Constrained Codes as Networks of Relations

Moshe Schwartz, *Member, IEEE*, and Jehoshua Bruck, *Fellow, IEEE*

***Abstract*—We address the well-known problem of determining the capacity of constrained coding systems. While the one-dimensional case is well understood to the extent that there are techniques for rigorously deriving the exact capacity, in contrast, computing the exact capacity of a two-dimensional constrained coding system is still an elusive research challenge. The only known exception in the two-dimensional case is an exact (however, not rigorous) solution to the $(1,\infty)$-run-length limited (RLL) system on the hexagonal lattice. Furthermore, only exponential-time algorithms are known for the related problem of counting the exact number of constrained two-dimensional information arrays.**

We present the first known rigorous technique that yields an exact capacity of a two-dimensional constrained coding system. In addition, we devise an efficient (polynomial time) algorithm for counting the exact number of constrained arrays of any given size. Our approach is a composition of a number of ideas and techniques: describing the capacity problem as a solution to a counting problem in networks of relations, graph-theoretic tools originally developed in the field of statistical mechanics, techniques for efficiently simulating quantum circuits, as well as ideas from the theory related to the spectral distribution of Toeplitz matrices.

Using our technique, we derive a closed-form solution to the capacity related to the Path–Cover constraint in a two-dimensional triangular array (the resulting calculated capacity is $0.72399217\ldots$). Path-Cover is a generalization of the well known one-dimensional $(0,1)$-RLL constraint for which the capacity is known to be $0.69424\ldots$.

***Index Terms*—Capacity of constrained systems, capacity of two-dimensional constrained systems, Fisher–Kasteleyn–Temperley (FKT) method, holographic reductions, networks of relations, spectral distribution of Toeplitz matrices.**

I. Introduction

WHILE most storage devices record information on a two-dimensional surface, they emulate a one-dimensional environment by spacing tracks of recorded data. The distance between adjacent tracks in common devices is an order of magnitude larger than the distance between adjacent symbols along the track. The next big leap in storage density may be achieved by reducing the distance between tracks. This in turn, requires a two-dimensional constrained-coding scheme to be employed.

A two-dimensional constrained system $\mathcal{S}$ is simply the union $\bigcup_{n,m\in\mathbb{N}}\mathcal{S}_{n,m}$ where $\mathcal{S}_{n,m}$ denotes the set of $n\times m$ arrays that satisfy some property. The common example of such a system is the (d,k)-run-length limited (RLL) constraint in which each row and each column of the array has runs of zeroes whose length is at least d and at most k. Other two-dimensional constraints forbid certain patterns in the arrays, such as the no-isolated-bit constraint in which every bit agrees with at least one of its four neighbors in the two-dimensional array.

An important measure associated with a constrained system is its *capacity*. Introduced by Shannon [30], the capacity of a constrained system $\mathcal{S}$ is defined as

$$\mathsf{cap}(S) \stackrel{\text{def}}{=} \lim_{n,m\to\infty}\frac{\log_2|S_{n,m}|}{nm}.$$

While the capacity of one-dimensional constraints is well understood, amazingly, there is still very little known about the capacity of two-dimensional systems.

In the case of two-dimensional (d,k)-RLL systems, Ito *et al.* [16] characterized the values of d and k for which the capacity is zero. General bounds on the capacity of (d,k)-RLL were given by Kato and Zeger [19], constructive lower bounds for (d,∞)-RLL by Halevy *et al.* [15], and nonconstructive asymptotically tight bounds for $(0,k)$-RLL by Schwartz and Vardy [29]. For the specific case of $(1,\infty)$-RLL, Calkin and Wilf [6] gave a numerical estimation method for the capacity using the transfer matrix method. Only for the $(1,\infty)$-RLL constraint on the hexagonal lattice, Baxter [3] gave an exact but not rigorous[1] analytical solution for the capacity using the corner transfer matrix method.

Other two-dimensional constraints do not fare any better. Several estimates for the capacity of the two-dimensional no-isolated-bit constraint exist. Halevy *et al.* [15] considered bit-stuffing encoders to constructively estimate this capacity. Non-constructively, Forchhammer and Laursen [12] estimated this capacity using random fields.

The method we present in this work is general enough to encompass a wide variety of constraints (both local and global), however, its expressive power is yet undetermined. We use only *mathematically rigorous* tools to obtain exact capacity solutions and polynomial-time algorithms, while pointing out places where nonrigorous practices were common. The method is based on a series of reductions:

1) A constrained system is first reduced to a network of relations in a way which enables us to connect the number of satisfying assignments to the network with the number of constrained arrays. Though this is usually done in a one-to-one manner, it is not mandated.
2) This network of relations is transformed to a weighted graph using holographic reductions in such a way that the

Manuscript received August 5, 2007; revised January 6, 2008. This work was supported in part by the Caltech Lee Center for Advanced Networking.

M. Schwartz was with the Department of Electrical Engineering, California Institute of Technology, Pasadena, CA, USA. He is now with the Department of Electrical and Computer Engineering, Ben-Gurion University, Beer-Sheva 84105, Israel (e-mail: schwartz@ee.bgu.ac.il).

J. Bruck is with the Department of Electrical Engineering, California Institute of Technology, Pasadena, CA 91125 USA (e-mail: bruck@paradise.caltech.edu).

Communicated by T. J. Richardson, Associate Editor for Coding Theory.

Digital Object Identifier 10.1109/TIT.2008.920245

[1] As Baxter notes in [4, p. 409]: "It is not mathematically rigorous, in that certain analyticity properties . . . are assumed, and the results . . . (which depend on assuming that various large-lattice limits can be interchanged) are used. However, I believe that these assumptions . . . are in fact correct."

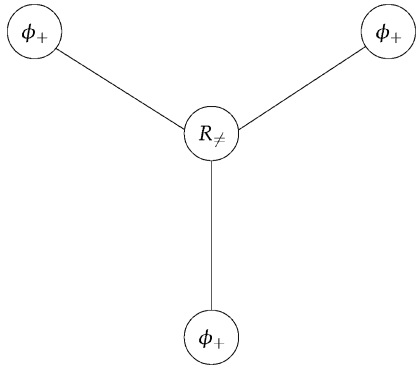


Fig. 1. A simple network of relations.

number of satisfying assignments to the network equals the weighted perfect matching of the graph. This is a many-to-many reduction in which the individual perfect matchings do not correspond in any one-to-one way to satisfying assignments, and the "interference" and cancellations between different matchings is the reason for the name *holographic* reductions.

3) Finally, the weighted perfect matching of the graph is expressed as a Pfaffian (or a linear combination of Pfaffians) of a certain skew-symmetric matrix, completing the series of reductions. Using the theory of spectral distribution of Toeplitz matrices, a limit involving the Pfaffian gives a closed-form solution for the capacity. The Pfaffian itself also provides a polynomial-time algorithm for counting the exact number of constrained arrays of any given size.

The paper is written with the goal of explaining our new method in a self-contained manner. We start by providing the background needed for the three key steps in Section II, starting with networks of relations, going through holographic reductions, and ending with the Fisher–Kasteleyn–Temperley (FKT) method which computes the weighted perfect matching of a graph. In Section III, we apply this background to an example constrained coding system and demonstrate how to derive its exact capacity using the theory of spectral distribution of Toeplitz matrices. We continue in Section IV by presenting a polynomial-time algorithm for counting the number of constrained arrays while taking the opportunity to introduce two generalizations to the method involving constraints on a torus and generalized relations. We conclude in Section V with a summary of the results and a list of open questions.

II. Background and Description of the New Technique

The background we are about to provide is described in a relatively self-contained manner, and is therefore quite lengthy. It is divided into three subsections, for which the following tiny example is an appetizer and also serves as a table of contents.

Suppose we are given a graph whose edges may be assigned either a $\mathbf{0}$ or a $\mathbf{1}$. Only not any assignment is possible: every vertex implements a local constraint on the values assigned to edges incident to it. Such graphs are called networks of relations and are described in Section II-A. In Fig. 1, we see a simple network whose three outer vertices are satisfied with any assignment to their single incident edge, while the middle vertex forbids all three incident edges to be assigned the same value.

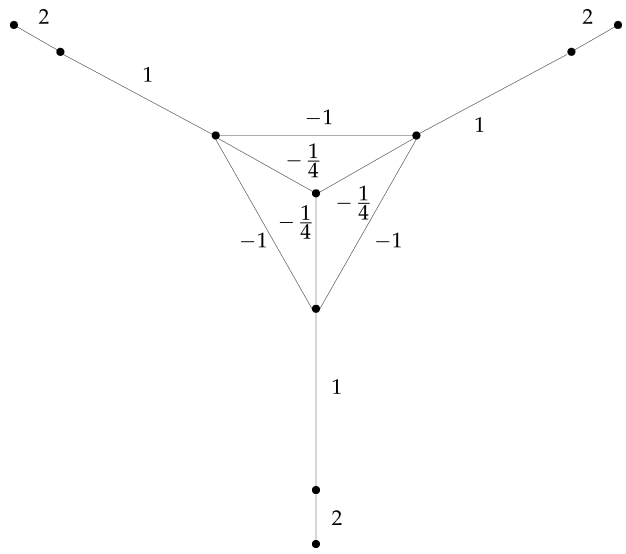


Fig. 2. The weighted graph corresponding to the network of relations from Fig. 1.

Obviously, the number of *satisfying assignments* in this example is 6.

Counting the number of satisfying assignments seems to be difficult when stated this way. But we can reduce this problem to a problem of finding the weighted perfect matching of some other graph. This is done using *holographic reductions* which are described in Section II-B. For this tiny example, the resulting weighted graph is shown in Fig. 2. The weighted perfect matching of the graph, which is the sum over all perfect matchings of the product of the weights of edges in the perfect matching, is indeed 6, as is the number of satisfying assignments to the original network of relations.

Finally, the weighted perfect matching of the graph is calculated using the FKT method, which is described in Section II-C. Loosely speaking, the adjacency matrix of the graph is modified by changing the signs of the entries to make it skew-symmetric

$$A = \begin{pmatrix} 0 & -1 & 1 & -\frac{1}{4} & 0 & -1 & 0 & 0 & 0 & 0 \\ 1 & 0 & -1 & -\frac{1}{4} & 0 & 0 & 0 & -1 & 0 & 0 \\ -1 & 1 & 0 & -\frac{1}{4} & 0 & 0 & 0 & 0 & 0 & -1 \\ \frac{1}{4} & \frac{1}{4} & \frac{1}{4} & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 2 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & -2 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 2 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & -2 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 2 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & -2 & 0 \end{pmatrix}.$$

The weighted perfect matching is then the Pfaffian of the modified matrix, which is (up to a sign) the square root of the determinant of the matrix. As if by magic, we again get

$$|\mathrm{Pf}(A)| = \sqrt{\det(A)} = 6.$$

A. Networks of Relations

We start our journey by introducing networks of relations. These networks were used in the context of relational databases and constraint-satisfaction problems, see, for example, [14], [8],

[9]. For more on the subject, the reader is referred to [7] and references therein.

Given some ground set Ω, a relation on n variables is a subset $R \subseteq \Omega^n$. Throughout the paper, we assume $\Omega = \{0, 1\}$. As will be apparent later on, by abuse of notation we will also consider a relation to be a function $R : \Omega^n \to \{0, 1\}$ which is simply the characteristic function associated with the subset R. We also define the *signature* of the relation to be the column vector

$$\mathrm{sig}(R) \stackrel{\mathrm{def}}{=} \begin{pmatrix} R(0,\ldots,0,0) \\ R(0,\ldots,0,1) \\ R(0,\ldots,1,0) \\ R(0,\ldots,1,1) \\ \vdots \\ R(1,\ldots,1,0) \\ R(1,\ldots,1,1) \end{pmatrix}.$$

A network of relations is a graph $G = (V, E)$ where we associate with each vertex $v \in V$ a relation R_v on $\deg(v)$ variables being the ordered set of incident edges on v. We can now assign every edge $e \in E$ a value from Ω and check whether all the relations are satisfied. For every such assignment of values to edges $A : E \to \Omega$, every vertex $v \in V$, and edges incident on v denoted $e_1^v, \ldots, e_{\deg(v)}^v$, we define A_v to be

$$A_v \stackrel{\mathrm{def}}{=} \left(A(e_1^v), \ldots, A(e_{\deg(v)}^v)\right).$$

We say that assignment A is a *satisfying assignment* if for every $v \in V$, the relation R_v is satisfied, i.e., $A_v \in R_v$. If we denote $E = (e_1, \ldots, e_{|E|})$, we will usually specify A by the vector of assignments to e_1 through $e_{|E|}$. Throughout the paper, we will be interested in counting the number of satisfying assignments to networks of relations.

Example 1: Let us take as a running example the network of relations shown in Fig. 3. We use the ground set $\Omega = \{0, 1\}$, define $R_{\neq}$ to be the not-all-equal relation on three variables, and ϕ_+ to be the accept-all relation on one variable

x_1	x_2	x_3	$R_{\neq}$
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	0

x_1	ϕ_+
0	1
1	1

We can easily see that there are exactly 18 distinct satisfying assignments to this network which we list as follows:

$$\begin{aligned} A \in \{&(0,0,1,0,0),(0,0,1,0,1),(0,0,1,1,0), \\ &(0,1,0,0,1),(0,1,0,1,0),(0,1,0,1,1), \\ &(0,1,1,0,0),(0,1,1,0,1),(0,1,1,1,0), \\ &(1,0,0,0,1),(1,0,0,1,0),(1,0,0,1,1), \\ &(1,0,1,0,0),(1,0,1,0,1),(1,0,1,1,0), \\ &(1,1,0,0,1),(1,1,0,1,0),(1,1,0,1,1)\}. \end{aligned}$$

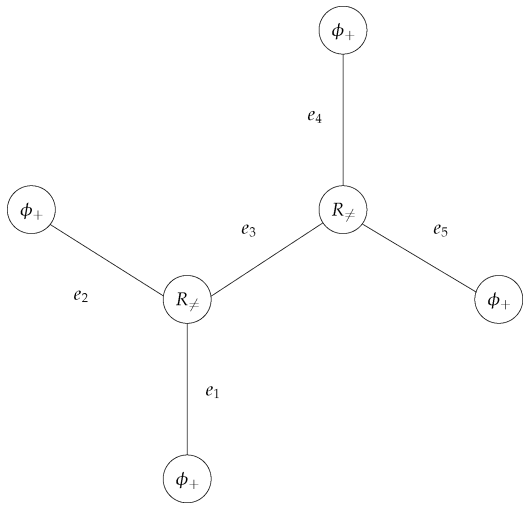


Fig. 3. The network of relations of Example 1.

If we wanted to be completely accurate, we should have included a numbering of the incident edges to each vertex of Fig. 3. However, since all the relations in this example are symmetric,[2] this is unnecessary. □

B. Holographic Reductions

Holographic reductions were introduced by Valiant in [36] and [37] to show certain counting problems may be solved in polynomial time. In a slightly different version they were introduced also by Valiant [35] to simulate quantum circuits efficiently in polynomial time. Though the notion of networks of relations does not appear as such in his work, Valiant shows a many-to-many reduction from such networks to weighted graphs. This reduction preserves the total number of solutions, i.e., the number of satisfying assignments to the original network of relations equals the weighted perfect matching of the resulting graph. The reduction itself is realized by replacing each of the vertices of the original network with a small gadget. In what follows, we will describe this reduction in practical terms. For a rigorous treatment of the method the reader is encouraged to read [36].

Let $G = (V, E)$ be a graph. A *perfect matching* is a subset of edges $M \subseteq E$ such that every vertex $v \in V$ is incident to exactly one of the edges in M. The set of all perfect matchings will be denoted $\mathrm{PM}(G)$. We can now assign complex weights[3] to the edges $w : E \to \mathbb{C}$, and define the weighted perfect matching of G to be

$$\mathrm{PerfMatch}(G) \stackrel{\mathrm{def}}{=} \sum_{M \in \mathrm{PM}(G)} \prod_{e \in M} w(e).$$

Our aim is to replace the vertices in the network of relations graph with gadgets (small subgraphs) which somehow capture the original relations. The gadgets are called *matchgates* and the resulting graph (the network of relations graph with vertices

[2]A *symmetric* relation is one that is closed under all permutations, i.e., if $(a_1, \ldots, a_n) \in R$ then so is $(a_{\pi(1)}, \ldots, a_{\pi(n)}) \in R$ where π is any permutation on $\{1, \ldots, n\}$.

[3]Throughout the paper, we use complex weights, though the method applies equally well to other fields.

replaced by matchgates) is called a *matchgrid*. At this point, just like in [36], we require the network of relations graph to be planar as well as all the matchgates we use, resulting in a planar matchgrid graph. This is perhaps the most restrictive requirement we face during the process. We are, however, able to use nonplanar graphs (either for the network of relations or for the matchgates), though at a cost of increased computational complexity. Such a nonplanar matchgrid will be introduced in Section IV. For now, however, we assume all graphs are planar.

A matchgate is defined as a graph $G = (V, E, X, Y)$ with vertex set V, edge set E, a set of input nodes $X \subseteq V$, and a set of output nodes $Y \subseteq V$, where X and Y are disjoint and $|X|+|Y|$ equals the number of variables in the original relation. For convenience, we can think of X and Y as drawn on the outer face[4] of the graph. The edges in the network of relations are copied to the matchgrid with weight 1, and are placed so as to connect input vertices of gadgets with output vertices of gadgets such that every input/output vertex is incident to exactly one of those edges.

The interaction of the matchgate with the outside world, i.e., the matchgrid, which should encapsulate the original relation, is given by a $2^{|X|} \times 2^{|Y|}$ matrix, called the *signature* of the matchgate, in the following manner: for each possible $Z \subseteq X \cup Y$ there is an entry containing $\mathrm{PerfMatch}(G-Z)$, where by $G-Z$ we denote the graph G with the vertices of Z removed as well as their incident edges. This is meant to simulate all possible ways of a global perfect matching (on the entire matchgrid) interacting with the matchgate, where the subset Z depends on whether edges between matchgates are chosen to be part of the global perfect matching. Such chosen edges already cover some of the interfacing input/output vertices of the matchgate, and so these are removed from G and a perfect matching of the remaining uncovered vertices in G is calculated. For convenience, we index the rows and columns of the signature of the matchgate by binary vectors in the obvious way, where for example, for the rows, $(0, \ldots, 0, 0)$ means no input vertex was removed (i.e., no input vertex is in Z), $(0, \ldots, 0, 1)$ means the last input vertex was removed, up to $(1, \ldots, 1, 1)$ which means all input vertices were removed.

Matchgates with only input vertices are called *recognizers*, those with only output vertices are called *generators*, and those with both are called *transducers*. The examples we show in this paper will only use recognizers and generators. This obviously restricts our networks of relations to be bipartite since we may only connect inputs with outputs. This, however, is not a severe restriction since most useful networks are bipartite by their nature, and when they are not, we could add auxiliary vertices on the edges (equality on two variables) to make them bipartite. By definition, the signature of a recognizer is a column vector, while the signature of a generator is a row vector.

Example 2: The recognizer matchgate seen in Fig. 4 contains four vertices and six edges. Three of the vertices (depicted as white circles) are input vertices and lie on the outer face of the graph. The shaded area is just used to highlight the perimeter of the matchgate and has no mathematical meaning to it. The signature of the matchgate is the following column vector:

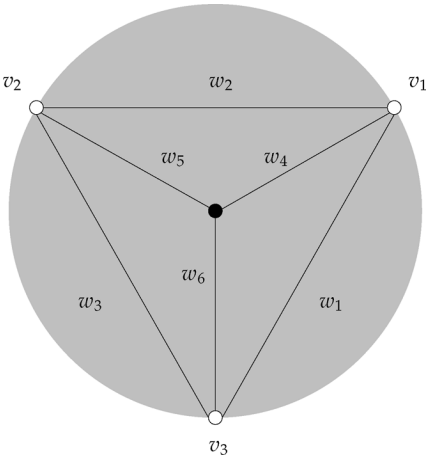


Fig. 4. The matchgate of example 2.

$$\begin{array}{cc} \text{index} & \text{signature} \\ (0,0,0) & \\ (0,0,1) & \\ (0,1,0) & \\ (0,1,1) & \\ (1,0,0) & \\ (1,0,1) & \\ (1,1,0) & \\ (1,1,1) & \end{array} \begin{pmatrix} w_1w_5 + w_2w_6 + w_3w_4 \\ 0 \\ 0 \\ w_4 \\ 0 \\ w_5 \\ w_6 \\ 0 \end{pmatrix}.$$

At this point, we note that had we said the matchgate was a generator instead of a recognizer, we would have gotten the exact same signature only as a row vector.

Let us examine a few of the entries in the signature. For the entry indexed by $(1,1,0)$ we have $Z = \{v_1, v_2\}$ which simulates a global perfect matching which already covers v_1 and v_2, so $G-Z$ contains only two surviving vertices which are v_3 and the inner vertex. Thus, $\mathrm{PerfMatch}(G-Z) = w_6$ because there is exactly one perfect matching covering v_3 and the inner vertex, and it contains just the edge with weight w_6.

For the entry indexed by $(0,0,0)$ we have $Z = \emptyset$ and it follows that there are three different perfect matchings covering the four vertices of $G-Z$. For each perfect matching we take the product of the weights of its edges, and sum over all perfect matchings to get $\mathrm{PerfMatch}(G-Z) = w_1w_5 + w_2w_6 + w_3w_4$.

Finally, for the entry indexed by $(0,0,1)$ we have $Z = \{v_3\}$ and so $G-Z$ contains three surviving vertices. Obviously, there is no perfect matching in a graph with an odd number of vertices and thus this entry is 0, as are the entries indexed by $(0,1,0)$, $(1,0,0)$, and $(1,1,1)$. □

Now, it would be extremely useful if we could get matchgates with signatures which equal the signatures of the relations we aim to replace. However, by Example 2, it is clearly seen that entries with an index of odd (even) weight are forced to be 0 if the matchgate has an even (odd) number of vertices. Unfortunately for us, a quick glance at the signature of a relation such as the not-all-equal relation on three variables seen in Example 1

[4]Loosely speaking, a planar graph may be drawn on the two-dimensional Euclidean plane by mapping vertices to distinct points, and edges to curves connecting them such that the curves do not intersect. Regions of the plane completely bounded by edges and vertices are called *inner faces*. The single unbounded face is called the *outer face* and is the region "outside" the graph.

reveals that it contains nonzero entries in both odd-weight and even-weight indices. To extend the expressive power of matchgates we now introduce a change of basis.

Without knowing it, all our examples thus far used the standard basis. A *basis* is an ordered set of vectors. In what follows, we will restrict ourselves to bases made up of two vectors of length 2 which are also linearly independent. It should be noted though, that in the general case those restrictions are unnecessary, and a basis should not be confused with the linear-algebra notion of a basis. The *standard basis* is defined as $\boldsymbol{b} = [(1,0),(0,1)]$. We will always denote the first vector in the basis as n which will play the role of a "logical" 0, and the second vector as p which will be a "logical" 1.

Let $\boldsymbol{\beta} = [\mathsf{n},\mathsf{p}] = [(\mathsf{n}_0,\mathsf{n}_1),(\mathsf{p}_0,\mathsf{p}_1)]$ be some basis. We define the basis translation matrix as

$$T_{\boldsymbol{\beta}} \stackrel{\text{def}}{=} \begin{pmatrix} \mathsf{n}_0 & \mathsf{n}_1 \\ \mathsf{p}_0 & \mathsf{p}_1 \end{pmatrix}.$$

Let Γ be some matchgate with n input/output vertices. Using this matrix, we can define the signature of Γ under the basis $\boldsymbol{\beta}$, which we denote as $\mathrm{sig}_{\boldsymbol{\beta}}(\Gamma)$, using two different equations depending on whether the matchgate is a generator or a recognizer

$$\mathrm{sig}_{\boldsymbol{\beta}}(\Gamma)\cdot T_{\boldsymbol{\beta}}^{\otimes n} = \mathrm{sig}_{\boldsymbol{b}}(\Gamma), \quad \text{for } \Gamma \text{ a generator} \tag{1}$$

$$T_{\boldsymbol{\beta}}^{\otimes n}\cdot \mathrm{sig}_{\boldsymbol{b}}(\Gamma) = \mathrm{sig}_{\boldsymbol{\beta}}(\Gamma), \quad \text{for } \Gamma \text{ a recognizer} \tag{2}$$

where $X^{\otimes n} \stackrel{\text{def}}{=} X\otimes\cdots\otimes X$ is n times the Kronecker product.

We can also query the value of individual entries in $\mathrm{sig}_{\boldsymbol{\beta}}(\Gamma)$ using the valG and valR operators for generators and recognizers, respectively. Given some $x \in \{\mathsf{n},\mathsf{p}\}^{\otimes n}$, we associate with it an index vector by substituting 0 for n and 1 for p. For example, with $\mathsf{n}\otimes\mathsf{p}\otimes\mathsf{n}$ we associate the index vector $(0,1,0)$. We now define $\mathrm{valG}_{\boldsymbol{\beta}}(\Gamma,x)$ to be the entry in $\mathrm{sig}_{\boldsymbol{\beta}}(\Gamma)$ with the index associated with x. By (1), this is simply the coefficient of x in the linear combination making up $\mathrm{sig}_{\boldsymbol{b}}(\Gamma)$. Similarly, $\mathrm{valR}_{\boldsymbol{\beta}}(\Gamma,x)$ is defined as the entry in $\mathrm{sig}_{\boldsymbol{\beta}}(\Gamma)$ with the index associated with x. By (2), this is simply the dot product $x\cdot\mathrm{sig}_{\boldsymbol{b}}(\Gamma)$.

Example 3: Returning to our running example, we shall build a generator matchgate for $R_{\neq}$ and also a recognizer matchgate for $R_{\neq}$. We have already noted that using the standard basis will not work in this case since the signature of $R_{\neq}$ has nonzero entries in both odd-weight and even-weight indices.

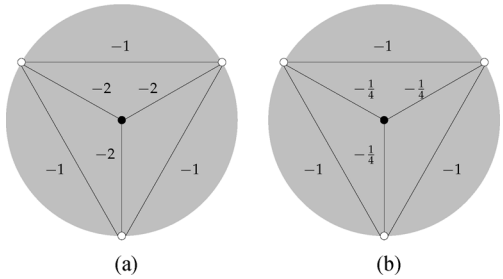
We will choose a basis $\boldsymbol{\beta}$ and set $\mathrm{sig}_{\boldsymbol{\beta}}(\Gamma)$ to be equal to the signature of the relation we want to replace. We will then calculate $\mathrm{sig}_{\boldsymbol{b}}(\Gamma)$ using (1) or (2) (depending on whether we want a generator or a recognizer) and hope that all the nonzero entries fall in either the even-weight indices, or the odd-weight indices.

We choose a basis which works well for self-dual relations (as is $R_{\neq}$) which is

$$\boldsymbol{\beta} = [\mathsf{n},\mathsf{p}] = [(1,1),(1,-1)]. \tag{3}$$

We notice that $T_{\boldsymbol{\beta}}^{\otimes n}$ is a Hadamard matrix. We start by building a generator for $R_{\neq}$. Substituting the values in (1) we get

$$(0,1,1,1,1,1,1,0)\cdot T_{\boldsymbol{\beta}}^{\otimes 3} = (6,0,0,-2,0,-2,-2,0).$$



(a) (b)

Fig. 5. Using the basis $\boldsymbol{\beta} = [(1,1),(1,-1)]$ we get weighted graphs for (a) a generator for $R_{\neq}$, and (b) a recognizer for $R_{\neq}$.

Indeed, as seen on the right, all the nonzero entries are in even-weight indices. We choose the matchgate used in Example 2 and so we need to equate the matchgate's signature with the signature of the relation under the basis $\boldsymbol{\beta}$

$$(w_1w_5 + w_2w_6 + w_3w_4, 0, 0, w_4, 0, w_5, w_6, 0) = (6,0,0,-2,0,-2,-2,0).$$

Choosing values for $w_1,\ldots,w_6$ that solve this equation is easy

$$w_1 = w_2 = w_3 = -1 \quad \text{and} \quad w_4 = w_5 = w_6 = -2.$$

There is more than one solution to this underconstrained set of equations, but any solution will do.

We also want a recognizer for $R_{\neq}$ and using (2)

$$T_{\boldsymbol{\beta}}^{\otimes 3}\cdot\begin{pmatrix} 3/4 \\ 0 \\ 0 \\ -1/4 \\ 0 \\ -1/4 \\ -1/4 \\ 0 \end{pmatrix} = \begin{pmatrix} 0 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 0 \end{pmatrix}$$

we again see all the nonzero values move to even-weight indices. Like before, we choose the matchgate used in Example 2 and therefore, we need to solve

$$\begin{pmatrix} w_1w_5 + w_2w_6 + w_3w_4 \\ 0 \\ 0 \\ w_4 \\ 0 \\ w_5 \\ w_6 \\ 0 \end{pmatrix} = \begin{pmatrix} 3/4 \\ 0 \\ 0 \\ -1/4 \\ 0 \\ -1/4 \\ -1/4 \\ 0 \end{pmatrix}.$$

We can now choose weights for the recognizer matchgate

$$w_1 = w_2 = w_3 = -1 \quad \text{and} \quad w_4 = w_5 = w_6 = -\frac{1}{4}.$$

It is interesting to note that using this Hadamard basis we get signatures that contain the Walsh transform of the signatures of the original relations (up to a missing normalizing factor).

The weighted graphs for the generator matchgate for $R_{\neq}$ and the recognizer matchgate for $R_{\neq}$ are shown in Fig. 5. □

Finally, we zoom out to examine the entire matchgrid. Let $\mathcal{M}$ be some matchgrid made up of r recognizers $A_1, \ldots, A_r$, and g generators $B_1, \ldots, B_g$, and let it contain a total of f connecting edges between the matchgates. We can think of each edge as carrying a value from $\boldsymbol{\beta} = [\mathsf{n}, \mathsf{p}]$, and so each possible assignment to edges is of the form $x \in \boldsymbol{\beta}^{\otimes f}$. By abuse of notation, let $\mathrm{valR}_{\boldsymbol{\beta}}(A_i, x)$ and $\mathrm{valG}_{\boldsymbol{\beta}}(B_i, x)$ stand for the normal valR and valG operators when we restrict x to the edges incident to A_i or B_i appropriately. We define a global property of the matchgrid called the *Holant*, as

$$\mathrm{Holant}(\mathcal{M}) \stackrel{\mathrm{def}}{=} \sum_{x \in \boldsymbol{\beta}^{\otimes f}} \left(\prod_{1 \leqslant j \leqslant g} \mathrm{valG}_{\boldsymbol{\beta}}(B_j, x) \right) \left(\prod_{1 \leqslant i \leqslant r} \mathrm{valR}_{\boldsymbol{\beta}}(A_i, x) \right).$$

We now have all the necessary definitions in place to state the main result of Valiant [36].

Theorem 4: For any matchgrid $\mathcal{M}$ over any basis $\boldsymbol{\beta}$, if $\mathcal{M}$ has weighted graph G then

$$\mathrm{Holant}(\mathcal{M}) = \mathrm{PerfMatch}(G).$$

While the connections between matchgrids and perfect matchings have been evident throughout this section, the connection to satisfying assignments for networks of relations is a little more subtle. Given a matchgrid built using this method, we can view edges between matchgates as carrying values from the basis $\boldsymbol{\beta} = [\mathsf{n}, \mathsf{p}]$, which we can think of as "logical" 0 and "logical" 1. The operators valG and valR in this basis query the signatures of the matchgates over $\boldsymbol{\beta}$, which are simply the signatures of the original relations. Thus, the Holant goes over all possible assignments of "logical" 0's and 1's to the edges between matchgates (which are the edges of the original network of relations), and queries the signatures of the relations for that assignment getting a value of either 0 or 1 for being unsatisfied or satisfied, respectively. Since we take the product of valG and valR, only if all relations are satisfied we get a contribution of 1 to the sum, thus counting exactly the satisfying assignments.

On the other hand, Theorem 4 is invariant under a change of bases, and it is the different handling of generators and recognizers which makes this possible. Thus, if we choose to view the same matchgrid using the standard basis, then by definition, valG and valR depend on the perfect matchings of the graph. This establishes a connection between $\mathrm{PerfMatch}(G)$ and the number of satisfying assignments to the original network of relations. For the rigorous proof of Theorem 4 the reader is referred to [36].

Example 5: We complete the matchgrid for the network of relations of Example 1. From Example 3, under the basis $\boldsymbol{\beta} = [(1,1), (1,-1)]$ we already have a generator matchgate for $R_{\neq}$ and a recognizer matchgate for $R_{\neq}$. The remaining relation is ϕ_+, but we note that we need both a generator and a recognizer for it. The resulting matchgrid is shown in Fig. 6. The skeptical reader is encouraged to verify that the weighted perfect matching of this graph is indeed 18, as is the total number of satisfying assignments to the original network of relations. □

C. The FKT Method

Unless weighted perfect matchings are easier to handle, the reduction described in the previous section is useless. Fortunately for us, the FKT method gives a simple expression for the weighted perfect matching of certain graphs which is also computable efficiently. The method was developed independently and concurrently by Fisher and Temperley [31], [10], and by Kasteleyn [17]. The motivation for their work was to find an alternative solution to the Ising problem, simpler than the original solution given by Onsager [26]. The solution more commonly used today, and the one we describe in this work, is due to the general method developed by Kasteleyn in [17] and in more detail in [18]. It is also interesting to note that very simple gadgets were employed in some occasions [11], [2] without reaching the general treatment given by Valiant which we described in the previous section.

Let G be a graph with weights on the edges, and let $A = (a_{i,j})$ be its $n \times n$ adjacency matrix where $a_{i,j}$ is the weight of the edge between vertices i and j. Since we are interested in graphs with perfect matchings we assume n is even. A perfect matching can obviously be described by the unordered partition π of the numbers $\{1, 2, \ldots, n\}$ into pairs, which we denote by $\mid p_1 p_2 \mid p_3 p_4 \mid \ldots \mid p_{n-1} p_n \mid$. Using this notation it follows that

$$\mathrm{PerfMatch}(G) = \sum_{\pi} a_{p_1,p_2} a_{p_3,p_4} \cdots a_{p_{n-1},p_n}$$

where π goes over all such unordered partitions. Since we only consider unordered partitions, we select for each partition a canonical representation in which $p_1 < p_2$, $p_3 < p_4$, up until $p_{n-1} < p_n$ (i.e., each pair is in ascending order), as well as $p_1 < p_3 < \cdots < p_{n-1}$ (i.e., the first elements of the pairs are in ascending order). Using this convention, we note that we only use the entries strictly above the main diagonal of the matrix A.

This expression seems very similar to another expression known as the Pfaffian which is defined as

$$\mathrm{Pf}(A) \stackrel{\mathrm{def}}{=} \sum_{\pi} \mathrm{sgn}(\pi) a_{p_1,p_2} a_{p_3,p_4} \cdots a_{p_{n-1},p_n}$$

where again π goes over all the canonical partitions and $\mathrm{sgn}(\pi)$ is the sign of π when considered as the permutation sending $i \to p_i$. This in itself is again reminiscent of the more widely used determinant. Indeed, since we only use the entries above the main diagonal, if we complete A so as to make it skew-symmetric, that is $a_{i,j} = -a_{j,i}$, then we get the well-known identity

$$[\mathrm{Pf}(A)]^2 = \det(A).$$

Thus, the Pfaffian of a skew-symmetric matrix is easily computed up to its sign by taking the square root of the determinant.

Returning to our problem of calculating the weighted perfect matching, we are faced with the problem caused by the added $\mathrm{sgn}(\pi)$ in the expression for the Pfaffian. This causes some of the perfect matchings to be counted with the wrong sign. It was

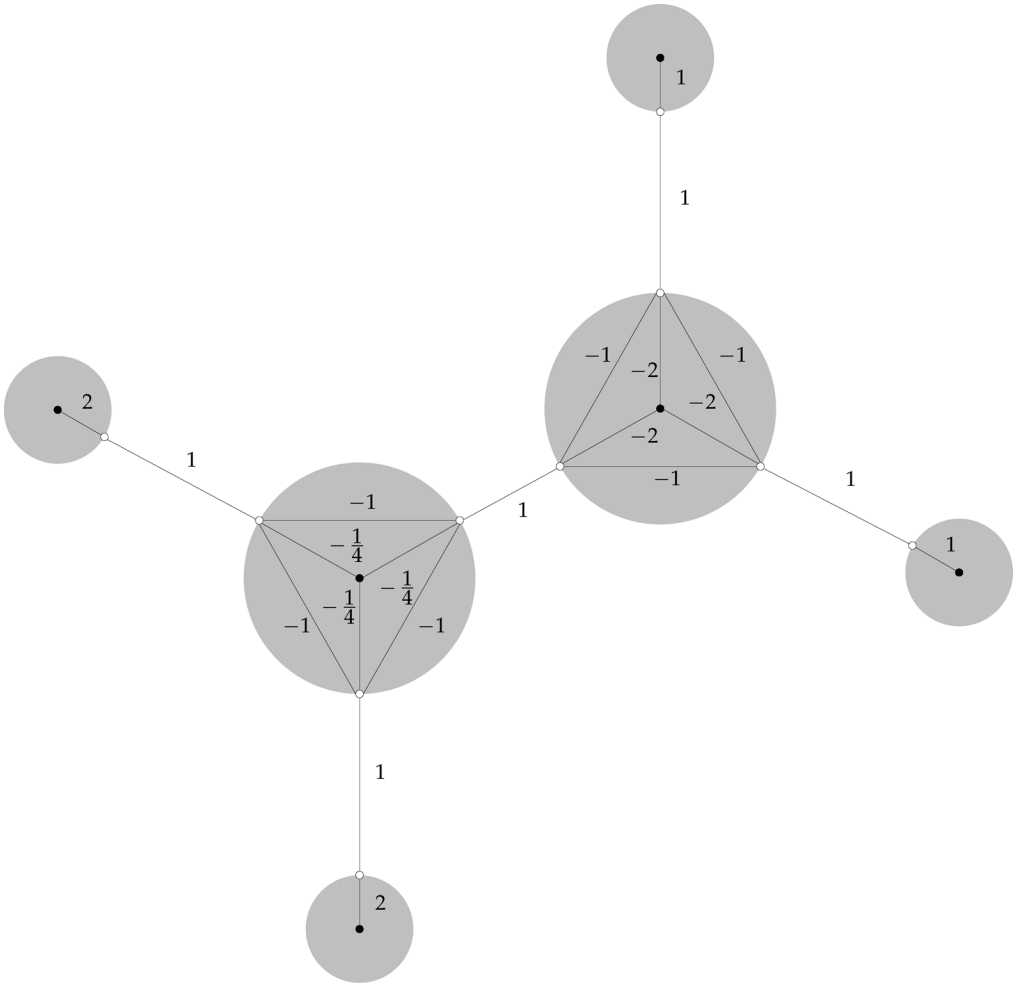


Fig. 6. The matchgrid for the network of relations from example 1.

the ingenious solution of Kasteleyn [17] to flip the signs of some of the entries of the original matrix A to compensate for $\mathrm{sgn}(\pi)$ and make the Pfaffian count all the original perfect matchings with the same sign.[5]

An *orientation* of an undirected graph G is simply an assignment of a direction to each of the edges of the graph. The solution given by Kasteleyn [17] requires a special orientation which we will now describe. Let M and M' be two perfect matchings in the graph G, and let $\oplus$ denote the symmetric difference operation between sets. Then $M \oplus M'$ is a set of cycles of even length in G. If we traverse any of those cycles in some direction, then some of the edges will be oriented in agreement with our traversal direction, and some will not. A *Pfaffian orientation* of a graph G is an orientation such that for each M and M', any cycle in $M \oplus M'$ has an odd number of edges oriented in agreement with the traversal direction. Note that since the cycles are always of even length, the traversal direction does not change the parity of the number of edges agreeing with it.

[5]The reader may notice at this point that $\mathrm{PerfMatch}(G)$ is simply the Hafnian of the matrix A, i.e., the Pfaffian without $\mathrm{sgn}(\pi)$. In fact, the Hafnian is to the Pfaffian as the permanent is to the determinant. However, both the Hafnian and the permanent are notoriously hard to handle and so it is worth the trouble to work with the Pfaffian and correct the sign problems.

Given a weighted graph G, and a Pfaffian orientation of its edges, we build a modified skew-symmetric adjacency matrix $A = (a_{i,j})$ as follows:

$$a_{i,j} = \begin{cases} 0, & \text{no edge between } i \text{ and } j \\ w(e_{i,j}), & \text{if } i \to j \\ -w(e_{i,j}), & \text{if } j \to i \end{cases}$$

where $i \to j$ denotes the edge between vertices i and j is oriented from i to j. Note that A is not the adjacency matrix of the graph G in the usual sense. Using this construction, Kasteleyn [17] showed that

$$\mathrm{PerfMatch}(G) = \pm \mathrm{Pf}(A)$$

where, using the Pfaffian orientation, either all perfect matchings are counted with a positive sign or all with a negative sign, depending on the chosen Pfaffian orientation only. Since in most cases we know the sign of the outcome, this unknown degree of freedom may be easily fixed.

It now remains a matter of finding out which graphs allow a Pfaffian orientation. Such graphs are called *Pfaffian orientable*. In his later work, Kasteleyn [18] showed that all planar graphs are Pfaffian orientable, which is the reason we required matchgrids to be planar in the previous section. For planar graphs, it was shown in [18] that if we orient the edges such that every

clockwise walk on a face of the graph has an odd number of edges agreeing, then that orientation is a Pfaffian orientation. As a result, a simple polynomial-time algorithm which finds such an orientation was also shown by Kasteleyn.

For further reading on generalized dimer problems with boundary conditions, the reader is referred to the excellent survey in [20] (and references therein). Planar graphs are not the only Pfaffian-orientable graphs. More results on Pfaffian-orientable graphs are given by the survey in [33], and the work in [25]. Advances in Pfaffians and perfect matchings may be found in [21]. Pfaffian orientations are also used to efficiently calculate some permanents, see [38], [27].

Example 6: For the last part of our running example we orient the edges of the graph to create a Pfaffian orientation. The resulting oriented graph is shown in Fig. 7. If we write down the modified adjacency matrix for the graph (after fixing some arbitrary ordering of the vertices) we get the matrix shown at the bottom of the page. Again, the skeptical reader will find out that

$$|\mathrm{Pf}(A)| = \sqrt{\det(A)} = 18$$

which is both the weighted perfect matching of the graph and the total number of satisfying assignments to the network of relations. □

III. Exact Calculation of the Capacity of the Path-Cover Constrained System

In this section, we address a specific two-dimensional constrained coding system and demonstrate how to apply our new technique (described in Section II) to derive an expression for its exact capacity. The derivation consists of the following steps.

A. Definition of the constrained system.
B. Reduction to a network of relations.
C. Holographic reduction to a matchgrid.
D. Pfaffian orientation of the graph.
E. Derivation of a closed-form expression for the exact capacity.

A. The Path-Cover Constrained System

Most constrained systems are easily defined by a finite set of forbidden patterns. For example, the *no-isolated-bit constraint* forbids a bit to be surrounded on all four sides by its complement. The famous $(1,\infty)$-*RLL constraint*, a variant of which will be discussed in the next section, forbids two adjacent $\mathbf{1}$'s. For our example, we choose a constraint we call the Path-Cover constraint (PC constraint), and which we motivate by first examining its one-dimensional version. If we are given a graph $G = (V, E)$, a path-cover for the graph is a set of simple paths (open or closed) of positive length, which are vertex disjoint, and which cover all the vertices. An alternative way of stating this constraint is that given a graph, we assign either a $\mathbf{0}$ or a $\mathbf{1}$ to each of the edges, such that when removing the edges with a $\mathbf{0}$, all the vertices remain with degree of either one or two.

In the one-dimensional case the graph G is simply the one-dimensional lattice with vertices $V = \{v_0, v_1, \ldots, v_n\}$ and edges $E = \{(v_i, v_{i+1}) \mid 0 \leqslant i \leqslant n-1\}$. It is easily seen that a valid PC assignment of values to edges is any assignment which does not contain two adjacent $\mathbf{0}$'s. Thus, the one-dimensional PC constraint is the famous one-dimensional $(0,1)$-RLL constraint (and with bit-flipping, the $(1,\infty)$-RLL constraint). The capacity in this case is known to be $\log_2[(1+\sqrt{5})/2] = 0.69424\ldots$.

Turning to two dimensions, we choose the two-dimensional triangular grid as the graph G: we tile the plane with regular triangles, place a vertex at the center of each triangle, and draw an edge between vertices whose triangles share a face. Again, we assign either a $\mathbf{0}$ or a $\mathbf{1}$ to the edges of the graph such that after removing the edges assigned a $\mathbf{0}$, all the remaining vertices are of degree either 1 or 2. We note that this time, the PC constraint is different from the usual two-dimensional RLL constraint. Also, unlike the two-dimensional RLL and the no-isolated-bit constraints, where values are assigned to vertices, in the PC constraints values are assigned to edges. See Fig. 8 for an illustration of the forbidden patterns in these three constraints.

$$A = \begin{pmatrix}
0 & -1 & 1 & -\frac{1}{4} & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 \\
1 & 0 & -1 & -\frac{1}{4} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 \\
-1 & 1 & 0 & -\frac{1}{4} & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
\frac{1}{4} & \frac{1}{4} & \frac{1}{4} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & -1 & 1 & 2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 0 & -1 & 2 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & -1 & 1 & 0 & 2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
0 & 0 & 0 & 0 & -2 & -2 & -2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 2 & 0 & 0 & 0 & 0 & 0 & 0 \\
1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 2 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -2 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 \\
0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0
\end{pmatrix}.$$

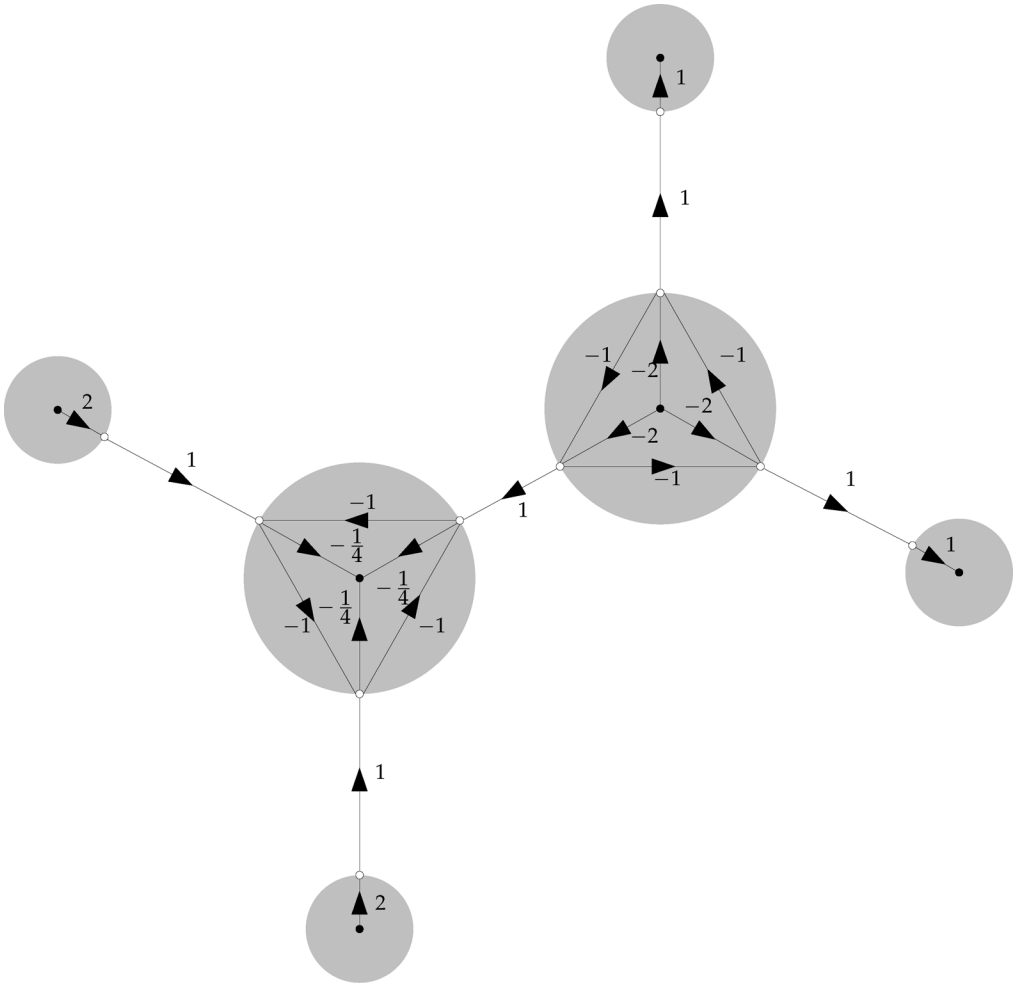


Fig. 7. The matchgrid for the network of relations from Example 1 with a Pfaffian orientation of the edges.

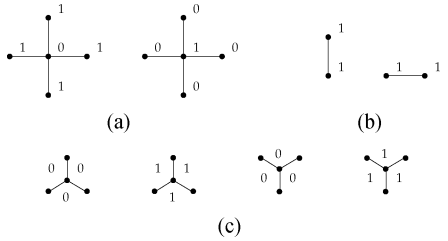


Fig. 8. The forbidden patterns of (a) the no-isolated-bit constraint, (b) the $(1, \infty)$-RLL constraint on the square lattice, and (c) the PC constraint on the triangular lattice.

B. The Network of Relations

There is a multitude of possible reductions of a constrained two-dimensional array on the plane to a network of relations. We show a simple reduction for which the constrained arrays are in an "almost" one-to-one correspondence with the satisfying assignments to the network.

We think of the triangular grid as drawn on a plane. We replace each vertex of the grid with the relation $R_{\neq}$ on three variables. This relation makes sure that the three adjacent cells do not contain the same bit, i.e., it eliminates the forbidden patterns of PC. It is easy to be convinced that, if we ignore the perimeter of the array, every constrained array induces exactly one satisfying assignment and *vice versa*. The resulting network of relations is shown in Fig. 9.

We do however have to take care of the perimeter of the array as well. To do so, we connect dangling edges to extra vertices of the accept-all relation ϕ_+. Each such vertex has the potential of multiplying the number of satisfying assignments by a factor of 2. But since we have only $O(n)$ such vertices, this does not change the capacity as calculated by counting the total number of satisfying assignments. The extra accept-all vertices are also shown in Fig. 9.

C. The Matchgrid

It is easily seen that the network of relations we built in the previous section is a bipartite graph by noting that the upright triangles share faces only with the inverted triangles, and *vice versa*. Conveniently for us, for the bulk of the network, we have just one type of relation, but we do have to specify some as recognizers and some as generators. We arbitrarily choose to build a generator $R_{\neq}$ in inverted triangles, and a recognizer $R_{\neq}$ in upright triangles. Those were already realized in Example 3 using the basis given in (3).

For the perimeter of the network, we need to implement ϕ_+ both as a generator and as a recognizer. Again, this was already realized in Example 3.

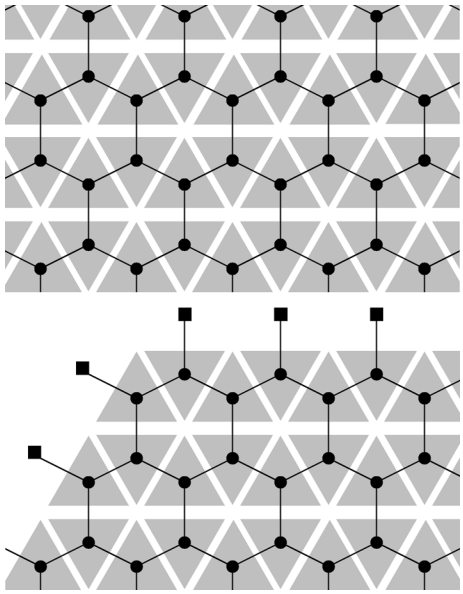


Fig. 9. The top image shows part of a network of relations for the PC constraint. Each filled circle represents the $R_{\neq}$ relation. The gray triangles show the original cells of the triangular grid. The bottom image shows the top left corner of the array with the filled squares representing the ϕ_+ relation.

D. The Pfaffian Orientation

Finding a Pfaffian orientation for the graph is an easy task. The orientation is not necessarily unique and some may in fact be quite complex to describe. However, the extremely regular nature of the graph suggests the existence of a simple orientation.

If we closely examine the network of relations in Fig. 9, we see that, apart from the perimeter of the array, it is made up of a single *basic block* and its translations. The simplest basic block is just a recognizer $R_{\neq}$ vertex and a generator $R_{\neq}$ vertex. This basic block may be oriented as shown in Fig. 10. It is also easy to verify Kasteleyn's orientation rule for planar graphs: every clockwise walk on an inner face has an odd number of edges agreeing. This may be verified both for the inner faces of the block, and the inner faces created by the joining of a few blocks.

Finally, we also have to orient the edges which correspond to the ϕ_+ matchgates which lie on the perimeter of the array. Those matchgates do not contain any inner face themselves, and do not form an inner face with the rest of the graph. Thus, they may be oriented arbitrarily.

E. The Exact Capacity

For mathematical convenience, from now on when we say "an $n \times n$ array" we mean an $n \times n$ set of basic blocks. In our case, the basic block depicted in Fig. 10 contains two $R_{\neq}$ vertices and (when viewed periodically) three edges from the original network which are to be assigned either a $\mathbf{0}$ or a $\mathbf{1}$. In light of the previous sections, the capacity of the constrained system S we are now examining is given by

$$\mathsf{cap}(S) = \lim_{n\to\infty} \frac{\log_2 |\mathrm{Pf}(A)|}{3n^2} = \lim_{n\to\infty} \frac{\log_2 \sqrt{\det(A)}}{3n^2}$$

where A is the skew-symmetric adjacency matrix of the matchgrid corresponding to the $n \times n$ constrained array. The 3 in the denominator comes from the fact that a basic block contains three bit storage positions (three edges from the original network to be assigned a value).

A derivation of an expression for the exact capacity largely depends on the ease of manipulating the matrix A. The first simplification is based on the observation that the matchgates for ϕ_+ contain just one edge which must be taken in any perfect matching, which also forces the edge connecting the matchgate to its single neighboring matchgate to be dropped. Since the weight of the edge is a constant (depending on whether it is a generator or a recognizer), and since we have only $O(n)$ such matchgates along the perimeter, we may ignore them altogether without changing the resulting capacity calculation. So from now on, by abuse of notation, let A denote the skew-symmetric adjacency matrix with the ϕ_+ matchgates and their connecting edges removed.

The components for a compact representation of A are the skew-symmetric matrix for the basic block (where the vertices are indexed as in Fig. 10)

$$B = \begin{pmatrix} 0 & -1 & 1 & -\frac{1}{4} & -1 & 0 & 0 & 0 \\ 1 & 0 & -1 & -\frac{1}{4} & 0 & 0 & 0 & 0 \\ -1 & 1 & 0 & -\frac{1}{4} & 0 & 0 & 0 & 0 \\ \frac{1}{4} & \frac{1}{4} & \frac{1}{4} & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & -1 & 1 & 2 \\ 0 & 0 & 0 & 0 & 1 & 0 & -1 & 2 \\ 0 & 0 & 0 & 0 & -1 & 1 & 0 & 2 \\ 0 & 0 & 0 & 0 & -2 & -2 & -2 & 0 \end{pmatrix}$$

and the matrix $\Delta_{i,j}$ (of the same dimensions as B) which is all zeroes except for position (i, j) which is 1. Furthermore, we need I_n, the $n \times n$ identity matrix, and the $n \times n$ matrix

$$U_n \stackrel{\mathrm{def}}{=} \begin{pmatrix} 0 & 1 & & & \\ & 0 & 1 & & \\ & & \ddots & \ddots & \\ & & & 0 & 1 \\ & & & & 0 \end{pmatrix}$$

with the unspecified positions being zero as well.

We have an $n \times n$ array of basic blocks which we order in the natural way (i.e., from left to right, and from the bottom up). This part of the graph is represented by the skew-symmetric adjacency matrix

$$I_n \otimes I_n \otimes B.$$

We still have to represent the edges between basic blocks in the same row

$$I_n \otimes U_n \otimes \Delta_{6,2} - I_n \otimes U_n^T \otimes \Delta_{6,2}^T$$

and the edges between basic blocks in different rows

$$U_n \otimes I_n \otimes \Delta_{7,3} - U_n^T \otimes I_n \otimes \Delta_{7,3}^T.$$

Thus, we get an expression for the skew-symmetric adjacency matrix A

$$A = I_n \otimes I_n \otimes B + I_n \otimes U_n \otimes \Delta_{6,2} - I_n \otimes U_n^T \otimes \Delta_{6,2}^T + U_n \otimes I_n \otimes \Delta_{7,3} - U_n^T \otimes I_n \otimes \Delta_{7,3}^T. \quad (4)$$

For the last step, we rely on the theory of spectral distribution of Toeplitz matrices (see Tilli [34]). Let us denote $Q \stackrel{\mathrm{def}}{=} [-\pi, \pi]$. For natural numbers $p, k \geqslant 1$, let an integrable p-variate function $f : Q^p \to \mathbb{C}^{k\times k}$ and a multi-index $n =$

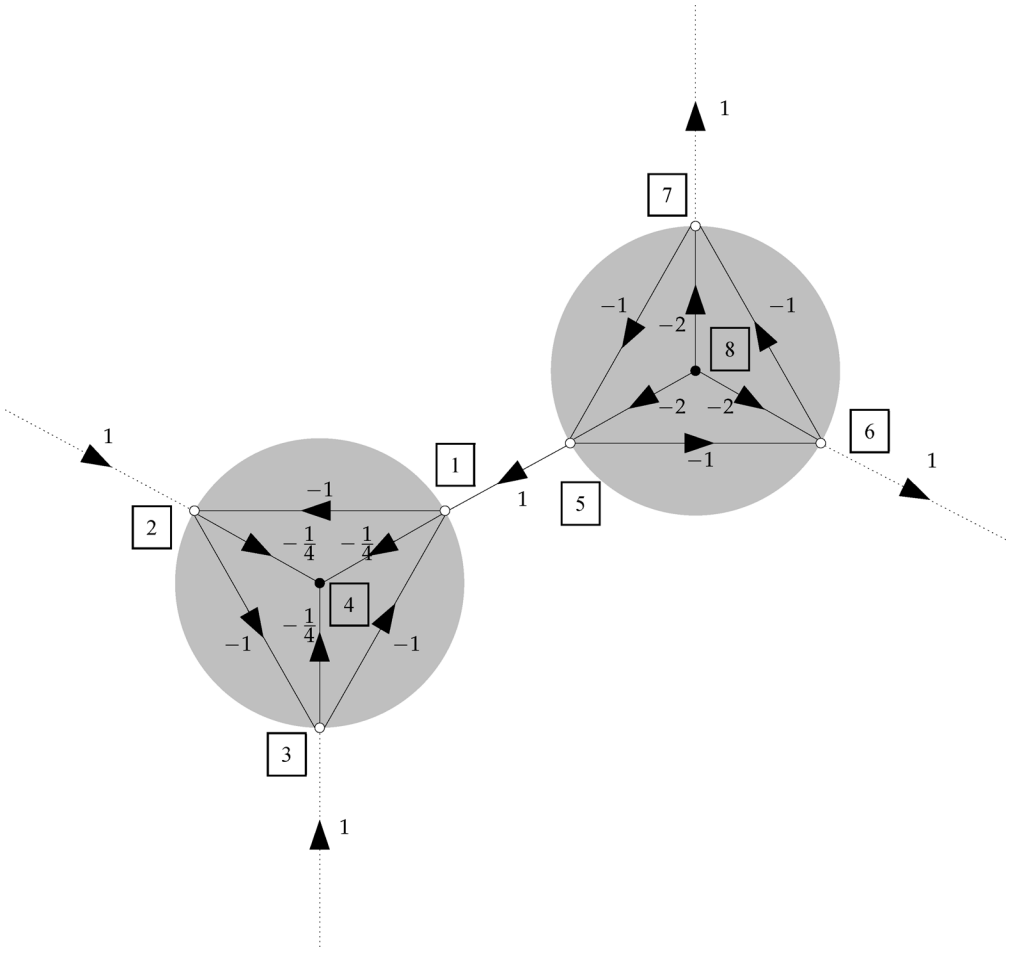


Fig. 10. A Pfaffian orientation of the basic block. The dotted edges denote the edges between translations of the basic block. The numbers in squares index the vertices.

$(n_1, \ldots, n_p)$, $n_i \geqslant 1$ be given. The p-level Toeplitz matrix $T_n(f)$ is defined as

$$T_n(f) \stackrel{\text{def}}{=} \sum_{j_1=-n_1+1}^{n_1-1} \cdots \sum_{j_p=-n_p+1}^{n_p-1} J_{n_1}^{(j_1)} \otimes \cdots \otimes J_{n_p}^{(j_p)} \otimes a_{j_1,\ldots,j_p}(f)$$

where $J_m^{(l)}$ denotes the matrix of order m whose i, j entry equals 1 if $j - i = l$ and equals zero otherwise, and where

$$a_{j_1,\ldots,j_p}(f) \stackrel{\text{def}}{=} \frac{1}{(2\pi)^p} \int_{Q^p} f(\phi) e^{-\boldsymbol{i}(j_1\phi_1 + \cdots + j_p\phi_p)} d\phi$$

is a matrix in $\mathbb{C}^{k\times k}$ and $\boldsymbol{i} = \sqrt{-1}$. The following theorem is due to Tilli [34].

Theorem 7: If $f : Q^p \to \mathbb{C}^{k\times k}$ is an integrable Hermitian matrix-valued function, then for any function F, uniformly continuous and bounded over $\mathbb{R}$ it holds that

$$\lim_{n\to\infty} \frac{1}{n_1 \ldots n_p} \sum_{j=1}^{kn_1\ldots n_p} F[\lambda_j(T_n(f))] = \frac{1}{(2\pi)^p} \int_{Q^p} \sum_{j=1}^{k} F[\lambda_j(f(\phi))] d\phi$$

where $\lambda_j(M)$ denotes the jth eigenvalue of M.

To apply this theorem to our needs we notice the following: First, for an $n \times n$ matrix X we have $\log_2 |\det(X)| = \sum_{i=1}^{n} \log_2 |\lambda_i(X)|$. Second, we notice that A from (4) is a two-level Toeplitz matrix, but it is skew-symmetric and not Hermitian as required. This is easily fixed by noting that $\boldsymbol{i}A$ is Hermitian, and since the order of A is a multiple of 4, then $\det(A) = \det(\boldsymbol{i}A)$. Thus, $\boldsymbol{i}A = T_n(f)$ where we define

$$f(\phi_1, \phi_2) = \boldsymbol{i}[B + e^{\boldsymbol{i}\phi_1}\Delta_{6,2} - e^{-\boldsymbol{i}\phi_1}\Delta_{6,2}^T + e^{\boldsymbol{i}\phi_2}\Delta_{7,3} - e^{-\boldsymbol{i}\phi_2}\Delta_{7,3}^T].$$

Before we continue, we need a lemma to connect the eigenvalues of f with those of $T_n(f)$.

Lemma 8: If $f : Q^p \to \mathbb{C}^{k\times k}$ is an integrable Hermitian matrix-valued function, and $m, M \in \mathbb{R}$ are constants such that

$$m \leqslant \lambda_j(f) \leqslant M$$

for all $1 \leqslant j \leqslant k$, then

$$m \leqslant \lambda_j(T_n(f)) \leqslant M$$

for all $1 \leqslant j \leqslant kn_1 \ldots n_p$.

Proof: First, it is an easy exercise to show that when f is Hermitian then $T_n(f)$ is Hermitian as well. Since the eigenvalues of a Hermitian matrix are all real, let $\underline{\lambda}(\cdot)$ and $\overline{\lambda}(\cdot)$ denote

the minimal and maximal eigenvalues, respectively. It is well known that for a Hermitian matrix H

$$\underline{\lambda}(H) = \min_{x \neq 0} \frac{x^* H x}{x^* x} \quad \text{and} \quad \overline{\lambda}(H) = \max_{x \neq 0} \frac{x^* H x}{x^* x} \tag{5}$$

where x^* denotes the conjugate transpose of x.

Let $m \in \mathbb{R}$ be a lower bound on the eigenvalues of f. By (5), for every vector x (including the 0 vector), $x^* f x \geqslant m x^* x$. From now on it is a matter of keeping track of the indices properly. Let $i = (i_1, \ldots, i_p)$ and $j = (j_1, \ldots, j_p)$ be multi-indices. By the definition of $T_n(f)$ we have $(T_n(f))_{i,j} = a_{j-i}(f)$. Let $x \neq 0$ be a vector of length $kn_1 \ldots n_p$ which we divide into contiguous blocks of length k that we index by the same multi-indices. Thus

$$\begin{aligned} x^* T_n(f) x &= \sum_i \sum_j x_i^* a_{j-i}(f) x_j \\ &= \sum_i \sum_j x_i^* \left[\frac{1}{(2\pi)^p} \int_{Q^p} f(\phi) e^{-\boldsymbol{i}(j-i)\cdot\phi} d\phi \right] x_j \\ &= \frac{1}{(2\pi)^p} \int_{Q^p} \sum_i \sum_j e^{-\boldsymbol{i}(j-i)\cdot\phi} x_i^* f(\phi) x_j d\phi \\ &\geqslant \frac{m}{(2\pi)^p} \int_{Q^p} \sum_i \sum_j e^{-\boldsymbol{i}(j-i)\cdot\phi} x_i^* x_j d\phi \\ &= \frac{m}{(2\pi)^p} \int_{Q^p} \left| \sum_j e^{-\boldsymbol{i}j\cdot\phi} x_j \right|^2 d\phi. \end{aligned} \tag{6}$$

We also have

$$x^* x = \sum_j |x_j|^2 = \frac{1}{(2\pi)^p} \int_{Q^p} \left| \sum_j e^{-\boldsymbol{i}j\cdot\phi} x_j \right|^2 d\phi. \tag{7}$$

Combining (6) and (7) it now follows that

$$\underline{\lambda}(T_n(f)) = \min_{x \neq 0} \frac{x^* T_n(f) x}{x^* x} \geqslant m$$

as we wanted to prove. The proof for the upper bound is symmetric. □

Getting back to our problem, since we are interested in $\log_2 |\det(\boldsymbol{i}A)| = \sum_i \log_2 |\lambda_i(\boldsymbol{i}A)|$, we want to bound the *magnitude* of the eigenvalues of $\boldsymbol{i}A = T_n(f(\phi_1, \phi_2))$. We do this by bounding the magnitude of the eigenvalues of $f(\phi_1, \phi_2)$ and using Lemma 8. We do not really need to find the exact values of $m, M \in \mathbb{R}$ such that $m \leqslant |\lambda_i(f(\phi_1, \phi_2))| \leqslant M$, but show that $M < \infty$ and $m > 0$.

Our first observation is that there exists $M \in \mathbb{R}$ such that $\overline{\lambda}(f(\phi_1, \phi_2)) \leqslant M < \infty$, since $f(\phi_1, \phi_2)$ is a matrix of constant size and bounded magnitude entries. Thus, using standard bounding techniques (for example, Gershgorin circle theorem), the maximal magnitude of an eigenvalue is bounded by a constant. To continue, a simple calculation shows that

$$\det(f(\phi_1, \phi_2)) = 21 - 4\cos\phi_1 - 4\cos\phi_2 - 4\cos(\phi_1 - \phi_2).$$

Thus, we have $\det(f(\phi_1, \phi_2)) \geqslant 9$. Since we also know that $\det(f(\phi_1, \phi_2)) = \prod_i \lambda_i(f(\phi_1, \phi_2))$ and $|\lambda_i(f(\phi_1, \phi_2))| \leqslant M$ and we have a constant number of elements in the product, it follows that there is a constant $m \in \mathbb{R}$, $m > 0$, such that $|\lambda_i(f(\phi_1, \phi_2))| \geqslant m$.

Now that we have bounded the magnitude of the eigenvalues of $f(\phi_1, \phi_2)$, and by Lemma 8, those of $T_n(f(\phi_1, \phi_2))$, we are ready to use Theorem 7. Let us define the function $F : \mathbb{R} \to \mathbb{R}$ as

$$F(x) \stackrel{\text{def}}{=} \begin{cases} \log_2 |m|, & |x| < m \\ \log_2 |x|, & m \leqslant |x| \leqslant M \\ \log_2 |M|, & |x| > M. \end{cases}$$

The function F is uniformly continuous and bounded over $\mathbb{R}$. Furthermore

$$\begin{aligned} \log_2 |\det(f(\phi_1, \phi_2))| &= \sum_i \log_2 |\lambda_i(f(\phi_1, \phi_2))| \\ &= \sum_i F(\lambda_i(f(\phi_1, \phi_2))) \end{aligned}$$

and also

$$\begin{aligned} \log_2 |\det(T_n(f))| &= \sum_i \log_2 |\lambda_i(T_n(f))| \\ &= \sum_i F(\lambda_i(T_n(f))). \end{aligned}$$

By Theorem 7, it now follows that

$$\begin{aligned} \mathsf{cap}(S) &= \lim_{n\to\infty} \frac{\log_2 \sqrt{\det(A)}}{3n^2} \\ &= \lim_{n\to\infty} \frac{1}{6n^2} \log_2 |\det(T_n(f))| \\ &= \frac{1}{24\pi^2} \int_{-\pi}^{\pi} \int_{-\pi}^{\pi} \log_2 |\det(f(\phi_1, \phi_2))| \, d\phi_1 d\phi_2 \\ &= \frac{1}{24\pi^2} \int_{-\pi}^{\pi} \int_{-\pi}^{\pi} \log_2 |21 - 4\cos\phi_1 - 4\cos\phi_2 \\ &\qquad - 4\cos(\phi_1 - \phi_2)| d\phi_1 d\phi_2 \\ &= 0.72399217\ldots. \end{aligned}$$

As far as we know, the number that we just calculated, namely, $0.72399217\ldots$, is the first known nontrivial exact capacity of a two-dimensional constrained system.

A couple of remarks regarding the calculation: If the eigenvalues of $\boldsymbol{i}A$ were *not* bounded away from 0, then we could not set $F(x)$ from Theorem 7 to behave like $\log_2 |x|$ on the interval containing the eigenvalues and still be bounded and uniformly continuous. This creates a problem[6] in using Theorem 7. It might be possible to get an equivalent result to Theorem 7 for $f(\phi)$ which are trigonometric polynomials of degree 1 (see [5]). If all else fails and the eigenvalues are not bounded away from 0, we can convert the equality of Theorem 7 to an inequality, stating that the limit is less than or equal to the integral. This is done by choosing the sequence of functions $F_\ell(x) \stackrel{\text{def}}{=} \max\{-\ell, \log_2 |x|\}$ (where obviously $F_\ell(x) \geqslant \log_2 |x|$), using Theorem 7, and finally using monotone convergence as $\ell \to \infty$.

[6]It should be noted that physicists have been using this claim without rigorous proof for over 40 years. The earliest examples to our knowledge are [24], and also [23] in which an early incorrect form of asymptotic matrix equivalence is used as proof.

IV. A Polynomial-Time Algorithm for Counting Constrained Arrays

In this section, we describe an algorithm for counting the exact number of constrained arrays of size $n \times n$ that runs in time polynomial in n. Though this algorithm applies to the example from the last section, we use the opportunity to switch examples and show two generalizations which extend the reach of our method. The first one we call *generalized relations*, and the second is the use of nonplanar graphs.

The constraint we use throughout this section may seem contrived, but it is instrumental in showing the added degrees of freedom provided by being able to count constrained arrays on other surfaces, such as a torus in this case, and the benefit of generalized relations. It is also interesting in the sense that local constraints enforced by the relations result in a global constraint. The constraint we examine is similar to the $(1,\infty)$-RLL constraint defined by the forbidden patterns shown in Fig. 8. A *violation* of the $(1,\infty)$-RLL constraint is a pair of adjacent $\mathbf{1}$'s where we have two kinds of violations: horizontal and vertical. We define the balanced-violation $(1,\infty)$-RLL constraint as the set of all binary $n \times n$ arrays such that the number of horizontal violations equals the number of vertical violations.

We will consider arrays which are placed on a torus. To be mathematically exact, we index the $n \times n$ array by $\mathbb{Z}_n \times \mathbb{Z}_n$, and since we are now working on the square grid, the positions adjacent to (x,y) are (working modulo n of course)

$$\{(x+1,y),(x-1,y),(x,y+1),(x,y-1)\}.$$

Thus, an $n \times n$ array A is balanced-violation $(1,\infty)$-RLL constrained, if

$$|\{A_{x,y} = A_{x+1,y} = 1\}| = |\{A_{x,y} = A_{x,y+1} = 1\}|$$

where all indices are taken modulo n. We note that the famous $(1,\infty)$-RLL constraint is a subset of this constraint in which the number of violations is further constrained to be zero.

A. Generalized Relations

Given a network of relations on a graph $G = (V,E)$, with every vertex $v \in V$ representing a relation $R_v \subseteq \Omega^{\deg(v)}$, we have considered assignments of values to the edges $A : E \to \Omega$. We called assignment A a satisfying assignment if every relation was satisfied, i.e., for every v, $A_v \in R_v$.

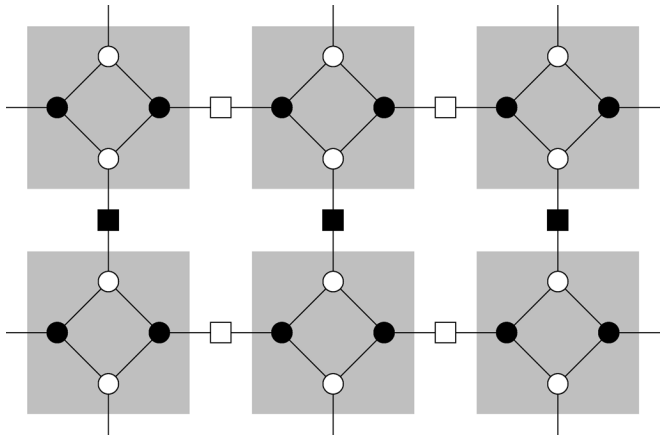


Fig. 11. Part of a network of relations for the balanced-violation $(1,\infty)$-RLL constraint. Circles represent the $R_=$ relation and squares represent the R_{nand} relations. Filled vertices are of the R^+ variant while empty ones are of the R^- variant. The gray squares show the original cells of the rectangular grid.

If we remember that we can think of a relation on n variables as a function $R : \Omega^n \to \{0,1\}$, then the number of satisfying assignments is also represented by

$$\sigma(G) = \sum_A \prod_{v \in V} R_v(A_v) \tag{8}$$

where the sum is over all $|\Omega|^{|E|}$ assignments. Obviously, only if all vertices are satisfied then the product is 1. Thus, the sum is counting only satisfying assignments.

The generalization is now immediately clear. A generalized relation on n variables is a function $R : \Omega^n \to \mathbb{C}$. The "number of satisfying assignments" is still defined by (8) but is not necessarily a nonnegative integer anymore. Intuitively, every assignment gets scored by the product of the "amount of satisfaction" of individual generalized relations. We also note that, although we only use $\mathbb{C}$, it may be replaced by other algebraic structures.

We now turn to designing the network of relations for our constrained system. We still use $\Omega = \{0,1\}$. Let $m \in \mathbb{N}$ be some integer. We will construct a set of networks $N_0, N_1, \ldots, N_{m-1}$ where in N_k, $0 \leqslant k \leqslant m-1$, we will use the following four relations shown at the bottom of the page, where $\omega_m \stackrel{\text{def}}{=} e^{2\pi \boldsymbol{i}/m}$. The network of relations is shown in Fig. 11.

x_1	x_2	x_3	$R_=^+$	$R_=^-$
0	0	0	1	1
0	0	1	0	0
0	1	0	0	0
0	1	1	0	0
1	0	0	0	0
1	0	1	0	0
1	1	0	0	0
1	1	1	ω_{2m}^{3k}	ω_{2m}^{-3k}

x_1	x_2	R_{nand}^+	R_{nand}^-
0	0	1	1
0	1	1	1
1	0	1	1
1	1	ω_m^k	ω_m^{-k}

The first thing to note about this network of relations is that the 4-cycles with two $R^{+}_{=}$ and two $R^{-}_{=}$ will zero out an assignment unless the wires incident to it are all $\mathbf{0}$ or all $\mathbf{1}$. Furthermore, in the case of an all-$\mathbf{1}$ assignment, the score of the assignment is multiplied twice by ω_{2m}^{3k} and twice by ω_{2m}^{-3k} and so, in total, both the all-$\mathbf{0}$ assignment and the all-$\mathbf{1}$ assignment contribute a multiplicative factor of 1 to the score of the assignment. Essentially, the 4-cycle is simply an equality relation on four variables. If we look at the gray cells of Fig. 11, which represent the original cells of the square lattice, then the 4-cycle inside each cell makes sure a single value, either a $\mathbf{0}$ or a $\mathbf{1}$, is assigned to edges exiting the cell in assignments with nonzero contribution. This value reflects the value of the cell in the constrained array.

We also have to somehow check for violations, i.e., two horizontally or vertically adjacent cells with value of $\mathbf{1}$ in the constrained array. To this end, the relations R^{+}_{nand} and R^{-}_{nand} are placed between the gray squares of Fig. 11. They check the value of two adjacent cells as reflected by the assignment to the edges exiting the two adjacent gray squares. Whenever two adjacent $\mathbf{1}$'s are detected, the relations penalize the number of satisfying assignments by a multiplicative factor of either ω_m^k for a vertical violation, or ω_m^{-k} for a horizontal one.

Now assume some assignment has v vertical violations and h horizontal ones. The score of this assignment is obviously $\omega_m^{k(v-h)}$. If we denote by $A_{v,h}$ the number of arrays with v vertical violations and h horizontal violations, and by σ_k the number of satisfying assignments to N_k, then

$$\sigma_k = \sum_{v=0}^{n^2}\sum_{h=0}^{n^2} A_{v,h}\omega_m^{k(v-h)}.$$

It follows that

$$\begin{aligned}\frac{1}{m}\sum_{k=0}^{m-1}\sigma_k &= \sum_{v=0}^{n^2}\sum_{h=0}^{n^2} A_{v,h}\frac{1}{m}\sum_{k=0}^{m-1}\omega_m^{k(v-h)} \\ &= \sum_{v\equiv h \pmod m} A_{v,h}.\end{aligned}$$

Since $0 \leqslant v, h \leqslant n^2$, if we choose any $m > n^2$ then

$$\frac{1}{m}\sum_{k=0}^{m-1}\sigma_k = \sum_{v\equiv h \pmod m} A_{v,h} = \sum_{v=h} A_{v,h}$$

which is exactly what we wanted to count in the first place. Two major issues remain to be dealt with: the fact that we have a nonplanar graph because of the torus, and the fact that we can calculate the above expression in polynomial time.

B. Counting Arrays on a Torus

We start in the usual manner: we arbitrarily choose to realize $R^{+}_{=}$ and R^{+}_{nand} as generators, while $R^{-}_{=}$ and R^{-}_{nand} as recognizers. For the network N_k, $0 \leqslant k \leqslant m-1$, we choose the basis $\boldsymbol{\beta} = [(1,1),(\omega_{2m}^{-k}, -\omega_{2m}^{-k})]$ and the resulting matchgate realizations are shown in Fig. 12.

If we ignore the fact that we are working on a torus, then finding a Pfaffian orientation for the graph is a task which may be done in polynomial time (see [18]). We now turn to handle the problem of working on a torus. If we can draw the graph on a surface with genus g without any edges crossing we call

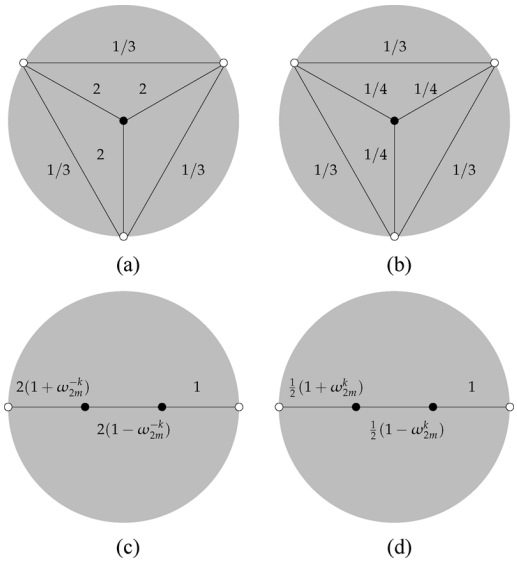


Fig. 12. Using the basis $\boldsymbol{\beta} = [(1,1),(\omega_{2m}^{-k}, -\omega_{2m}^{-k})]$ we get weighted graphs for (a) a generator for $R^{+}_{=}$, (b) a recognizer for $R^{-}_{=}$, (c) a generator for R^{+}_{nand}, and (d) a recognizer for R^{-}_{nand}.

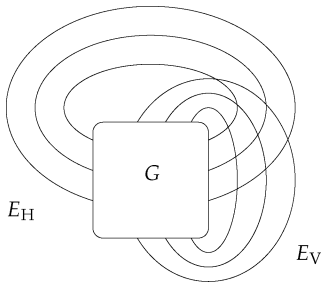


Fig. 13. A drawing of a graph G of genus 1 on the plane with only the horizontal wraparound edges E_{H} crossing the vertical wraparound edges E_{V}.

it a graph of genus g. Kasteleyn stated, without proof, that the perfect matching of a graph of genus g may be calculated using a linear combination of 4^g Pfaffians. A proof of this statement may be found in [13], and for more general surfaces, in [32].

In our case, a torus is a surface with genus 1, and the linear combination takes on a very simple form. First, it is a well-known fact that any graph G of genus 1 we can draw on the plane with no edges crossing, except for a set of horizontal wraparound edges E_{H} and vertical wraparound edges E_{V} which do cross.[7] See Fig. 13 for a sketch.

It was shown in [32], that for any graph G there exists what is called a *crossing orientation* of the edges such that

$$\mathrm{Pf}(A) = \pm \sum_{M\in \mathrm{PM}(G)} (-1)^{\kappa(M)} \prod_{e\in M} w(e) \tag{9}$$

where A is the skew-symmetric adjacency matrix of G and $\kappa(M)$ is the number of crossings in the perfect matching M. If not for $(-1)^{\kappa(M)}$, this would be the expression for $\mathrm{PerfMatch}(G)$. Since the only possible crossings occur in the wraparound edges, for a perfect matching M let h_M denote the number of horizontal wraparound edges in M, and v_M the number of vertical wraparound edges in M. Obviously, the

[7] In the general genus-g case this is called a *plane model* or a *pasting map* (see [32] and references therein).

TABLE I
THE SIGNS GIVEN TO PERFECT MATCHINGS OF G BY $\mathrm{Pf}(A_1),\ldots,\mathrm{Pf}(A_4)$

type	$\mathrm{Pf}(A_1)$	$\mathrm{Pf}(A_2)$	$\mathrm{Pf}(A_3)$	$\mathrm{Pf}(A_4)$
(e, e)	+	+	+	+
(e, o)	+	+	−	−
(o, e)	+	−	+	−
(o, o)	−	+	+	−

number of crossings in M is $\kappa(M) = h_M v_M$. We partition all the perfect matchings according to the parity of h_M and v_M calling them (e, e), (e, o), (o, e), (o, o), where e stands for even, and o stands for odd, while the first entry is the parity of h_M.

The crossing orientation for a torus was shown in [32] to take on a very simple form. We orient all the edges of G except for those in $E_{\mathrm{H}} \cup E_{\mathrm{V}}$. This is always possible since no other edges are crossing. Then we orient the edges of E_{H} as if E_{V} did not exist. Again, this is possible since $G - E_{\mathrm{V}}$ is planar. To complete the orientation, we also orient the edges of E_{V} as if E_{H} did not exist. We call the resulting skew-symmetric adjacency matrix A_1. By (9), $\mathrm{Pf}(A_1)$ counts all the perfect matchings with the correct sign except for those of type (o, o), since that is the only case with an odd number of crossings. We may now flip the signs of the weights on E_{H} while keeping the orientation (hence, keeping the same $\pm$ as in (9)), and get a matrix A_2. If we just flip the signs of the weights on E_{V} we get A_3. And finally, if we flip the signs of the weights on both E_{H} and E_{V} we get A_4. In Table I, we see how the different Pfaffians count the four types of perfect matchings of the original graph G. It is easily seen now that

$$\mathrm{PerfMatch}(G) = \pm\frac{1}{2}\left[\mathrm{Pf}(A_1)+\mathrm{Pf}(A_2)+\mathrm{Pf}(A_3)-\mathrm{Pf}(A_4)\right].$$

It follows that each of the σ_k, $0 \leqslant k \leqslant m-1$, may be calculated up to a sign by a linear combination of four Pfaffians. Moreover, we note that the Pfaffian orientation of the graph is the same for $G_0,\ldots,G_{m-1}$ which correspond to the networks $N_0,\ldots,N_{m-1}$. Thus, the $\pm$ is the same for all and the linear combination of $4m$ Pfaffians will give us $\sum_{k=0}^{m-1}\sigma_k$ up to a sign.

C. Algorithm Complexity

Our task is now to show that we can calculate the linear combination of $4m$ Pfaffians in time polynomial in n. We choose the smallest possible m, i.e., $m = n^2+1$, and then we have $4(n^2+1)$ Pfaffians of matrices of size $cn^2 \times cn^2$ where c is the number of vertices in the basic block, a constant. Thus, if we can show that we can calculate a Pfaffian of an $\ell \times \ell$ matrix in time polynomial in ℓ, then we are done.

There is, however, another complication we have to take care of. The entries of our matrices are complex numbers with infinite precision. We, on the other hand, may only work with some fixed precision of p binary digits before and after the binary point, i.e., represent numbers which are $\leqslant 2^p$ with absolute error $\leqslant 2^{-p}$. Thus, to make sure our algorithm is indeed polynomial in ℓ, we will have to show we can calculate the entries with precision p in polynomial time, run a polynomial-time algorithm with fixed precision, and then recover the correct result in spite of the initial error of approximation and the accumulated error while running the algorithm.

We start by describing the algorithm for calculating the Pfaffian. There are several known polynomial-time algorithms, but in order to easily bound the error, we prefer to use a division-free algorithm. Such algorithms are described in [28] and [22]. We will use the algorithm by Rote [28] and describe it for completeness.

Let A_ℓ be an $\ell \times \ell$ skew-symmetric matrix, ℓ even,[8] and of the form

$$A_\ell = \left(\begin{array}{cc|c} 0 & a & r \\ -a & 0 & -s^T \\ \hline -r^T & s & A_{\ell-2} \end{array}\right)$$

with $A_{\ell-2}$ being an $(\ell-2)\times(\ell-2)$ skew-symmetric matrix. Let us also define the $\ell \times \ell$ skew-symmetric "identity" matrix B_ℓ by

$$B_\ell \stackrel{\mathrm{def}}{=} \begin{pmatrix} 0 & 1 & & & & & \\ -1 & 0 & & & & & \\ & & 0 & 1 & & & \\ & & -1 & 0 & & & \\ & & & & \ddots & & \\ & & & & & 0 & 1 \\ & & & & & -1 & 0 \end{pmatrix}.$$

Finally, given A_ℓ, we define the formal power series $G_{A_\ell}(\lambda)$ as

$$G_{A_\ell}(\lambda) \stackrel{\mathrm{def}}{=} -\lambda^2 + a + \sum_{i=1}^{\infty} r B_{\ell-2}(A_{\ell-2}B_{\ell-2})^{i-1}s\lambda^{-2i}.$$

Given these definitions, it was shown in [28] that $\mathrm{Pf}(A_\ell)$ is the coefficient of λ^0 in the formal power series $P_{A_\ell}(\lambda)$, defined recursively by

$$P_{A_\ell}(\lambda) \stackrel{\mathrm{def}}{=} G_{A_\ell}(\lambda)\cdot P_{A_{\ell-2}}(\lambda).$$

Several observations regarding this algorithm may be easily made. We use very loose bounds which are good enough to show that the algorithm is indeed polynomial with the desired accuracy, while allowing a very simple analysis. We have to find the coefficients of the terms in $\ell/2$ power series $G_{A_\ell}, G_{A_{\ell-2}},\ldots,G_{A_2}$, and in each of those it suffices to compute the coefficients of the terms up to that of $\lambda^{-\ell}$. Hence, we need to compute $\ell^2/4$ coefficients of the form $rB_j(A_jB_j)^i s$. Each of these is made up of a product of up to $\ell/2+3 \leqslant \ell$ (for $\ell \geqslant 6$) matrices of size at most $\ell \times \ell$. We can use a simple tree for finding the product: in the first round, we partition the ℓ matrices into pairs and take their product. In the second round, we take the resulting $\ell/2$ matrices and repeat the process, and so on. In each round, we use the results from the previous round, and in total we have at most $\lceil \log_2 \ell \rceil$ rounds.

In the next stage, after computing all the coefficients we need to convolve them to compute the coefficient of λ^0 in P_{A_ℓ}. This, again, may be implemented by taking the product of at most ℓ matrices of size at most $\ell^2 \times \ell^2$ in $\lceil \log_2 \ell \rceil$ rounds as we did in the previous stage. The matrices simply hold the coefficients in staggered rows padded with zeroes, to simulate the convolution. This process is then repeated for each of the Pfaffians and the resulting numbers summed, and since the number of Pfaffians required in our case is also $O(\ell)$, we have a polynomial-time algorithm.

[8] We note that the restriction to even ℓ does not hurt us since our graphs will always have an even number of vertices or else no perfect matching exists.

In summary, we need to compute the linear combination of $c'\ell$ Pfaffians, c' a constant, where for each Pfaffian we take the product of matrices of size at most $\ell^2 \times \ell^2$ in at most $2\lceil \log_2 \ell \rceil$ rounds. It remains for us to prove that we can do so with fixed precision arithmetic using a polynomial number of digits. As mentioned before, let us work with some fixed precision of p binary digits before and after the binary point, i.e., represent numbers which are $\leqslant 2^p$ with absolute error $\leqslant 2^{-p}$. If we show that for some p which is polynomial in ℓ we can still recover the wanted result despite the errors introduced along the way, then we are done.

We now follow a similar line of reasoning used by Valiant in [36], which we describe here for completeness, and adapt it to our case. Let $Y \subset \mathbb{C}$ denote the set of all values appearing as entries in the initial matrices whose Pfaffians we want to compute. Define $D \stackrel{\text{def}}{=} \max\{|y| \mid y \in Y\}$. We note that because of the edges between matchgates with weight 1, we have $D \geqslant 1$. We now run the above mentioned algorithm with a fixed precision of p binary digits before and after the binary point, resulting in absolute error of at most 2^{-p} in each roundoff operation.

We need to bound both the largest modulus of any entry in the matrices used during the algorithm, as well as the resulting maximal absolute error. Before we do that, let F_i denote an upper bound on the largest modulus of any value computed in the ith round of the *exact* algorithm, i.e., with infinite precision. Obviously $F_0 = D$ and we can use $F_i \geqslant F_{i-1}^2 \ell^2$ since we are taking the product of two $\ell^2 \times \ell^2$ matrices with entries with maximal modulus F_{i-1}. Certainly, taking $F_i = (D\ell^2)^{2^i}$ gives an upper bound which is strong enough for our needs. We note that $F_i \geqslant 1$ necessarily.

Now let ϵ_i denote an upper bound on the absolute error in the modulus of any entry in the ith round of the *fixed precision* algorithm, which is caused by the roundoff operations. We can take $\epsilon_0 = 2^{-p}$, and we will make sure that $2^{-p} \leqslant \epsilon_i \leqslant \frac{1}{4c'\ell}$, where we recall that $c'\ell$ is the number of Pfaffians in the linear combination. These bounds on ϵ_i will be kept throughout the computation by noting that $\epsilon_i \leqslant \epsilon_{i+1}$ and by bounding the error in the last round by choosing a suitable precision p.

Let U and V denote the modulus of two entries after $i-1$ rounds in the *exact* algorithm. Then in the *fixed precision* algorithm, the maximal error in modulus when multiplying the corresponding values is bounded by taking

$$\begin{aligned}(U+\epsilon_{i-1})(V+\epsilon_{i-1}) + 2^{-p} - UV &= (U+V)\epsilon_{i-1} + \epsilon_{i-1}^2 + 2^{-p} \\ &\leqslant 2F_{i-1}\epsilon_{i-1} + \epsilon_{i-1}^2 + 2^{-p} \stackrel{\text{def}}{=} \epsilon'.\end{aligned}$$

Thus, after ℓ^2 summations (which incur an added 2^{-p} error) we get the maximal absolute error in the modulus of an entry after the ith round to be at most $\ell^2(\epsilon' + 2^{-p})$, i.e.,

$$\begin{aligned}\epsilon_i &\stackrel{\text{def}}{=} \ell^2(2F_{i-1}\epsilon_{i-1} + \epsilon_{i-1}^2 + 2\cdot 2^{-p}) \\ &\leqslant 5\ell^2 F_{i-1}\epsilon_{i-1} \leqslant (5\ell^2)^i F_{i-1}F_{i-2}\ldots F_0\epsilon_0 \\ &\leqslant (5\ell^2)^i (D\ell^2)^{2^i} 2^{-p}\end{aligned}$$

where we used the fact that $\epsilon_{i-1}^2 \leqslant F_{i-1}\epsilon_{i-1}$ and $2^{-p} \leqslant \epsilon_{i-1} \leqslant F_{i-1}\epsilon_{i-1}$.

Since we are interested in $2\log_2 \ell + O(1)$ rounds, then for $p = O(\ell^2)(\log_2 D + \log_2 \ell)$ binary places after the binary point, the absolute error in modulus in any of the entries is small, $\epsilon_i \leqslant \frac{1}{4c'\ell}$. Furthermore, since no entry is larger in magnitude than $F_i = (D\ell^2)^{2^i}$, then $p = O(\ell^2)(\log_2 D + \log_2 \ell)$ binary places to the left of the binary point will suffice as well. It now follows that the sum of $c'\ell$ Pfaffians, each computed to within an absolute error of $\frac{1}{4c'\ell}$, has an absolute error of magnitude at most $\frac{1}{4}$. Since we know the sum of the Pfaffians should be an integer, we can easily round the result to the nearest one and get the correct value.

Finally, we have to consider whether we can compute the entries of the initial matrices, whose Pfaffians we want to compute, to p digits of precision in polynomial time. The entries which contain integer constants or even rational numbers, are obviously easy to compute to within a polynomial number of binary digits in polynomial time. The slightly more complicated case is that of the entries which contain $\sin(2\pi k/m)$ and $\cos(2\pi k/m)$. The constant π is easily computed to within a polynomial number of digits using, for example, the Bailey, Borwein, and Plouffe (BBP) algorithm [1]. Then $\sin(2\pi k/m)$ may be computed using a simple Maclaurin series, only we have to consider two sources for error: the error in the approximation of π, and the error caused by computing only the first n terms in the series.

Let $0 < x = 2\pi k/m < 2\pi$, and let us examine $\sin(x+\delta)$ where δ is a function of the absolute error introduced by the computation of π:

$$\sin(x+\delta) = (x+\delta) - \frac{(x+\delta)^3}{3!} + \frac{(x+\delta)^5}{5!} + \cdots \pm R_n$$

where

$$|R_n| \leqslant \frac{(x+\delta)^n}{n!}.$$

We can assume $|\delta| \leqslant x$. By the Stirling approximation we have $|R_n| = 2^{-\Omega(n)}$ for $n > 4\pi e$. Now

$$\frac{(x+\delta)^i}{i!} \leqslant \frac{x^i}{i!} + \left|\frac{2^i x^{i-1}\delta}{i!}\right|$$

since $|\delta| \leqslant x$. Thus, the absolute error ϵ in computing $\sin(x)$ is at most

$$\begin{aligned}\epsilon &= 2^{-\Omega(n)} + 2|\delta| \sum_{i=1,3,5,\ldots} \frac{(2x)^{i-1}}{i!} \\ &\leqslant 2^{-\Omega(n)} + 2|\delta| \sum_{i=1}^{\infty} \frac{(4\pi)^{i-1}}{(i-1)!} \\ &= 2^{-\Omega(n)} + 2|\delta| e^{4\pi}\end{aligned}$$

but since δ may be made exponentially small in polynomial time, so is ϵ. A similar analysis applies to $\cos(2\pi k/m)$ as well.

V. Conclusion and Open Problems

We presented a general method that enables the calculation of the exact capacity of some two-dimensional constrained systems, as well as a polynomial-time algorithm for counting the exact number of constrained arrays in the system. The method uses a series of reductions, from a given constrained system to a network of relations and then to a weighted graph. It is the theory of spectral distribution of Toeplitz matrices that allows us to find the limit of the determinant of the modified adjacency matrix of that weighted graph and in turn yields the capacity of the system.

While we were able to rigorously compute the exact capacity of the PC constraint in a two-dimensional system, sadly, we have not been able thus far to come up with an exact and rigorous closed-form solution to the hard-square entropy constant, i.e., the capacity of the two-dimensional $(1,\infty)$-RLL constraint.

This raises the key open question: What is the expressive power of our proposed method?

While performing the reductions associated with the proposed method, one may "get stuck" at two different stages: i) not being able to find a basis for the holographic reduction, or ii) getting a modified adjacency matrix with eigenvalues not bounded away from 0. Without knowing the expressive power of this method we do not know whether we reached a dead end, or simply took the wrong path in fixing some of the many degrees of freedom the method offers. Those degrees of freedom generate some more open problems.

- Is there a systematic or best way of reducing a constrained system to a network of relations (perhaps generalized relations)? The wrong reduction may lead to a dead end in any of the next stages of the reduction.
- Given a certain basis, what are the sets of matchgates that are realizable together?
- Can we generalize holographic reductions to nonbinary alphabets? This would perhaps enable us to create planar networks of relations for "wider" constraints such as the currently binary nonplanar no-isolated-bit and the general (d, k)-RLL constraints.
- How do we choose generators, recognizers, or transducers? Can we break down relations on a large number of variables, to smaller relations (perhaps creating generator/recognizer conflicts)? For example, the 4-cycles in Fig. 11 are essentially emulating a transducer matchgate with two inputs and two outputs. Replacing it with a single equality matchgate which is either a generator or a recognizer eliminates all possible bases for a holographic reduction.

We trust and hope that these interesting open problems will be the subject of future research.

ACKNOWLEDGMENT

The first author would like to thank Matthew Cook for introducing him to networks of relations. Both authors would like to thank Albrecht Böttcher, Paolo Tilli, and Harold Widom, for providing help with the theory of Toeplitz determinants. The authors also wish to thank Peter Keevash and Eyal Rozenman for commenting on earlier versions of this work. Finally, the authors would like to thank the two anonymous reviewers, whose comments improved the presentation of this paper.

REFERENCES

[1] D. Bailey, P. Borwein, and S. Plouffe, "On the rapid computation of various polylogarithmic constants," *Math. Comp.*, vol. 66, no. 218, pp. 903–913, Apr. 1997.

[2] F. Barahona, "On the computation complexity of Ising spin glass models," *J. Phys. A: Math. Gen.*, vol. 15, pp. 3241–3252, 1982.

[3] R. J. Baxter, "Hard hexagons: Exact solution," *J. Phys. A: Math. Gen.*, vol. 13, pp. L61–L70, 1980.

[4] R. J. Baxter, *Exactly Solved Models in Statistical Mechanics*. New York: Academic, 1982.

[5] A. Böttcher, personal communication, Jan. 2007.

[6] N. Calkin and H. Wilf, "The number of independent sets in the grid graph," *SIAM J. Discr. Math.*, vol. 11, pp. 54–60, 1998.

[7] M. Cook, "Networks of Relations," Ph.D. dissertation, Calif. Inst. Technol., Pasadena, CA, 2005 [Online]. Available: http://paradise.caltech.edu/papers/thesis011.pdf

[8] N. Creignou, S. Khanna, and M. Sudan, *Complexity Classifications of Boolean Constraint Satisfaction Problems*. Philadelphia, PA: SIAM, 2001.

[9] R. Dechter, *Constraint Processing*. San Francisco, CA: Morgan Kaufmann, 2003.

[10] M. E. Fisher, "Statistical mechanics of dimers on plane lattice," *Phys. Rev.*, vol. 124, no. 6, pp. 1664–1672, Dec. 1961.

[11] M. E. Fisher, "On the dimer solution of planar Ising models," *J. Math. Phys.*, vol. 7, no. 10, pp. 1776–1781, Oct. 1966.

[12] S. Forchhammer and T. V. Laursen, "A model for the two-dimensional no isolated bit constraint," in *Proc. IEEE Int. Symp. Information Theory*, Seattle, WA, Jul. 2006, pp. 1189–1193.

[13] A. Galluccio and M. Loebl, "On the theory of Pfaffian orientations. I. Perfect matchings and permanents," *Elec. J. Comb.*, vol. 6, 1999.

[14] D. Geiger, "Closed systems of functions and predicates," *Pacific J. Math.*, vol. 27, pp. 95–100, 1968.

[15] S. Halevy, J. Chen, R. M. Roth, P. H. Siegel, and J. K. Wolf, "Improved bit-stuffing bounds on two-dimensional constraints," *IEEE Trans. Inf. Theory*, vol. 50, no. 5, pp. 824–838, May 2004.

[16] H. Ito, A. Kato, Z. Nagy, and K. Zeger, "Zero capacity region of multidimensional run length constraints," *Elec. J. Comb.*, vol. 6, 1999.

[17] P. W. Kasteleyn, "The statistics of dimers on a lattice. I. The number of dimer arrangements on a quadratic lattice," *Physica*, vol. 27, pp. 1209–1225, 1961.

[18] P. W. Kasteleyn, "Graph theory and crystal physics," in *Graph Theory and Theoretical Physics*, F. Harary, Ed. New York: Academic, 1967, pp. 43–110.

[19] A. Kato and K. Zeger, "On the capacity of two-dimensional run-length constrained channels," *IEEE Trans. Inf. Theory*, vol. 45, no. 5, pp. 1527–1540, Jul. 1999.

[20] R. Kenyon, "The planar dimer model with boundary: A survey," in *CRM Monograph Series*, M. Baake and R. Moody, Eds. Providence, RI: AMS, 2000, pp. 307–328.

[21] G. Kuperberg, "An exploration of the permanent-determinant method," *Elec. J. Comb.*, vol. 5, 1998.

[22] M. Mahajan, P. R. Subramanya, and V. Vinay, "The combinatorial approach yields an NC algorithm for computing Pfaffians," *Discr. Appl. Math.*, vol. 143, pp. 1–16, 2004.

[23] E. W. Montroll, "Lattice statistics," in *Applied Combinatorial Mathematics*, E. F. Beckenbach, Ed. New York: Wiley, 1964, pp. 96–143.

[24] E. W. Montroll, R. B. Potts, and J. C. Ward, "Correlations and spontaneous magnetization of the two-dimensional Ising model," *J. Math. Phys.*, vol. 4, no. 2, pp. 308–322, 1963.

[25] S. Norine, "Drawing Pfaffian graphs," in *Lecture Notes in Computer Science*. Berlin, Germany: Springer-Verlag, 2005, vol. 3383, pp. 371–376.

[26] L. Onsager, "Crystal statistics. I. A two-dimensional model with an order-disorder transition," *Phys. Rev.*, vol. 65, pp. 117–149, 1944.

[27] N. Robertson, P. D. Seymour, and R. Thomas, "Permanents, Pfaffian orientations, and even directed circuits," *Ann. Math.*, vol. 150, pp. 929–975, 1999.

[28] G. Rote, "Division-free algorithms for the determinant and the Pfaffian: Algebraic and combinatorial approaches," in *Lecture Notes in Computer Science*. Berlin, Germany: Springer-Verlag, 2001, vol. 2122, pp. 119–135.

[29] M. Schwartz and A. Vardy, "Tight asymptotic bounds on the capacity of multi-dimensional $(0, k)$-RLL," in *Proc 16th AAECC , Las Vegas, NV, Feb. 2006 (Lecture Notes in Computer Science)*. Berlin, Germany: Springer-Verlag, 2006, vol. 3857, pp. 225–234.

[30] C. E. Shannon, "A mathematical theory of communication," *Bell Syst. Tech. J.*, vol. 27, pp. 379–423, Jul. 1948.

[31] H. N. V. Temperley and M. E. Fisher, "Dimer problem in statistical mechanics—An exact result," *Phil. Mag.*, vol. 6, pp. 1061–1063, 1960.

[32] G. Tesler, "Matchings in graphs on nonorientable surfaces," *J. Combin. Theory Ser. B*, vol. 78, pp. 198–231, 2000.

[33] R. Thomas, "A survey of Pfaffian orientations of graphs," in *Proc. Int. Congress of Mathematicians, ICM2006*, Madrid, Spain, 2006, pp. 963–984.

[34] P. Tilli, "A note on the spectral distribution of Toeplitz matrices," *Linear and Multilinear Algebra*, vol. 45, pp. 147–159, 1998.

[35] L. G. Valiant, "Quantum circuits that can be simulated classically in polynomial time," *SIAM J. Comput.*, vol. 31, no. 4, pp. 1229–1254, 2002.

[36] L. G. Valiant, "Holographic algorithms," in *Proc. 45th Annu. IEEE Symp. Foundations of Computer Science (FOCS2004)*, Rome, Italy, Oct. 2004, pp. 306–315.

[37] L. G. Valiant, "Holographic circuits," in *Proc. ICALP32 (Lecture Notes in Computer Science)*. Berlin, Germany: Springer-Verlag, 2005, vol. 3980, pp. 1–15.

[38] V. V. Vazirani and M. Yannakakis, "Pfaffian orientations, 0–1 permanents, and even cycles in directed graphs," *Discr. Appl. Math.*, vol. 25, pp. 179–190, 1989.