Polynomial Constraint Satisfaction Problems, Graph Bisection, and the Ising Partition Function

ALEXANDER D. SCOTT
University of Oxford
and
GREGORY B. SORKIN
IBM Research

We introduce a problem class we call Polynomial Constraint Satisfaction Problems, or PCSP. Where the usual CSPs from computer science and optimization have real-valued score functions, and partition functions from physics have monomials, PCSP has scores that are arbitrary multivariate formal polynomials, or indeed take values in an arbitrary ring.

Although PCSP is much more general than CSP, remarkably, all (exact, exponential-time) algorithms we know of for 2-CSP (where each score depends on at most 2 variables) extend to 2-PCSP, at the expense of just a polynomial factor in running time. Specifically, we extend the reduction-based algorithm of Scott and Sorkin; the specialization of that approach to sparse random instances, where the algorithm runs in polynomial expected time; dynamic-programming algorithms based on tree decompositions; and the split-and-list matrix-multiplication algorithm of Williams.

This gives the first polynomial-space exact algorithm more efficient than exhaustive enumeration for the well-studied problems of finding a maximum bisection of a graph, and calculating the partition function of an Ising model. It also yields the most efficient algorithm known for certain instances of counting and/or weighted Maximum Independent Set. Furthermore, PCSP solves both optimization and counting versions of a wide range of problems, including all CSPs, and thus enables samplers including uniform sampling of optimal solutions and Gibbs sampling of all solutions.

Categories and Subject Descriptors: F.2.2 [**Analysis of Algorithms and Problem Complexity**]: Nonnumerical Algorithms and Problems—*Computations on discrete structures*

General Terms: Algorithms, Theory

Additional Key Words and Phrases: Exponential-time algorithm, exact algorithm, constraint satisfaction, discrete optimization, partition function, generating function

Author's address: A.D. Scott, Mathematical Institute, University of Oxford, 24-29 St Giles', Oxford, OX1 3LB, UK, scott@maths.ox.ac.uk. Research supported in part by EPSRC grant GR/S26323/01.
Author's address: G.B. Sorkin, Department of Mathematical Sciences, IBM T.J. Watson Research Center, Yorktown Heights NY 10598, USA, sorkin@watson.ibm.com.

1. INTRODUCTION

Many decision and optimization problems, such as 3-colorability, Max Cut, Max k-Cut, and Max 2-Sat, belong to the class Max 2-CSP of constraint satisfaction problems with at most two variables per constraint. A Max 2-CSP (or for short, simply CSP) instance defines a "score function" or "soft constraint" on each vertex and edge of a "constraint graph", and its solution is a vertex coloring or "assignment" maximizing the total score.

We define a more general class we call Polynomial 2-CSP, or simply 2-PCSP or PCSP. (The PCSP definition extends obviously to longer clauses, but much less is known about algorithms even for 3-CSP.) Comparing with a CSP, the vertex and edge scores of a PCSP are (potentially multivariate) polynomials (rather than real numbers), the score of a coloring is the product of all vertex and edge scores (rather than their sum), and an instance's "value", or "partition function", is the sum of scores over all colorings (rather than the maximum). In fact the scores may be "generalized polynomials", whose powers are not restricted to positive integers but may be arbitrary reals, but we use "polynomial" for want of a better term.

Compared with the "max–sum" CSP formulation typical in the CSP community, PCSP has a "sum–product" form commonly seen as the "partition function" in statistical physics: a sum over configurations, of a product of exponentials of "energies". In the physics settings we are familiar with, the partition function is always a product of monomials (sometimes multivariate, for example with energy and magnetization), whereas our PCSPs permit polynomials. With general polynomials, PCSP is closed under a variety of operations enabling its (relatively) efficient solution by a number of algorithmic approaches.

PCSP includes many optimization problems not in the class CSP, including Maximum Bisection, Maximum Clique, Sparsest Cut, various judicious partitioning problems, Max Ones 2-Sat, and many others. Capitalizing on the fact that a PCSP's score functions may be multivariate polynomials, in all these cases, we simply track several score functions at once. For example, for Max Bisection we track the size of one side of the partition as well as the size of the cut.

PCSPs give a powerful technique for counting and sampling solutions. Given a CSP, there is a natural way to obtain a corresponding PCSP. Solving the PCSP gives a generating function for the number of CSP assignments of each possible score, which allows us to count or (by successively solving subinstances) to sample uniformly from solutions of maximum score, or to sample all solutions according to the Gibbs distribution or any other score-based distribution.

PCSP is interesting because, despite being more general than CSP, all the best CSP algorithms we know of can be extended to solve PCSP, with the same running-time bounds. (Exact CSP algorithms are generally exponential-time, and solving a PCSP takes just a polynomial factor longer than solving a CSP instance of the same size, for the extra overhead of adding and multiplying polynomials rather than reals.) Specifically, four algorithms that extend are: the split-and-list matrix-multiplication algorithm of Williams [Williams 2004]; the reduction-based algorithm of Scott and Sorkin [Scott and Sorkin 2007]; the specialization of that approach to sparse random instances, where the algorithm runs in polynomial expected time [Scott and Sorkin 2006b]; and dynamic-programming algorithms based

on tree decompositions [Jansen et al. 2005; Scott and Sorkin 2007; Kneis et al. 2009]. Both the definition of PCSP and all our algorithms only require us to take sums and products of polynomial "scores". It is thus natural to work in a still more general class that we call Ring CSP, where each score function and the "value" of an instance are given by element of an arbitrary ring R; PCSPs are a special case where R is a polynomial ring.

2. OUTLINE

Following a small amount of notation in Section 3, Section 4 defines the classes CSP, PCSP and RCSP. Section 5 provides examples of CSPs and related PCSPs; because this includes many problems beyond CSP, such as graph bisection, the Ising partition function, and judicious partitioning, it is a focal point of the paper.

In Sections 6–9, we show how four CSP algorithms extend to PCSP/RCSP, and analyze them in terms of the number of ring operations (or sums and products of polynomials in the case of PCSPs) that they require. In Section 10 we consider the time and space complexity of the ring operations for PCSPs. We put the two together in Section 11, summarizing the time and space complexity of each algorithm, and noting the circumstances favoring one algorithm over another.

In addition to counting maximum CSP solutions, a PCSP's partition function also enables construction of an optimal solution, and various forms of perfect random sampling from optimal solutions or all solutions; this is taken up in Section 12.

3. NOTATION

In the next section we will define the class of CSPs and our new class PCSP of Polynomial CSPs. An instance of either has a "constraint graph" $G = (V, E)$ with vertex set V and edge set E, and we reserve the symbols G, V and E for these roles. An instance of CSP or PCSP also has a domain of *values* or *colors* that may be assigned to the vertices (variables), for example $\{\text{true}, \text{false}\}$ for a satisfiability problem, or a set of colors for a graph coloring problem. In general we will denote this domain by $[k]$, interpreted as $\{0, \ldots, k-1\}$ or (it makes no difference) $\{1, \ldots, k\}$. At the heart of both the CSP and PCSP instance will be cost or "score" terms $s_v(i)$ (for $v \in V(G)$, $i \in [k]$) and $s_{xy}(i, j)$ (for $xy \in E(G)$, $i, j \in [k]$).

For CSPs the scores s are real numbers, while for PCSPs they are polynomials which we write in the variable z if univariate, or if multivariate over z, w or $z, w_1, w_2, \ldots$. Just as CSP scores s need not be positive integers, exponents in the PCSP score "polynomials" may be fractional or negative (or both). (The term "fractional polynomial" is used in the literature of statistical regression.)

We use $\uplus$ to indicate disjoint union, so $V_0 \uplus V_1 = V$ means that V_0 and V_1 partition V, i.e., $V_0 \cap V_1 = \emptyset$ and $V_0 \cup V_1 = V$. The notation $O^\star$ hides polynomial factors in any parameters, so for example $O(n2^{5+19m/100})$ is contained in $O^\star\big(2^{19m/100}\big)$.

4. CSP, POLYNOMIAL CSP, AND RING CSP

4.1 CSP

Let us begin by defining the problem class CSP over a domain of size k. An *instance I of CSP with constraint graph $G = (V, E)$ and domain $[k]$* has the following

ingredients:

(1) a real number $s_\emptyset$;
(2) for each vertex $v \in V$ and color $i \in [k]$, a real number $s_v(i)$;
(3) for each edge $xy \in E$ and any colors $i, j \in [k]$, a real number $s_{xy}(i, j)$.

We shall refer to these quantities as, respectively, the *nullary score*, the *vertex scores*, and the *edge scores*. Note that we want only one score for an edge $\{i, j\}$ with given colors $\{x, y\}$ assigned its respective endpoints, so $s_{xy}(i, j)$ and $s_{yx}(j, i)$ are taken to be equivalent names for the same score (or one may simply assume that $x < y$).

Given an *assignment* (or *coloring*) $\sigma\colon V \to [k]$, we define the *score of* σ to be the real number

$$I(\sigma) := s_\emptyset + \sum_{v \in V} s_v(\sigma(v)) + \sum_{xy \in E} s_{xy}(\sigma(x), \sigma(y)).$$

The instance's solution is the maximum possible score

$$\max_{\sigma: V \to [k]} I(\sigma) = \max_{\sigma: V \to [k]} \Big(s_\emptyset + \sum_{v \in V} s_v(\sigma(v)) + \sum_{xy \in E} s_{xy}(\sigma(x), \sigma(y)) \Big) \tag{1}$$

or the assignment σ achieving the maximum; it is straightforward to get either from the other. For obvious reasons, (1) is commonly referred to as a *max–sum* formulation.

In this paper we take a generating-function approach, among other things enabling us to count the number of solutions satisfying various properties. Given an instance I of CSP, the corresponding generating function is the polynomial

$$\sum_{\sigma: V \to [k]} z^{I(\sigma)}. \tag{2}$$

More generally, if we want to keep track of several quantities simultaneously, say $I(\sigma), J(\sigma), \cdots$, we can consider a multivariate generating function $\sum_\sigma z^{I(\sigma)} w^{J(\sigma)} \cdots$. Calculating the generating function in the obvious way, by running through all $k^{|V|}$ assignments, is clearly very slow, so it is desirable to have more efficient algorithms.

In working with generating functions, we borrow some notions from statistical physics. We think of the score $I(\sigma)$ as a "Hamiltonian" measuring the "energy" of a configuration σ. Thus edge scores correspond to "pair interactions" between adjacent sites, while vertex scores measure the effect of a "magnetic field". (From this perspective, the nullary score is just a constant that disappears after normalization.) The generating function is then the "partition function" for this model. (The partition function is often written in the form $\sum_\sigma \exp(-\beta I(\sigma))$, where β is known as the *inverse temperature*, but substituting z for $e^{-\beta}$ yields the partition function in polynomial form.)

A crucial element of our approach is that the score $I(\sigma)$ can be broken up as a sum of local interactions, and thus an expression such as $z^{I(\sigma)}$ can be expressed as a product of monomials corresponding to local interactions:

$$z^{I(\sigma)} = z^{s_\emptyset} \cdot \prod_{v \in V} z^{s_v(\sigma(v))} \cdot \prod_{xy \in E} z^{s_{xy}(\sigma(x), \sigma(y))}.$$

In order to provide a framework for this approach, we introduce a generalized version of constraint satisfaction where the scores are polynomials (in some set of variables) instead of real numbers, and the score of an assignment is taken as a product rather than a sum.

4.2 Polynomial CSP

An *instance I of Polynomial CSP, with constraint graph G and domain* $[k]$, has the following ingredients:

(1) a set of formal variables, and various polynomials over the reals in these variables:
(2) a polynomial $p_\emptyset$;
(3) for each vertex $v \in V$ and color $i \in [k]$, a polynomial $p_v(i)$;
(4) for each edge $xy \in E$ and pair of colors $i, j \in [k]$, a polynomial $p_{xy}(i, j)$.

We refer to these three types of polynomial as, respectively, the *nullary polynomial*, the *vertex polynomials*, and the *edge polynomials*. (Reiterating from Section 3, we use "polynomial" in a general sense, allowing negative and fractional powers.) We want only one polynomial for a given edge with given colors on its endpoints, so again we either take $p_{xy}(i, j)$ and $p_{yx}(j, i)$ to be equivalent or simply assume that $x < y$.

Given an assignment $\sigma\colon V \to [k]$, we define the *score of* σ to be the polynomial

$$I(\sigma) := p_\emptyset \cdot \prod_{v \in V} p_v(\sigma(v)) \cdot \prod_{xy \in E} p_{xy}(\sigma(x), \sigma(y)). \tag{3}$$

We then define the *partition function Z_I of I* by

$$Z_I = \sum_{\sigma\colon V \to [k]} I(\sigma) = \sum_{\sigma\colon V \to [k]} p_\emptyset \cdot \prod_{v \in V} p_v(\sigma(v)) \cdot \prod_{xy \in E} p_{xy}(\sigma(x), \sigma(y)). \tag{4}$$

The partition function is the "solution" of a PCSP instance. In contrast with the max–sum formulation of (1), (4) is referred to as a *sum–product* formulation. The sense of the PCSP formulation will become clearer with examples in the next section.

4.3 Ring CSP

The definition of $I(\sigma)$ in equation (3), and for the most part our analysis in later sections, requires only addition and multiplication of scores. It therefore makes sense to work in a still more general context where the score functions are elements of an arbitrary ring. We define an *instance I of RCSP over a ring R, with constraint graph G and domain* $[k]$, to have the following ingredients:

(1) a ring element $r_\emptyset$;
(2) for each vertex $v \in V$ and color $i \in [k]$, a ring element $r_v(i)$;
(3) for each edge $xy \in E$ and pair of colors $i, j \in [k]$, a ring element $r_{xy}(i, j)$.

The *score* of an assignment $\sigma\colon V \to [k]$ is the ring element

$$I(\sigma) := r_\emptyset \cdot \prod_{v \in V} r_v(\sigma(v)) \cdot \prod_{xy \in E} r_{xy}(\sigma(x), \sigma(y)),$$

and the *partition function* Z_I *of* I is

$$Z_I = \sum_{\sigma\colon V \to [k]} I(\sigma).$$

Thus PCSPs are the special case of RCSPs where the ring R is a polynomial ring over the reals.

5. EXAMPLES

In this section we show how some standard problems can be written as PCSPs. First we show that for every CSP there is a naturally corresponding PCSP; then we illustrate how various problems, including some that are not CSPs, can be expressed as PCSPs. We will turn to algorithms in later sections, but for now bear in mind that a PCSP over a constraint graph G can be solved about as efficiently as a CSP over the same graph.

5.1 Generating function of a simple CSP

Any CSP can naturally be encoded as a PCSP by considering its generating function.

Definition 1 Generating function of an instance. *Given an instance I of CSP on a constraint graph $G = (V, E)$ and variable domain $[k]$, we can define a corresponding instance I^* of PCSP with the same graph and variable domain, and polynomials*

$$\begin{aligned} & & p_\emptyset &= z^{s_\emptyset} \\ (\forall v \in V,\ \forall i \in [k]) & & p_v(i) &= z^{s_v(i)} \\ (\forall xy \in E,\ \forall i,j \in [k]) & & p_{xy}(i,j) &= z^{s_{xy}(i,j)}. \end{aligned}$$

The connection between I and I^* is given by the following simple observation.

Lemma 2. *Let I be an instance of CSP, and let I^* be the corresponding PCSP instance. Then the partition function Z_{I^*} is the generating function* (2) *for the instance I.*

Proof. For any assignment σ, we have

$$\begin{aligned} I^*(\sigma) &= z^{s_\emptyset} \cdot \prod_{v \in V} z^{s_v(i)} \cdot \prod_{xy \in E} z^{s_{xy}(i,j)} \\ &= z^{s_\emptyset + \sum_{v \in V} s_v(i) + \sum_{xy \in E} s_{xy}(i,j)} \\ &= z^{I(\sigma)}. \end{aligned}$$

It therefore follows that the partition function

$$Z_{I^*} = \sum_\sigma I^*(\sigma) = \sum_\sigma z^{I(\sigma)}$$

is the generating function for the original constraint satisfaction problem. □

Similar results are easily seen to hold for generating functions in more than one variable.

5.2 Max Cut and Max Dicut

Max Cut provides a simple illustration of Definition 1 and Lemma 2. Let us first write Max Cut as a CSP (Example 1) and then construct the corresponding PCSP (Example 2).

Example 1 Max Cut CSP. *Given a graph $G = (V, E)$, set $k = 2$ and define a CSP instance I by*

$$\begin{aligned} & s_\emptyset = 0 \\ (\forall v \in V) \quad & s_v(0) = s_v(1) = 0 \\ (\forall xy \in E) \quad & s_{xy}(0,1) = s_{xy}(1,0) = 1 \ , \ s_{xy}(0,0) = s_{xy}(1,1) = 0. \end{aligned}$$

With $\sigma^{-1}(i) = \{v \colon \sigma(v) = i\}$, note that $(V_0, V_1) = (\sigma^{-1}(0), \sigma^{-1}(1))$ is a partition of V, and

$$I(\sigma) = \sum_{\substack{xy \in E \colon \\ \sigma(x)=0,\, \sigma(y)=1}} 1 = e(V_0, V_1) \tag{5}$$

is the size of the cut induced by σ. The corresponding PCSP instance is obtained as in Definition 1.

Example 2 Max Cut PCSP. *Given a graph $G = (V, E)$, set $k = 2$ and define a PCSP instance I by*

$$\begin{aligned} & p_\emptyset = 1 \\ (\forall v \in V) \quad & p_v(0) = p_v(1) = 1 \\ (\forall xy \in E) \quad & p_{xy}(0,1) = p_{xy}(1,0) = z \quad , \quad p_{xy}(0,0) = p_{xy}(1,1) = 1. \end{aligned}$$

(In all such cases, a 1 on the right hand side may be thought of as z^0.)

By Lemma 2, the partition function Z_I is therefore the generating function for cuts:

$$Z_I = \sum 2c_i z^i, \tag{6}$$

where c_i is the number of cuts of size i, and the factor 2 appears because each cut (V_0, V_1) also appears as (V_1, V_0). The size of a maximum cut is the degree of Z_I, and the number of maximum cuts is half the leading coefficient.

Note that the partition function (6) is the partition function of the Ising model with no external field (see below for a definition). Thus we have recovered the familiar fact that, up to a change of variables, the partition function of the Ising model is the generating function for cuts.

We can also encode weighted instances of Max Cut with edge weights $\rho \colon E \to \mathbb{R}$ by modifying the third line of the definition above to

$$p_{xy}(0,1) = p_{xy}(1,0) = z^{\rho(xy)} \qquad \forall xy \in E.$$

Similarly, we can encode weighted Max Dicut (maximum directed cut) by setting

$$\begin{aligned} & p_\emptyset = 1 \\ (\forall v \in V) \quad & p_v(0) = p_v(1) = 1 \\ (\forall xy \in E) \quad & p_{xy}(0,1) = z^{\rho(xy)} \ , \ p_{xy}(1,0) = z^{\rho(yx)} \ , \ p_{xy}(0,0) = p_{xy}(1,1) = 1, \end{aligned}$$

where $\rho(xy)$ denotes the weight of the *directed* edge xy, and we define $\rho(xy) = 0$ if there is no such edge. Max k-Cut is encoded the same way, only with k-valued variables in place of binary ones.

5.3 Graph Bisection

A slight generalization of the previous example allows us to handle graph bisection. This is important because most CSP algorithms (other than dynamic programming) can solve Max Cut, but cannot be applied to Max Bisection because there is no way to force them to generate a balanced cut. A PCSP can do this by using a second variable: z tracks the size of the cut, while w tracks the size of one side.

EXAMPLE 3 CUT GENERATING FUNCTION PCSP. *Given a graph $G = (V, E)$, set $k = 2$ and define a PCSP instance I by*

$$\begin{array}{ll} & p_\emptyset = 1 \\ (\forall v \in V) & p_v(0) = 1 \quad , \quad p_v(1) = w \\ (\forall xy \in E) & p_{xy}(0,1) = p_{xy}(1,0) = z \quad , \quad p_{xy}(0,0) = p_{xy}(1,1) = 1. \end{array}$$

By Lemma 2, the partition function Z_I is a bivariate generating function for cuts:

$$Z_I = \sum c_i w^s z^i, \tag{7}$$

where c_i is the number of cuts of size i with s vertices in the "1" side of the cut. This formulation simultaneously encodes Max Cut (the size of a maximum cut is the largest power of z), Max Bisection (the largest power of z in a monomial with $w^{\lfloor n/2 \rfloor}$), Min Bisection (similarly), and Sparsest Cut (the smallest ratio of the power of z to that of w in any monomial), and counting versions of all of these (simply attending to the corresponding coefficients and adjusting for double-counting as appropriate). Instances with vertex and/or edge weights can be modeled using non-unit powers of w and z respectively. The fact that Max and Min Bisection can be modeled as PCSPs has algorithmic implications.

5.4 The Ising model

The same PCSP also encodes the Ising model.

EXAMPLE 4 ISING CSP MODEL. *The Ising model with edge weights J and external field h on a graph G is defined in terms of its Hamiltonian H. For an assignment $\sigma \colon V \to \{0, 1\}$, we define*

$$H(\sigma) = J \sum_{xy \in E} \delta(\sigma(x), \sigma(y)) + h \sum_{v \in V} \sigma(v).$$

Here $\delta(a, b)$ is the delta function, returning 0 if $a = b$, and 1 otherwise, J is the *interaction strength*, and h is the *external magnetic field*. In analogy with (5), taking $V_i = \sigma^{-1}(i)$, we may rewrite H as

$$H(\sigma) = Je(V_0, V_1) + h|V_1|.$$

Note that H is an instance of (plain) CSP.

The *partition function* of the Ising model at *inverse temperature* β is

$$Z_{\text{Ising}} = \sum_{\sigma} e^{-\beta H(\sigma)} = \sum_{V_0 \uplus V_1 = V} w^{|V_1|} z^{e(V_0,V_1)}, \tag{8}$$

where we have written $w = e^{-\beta h}$ and $z = e^{-\beta J}$, and the last sum is taken over ordered pairs (V_0, V_1) that partition V. With this change of variables, the Ising partition function is given by Example 3.

More generally, the Potts model can be written in a similar way.

5.5 Max Independent Set and Max Clique

Maximum Independent Set (MIS) is easily expressed as a CSP:

Example 5 MIS CSP.

$$\begin{aligned} & s_\emptyset = 0 \\ (\forall v \in V) \quad & s_v(0) = 0\ ,\ s_v(1) = 1 \\ (\forall xy \in E) \quad & s_{xy}(0,0) = s_{xy}(0,1) = s_{xy}(1,0) = 0\ ,\ s_{xy}(1,1) = -2. \end{aligned}$$

Maximum clique cannot be modeled in the same way, because the clique constraint is enforced by non-edges, which are not an element of the model. (We use clique in the sense of a complete subgraph, not the stronger definition as a *maximal* complete subgraph.) Of course a maximum clique in G corresponds to a maximum independent set in its complement graph $\overline{G}$, but for our purposes this can be very different, as we typically parametrize running time in terms of the number of edges, and a Max Clique instance on a sparse graph G with $|E|$ edges becomes an MIS instance on the dense graph $\overline{G}$ with $\binom{n}{2} - |E|$ edges. We discuss this further in Section 13. However, by introducing a second variable as we did for Max Bisection, it is possible to model a Max Clique instance as a PCSP whose constraint graph is the input graph G.

Example 6 Max Clique PCSP.

$$\begin{aligned} & s_\emptyset = 1 \\ (\forall v \in V) \quad & s_v(0) = 1\ ,\ s_v(1) = w \\ (\forall xy \in E) \quad & s_{xy}(0,0) = s_{xy}(0,1) = s_{xy}(1,0) = 1\ ,\ s_{xy}(1,1) = z. \end{aligned}$$

An assignment σ has score $I(\sigma) = w^{|V_1|} z^{e(G|_{V_1})}$, the power of w counting the number of vertices in the chosen set $V_1 = \sigma^{-1}(1)$, and the power of z counting the number of edges induced by that set. In the partition function, k-cliques correspond to terms of the form $w^k z^{\binom{k}{2}}$. Of course, independent sets of cardinality k correspond to terms with $w^k z^0$: the PCSP simultaneously counts maximum cliques and maximum independent sets (among other things).

The same approach also counts maximum-weight cliques and independent sets, in a graph with vertex and/or edge weights ρ, if we introduce a third formal variable w' and set the scores as before except for $s_v(1) = w w'^{\rho(v)}$ and $s_{xy}(1,1) = z w'^{\rho(x,y)}$. Cliques are represented in terms $w^k z^{\binom{k}{2}} w'^{\rho}$, and maximum-weight cliques in such terms with the largest power of w'.

5.6 Judicious partitions

Similar techniques may be applied to various *judicious partitioning* problems [Bollobás and Scott 1999; Scott 2005], such as finding a cut of a graph which minimizes $\max\{e(V_0), e(V_1)\}$.

Example 7 Judicious bipartition PCSP.

$$\begin{aligned} & s_\emptyset = 1 \\ (\forall v \in V) \quad & s_v(0) = s_v(1) = 1 \\ (\forall xy \in E) \quad & s_{xy}(0,1) = s_{xy}(1,0) = 1 \ , \ s_{xy}(0,0) = z_1 \ , \ s_{xy}(1,1) = z_2. \end{aligned}$$

A simple calculation shows that $I(\sigma) = z_1^{e(V_0)} z_2^{e(V_1)}$, where $V_i = \sigma^{-1}(i)$. Thus the problem of minimizing $\max\{e(V_0), e(V_1)\}$ (or, for instance, the problem of maximizing $\min\{e(V_0), e(V_1)\}$) can be solved by examining the partition function.

We might further ask for a *bisection* that minimizes $\max\{e(V_0), e(V_1)\}$, and this is easily achieved by introducing a third variable: we set $s_v(0) = w$ and examine only terms of the partition function that have degree $\lfloor |V|/2 \rfloor$ in w.

5.7 Simultaneous assignments

We can think of the foregoing example, of finding a balanced bisection minimizing $\max\{e(V_0), e(V_1)\}$, as a case of having more than one CSP on a single set of variables. In other examples, we might have two instances of Max Sat that we wish to treat as a bi-criterion optimization problem; or a Max Sat instance to optimize subject to satisfaction of a Sat instance; or we might wish to find a partition of a vertex set yielding a large cut for two different graphs on the same vertices. All such problems are now straightforward: use a variable z to encode one problem and a variable w to encode the second. (Thus, the initial edge scores are of the form $z^{\text{firstscore}} w^{\text{secondscore}}$.)

5.8 H-coloring

The H-coloring of a graph G has received much recent attention, including the book [Hell and Nešetřil 2004], the application to soft constraints in [Bulatov and Grohe 2005], and the application to asymmetric hard constraints in [Dyer et al. 2006]. H-coloring can be encoded as a PCSP by giving every edge of G the same (symmetric) score function. However, PCSP can also give a different (and asymmetric) score function to each edge, and indeed this is an essential feature as it leaves the class closed under various reductions.

6. SPLIT-AND-LIST, MATRIX-MULTIPLICATION ALGORITHM

The algorithmic approach introduced by Williams [Williams 2004], which he calls "split and list", is characterized by its use of fast matrix multiplication. For binary 2-CSPs it runs in time $O^\star\big(2^{\omega n/3}\big)$, where ω is the "matrix multiplication exponent" such that two $n \times n$ matrices can be multiplied together in time $O(n^\omega)$. It also requires exponential space, $O(2^{2n/3})$. The best known bound on the matrix multiplication constant is $\omega < 2.37\ldots$, from the celebrated algorithm of Coppersmith and Winograd [Coppersmith and Winograd 1990], which works over arbitrary rings.

Koivisto [Koivisto 2006b] extended Williams' algorithm from CSP (with a max–sum formulation) to counting-CSP (with sum–product formulation), and Koivisto's version transfers immediately to the PCSP and RCSP context. While the sum–product formulation is common in statistical physics, Koivisto's use of it appears to be one of the few besides the present work in the field of exact algorithms. For Williams' algorithm, the sum–product formulation is not only more general but also simpler.

6.1 Algorithm

We follow the algorithm of Koivisto [Koivisto 2006b]. Arbitrarily partition the variables V into 3 equal-sized sets, A, B and C. Explicitly list every possible assignment $\alpha\colon A \mapsto [k]$ of variables in A and similarly every assignment β for B and γ for C. Explicitly construct M_{AA}, a $k^{|A|} \times k^{|A|}$ diagonal matrix with elements

$$M_{AA}(\alpha,\alpha) := \prod_{v\in A} p_v(\alpha(v)) \cdot \prod_{xy\in E\cap\{A\times A\}} p_{xy}(\alpha(x),\alpha(y)),$$

and likewise M_{BB} and M_{CC}. Also explicitly construct M_{AB}, a $k^{|A|} \times k^{|B|}$ matrix with elements

$$M_{AB}(\alpha,\beta) := \prod_{xy\in E\cap\{A\times B\}} p_{xy}(\alpha(x),\beta(y)),$$

and likewise M_{BC} and M_{CA}. Finally, explicitly calculate

$$M = p_\emptyset M_{AA} M_{AB} M_{BB} M_{BC} M_{CC} M_{CA},$$

the multiplication by $p_\emptyset$ being a scalar product and the others matrix products. Triples (α,β,γ) are in one to one correspondence with assignments $\sigma\colon V \mapsto [k]$, so the terms of the trace of M,

$$\operatorname{tr}(M) = \sum_\alpha \sum_\beta \sum_\gamma p_\emptyset M_{AA}(\alpha,\alpha) M_{AB}(\alpha,\beta) M_{BB}(\beta,\beta) M_{BC}(\beta,\gamma) M_{CC}(\gamma,\gamma) M_{CA}(\gamma,\alpha),$$

are in one to one correspondence with the terms of $I(\sigma)$, merely distinguishing whether each monadic score is associated with A, B, or C, and each dyadic score with $A \times A$, $A \times B$, etc. Thus, the partition function is the trace,

$$Z_I = \operatorname{tr}(M).$$

6.2 Complexity

Given that two $n \times n$ matrices over an arbitrary ring can be multiplied using $O(n^\omega)$ ring operations, we have established the following result.

THEOREM 3. *Let R be a ring and let G be a graph with n vertices. Let I be any RCSP over R, with constraint graph G and domain $[k]$. Then the algorithm above calculates the partition function Z_I with $O\left(k^{\omega n/3}\right)$ ring operations.*

For PCSPs, as we discuss in Section 10, under modest conditions each ring operation takes time $O^\star(1)$, that is, time polynomial in the input size. In this case, the algorithm will run in time and space $O^\star\left(k^{\omega n/3}\right)$.

6.3 A remark

This version of the algorithm is simpler than Williams' because matrix multiplication provides exactly the sum-of-products that RCSP calls for, not the maximum-of-sums needed for the usual 2-CSP. To get the latter, Williams combines the monadic scores with the dyadic ones (this would correspond to incorporating M_{AA} into M_{AB}); "guesses" an optimal score s and how it is partitioned as $s = s_{AB}+s_{BC}+s_{CA}$ among $A \times B$, $B \times C$, and $C \times A$; defines zero-one matrices so that $M_{AB}(\alpha, \beta) = 1$ if the assignments α and β produce the desired value s_{AB}, and 0 otherwise; and multiplies these binary matrices to find (and count) triangles, which correspond to assignments (α, β, γ) yielding $s_{AB} + s_{BC} + s_{CA}$. Williams' original algorithm must iterate over all guesses, of which there are $\Theta(m^3)$ even for 0–1 score functions, while no such guessing is involved in the sum–product version.

7. A REDUCTIVE ALGORITHM

The preponderance of algorithms for CSPs such as Max 2-Sat work by reduction: they reduce an input instance to one or more "smaller" instances in such a way that the solutions to the latter, found recursively, yield a solution to the former. The fastest polynomial-space algorithm for general Max 2-CSP is the reductive algorithm of [Scott and Sorkin 2007], running in time $O^\star\big(k^{19m/100}\big)$ for an instance with m 2-variable constraints and domain $[k]$. While we will not describe that paper's "Algorithm B" in detail, if an instance I has (a constraint graph with) a vertex of degree 0, 1, or 2, a Type 0, 1, or 2 reduction (respectively) replaces I with an equivalent, smaller instance I'. Otherwise, a Type 3 reduction replaces I with k smaller instances $I_0, \ldots, I_{k-1}$, the largest of whose solutions is the solution to I. If a Type 3 reduction results in a disconnected graph, the components are solved separately and summed.

The algorithm extends to PCSP (and indeed RCSP), the key point being that the Type 0, 1, 2, and 3 CSP reductions have PCSP (and RCSP) analogues. By working with polynomials rather than real numbers we are able to carry substantially more information: for instance, the old CSP reductions correspond to the new PCSP reductions with the polynomials truncated to their leading terms.

7.1 Complexity

We exhibit the extended reductions below, where it will also be evident that each can be performed using $O(k^3)$ ring operations. The extension of [Scott and Sorkin 2007]'s Algorithm B to RCSPs can then be seen to satisfy the following.

THEOREM 4. *Let R be a ring and let G be a graph with n vertices and m edges. Let I be any RCSP over R, with constraint graph G and domain $[k]$. Then the extended Algorithm B calculates the partition function Z_I in polynomial space and with $O^\star\big(k^{19m/100}\big)$ ring operations.*

7.2 The reductions

A Type 0 reduction expresses the partition function of an RCSP as a product of partition functions of two smaller instances (or in an important special case, a single such instance). Type 1 and 2 reductions each equate the generating function of an instance to that of an instance with one vertex less. Finally, a Type 3 reduction

produces k instances, whose partition functions sum to the partition function of the original instance. In each case, once the reduction is written down, verifying its validity is just a matter of checking a straightforward identity.

A word of intuition, deriving from the earlier CSP reductions, may be helpful (though some readers may prefer to go straight to the equations). A CSP 2-reduction was performed on a vertex v of degree 2, with neighbors u and w. The key observation was that in a maximization problem, the optimal assignment $\sigma(v)$ is a function of the assignments $\sigma(u)$ and $\sigma(w)$. That is, given any assignments $\sigma(u) = i$ and $\sigma(w) = j$, the optimal total $\tilde{s}_{uw}(i,j)$ of the scores of the vertex v, the edges uv and vw, and the edge uw (if present, and otherwise taken to be the 0 score function) is

$$\tilde{s}_{uw}(i,j) = \max_l \big\{ s_{uw}(i,j) + s_{uv}(i,l) + s_v(l) + s_{vw}(l,j) \big\}. \tag{9}$$

By deleting v from the CSP instance and replacing the original score s_{uw} with the score $\tilde{s}_{uw}$, we obtain a smaller instance with the same maximum value. A 2-reduction in the RCSP context works the same way, except that the max-of-sums of (9) is replaced by the sum-of-products of (11).

Type 0 Reduction. Suppose I is an RCSP instance whose constraint graph G is disconnected. Let $V = V_1 \cup V_2$ be a nontrivial partition such that $e(V_1, V_2) = 0$. Let I_1 and I_2 be the subinstances obtained by restriction to V_1 and V_2, except that we define the nullary scores by

$$p_\emptyset^{(I_1)} = p_\emptyset^{(I)}\ ,\ p_\emptyset^{(I_2)} = 1.$$

A straightforward calculation shows that, for any assignment $\sigma\colon V \to [k]$, we have

$$\begin{aligned} I(\sigma) &= p_\emptyset \cdot \prod_{v\in V} p_v(\sigma(v)) \cdot \prod_{xy\in E} p_{xy}(\sigma(x),\sigma(y)) \\ &= \left(p_\emptyset \cdot \prod_{v\in V_1} p_v(\sigma(v)) \cdot \prod_{xy\in E(G[V_1])} p_{xy}(\sigma(x),\sigma(y)) \right) \left(1 \cdot \prod_{v\in V_2} p_v(\sigma(v)) \cdot \prod_{xy\in E(G[V_2])} p_{xy}(\sigma(x),\sigma(y)) \right) \\ &= I_1(\sigma_1) I_2(\sigma_2), \end{aligned}$$

where σ_i denotes the restriction of σ to V_i. It follows easily that

$$Z_I = Z_{I_1} Z_{I_2}.$$

Thus in order to calculate Z_I it suffices to calculate Z_{I_1} and Z_{I_2}.

When one component of G is an isolated vertex v, with $V' = V \setminus v$, one term becomes trivial: $Z_I = \big(p_\emptyset \cdot \sum_{i\in[k]} p_v(i)\big) \cdot Z_{I_2}$. So in this case

$$Z_I = Z_{\widetilde{I}},$$

where $\widetilde{I}$ is the instance obtained from I by deleting v, defining

$$p_\emptyset(\widetilde{I}) = p_\emptyset \cdot \sum_{i\in[k]} p_v(i)$$

(a trivial calculation), and leaving all other scores unchanged. This reduces the computation of the partition function of I to that of an instance one vertex smaller.

Type 1 Reduction. Suppose that I is an instance with constraint graph G, and $v \in V$ has degree 1. Let w be the neighbor of v. We shall replace I by an "equivalent" instance $\widetilde{I}$ (one with the same partition function) with constraint graph $G \setminus v$.

We define the instance $\widetilde{I}$ by giving w vertex scores

$$\widetilde{p}_w(i) = p_w(i) \sum_{j=0}^{k-1} (p_{wv}(i,j) \cdot p_v(j)). \tag{10}$$

All other scores remain unchanged except for $p_v(*)$ and $p_{vw}(*,*)$, which are deleted along with v.

To show that $Z_I = Z_{\widetilde{I}}$, let $\sigma \colon V \setminus v \to [k]$ be any assignment and, for $j \in [k]$, extend σ to $\sigma^j : V \to [k]$ defined by $\sigma^j(v) = j$ and $\sigma^j|_{V \setminus v} = \sigma$. Using (10), we have

$$\begin{aligned}
\widetilde{I}(\sigma) &= \widetilde{p}_\emptyset \cdot \prod_{x \in V \setminus v} \widetilde{p}_x(\sigma(x)) \cdot \prod_{xy \in E(G \setminus v)} \widetilde{p}_{xy}(\sigma(x), \sigma(y)) \\
&= \left(p_\emptyset \cdot \prod_{x \in V \setminus \{v,w\}} p_x(\sigma(x)) \cdot \prod_{xy \in E(G \setminus v)} p_{xy}(\sigma(x), \sigma(y)) \right) \cdot \left(p_w(\sigma(w)) \cdot \sum_{j=0}^{k-1} p_{wv}(\sigma(w), j) \cdot p_v(j) \right) \\
&= \sum_{j=0}^{k-1} p_\emptyset \cdot \prod_{v \in V} p_v(\sigma^j(v)) \cdot \prod_{xy \in E} p_{xy}(\sigma^j(x), \sigma^j(y)) \\
&= \sum_{j=0}^{k-1} I(\sigma^j)
\end{aligned}$$

and so

$$\begin{aligned}
Z_I &= \sum_{\sigma : V \to [k]} I(\sigma) \\
&= \sum_{\sigma : V \setminus v \to [k]} \sum_{j=0}^{k-1} I(\sigma^j) \\
&= \sum_{\sigma : V \setminus v \to [k]} \widetilde{I}(\sigma) \\
&= Z_{\widetilde{I}}.
\end{aligned}$$

Type 2 Reduction. Suppose that I is an instance with constraint graph G, and $v \in V$ has degree 2. Let u and w be the neighbors of v in G. We define an instance $\widetilde{I}$ with constraint graph $\widetilde{G}$, which will have fewer vertices and edges than G. $\widetilde{G}$ is obtained from G by deleting v and adding an edge uw (if the edge is not already present). By analogy with the plain CSP 2-reduction from [Scott and Sorkin 2007] and recapitulated above around (9), we define $\widetilde{I}$ by setting, for $i, j \in [k]$,

$$\widetilde{p}_{uw}(i,j) = p_{uw}(i,j) \cdot \sum_{l=0}^{k-1} p_{uv}(i,l) p_v(l) p_{vw}(l,j), \tag{11}$$

where we take $p_{uw}(i,j) = 1$ if edge uw was not previously present. All other scores remain unchanged except for $p_v(*)$, $p_{vu}(*,*)$ and $p_{vw}(*,*)$, which are deleted.

To show that $Z_I = Z_{\widetilde{I}}$, let $\sigma\colon V \setminus v \to [k]$ be any assignment. As before, we write σ^l for the assignment with $\sigma^l|_{V\setminus v} \equiv \sigma$ and $\sigma^l(v) = l$. Then

$$\begin{aligned}
\widetilde{I}(\sigma) &= \widetilde{p}_\emptyset \cdot \prod_{x\in V\setminus v} \widetilde{p}_x(\sigma(x)) \cdot \prod_{xy\in E(\widetilde{G})} \widetilde{p}_{xy}(\sigma(x),\sigma(y)) \\
&= p_\emptyset \prod_{x\in V\setminus v} p_x(\sigma(x)) \cdot \prod_{xy\in E(G\setminus v)} p_{xy}(\sigma(x),\sigma(y)) \cdot \sum_{l=0}^{k-1} p_{uv}(\sigma(u),l)p_v(l)p_{vw}(l,\sigma(w)) \\
&= \sum_{l=0}^{k-1} p_\emptyset \prod_{x\in V\setminus v} p_x(\sigma^l(x)) \cdot \prod_{xy\in E(G\setminus v)} p_{xy}(\sigma^l(x),\sigma^l(y)) \\
&= \sum_{l=0}^{k-1} I(\sigma^l).
\end{aligned}$$

As with Type 1 reductions, this implies that $Z_I = Z_{\widetilde{I}}$.

Type 3 Reduction. Suppose that I is an instance with constraint graph G, and $v \in V$ has degree 3 *or more*. Let U be the set of neighbors of v in G. We define k instances $\widetilde{I}_0, \ldots, \widetilde{I}_{k-1}$ each with constraint graph $\widetilde{G} = G \setminus v$. For $i \in [k]$, the ith instance $\widetilde{I}_i$ corresponds to the set of assignments where we take $\sigma(v) = i$.

We define nullary scores for $\widetilde{I}_i$ by setting

$$\widetilde{p}^{(i)}_\emptyset = p_\emptyset \cdot p_v(i)$$

and, for each neighbor $u \in U$ of v, and each $j \in [k]$, vertex scores

$$(\widetilde{p}^{(i)})_u(j) = p_u(j) \cdot p_{uv}(j,i).$$

All other scores remain unchanged from I except for $p_v(*)$ and $p_{v*}(*,*)$, which are deleted along with v.

For any assignment $\sigma\colon V \setminus v \to [k]$, and $i \in [k]$ we write (as usual) $\sigma^i\colon V \to [k]$ for the assignment with $\sigma^i|_{V\setminus v} \equiv \sigma$ and $\sigma^i(v) = i$. Then, writing $W = V \setminus (v \cup \Gamma(v))$,

$$\begin{aligned}
\widetilde{I}_i(\sigma) &= \widetilde{p}_\emptyset \cdot \prod_{x\in V\setminus v} \widetilde{p}_x(\sigma(x)) \cdot \prod_{xy\in E(G\setminus v)} \widetilde{p}_{xy}(\sigma^i(x),\sigma^i(y)) \\
&= \Big(p_\emptyset p_v(i)\Big) \cdot \Big(\prod_{x\in W} p_x(\sigma(x)) \cdot \prod_{u\in U} \big[p_u(\sigma(u))p_{uv}(\sigma(u),i)\big]\Big) \cdot \Big(\prod_{xy\in E(G\setminus v)} p_{xy}(\sigma(x),\sigma(y))\Big) \\
&= p_\emptyset \prod_{x\in V} p_x(\sigma^i(x)) \cdot \prod_{xy\in E} p_{xy}(\sigma^i(x),\sigma^i(y)) \\
&= I(\sigma^i).
\end{aligned}$$

Then

$$\begin{aligned} Z_I &= \sum_{\sigma: V \to [k]} I(\sigma) \\ &= \sum_{\sigma: V \setminus v \to [k]} \sum_{i=0}^{k-1} I(\sigma^i) \\ &= \sum_{\sigma: V \setminus v \to [k]} \sum_{i=0}^{k-1} \widetilde{I}_i(\sigma) \\ &= \sum_{i=0}^{k-1} \sum_{\sigma: V \setminus v \to [k]} \widetilde{I}_i(\sigma) \\ &= \sum_{i=0}^{k-1} Z_{\widetilde{I}_i}. \end{aligned}$$

Thus the partition function for I is the sum of the partition functions for the $\widetilde{I}_i$.

We can also write down another reduction, generalizing the Type 0 reduction above, although we will not employ it here.

Cut Reduction. Suppose that I is an instance with constraint graph G, and let $V_0 \subset V$ be a vertex cut in G. Let $G_1, \ldots, G_r$ be the components of $G \setminus V_0$. We can calculate Z_I by dividing assignments into classes depending on their restriction to V_0. Indeed, for each assignment $\sigma_0 \colon V_0 \to [k]$, let us define

$$I_0 = p_\emptyset \cdot \prod_{v \in V_0} p_v(\sigma_0(v)) \prod_{xy \in E(G[V_0])} p_{xy}(\sigma_0(x), \sigma_0(y)),$$

and r instances $I_1^{\sigma_0}, \ldots, I_r^{\sigma_0}$ where $I_t^{\sigma_0}$ has score polynomials

$$\begin{aligned} & & \widetilde{p}_\emptyset &= 1 \\ \forall v \notin V_0 & & \widetilde{p}_v(i) &= p_v(i) \cdot \prod_{w \in V_0} p_{wv}(\sigma_0(w), i) \\ \forall vw \in E[G \setminus V_0] & & \widetilde{p}_{vw}(i,j) &= p_{vw}(i,j), \end{aligned}$$

where we take $p_{wv}(i,j) = 1$ if $wv \notin E$.

Then for any $\sigma \colon V \to [k]$, we have

$$I(\sigma) = I_0(\sigma|V_0) \cdot \prod_{i=1}^{r} I_i^{\sigma|V_0}(\sigma|V_i)$$

and so

$$Z_I = \sum_{\sigma_0: V_0 \to [k]} I_0(\sigma_0) \prod_{i=1}^{r} Z_{I_i^{\sigma_0}}.$$

Thus the partition function Z_I is the sum of $|V_0|^k$ products: if $|V_0|$ is small, this provides an effective reduction to smaller cases. Note that this is what we have

done in the Type 2 reduction above, where we also use the fact that, for a cutset of size two, the partition functions for the subinstance consisting of the single vertex v can be encoded using constraints on its neighbors. Unfortunately, this is not in general possible for larger cutsets.

8. AN ALGORITHM FOR SPARSE SEMI-RANDOM INSTANCES

An algorithm closely related to the CSP Algorithm B referenced above, and using the same reductions, solves sparse boolean CSP instances in expected linear time [Scott and Sorkin 2006b]. Specifically, for $\lambda = \lambda(n) \geq 0$ and a random graph $G(n, c/n)$ with $c \leq 1 + \lambda n^{-1/3}$ (below the giant-component threshold or in its so-called scaling window), a boolean CSP instance I with constraint graph G is solved in expected time $O(n)\exp(1 + \lambda^3)$.

8.1 Complexity

Because the reductions extend to RCSP, it follows the algorithm extends immediately to boolean RCSP instances on such a constraint graph.

THEOREM 5. *Let $\lambda = \lambda(n) \geq 0$. For $c \leq 1 + \lambda n^{-1/3}$, let $G \in \mathcal{G}(n, c/n)$ be a random graph, and let I be any boolean RCSP over any ring R with constraint graph G. Then the algorithm above calculates the partition function Z_I with an expected $O(n)\exp(1 + \lambda^3)$ ring operations.*

9. DYNAMIC PROGRAMMING FOR GRAPHS OF SMALL TREEWIDTH

Roughly, a *tree decomposition* of a graph $G = (V, E)$ consists of a tree T each of whose vertices is associated with a "bag" $B \subseteq V$, such that every vertex of G appears in some bag, every edge of G has both endpoints in some bag, and the set of bags containing any given vertex v of G forms a subtree of T. The "width" of a tree decomposition is one less than the cardinality of the largest bag, and the treewidth of G is the minimum width of any tree decomposition. For our purposes we will assume that a tree decomposition is given, though for graphs of constant-bounded treewidth, a minimum-treewidth decomposition can be found in linear time [Robertson and Seymour 1995; Reed 1992; Bodlaender 1996].

Efficient algorithms for various sorts of constraint satisfaction and related problems on graphs of small treewidth have been studied since at least the mid-1980s, with systematic approaches dating back at least to [Dechter and Pearl 1987; 1989; Arnborg and Proskurowski 1989]. A special issue of Discrete Applied Mathematics was devoted to this and related topics in 1994 [Arnborg et al. 1994], and the field remains an extremely active area of research.

A recent series of results on treewidth is Monien and Preis's proof that every cubic graph with m edges has bisection width at most $(1/6 + o(1))m$ [Monien and Preis 2001], Fomin and Høie's use of this to show that the pathwidth of a cubic graph is bounded by $(1/6 + o(1))m$ [Fomin and Høie 2006], and the use of this to prove that any graph has treewidth at most $(13/75 + o(1))m$ [Scott and Sorkin 2007; Kneis et al. 2009].

Focusing on two problems of particular interest to us, as a key part of an approximation result of Jansen, Karpinski, Lingas and Seidel [Jansen et al. 2005], it was shown that given a graph G and a tree decomposition T of width $b - 1$, an exact

maximum (or minimum) bisection of G can be found in time *and space* $O^\star\big(2^b\big)$, using dynamic programming on the tree decomposition. To find a maximum *clique* in time $O^\star\big(2^b\big)$ is even easier, and may be done in polynomial space, using the well-known fact that any clique of G is necessarily contained in a single bag of T.

Here, we generalize these results for bisection and clique to the entire class RCSP. While the extension is reasonably straightforward, the details are confusing enough that it is valuable to have the general result available in this neatly packaged form, and not have to apply the approach (or wonder if it can be applied) to a particular problem.

9.1 Complexity

THEOREM 6. *Let G be a graph with n vertices, m edges, and a tree decomposition T of width $b-1$. Let I be any RCSP instance over G. Given I and T, the partition function Z_I can be calculated in space $O^\star(()\, k^{b-1})$ and with $O^\star\big(k^b\big)$ ring operations.*

We sketch the proof before detailing it. We express an assignment score as a product of *bag scores*, f_B for a bag B. In computing the partition function, we reduce on any leaf bag B_1 of T and its parent B_2, in a way closely related to the Type 1 reductions of Section 7: variables appearing *only* in B_1 can be "integrated out" to yield a function f'_{B_1} on a smaller set of variables. By the definition of a tree decomposition, that smaller set is a subset of the variables of B_2, so we can absorb f'_{B_1} into f_{B_2} by defining $f'_{B_2} = f_{B_2} \cdot f'_{B_1}$. Deleting B_1 from T, and forgetting f_{B_1}, completes the reduction. We now present this more formally.

PROOF. We first show how to express the score of an assignment as a product of bag scores. Let $\mathcal{B}$ be the collection of all bags. While a vertex is typically found in several bags, associate each vertex $v \in V(G)$ with just one bag that contains it; similarly, associate each edge $e \in E(G)$ with some bag containing both its endpoints. For a bag B, let $V(B)$ be the set of associated vertices and $E(B)$ the set of associated edges, so that $\uplus_{B\in\mathcal{B}} V(B) = V(G)$ and $\uplus_{B\in\mathcal{B}} E(B) = E(G)$. Define each bag score as the product of its associated vertex and edge scores:

$$f_B(\sigma|_B) = \prod_{v\in V(B)} p_v(\sigma(v)) \cdot \prod_{xy\in E(B)} p_{xy}(\sigma(x),\sigma(y)). \tag{12}$$

(Note that for an edge $xy \in E(B)$, the endpoints x and y need not be in $V(B)$, but they must be in B itself, and thus $f_B(\sigma|_B)$ is well defined, depending only on assignments of vertices in B.) It is clear that for the original PCSP instance I,

$$\begin{aligned} Z_I &:= \sum_{\sigma\colon V\to[k]} p_\emptyset \cdot \prod_{v\in V} p_v(\sigma(v)) \cdot \prod_{xy\in E} p_{xy}(\sigma(x),\sigma(y)) \\ &= \sum_{\sigma\colon V\to[k]} p_\emptyset \cdot \prod_{B\in\mathcal{B}} f_B(\sigma|_B). \end{aligned} \tag{13}$$

We begin the algorithm by explicitly computing each bag score. Each such computation takes $O(b^2k^{b-1})$ ring operations (for a bag of size $b-1$, each of the k^{b-1} assignments requires evaluating $O(b)$ vertex scores and $O(k^2)$ edge scores), and the resulting bag score function is explicitly represented as a table of size at most k^{b-1}.

We now show a reduction that will preserve the form of (13), but with one bag fewer. After reduction, the bag scores themselves will no longer depend simply on vertex and edge scores as in (12), which is why we work with (13) instead.

Let B_1 be a leaf bag and B_2 its parent. Let $V' = B_1 \setminus B_2$. By the nature of a tree decomposition, V' is the set of vertices found exclusively in B_1. Let $V'' = V \setminus V'$. Correspondingly, we will use σ' and σ'' for assignments from these sets to $[k]$, and (σ', σ'') for a complete assignment from V to $[k]$. In the following rewriting of (13), lines (15) and (16) are implicit definitions of the functions f'_{B_1} and f'_{B_2}.

$$Z_I = \sum_{\sigma\colon V\to[k]} p_\emptyset \cdot \left(\prod_{B\in\mathcal{B}\setminus\{B_1,B_2\}} f_B(\sigma''|_B)\right) \cdot f_{B_2}(\sigma|_{B_2}) \cdot f_{B_1}(\sigma|_{B_1}) \tag{14}$$

$$= \sum_{\sigma''\colon V''\to[k]} p_\emptyset \cdot \left(\prod_{B\in\mathcal{B}\setminus\{B_1,B_2\}} f_B(\sigma''|_B)\right) \cdot f_{B_2}(\sigma''|_{B_2}) \cdot \sum_{\sigma'\colon V'\to[k]} f_{B_1}((\sigma'',\sigma')|_{B_1})$$

$$= \sum_{\sigma''\colon V''\to[k]} p_\emptyset \cdot \left(\prod_{B\in\mathcal{B}\setminus\{B_1,B_2\}} f_B(\sigma''|_B)\right) \cdot f_{B_2}(\sigma''|_{B_2}) \cdot f'_{B_1}(\sigma''|_{B_1\cap B_2}) \tag{15}$$

$$= \sum_{\sigma''\colon V''\to[k]} p_\emptyset \cdot \left(\prod_{B\in\mathcal{B}\setminus\{B_1,B_2\}} f_B(\sigma''|_B)\right) \cdot f'_{B_2}(\sigma''|_{B_2}). \tag{16}$$

This reduces the number of bags by one, as desired.

Note that the definition implicit in (15), that for any $\sigma''\colon V'' \to [k]$,

$$f'_{B_1}(\sigma''|_{B_1\cap B_2}) := \sum_{\sigma'\colon V'\to[k]} f_{B_1}((\sigma'',\sigma')|_{B_1}), \tag{15$'$}$$

is well defined because the right-hand side depends only on values of (σ'', σ') on the set $B_1 \cap B_2$. Computing f'_{B_1} adds nothing to the algorithm's space requirement, as the table for f'_{B_1} is smaller than and replaces that for f_{B_1}. The time required is that for $O(k^{|B_1|})$ ring operations, since for each of the $k^{|B_1\cap B_2|}$ assignments $(\sigma'', \sigma')|_{B_1\cap B_2}$, the right-hand side takes a summation over the $k^{|B_1\setminus B_2|}$ assignments σ' of V', with each function evaluation done by table lookup. (We remark that the summation over assignments in (15′) is why the bag scores cease to depend simply on vertex and edge scores as in (12).)

Likewise, the definition implicit in (16), that for any $\sigma''\colon V'' \to [k]$,

$$f'_{B_2}(\sigma''|_{B_2}) := f_{B_2}(\sigma''|_{B_2}) \cdot f'_{B_1}((\sigma'')|_{B_1\cap B_2}), \tag{16$'$}$$

is well defined because the right-hand side depends only on values of (σ'', σ') on the set B_2. Computing f'_{B_2} again adds nothing to the algorithm's space requirement, as the table is the same size as that for f'_{B_2}, and takes $O(k^{b-1})$ ring operations.

In going from (14) to (16) we reduced the decomposition tree size by one, using $O(k^{b-1})$ ring operations. In the process we did not increase the size of any bag (specifically, B_2 did not acquire any new variables from the deleted B_1), and thus this same time bound applies throughout the algorithm. We conclude that altogether the algorithm takes space $O^\star\big(k^{b-1}\big)$ and uses $O^\star\big(k^{b-1}\big)$ ring operations. □

10. CONTROLLING THE RING-OPERATION COMPLEXITY

In this section, we restrict to Polynomial CSPs, and consider the complexity of performing the ring operations in our algorithms. In general, PCSP instances can be intractable. For example, consider a boolean ($[k] = \{0, 1\}$) PCSP instance I where all dyadic scores are 1, and the monadic scores are $p_v(0) = 1$, $p_v(1) = z^{\rho_v}$, with mutually incommensurable real values ρ_v. Then each of the 2^n variable assignments corresponds to a different monomial in the partition function $Z(I)$. Since the output is of size $\Omega(2^n)$, there is no hope of a time- or space-efficient algorithm.

However, some modest restrictions suffice to control the size of the elements of the polynomial ring with which we are working (including the output itself), and thus the time and space efficiency of the various algorithms.

10.1 Polynomially bounded instances

The simplest well-behaved case is when an instance is over a single variable, and all score polynomials are of low degree, with small coefficients. This is easily formalized and generalized.

DEFINITION 7 POLYNOMIALLY BOUNDED. *A family $\mathcal{F}$ of instances of PCSP is* polynomially bounded *if for an instance $I \in \mathcal{F}$ having n variables and m clauses, with respect to $m + n$:*

—*the number of formal variables is $O(1)$,*
—*all powers are integers with absolute value $O^\star(1)$, and*
—*coefficients are integers of length $O^\star(1)$, or sums and products of coefficients are deemed to be elementary operations.*

For convenience we will speak of "a polynomially bounded instance".

LEMMA 8. *In any of the four algorithms described, over a fixed domain $[k]$, solving a polynomially bounded instance I with n variables and m clauses, each ring operation can be performed in time and space $O^\star(1)$.*

PROOF. Say that there are c variables, each with maximum degree Δ. As a sum of products, the partition function itself has maximum degree (in each variable) at most $(m + n + 1)\Delta$. This is also true of any polynomial that arises in the course of the computation. This can be verified directly by looking at the details of each algorithm. Alternatively, note that the algorithms compute "obliviously" on the polynomials (the operations performed depend on the constraint graph but not on the polynomials), the algorithms never subtract, and thus if on any input an algorithm ever produced terms of higher degree than the degree bound on the partition function, it would certainly do so when each input score polynomial is $1 + z + \cdots + z^\Delta$ (or the equivalent multivariate polynomial), and (in the absence of any negative coefficients) such terms would survive to appear in the partition function, a contradiction.

Since each intermediate polynomial has maximum degree at most $(m+n+1)\Delta$, it has at most $((m + n + 1)\Delta)^c = O^\star(1)$ terms, and two such polynomials can be multiplied with $O^\star(1)$ integer operations.

If we wish to further break down the time as a function of the lengths of the integer coefficients, each arises as a sum of at most k^{m+n+1} products of at most $m+n+1$

input coefficients, and therefore has length at most $(m+n+1)\log k \cdot (m+n+1) = O^\star(1)$ times that of the longest input coefficient. Thus, adding and multiplying coefficients can also be done in time $O^\star(1)$. □

10.2 Pruned instances

An alternative for a PCSP in a single variable z and with positive coefficients is to allow arbitrary (real) powers in the score polynomials, but to avoid the blowup in our motivating example by demanding not the full partition function but just its leading term (the one with the highest power of z). We now generalize this approach to multivariate polynomials with one distinguished variable z.

DEFINITION 9 z-PRUNABLE. *An instance of PCSP with n variables and m clauses (more precisely a family of instances) is* z-prunable *if, with respect to $m+n$:*

—*the number of formal variables is $O(1)$,*

—*z may have arbitrary real powers,*

—*the remaining variables $w_1, w_2, \ldots$ have integer powers with absolute value no more than $O^\star(1)$, and*

—*coefficients are nonnegative integers of length $O^\star(1)$, or coefficients are nonnegative reals and sums and products of coefficients are deemed to be elementary operations,*

where the $O(\cdot)$ notation is with respect to $m+n$.

Note that a PCSP arising as the generating function of a CSP (see Section 5.1) is always z-prunable: there is just a single formal variable z, and all coefficients are 1.

Pruned polynomial. Given a polynomial p in one variable z, we define the *pruned polynomial* $(p)_z$ to be the polynomial obtained by removing all but the leading term. If p is a polynomial in variables $z, w_1, w_2, \ldots$, we obtain $(p)_z$ by throwing away all terms T such that there is a term of the form cTz^i, where $c > 0$ is real, and $i > 0$. For instance,

$$(2z^2 + 3z + 700 + zw_1 + z^2w_1 + zw_2 + z^{10}w_1w_2)_z = 2z^2 + z^2w_1 + zw_2 + z^{10}w_1w_2.$$

Pruning a partition function often preserves all the information of interest. Consider for example a PCSP I arising from a simple maximization CSP, say weighted Max Cut. In the partition function Z_I, the leading term's power is the weight of a largest cut, and its coefficient the number of such cuts. (For this informal discussion, we will simply multiply-count cuts that are symmetric to one another.) With real edge weights, Z_I can have exponentially many terms, but the weight of a largest cut, and the number of such cuts, is preserved in the pruned partition function $(Z_I)_z$, which consists *solely* of that leading term.

More generally, consider a CSP where we wish to maximize one (real-valued) parameter for one or more values of several other (integer-valued) parameters. For example, the maximum bisection of an edge-weighted graph means maximizing the weight of cut edges, when the number of vertices in one side of the partition is held to $n/2$. Terms in the partition function Z_I of the corresponding bivariate PCSP have real powers of the edge-weighing variable z, and integral powers of the variable w that counts vertices in partition 1. Again (by definition) pruning

to $(Z_I)_z$ preserves the maximum power of z and the corresponding coefficient for any fixed power of w, so from $(Z_I)_z$ we may read off the weight of a maximum bisection and the number of such bisections, and indeed the weight and cardinality of maximum cuts of any specified sizes (say with exactly $n/3$ vertices in one side of the partition).

Pruning in the algorithms. The key point about pruning is that for polynomials p and q (in any set of variables) if coefficients are nonnegative then $(p+q)_z = (p_z + q_z)_z$ and $(pq)_z = ((p)_z(q)_z)_z$.

With either of the reductive algorithms described, the pruned partition function can be obtained efficiently, first pruning the input instance, then pruning as we go. (Correctness of this can be seen inductively, working backwards from the last reduction.) It is a trivial observation that for each reduction R (of Type 0–3), and any instance I, $(R(I))_z = (R(I_z))_z$. In particular, if we perform a full sequence of reductions and z-prune at every stage, we will end up with $(Z_I)_z$.

Pruning can also be done for tree decomposition-based dynamic-programming algorithms. Referring to the proof of Theorem 6, we simply note that the pruned partition function can be obtained from pruned versions of f_{B_1}, f'_{B_1}, f_{B_2}, and f'_{B_2}.

This establishes the following lemma.

LEMMA 10. *Given a z-prunable PCSP instance I with n variables and m clauses over a fixed domain $[k]$, the pruned partition function can be calculated by the reductive or dynamic programming algorithms (see Theorems 4, 5, and 6) with each ring operation taking time and space $O^\star(1)$.*

It is not likely that the PCSP extension of Williams' algorithm can accommodate real-valued powers, even with pruning. One problem is the use of fast matrix multiplication as a "black box": it is unlikely that we can prune within the multiplication algorithm. Another piece of evidence against being able to accommodate real-valued PCSP powers is that in the CSP setting, Williams was unable to accommodate real-valued scores (the analogue of the PCSP's powers), and a solution for PCSP would imply one for CSP. Williams noted that one could work around real-valued scores by approximation methods, for example multiplying all scores by a large constant and then replacing each with the nearest integer, and this can also be done in the PCSP setting.

11. SUMMARY OF PCSP ALGORITHMIC RESULTS

Combining the results in Sections 6–9, phrased in terms of the number of ring operations, with the bounds in Section 10 on the complexity of these operations for PCSPs, gives the following general conclusions. Some applications are presented in Section 13.

THEOREM 11. *The PCSP extension of Williams split-and-list algorithm [Williams 2004] solves any polynomially bounded PCSP instance with n variables over domain $[k]$ in time and space $O^\star\big(k^{\omega n/3}\big)$, where ω is the matrix-multiplication exponent for the ring of polynomials over reals.*

Parametrizing by vertices, this exponential-space algorithm is the only known algorithm faster than $O^\star(k^n)$ for PCSP, just as it is for CSP. In particular, it is the only known algorithm efficient for dense general PCSP instances.

THEOREM 12. *The PCSP extension of of Scott and Sorkin's reductive algorithm [Scott and Sorkin 2007] solves any polynomially bounded PCSP instance (or finds the pruned partition function of any prunable PCSP instance) with n variables, m clauses, and over domain $[k]$, in time $O^\star\big(k^{19m/100}\big)$ and space $O^\star(1)$.*

This algorithm is suitable for sparse instances (if $m > (100/19)n$ then the naive $O(k^n)$ algorithm is better). It is not as fast as dynamic programming (see Theorem 14), but unlike that and Williams' algorithm it runs in polynomial space, and is potentially practical.

THEOREM 13. *When $G = G(n, c/n)$ is an Erdős-Rényi random graph with $c \leq 1 + \lambda n^{-1/3}$, an extension of Scott and Sorkin's expected-linear-time CSP algorithm [Scott and Sorkin 2006b] solves any boolean PCSP instance with constraint graph G in expected time $O^\star(1)\exp(1 + \lambda^3)$ and in space $O^\star(1)$.*

This polynomial-space, polynomial-expected-time algorithm is of course the most efficient for sparse semi-random instances (below the giant-component threshold or in its scaling window).

THEOREM 14. *Tree decomposition-based dynamic programming can solve any polynomially bounded PCSP instance (or find the pruned partition function of any prunable PCSP instance) with n variables, m clauses, over domain $[k]$, and having treewidth $b - 1$, in time and space $O^\star\big(k^b\big)$ or (since $b - 1 \leq (13/75 + o(1))m$, from [Scott and Sorkin 2007; Kneis et al. 2009]) time and space $O^\star\big(k^{(13/75+o(1))m}\big)$.*

This exponential-space algorithm is suitable for instances of small treewidth, including sparse instances with $m < (75/13)n$.

12. CONSTRUCTING AND SAMPLING SOLUTIONS

Wherever we can compute a CSP's maximum value, we can also produce a corresponding assignment; and wherever we can count assignments producing a given value, we can also do exact random sampling from these assignments. The method is standard, and we illustrate with sampling. We construct our assignment one variable at a time, starting from the empty assignment. Given a partial assignment $\sigma_0 \colon V_0 \to [k]$, and a vertex $v \notin I_0$, we calculate the partition functions

$$Z_{I;\sigma_0(i)} = \sum_{\sigma \colon \sigma|_{V_0}=\sigma_0,\, \sigma(v)=i} I(\sigma), \tag{17}$$

and use these to determine the conditional distribution of $\sigma(v)$ given that $\sigma|_{I_0} = \sigma_0$.

This enables us to sample from a variety of distributions. For instance, we get the following result.

THEOREM 15. *Let G be a graph with m edges. Then in time $O^\star\big(2^{19m/100}\big)$ and space $O^\star(1)$ we can sample uniformly at random from the following distributions:*

—maximum cuts, maximum bisections, minimum bisections

—maximum independent sets, cliques of maximal size

—independent sets of any fixed size, cliques of any fixed size.

In the same time, we can sample from the equilibrium (Gibbs) distribution of the Ising model with any fixed interaction strength and external magnetic field.

A similar result holds for edge- and/or vertex-weighted graphs, except that we sample at random only from optimal assignments. Many other problems can be expressed in this framework. For example, we can count and sample proper k-colorings in time $O^\star\big(k^{19m/100}\big)$ and space $O^\star(1)$, or in time and space $O^\star\big(k^{(13/75+o(1))m}\big)$.

If we wish to exhibit just one optimal assignment deterministically, we can do so by a small modification of the sampling approach above: at each step, assign the next vertex v the smallest i such that $\sigma_0(i)$ can be extended to an optimal assignment (this can be read off from $Z_{I;\sigma(i)}$ given by (17)).

13. APPLICATIONS AND CONCLUSIONS

The PCSP / RCSP framework encompasses many problems that are not in CSP, yet can be solved by extensions of any general CSP algorithm we know of, with essentially equal efficiency. For several problems, this yields the best algorithms known.

13.1 Graph bisection

Jansen, Karpinski, Lingas and Seidel [Jansen et al. 2005] showed how to use dynamic programming based on tree decomposition to find a maximum or minimum bisection of a graph. Combined with the bounds from [Scott and Sorkin 2007; Kneis et al. 2009] on the treewidth of a graph with m edges, this gave an algorithm with a time and space bound of $O^\star\big(2^{(13/75+o(1))m}\big)$.

Until now, no polynomial-space algorithm was known for this problem other than the naive enumeration running in time $O^\star(2^n)$. Once bisection is phrased as a PCSP, our results (Theorem 12) immediately imply that in time $O^\star\big(2^{19m/100}\big)$ and polynomial space, we can solve maximum and minimum bisection, count maximum and minimum bisections, and indeed count bisections of all sizes. We can do this also in weighted bisection models, as long as they are polynomially bounded or z-prunable. Our results (Theorem 11) also mean that dense instances of bisection can be solved by Williams' algorithm, in time and space $O^\star\big(2^{\omega n/3}\big)$, again yielding the full generating function of cuts.

13.2 Cliques and Independent Sets

Maximum Independent Set is in Max 2-CSP and has been studied extensively, but, surprisingly, PCSP offers an advantage for some MIS variants. For simple MIS, for any graph with average vertex degree d, all four of our general-purpose PCSP algorithms are dominated either by [Bourgeois et al. 2008] (for average degree ≤ 3, time $O^\star\big(2^{0.1331n}\big)$) or by [Fomin et al. 2006] (time $O^\star\big(2^{0.288n}\big)$, otherwise). However, for weighted MIS, counting MIS, or weighted counting MIS, the fastest algorithm we are aware of is the vertex-parametrized $O^\star\big(2^{0.3290n}\big)$ algorithm of Fürer and Kasiviswanathan [Fürer and Kasiviswanathan 2005], based on an improved analysis of an algorithm of Dahllöf, Jonsson, and Wahlström [Dahllöf et al. 2005]. For any of these problems the $O^\star\big(2^{19m/100}\big)$ reductive algorithm of Theorem 12 is the fastest polynomial-space algorithm known if the average degree is below about 3.46, and the $O^\star\big(2^{(13/75+o(1))m}\big)$ exponential-space dynamic-programming algorithm of Theorem 14 the fastest algorithm known if the average degree is below about 3.79.

We noted earlier that Maximum Clique is not in the class 2-CSP, because constraints are given by non-edges, not edges, but that it is in the class PCSP. This

is nice, but does not currently offer any algorithmic advantage. If G has average degree $d = o(n)$, the following simple algorithm has running time $O^\star\big(2^{o(n)}\big)$, dominating all other algorithms under consideration here. For any k of our choosing, any clique either is contained in the $\leq n/k$-order subgraph induced by vertices of degree $\geq kd$, which is checkable in time $O^\star\big(2^{n/k}\big)$, or contains a vertex of degree $\leq kd$, which is checkable in time $O^\star\big(2^{kd}\big)$. Taking $k = \sqrt{n/d}$ gives running time $O^\star\Big(2^{\sqrt{nd}}\Big)$. If on the other hand d is not $o(n)$ then certainly $d = \omega(1)$. This rules out edge-parametrized algorithms, leaving us only Williams' algorithm, which is less efficient than applying a vertex-parametrized MIS algorithm such as that of [Fürer and Kasiviswanathan 2005] to the complement graph $\overline{G}$.

13.3 Ising partition function

The Ising partition function, a canonical object in statistical physics, is a PCSP, and can be solved by any of the algorithms presented here. We know of no other algorithm to find the partition function other than naive enumeration over all 2^n configurations.

13.4 Potts and other q-state models

The Potts (q-coloring) partition function can trivially be evaluated in time $O^\star(q^n)$, but it is by no means obvious that one can remove the dependence on q. A breakthrough result of Björklund, Husfeldt, Kaski, and Koivisto [Björklund et al. 2008] (see also [Björklund and Husfeldt 2006; Koivisto 2006a; Björklund et al. ; Björklund et al. 2007; Björklund and Husfeldt 2008]) shows how to evaluate the Potts partition function (and therefore by the Fortuin-Kasteleyn identity also the Tutte polynomial) using the Inclusion–Exclusion method, in time $O^\star(2^n)$ in exponential space, or time $O^\star(3^n)$ in polynomial space.

Our PCSP model includes the Potts model and more general models where the "energy" is not simply a function of equality or inequality of neighboring colors. Our algorithms unfortunately do have have exponential dependence on q, but with the greater generality of PCSP this may be unavoidable: as noted in [Björklund et al. 2008], Traxler [Traxler 2008] shows that if the q-state Potts model is generalized even to 2-CSPs on variables with q states, exponential dependence on q cannot be avoided, at least in the vertex-parametrized case, assuming the Exponential Time Hypothesis.

13.5 Conclusions

As we have shown, the PCSP formulation is clean and offers quantifiable advantages for several important problems. However, the greatest benefit is its broad applicability. For example, exponential time algorithms have not previously been considered for problems such as judicious partitioning, and for virtually any form of this problem such algorithms now follow instantly from its membership in PCSP. Moreover, when solving optimization problems coded into PCSP form, we automatically get solutions to the corresponding counting problems.

Finally, we note that algebraic formulations appear to be playing a newly important role in the development of exponential-time algorithms for CSPs and related problems. Our work along these lines began with [Scott and Sorkin 2006a], and

we have already discussed Koivisto's sum-of-products variation on Williams' split-and-list algorithm [Koivisto 2006b]. The methods leading to the $O^\star(2^n)$ Tutte polynomial computation [Björklund et al. 2008] work in a fairly general algebraic setting, and a new algorithm of Williams [Williams 2008] that finds a k-path in an order-n graph in time $O^\star\big(2^k\big)$ works in an algebra carefully chosen for properties including making walks other than simple paths cancel themselves out. PCSPs should be considered one piece of this new algebraic toolkit.

REFERENCES

Arnborg, S., Hedetniemi, S., and (Eds.), A. P. 1994. *Discrete Applied Mathematics 54,* 2-3. special issue.

Arnborg, S. and Proskurowski, A. 1989. Linear time algorithms for NP-hard problems restricted to partial k-trees. *Discrete Applied Mathematics 23,* 1, 11–24.

Björklund, A. and Husfeldt, T. 2006. Inclusion-exclusion algorithms for counting set partitions. In *Proceedings of the 47th IEEE Symposium on Foundations of Computer Science (FOCS 2006)*. 575–582.

Björklund, A. and Husfeldt, T. 2008. Exact algorithms for exact satisfiability and number of perfect matchings. *Algorithmica 52,* 2, 226–249.

Björklund, A., Husfeldt, T., Kaski, P., and Koivisto, M. 2007. Fourier meets Möbius: fast subset convolution. In *Proceedings of the 39th Annual ACM Symposium on Theory of Computing (STOC 2007)*. ACM, New York, 67–74.

Björklund, A., Husfeldt, T., Kaski, P., and Koivisto, M. 2008. Computing the Tutte polynomial in vertex-exponential time. In *Proceedings of the 49th IEEE Symposium on Foundations of Computer Science (FOCS 2008)*.

Björklund, A., Husfeldt, T., and Koivisto, M. Set partitioning via inclusion-exclusion. *SIAM Journal on Computing*. special issue dedicated to selected papers from FOCS 2006, to appear.

Bodlaender, H. L. 1996. A linear-time algorithm for finding tree-decompositions of small treewidth. *SIAM J. Comput. 25,* 6, 1305–1317.

Bollobás, B. and Scott, A. D. 1999. Exact bounds for judicious partitions of graphs. *Combinatorica 19,* 4, 473–486.

Bourgeois, N., Escoffier, B., and Paschos, V. T. 2008. An $O^*(1.0977^n)$ exact algorithm for max independent set in sparse graphs. In *IWPEC*, M. Grohe and R. Niedermeier, Eds. Lecture Notes in Computer Science, vol. 5018. Springer, 55–65.

Bulatov, A. and Grohe, M. 2005. The complexity of partition functions. *Theoret. Comput. Sci. 348,* 2-3, 148–186.

Coppersmith, D. and Winograd, S. 1990. Matrix multiplication via arithmetic progressions. *J. Symbolic Comput. 9,* 3, 251–280.

Dahllöf, V., Jonsson, P., and Wahlström, M. 2005. Counting models for 2SAT and 3SAT formulae. *Theoret. Comput. Sci. 332,* 1-3, 265–291.

Dechter, R. and Pearl, J. 1987. Network-based heuristics for constraint-satisfaction problems. *Artif. Intell. 34,* 1, 1–38.

Dechter, R. and Pearl, J. 1989. Tree clustering for constraint networks (research note). *Artif. Intell. 38,* 3, 353–366.

Dyer, M. E., Goldberg, L. A., and Paterson, M. 2006. On counting homomorphisms to directed acyclic graphs. In *ICALP (1)*. 38–49.

Fomin, F. V., Grandoni, F., and Kratsch, D. 2006. Measure and conquer: a simple $O(2^{0.288n})$ independent set algorithm. In *Proceedings of the 17th annual ACM-SIAM symposium on Discrete Algorithms (SODA 2006)*. ACM, New York, NY, USA, 18–25.

Fomin, F. V. and Høie, K. 2006. Pathwidth of cubic graphs and exact algorithms. *Inform. Process. Lett. 97,* 5, 191–196.

Fürer, M. and Kasiviswanathan, S. P. 2005. Algorithms for counting 2-SAT solutions and colorings with applications. Tech. Rep. 33, Electronic Colloquium on Computational Complexity.

Hell, P. and Nešetřil, J. 2004. *Graphs and homomorphisms*. Oxford Lecture Series in Mathematics and its Applications, vol. 28. Oxford University Press, Oxford.

Jansen, K., Karpinski, M., Lingas, A., and Seidel, E. 2005. Polynomial time approximation schemes for max-bisection on planar and geometric graphs. *SIAM J. Comput. 35,* 1, 110–119 (electronic).

Kneis, J., Mölle, D., Richter, S., and Rossmanith, P. 2009. A bound on the pathwidth of sparse graphs with applications to exact algorithms. In *SIAM J. Discrete Math.* Vol. 23. SIAM, 407–427.

Koivisto, M. 2006a. An $O^*(2^n)$ algorithm for graph colouring and other partitioning problems via inclusion-exclusion. In *Proceedings of the 47th IEEE Symposium on Foundations of Computer Science (FOCS 2006)*. 583–590.

Koivisto, M. 2006b. Optimal 2-constraint satisfaction via sum-product algorithms. *Inform. Process. Lett. 98,* 1, 24–28.

Monien, B. and Preis, R. 2001. Upper bounds on the bisection width of 3- and 4-regular graphs. In *Mathematical foundations of computer science, 2001 (Mariánské Lázně)*. Lecture Notes in Comput. Sci., vol. 2136. Springer, Berlin, 524–536.

Reed, B. A. 1992. Finding approximate separators and computing tree width quickly. In *STOC*. ACM, 221–228.

Robertson, N. and Seymour, P. D. 1995. Graph minors XIII: The disjoint path problem. *J. Combinatorial Theory (Series B) 63*, 65–110.

Scott, A. D. 2005. Judicious partitions and related problems. In *Surveys in combinatorics 2005*. London Math. Soc. Lecture Note Ser., vol. 327. Cambridge Univ. Press, Cambridge, 95–117.

Scott, A. D. and Sorkin, G. B. 2006a. Generalized constraint satisfaction problems. Tech. Rep. cs:DM/0604079v1, arxiv.org. Apr. See http://arxiv.org/abs/cs.DM/0604079.

Scott, A. D. and Sorkin, G. B. 2006b. Solving sparse random instances of Max Cut and Max 2-CSP in linear expected time. *Comb. Probab. Comput. 15,* 1-2, 281–315.

Scott, A. D. and Sorkin, G. B. 2007. Linear-programming design and analysis of fast algorithms for Max 2-CSP. *Discrete Optim. 4,* 3-4, 260–287.

Traxler, P. 2008. The time complexity of constraint satisfaction. In *Proceedings of the 3rd International Workshop on Parameterized and Exact Computation (IWPEC 2008)*. Lecture Notes in Computer Science, vol. 5018. Springer, 190–201.

Williams, R. 2004. A new algorithm for optimal constraint satisfaction and its implications. In *Proc. 31st International Colloquium on Automata, Languages and Programming (ICALP)*.

Williams, R. 2008. Finding paths of length k in $O^*(2^k)$ time. Tech. Rep. cs.DS/0807.3026v3, arxiv.org. Nov. See http://arxiv.org/abs/cs.DM/0604080.