightening discussions and continuous support.

References

[1] F. Barahona. On the computational complexity of Ising spin glass models. *J. Phys. A*, 15, 1982.

[2] F. Barahona. Balancing signed toroidal graphs in polynomial time. *Preprint University of Chile*, 1983. Unpublished manuscript.

[3] F. Barahona, M. Grötschel, M. Jünger, and G. Reinelt. An application of combinatorial optimization to statistical physics and circuit layout design. *Operations Research*, 36:493–513, 1988.

[4] C. De Simone, M. Diehl, M. Jünger, P. Mützel, G. Reinelt, and G. Rinaldi. Exact ground states of Ising spin glasses: New experimental results with a branch and cut algorithm. *Jour. of Stat. Phisycs*, 80:487–496, 1995.

[5] C. De Simone, M. Diehl, M. Jünger, P. Mützel, G. Reinelt, and G. Rinaldi. Exact ground states of $2d \pm j$ Ising spin glasses. *Jour. of Stat. Phisycs*, 84:1363–1371, 1996.

[6] C. Delorme and S. Poljak. Laplacian eigenvalues and the maximum cut problem. *Math. Prog.*, 62:557–574, 1993.

[7] M.E. Fisher. On the dimer solution of planar Ising models. *Journal of Mathematical Physics 7*, 10, 1966.

[8] A. Galluccio and M. Loebl. A theory of pfaffian orientations I: Perfect matchings and permanents. *Electronic Jour. Combinatorics*, 6(1), 1999.

[9] M. R. Garey and D. S. Johnson. *Computers and Intractability.* W. H. Freeman and Company, New York, 1979.

[10] A. George. Nested dissection of a regular finite element mesh. *SIAM J. Numer. Anal.*, 10:345–363, 1973.

[11] M. X. Goemans and D. P. Williamson. Improved approximation algorithms for maximum cut and satisfiability problems using semidefinite programming. *J. of ACM*, 42:1115–1145, 1995.

[12] M. Junger, G. Reinelt, H. Rieger, and G. Rinaldi. *Algorithmic Techniques in Physics.* Dagstuhl-Seminar-Report 197, Dagstuhl, 1997.

[13] M. Kac and J.C. Ward. A combinatorial solution of the two-dimensional Ising model. *Physical Review*, 88, 1952.

[14] M. Kardar and L. Saul. The 2D $\pm$ Ising spin glass: exact partition functions in polynomial time. *Nuclear Physics B*, 432:641–667, 1994.

[15] P. W. Kasteleyn. Dimer statistics and phase transitions. *Jour. Math. Physics*, 4:287–293, 1963.

[16] P.W. Kasteleyn. Graph theory and crystal physics. In *Graph theory and theoretical physics*, New York, 1967. Academic Press.

[17] R.J. Lipton, D.J. Rose, and R.E. Tarjan. Generalized nested dissection. *SIAM J. Numer. Anal.*, 16:346–358, 1979.

[18] S. Poljak and F. Rendl. Solving the max-cut problem using eigenvalues. *Discr. Appl. Math.*, 62:249–278, 1995.

[19] B.L. van der Waerden. Die lange reichweite der regelmassigen atomanordnung in mischkristallen. *Z. Physik*, 118:473, 1941.

[20] J. Vondrak. ??? Master's thesis, Charles University, Prague, 1999.