Ising model partition-function computation as a weighted counting problem

Shaan Nagy,[1,2] Roger Paredes,[3] Jeffrey M. Dudek,[1] Leonardo Dueñas-Osorio,[3] and Moshe Y. Vardi[1]
[1]*Department of Computer Science, Rice University, Houston, Texas 77005, USA*
[2]*Department of Computer Sciences, University of Wisconsin-Madison, Madison, Wisconsin 53706, USA*
[3]*Department of Civil and Environmental Engineering, Rice University, Houston, Texas 77005, USA*

(Received 19 February 2024; accepted 27 March 2024; published 6 May 2024)

While the Ising model is most often used to understand physical phenomena, its natural connection to combinatorial reasoning also makes it one of the best models to probe complex systems in science and engineering. We bring a computational lens to the study of Ising models, where our computer-science perspective is twofold: On the one hand, we show that partition function computation (#Ising) can be reduced to weighted model counting (WMC). This enables us to take off-the-shelf model counters and apply them to #Ising. We show that one model counter (`TensorOrder`) outperforms state-of-the-art tools for #Ising on midsize and topologically unstructured instances, suggesting the tool would be a useful addition to a portfolio of partition function solvers. On the other hand, we consider the computational complexity of #Ising and relate it to the logic-based counting of constraint-satisfaction problems or #CSP. We show that known dichotomy results for #CSP give an easy proof of the hardness of #Ising and provide intuition on where the difficulty of #Ising comes from.

DOI: 10.1103/PhysRevE.109.055301

I. INTRODUCTION

The Ising spin glass model is a fundamental tool in statistical mechanics to study many-body physical systems [1,2]. One important property of an Ising model is its *partition function* (also called its *normalization probability*) [3]. Spin-glass models have also been recently leveraged in problems within data science, e.g., community detection [4].

Similarly, weighted counting is a fundamental problem in artificial intelligence, with applications in probabilistic reasoning, planning, inexact computing, and engineering reliability [5–10]. The task is to count the total weight, subject to a given weight function, of the solutions to a given set of constraints [9]. Weighted counting is a core task for many algorithms that normalize or sample from probabilistic distributions (arising from graphical models, conditional random fields, and skip-gram models, among others [11]).

From a complexity-theoretic view, both weighted counting and the computation of the Ising-model partition function lie in the complexity class #P-Complete under standard assumptions (e.g., weighted counting instances with rational, log-linear weights) [1,12–14]. Although proving the difficulty of #P-Complete problems is still open, it is widely believed that #P-Complete problems are fundamentally hard. Membership in #P-Complete means that the two problems are in some sense equivalent: computing an Ising model partition function can be done through a single weighted counting query with only a moderate (meaning polynomial) amount of additional processing, and vice versa.

In this work, we demonstrate that weighted counting provides useful lenses through which to view the problem of computing the partition function of an Ising spin glass model. We focus here on two closely related formalizations of weighted counting: *weighted model counting (WMC) problems* [9], and *weighted constraint satisfaction problems (w#CSPs)*. Section II introduces Ising models and weighted counting.

Through one lens, viewing the partition function of an Ising model as a WMC problem allows us to compute the partition function in practice. The study of WMC problems has largely focused on the development of practical tools (called *counters*) for solving WMC instances. This has resulted in a huge variety of counters [13,15–23], which, despite the computational difficulty of counting, have been used to solve large, useful applied problems in a variety of fields (e.g., informatics, infrastructure reliability, etc.) [24].

We show in Sec. III that powerful, off-the-shelf weighted model counters can be used to compute Ising partition functions. In particular, we find that the counter `TensorOrder` [15,16] outperforms a variety of other counters and traditional approaches to computing the partition function of Ising models for midsized square lattices and highly disordered topologies, including direct tensor network approaches in computational physics [25,26]. `TensorOrder`'s success on disordered topologies makes previously challenging Ising instances easier in practice. Thus, `TensorOrder` shows promise as part of a portfolio of Ising parition function solvers. Moreover, `TensorOrder` is able to outperform these approaches while still *exactly* computing the partition function, unlike other methods (e.g., `CATN` [25] and `Cotengra` [26]) that, despite typically computing results to machine precision, produce approximate counts with no accuracy guarantees, which are desirable in safety-critical applications. The key idea of `TensorOrder` is to divide the computation of a WMC problem into two phases—a planning phase of high-level reasoning, followed by an execution phase of low-level computation—in such a way that well-known high-performance computing libraries (`FlowCutter` [27] and `numpy` [28]) can be used directly for each phase.

Through another lens, viewing the partition function of an Ising model as a w#CSP gives us foundational insights into its computational complexity. It is useful to consider w#CSPs here since their study has largely been from a complexity-theoretic perspective and has led to deep results for the computational complexity of counting (e.g., Theorem 1) [8]. In Sec. IV we highlight how this w#CSP dichotomy theorem can be used to better understand the complexity and structure of Ising model partition function computation. Specifically, our analysis suggests that the difficulty of Ising partition function computation can be traced to the difficulty of answering the following problem: A partial assignment of spins to lattice sites defines a set of configurations. How does changing one lattice site's spin affect the probability of the set of configurations agreeing with the partial assignment?

We conclude in Sec. V and offer ideas for future research and development.

II. PRELIMINARIES

The Ising model is well known in computer science, and it appears in a number of interesting applications due to its expressiveness and dissimilarity from traditional computer science problems [29]. The relationship between Ising models and SAT is especially rich. Critical topological and hardness insights in SAT can be related to topological features of Ising models [30], and each problem has inspired many computational techniques for the other [31–33]. More than SAT, combinatorial optimization has seen a rash of Ising-based and Ising-inspired software and hardware techniques, particularly in quantum computing [34,35]. In adiabatic quantum optimization, the Ising model is used to represent a wide range of discrete optimization problems [29], which can then be solved using quantum devices [36]. These techniques find application in AI as well [37]. Additionally, methods developed outside statistical physics, such as engineering reliability methods, have been used in the study of the Ising model [38].

To inform our discussion of Ising models, we give a versatile yet formal definition of the *Ising model*. In particular, we define the *partition function* of an Ising model.

A. Ising models

Definition 1. An *Ising model* is a tuple (Λ, J, h) where (using $\mathbb{R}$ to denote the real numbers):

Λ is a set whose elements are called *lattice sites*

$J : \Lambda^2 \to \mathbb{R}$ is a function whose output $J(i, j)$ is called the *interaction* of $i, j \in \Lambda$ and

$h : \Lambda \to \mathbb{R}$ is a function, called the *external field*.

A *configuration* is a function $\sigma : \Lambda \to \{-1, 1\}$, which assigns a *spin* (either -1 or 1) to each lattice site. The (classical) *Hamiltonian* is a function H that assigns an *energy* to each configuration, as follows:

$$H(\sigma) \equiv -\sum_{i,j\in\Lambda} J(i, j)\sigma(i)\sigma(j) - \sum_{j\in\Lambda} h(j)\sigma(j). \tag{1}$$

Notationally, J, σ, and h accept arguments as subscripts and Λ is implicit. Thus, the Hamiltonian of an Ising model is often written

$$H(\sigma) \equiv -\sum_{i,j} J_{ij}\sigma_i\sigma_j - \sum_{j} h_j\sigma_j. \tag{2}$$

An Ising model can be interpreted physically as a set of discrete magnetic moments, each with a "spin" of -1 or 1. The entries of J indicate the strength of local interactions between two magnetic moments. The signed entries of h indicate the strength and orientation of an external magnetic field. Also note that the set of Ising Hamiltonians can be seen exactly as the set of degree-two polynomials over Λ with no constant term (when no self-interaction is assumed). A key feature of an Ising model is its set of *ground states*, which are configurations that minimize the Hamiltonian (i.e., minimal-energy configurations). Finding a ground state of a given Ising model is famously NP-hard [1] (and a suitable decision variant is NP-complete). This connection between Ising models and Boolean satisfiability (SAT) has been especially important for the study of SAT phase transitions, where the behavior of randomized SAT problems can be understood through the behavior of *spin-glass* models [39].

An important problem for Ising models is the computation of the *partition function*:

Definition 2. The *partition function* (or *normalization probability*) of an Ising model (Λ, J, h) with Hamiltonian H at parameter $\beta \geqslant 0$ (called the *inverse temperature*) is

$$Z_\beta \equiv \sum_{\sigma:\Lambda\to\{-1,1\}} e^{-\beta H(\sigma)}. \tag{3}$$

As its other name suggests, the partition function serves as a normalization constant when computing the probability of a configuration. These probabilities are given by a Boltzmann distribution: for a configuration σ, we have $P_\beta(\sigma) = \frac{e^{-\beta H(\sigma)}}{Z_\beta}$.

There are several special cases of the Ising model that are studied in research and practice. A few of the most common ones are the following.

Ferromagnetic and Antiferromagnetic Ising Models. The term "Ising model" is sometimes (especially historically) used to refer to the *ferromagnetic Ising model*, which is the special case where all interactions are non-negative, i.e., where $J_{ij} \geqslant 0$ for all $i, j \in \Lambda$. Interactions where $J_{ij} < 0$ are called *antiferromagnetic*.

2D Ising Models. In a *2D Ising model* the elements of Λ are vertices of a two-dimensional grid, and $J_{ij} = 0$ unless vertices i and j are adjacent in the grid.

Sparse Interactions. In many Ising models J can be represented as a sparse matrix, i.e., J is 0 on most values. For example, 2D Ising models are sparse.

No External Field. A common simplification is to assume that there is no external magnetic field, which means considering only Ising models where $h_i = 0$ for all $i \in \Lambda$.

No Self-Interaction. In most studied cases (including this paper) $J_{ii} = 0$ for all $i \in \Lambda$. Interactions of a particle with itself only contribute a constant additive factor to the Hamiltonian.

Symmetric/Triangular Interactions. We can always assume without loss of generality that, for all distinct $i, j \in \Lambda$, either $J_{ij} = 0$ or $J_{ji} = 0$. Similarly, we can instead assume without loss of generality that J is symmetric. These assumptions are mutually exclusive unless J is diagonal (which is uninteresting; see prior point).

Additionally, there is often interest in describing the physical connectivity of the lattice sites, as in 2D (planar) Ising models. This physical connectivity is sometimes called the *topology* of the model, and information about a model's topology is useful in analyzing it. Such topologies may be represented as graphs, defined below.

Definition 3 ((Simple) Graphs). An undirected graph (or, graph, for short) is a pair (V, E) where

V is a set whose elements are called *vertices*.

E is a collection of sets of the form $\{v, w\}$, where v and w are distinct elements in V. Each set $\{v, w\}$ represents an (undirected) *edge* between vertices v and w.

Typically, we restrict our attention to graphs that are *finite* (i.e., V is finite) and *simple* (a vertex cannot have an edge with itself, and no two vertices can share more than one edge).

B. Weighted counting problems

In computer science, the most foundational problems are decision problems, simple yes or no questions. One well-studied class of decision problems is the class of constraint satisfaction problems (CSPs). Broadly speaking, CSPs are problems in which one asks whether a solution exists that satisfies some given constraints. One famous example is Boolean satisfiability (SAT) in which one is given a Boolean formula and asked whether the formula can be made true by any assignment to its variables.

Connections between statistical models and CSPs are well known in physics [40]; the solutions of CSPs can be mapped to the ground states of spin glasses. By extension, counting the number of solutions to CSPs can be mapped to partition function evaluations in the zero-temperature limit. These deep connections between spin glasses and CSPs are exploited by celebrated counting algorithms such as the survey propagation algorithm and the cavity method [41].

Computer scientists are often also interested in counting problems, which focus on counting the number of solutions in a solution space that satisfy certain constraints. For example, one might wish to know the number of satisfying solutions to a Boolean formula or the number of 3-colorings for a particular graph. One might also be interested in weighted counting problems, where one assigns weights to particular elements of a solution space and sums the weights over the solution space. Computation of the partition function of an Ising model is a good example of this case. We assign a weight $(e^{-\beta H(\sigma)})$ to each configuration of the model and sum these weights over all configurations to determine the partition function. This sort of problem is common across many fields and, in many cases, is computationally intractable.

In the following two subsection, we discuss two computational modes which generalize CSPs and SAT from decision problems to weighted counting problems, w#CSPs and WMC. As in the decision case, weighted counting problems describe a general sort of problem, the w#CSP framework captures many such problems, and WMC is a well-studied problem expressible in the w#CSP framework. Casting Ising model partition function computation in these two modes provides deep theoretical insight and extends the range of techniques that can be used to compute partition functions.

1. Weighted counting constraint satisfaction problems

Many (but not all) weighted counting problems can be expressed as Weighted Counting Constraint Satisfaction Problems (w#CSPs). Framing weighted counting problems as w#CSPs introduces a standard form, which allows work done on the standard form to be applied to many different weighted counting problems. This is especially evident when analyzing the hardness of w#CSPs as we discuss later (see Theorem 1).

In defining w#CSPs, we use $\mathbb{C}$ to denote the complex numbers and we use $\mathbb{Z}_+$ to denote the positive integers.

Definition 4 (Constraint Language). Let D be a finite set and, for each $k \in \mathbb{Z}_+$, let A_k be the set of all functions $F : D^k \to \mathbb{C}$. Then a *constraint language* $\mathcal{F}$ is a subset of $\bigcup_{k\in\mathbb{Z}_+} A_k$.

We write $\#CSP(\mathcal{F})$ as the weighted #CSP problem corresponding to $\mathcal{F}$.

Definition 5 (Weighted #CSP: Instance and Instance Function). Let $\mathcal{F}$ be a constraint language. Then an *instance of* $\#CSP(\mathcal{F})$ is a pair (I, n) where $n \in \mathbb{Z}_+$ and I is a finite set of formulas each of the form $F(x_{i_1}, \dots, x_{i_k})$, where $F \in \mathcal{F}$ is a k-ary function and $i_1, \dots, i_k \in [n]$ with each x_{i_j} being a variable ranging over D. Note that k and $i_1, \dots, i_k$ may differ in each formula in I.

Given an instance (I, n), we define the corresponding *instance function* $F_I : D^n \to \mathbb{C}$ to be the following conjunction over I:

$$F_I(y) = \prod_{F(x_{i_1},\dots,x_{i_k})\in I} F(y[i_1], \dots, y[i_k]). \tag{4}$$

The output of the instance (I, n) is $Z(I) = \sum_{y\in D^n} F_I(y)$, analogous to the partition function of an Ising model.

As we will see, representing problems as w#CSPs allows us access to a rich body of theoretical and computational work. In Sec. III we will see that converting a problem to a w#CSP allows us to easily assess its computational complexity—how the number of operations needed to solve the problem scales with input size. Converting a problem to a w#CSP also allows access to a slew of standardized computational tools. For example, a common approach for computing the partition function of an Ising model is to represent the problem as a tensor network. In fact, all w#CSPs can be converted to tensor networks, so recognizing partition function computation as a w#CSP makes the applicability of tensor networks immediate. In Sec. IV we show that Ising partition function computation is a w#CSP and discuss the deep theoretical consequences.

2. Weighted model counting

All weighted counting problems (and their w#CSPs forms) can be reduced to weighted model counting (WMC). By converting a w#CSP to WMC, we can take advantage of well-developed existing solvers which, in many cases, outperform problem-specific methods. Propositional model counting or the Sharp Satisfiability problem (#SAT) consists of counting the number of satisfying assignments of a given Boolean formula. Without loss of generality, we focus on formulas in conjunctive normal form (CNF).

Definition 6 (Conjunctive Normal Form (CNF)). A formula ϕ over a set X of Boolean variables is in CNF when written as

$$\phi = \bigwedge_{i=1}^{m} C_i = \bigwedge_{i=1}^{m} \left(\bigvee_{j=1}^{k_i} l_{ij} \right), \tag{5}$$

where every clause C_i is a disjunction of $k_i \leqslant |X|$ literals, and every literal l_{ij} is a variable in X or its negation.

Given a truth-value assignment $\tau : X \to \{0, 1\}$ (such as a microstate of a physical system), we use $\phi(\tau)$ to denote the formula that results upon replacing the variables $x \in X$ of ϕ by their respective truth values $\tau(x)$, and say τ is a satisfying assignment of ϕ when $\phi(\tau) = 1$. Thus, given an instance ϕ of #SAT, one is interested in the quantity $\sum_{\tau \in [X]} \phi(\tau)$, where $[X]$ denotes the set of truth-value assignments.

Literal weighted model counting (WMC) is a generalization of #SAT in which every truth-value assignment is associated to a real weight. Formally, WMC is defined as follows.

Definition 7 (Weighted Model Counting). Let ϕ be a formula over a set X of Boolean variables, and let $W : X \times \{0, 1\} \to \mathbb{R}$ be a function (called the weight function). Let $[X]$ denote the set of truth-value assignments $\tau : X \to \{0, 1\}$. Analogous to the Ising partition function and w#CSP instance output, we define the weighted model count of ϕ w.r.t. W is

$$W(\phi) \equiv \sum_{\tau \in [X]} \phi(\tau) \cdot \prod_{x \in X} W(x, \tau(x)). \tag{6}$$

One advantage of WMC with respect to #SAT is that it captures problems of practical interest such as probabilistic inference [42] more naturally. Network reliability [43] and Bayesian inference [10] are specific instances of probabilistic inference problems. Thus, the development of practical WMC solvers remains an active area of research, where algorithmic advances can enable the solution of difficult combinatorial problems across various fields.

In this paper we cast the Ising model partition function computation as a well-understood problem of weighted model counting, which, in turn, enables its computation via actively developed off-the-shelf WMC solvers. Significantly, this paper gives empirical evidence that the runtime of *exact* model counters outperforms *approximate* state-of-the-art physics-based tools currently used for partition function computations [44] in some cases. In particular, some model counters (i.e., `TensorOrder`) do better for Ising instances with sufficiently disordered topologies.

III. ISING WEIGHTED MODEL COUNTING

Computing the partition function of an Ising model amounts to taking a weighted sum over possible states. In the realm of model counting, this type of problem is expressible as a weighted model counting (WMC) problem (Definition 7). Reducing partition function computation to weighted model counting allows us to take advantage of various dedicated WMC solvers which can handle very large formulas [24]. In this section we develop a succinct reduction from partition function computation to WMC, and we show that dedicated WMC solvers (in particular `TensorOrder` [15,16]) outperform existing tools for partition function computation (CATN [25] and Cotengra [26]) for midsized square lattices and sufficiently disordered topologies. This suggests that model counters constitute a valuable addition to a portfolio of Ising solvers.

TABLE I. Truth-table of formulas $p_{ij} = [(x_i \Leftrightarrow x_j) \Leftrightarrow x_{ij}]$ and sign of $\sigma_i\sigma_j$. Note that every assignment τ such that $p_{ij}(\tau) = 1$ is consistent with encoding sough after in Eq. (7).

x_i	x_j	x_{ij}	p_{ij}	σ_i	σ_j	$\sigma_i\sigma_j$	Eq. (7)
0	0	1	1	−	−	+	True
1	0	0	1	+	−	−	True
0	1	0	1	−	+	−	True
1	1	1	1	+	+	+	True
0	0	0	0	−	−	+	False
1	0	1	0	+	−	−	False
0	1	1	0	−	+	−	False
1	1	0	0	+	+	+	False

A. Ising model partition function as WMC

We now reduce the problem of computing the partition function of the Ising model, Z_β (Definition 2), to one of weighted model counting (Definition 7). Formally, our reduction takes an Ising model and constructs an instance $W(\phi)$ of weighted model counting such that $Z_\beta = W(\phi)$. The reduction consists of two steps. In Step 1 we construct a CNF formula, ϕ, over a set X of Boolean variables, such that the truth-value assignments $\tau \in \{0, 1\}^X$ that satisfy the formula correspond exactly to the configurations $\sigma \in \{\pm 1\}^\Lambda$ of the Ising model. In Step 2, we construct a literal weight function, W, such that for every satisfying assignment, τ, the product of literal weights, $\prod_{x \in X} W(x, \tau(x))$, is equal to the Boltzmann weight of the corresponding Ising model configuration σ. Together, Step 1 and Step 2 ensure that $Z_\beta = W(\phi)$, which we make explicit at the end of this section. We next show the details of the two steps.

1. Step 1 of 2: The Boolean formula in CNF

To construct the Boolean formula ϕ in CNF, we first introduce the set X of Boolean variables x_i, with $i \in \Lambda$, and Boolean variables x_{ij}, with $i, j \in \Lambda$ and $i \neq j$.[1] We use the variable x_i to represent the spin state $\sigma_i \in \{\pm 1\}$ for every $i \in \Lambda$. We use the variable x_{ij} to represent the interaction $\sigma_i\sigma_j \in \{\pm 1\}$ for every $i \neq j$. By convention, we associate the truth-value one with the positive sign, and the truth-value zero with the negative sign. The semantic relation between truth-value assignments $\tau \in \{0, 1\}^X$ and Ising model configuration $\sigma \in \{\pm 1\}^\Lambda$ is formally encoded by the equations below and

[1] Whenever $J_{ij} = J_{ji} = 0$, we can ignore variable x_{ij} for $i \neq j$.

Table I:

$$\sigma_i = 2\tau(x_i) - 1,\text{for } i \in \Lambda,$$
$$\sigma_i\sigma_j = 2\tau(x_{ij}) - 1, \quad \text{for } i, j \in \Lambda : i \neq j. \tag{7}$$

The first equation ensures that truth-value assignments $\tau(x_i) = 1$ and $\tau(x_i) = 0$ correspond exactly to spin states $\sigma_i = +1$ and $\sigma_i = -1$ respectively. The second equation ensures that truth-value assignments, $\tau(x_{ij}) = 1$ and $\tau(x_{ij}) = 0$, correspond exactly to positive and negative interactions, $\sigma_i = \sigma_j$ and $\sigma_i \neq \sigma_j$, respectively. As not every truth-value assignment $\tau \in \{0, 1\}^X$ describes a valid Ising configuration, we ensure that the set of truth-value assignments that satisfy the relations of Eq. (7) must also be such that $\tau(x_{ij}) = 1$ exactly when $\tau(x_i) = \tau(x_j)$ (i.e., when the spins have equal sign $\sigma_i = \sigma_j$), and $\tau(x_{ij}) = 0$ otherwise. This is consistent with the signs of interactions, which are positive for equal spin states and negative otherwise. Hence, the set of assignments that describe valid configurations via the encoding of Eq. (7) is exactly the set of satisfying assignments of the Boolean formula:

$$\phi \equiv \bigwedge_{i,j:i\neq j} p_{ij}, \tag{8}$$

with $p_{ij} = [(x_i \Leftrightarrow x_j) \Leftrightarrow x_{ij}]$. We can equivalently express p_{ij} in CNF as the conjunction of the clauses $(x_i \vee \bar{x}_j \vee \bar{x}_{ij})$, $(x_i \vee x_j \vee x_{ij})$, $(\bar{x}_i \vee x_j \vee \bar{x}_{ij})$, and $(\bar{x}_i \vee \bar{x}_j \vee x_{ij})$. The correctness of this CNF encoding can be shown by first noting that a satisfying assignment of ϕ must satisfy all predicates p_{ij}. Then, from the truth table of predicate p_{ij} depicted in Table I, it follows that Eq. (7) maps every satisfying assignment τ of ϕ to an Ising model configuration σ, and vice versa. Thus, the satisfying assignments of ϕ and the configurations of the Ising model are in one-to-one correspondence.

2. *Step 2 of 2: The literal-weight function*

To construct the literal-weight function W that assigns weights to variables depending on the values they take, first note that the Boltzmann weight of an arbitrary configuration σ of the Ising model can be written as

$$e^{-\beta H(\sigma)} = \left(\prod_{i\in\Lambda} e^{\beta h_i \sigma_i}\right) \cdot \left(\prod_{i,j\in\Lambda} e^{\beta J_{ij}\sigma_i\sigma_j}\right). \tag{9}$$

We construct W such that the first term in the right-hand side of Eq. (9) is the product of literal weights $W(x_i, \tau(x_i))$, for every $i \in \Lambda$, and the second term of the same expression is the product of literal weights $W(x_{ij}, \tau(x_{ij}))$, for every $i \neq j$. Specifically, we introduce the literal-weight function:

$$W(x_i, \tau(x_i)) = e^{\beta h_i[2\tau(x_i)-1]}, \text{ for } i \in \Lambda,$$
$$W(x_{ij}, \tau(x_{ij})) = e^{\beta J_{ij}[2\tau(x_{ij})-1]}, \text{ for } i \neq j, \tag{10}$$

and observe that when $\phi(\tau) = 1$, or equivalently when Eq. (7) maps τ to an Ising model configuration σ, we find that the product of literal weights, $\prod_{x\in X} W(x, \tau(x))$, is equal to the Boltzmann weight $e^{-\beta H(\sigma)}$ of the respective Ising model configuration.

3. *The partition function equals the weighted model count*

Given the Boolean formula ϕ in Eq (8), and given the literal weight function W in Eq. (10), we can verify the equivalence between the partition-function value Z_β and the weighted model count, $W(\phi)$, by writing the latter as a summation of the form $\sum_{\tau\in R}\prod_{x\in X} W(x, \tau(x))$, where $R = \{\tau : \phi(\tau) = 1\}$ is the set of satisfying assignments of ϕ. Moreover, Eq. (7) establishes a one-to-one correspondence between the satisfying assignments $\tau \in R$ and the configurations $\sigma \in \{\pm 1\}^\Lambda$ of the Ising model. Since $\prod_{x\in X} W(x, \tau(x)) = e^{-\beta H(\sigma)}$ for every satisfying assignment $\tau \in R$, we conclude that $Z_\beta = W(\phi)$.

Next, we review off-the-shelf weighted model counters and evaluate their performance relative to state-of-the-art solvers (developed by the physics community) used to compute Ising partition functions.

B. Weighted model counters

Increasingly, model counters are finding applications in fundamental problems of science and engineering, including computations of the normalizing constant of graphical models [45], the binding affinity between molecules [46], and the reliability of networks [43]. Thus, the development of model counters remains an active area of research that has seen significant progress over the past two decades [16,47].

In terms of their algorithmic approaches, exact weighted model counters can be grouped into three broad categories: direct reasoning, knowledge compilation, and dynamic programming. Solvers using direct reasoning (e.g., `Cachet` [48]) exploit the structure of the CNF representation of the input formula to speed up computation. Solvers using knowledge compilation (e.g., `miniC2D` [18]) devote effort to converting the input formula to an alternate representation in which the task of counting is computationally efficient, thereby shifting complexity from counting to compilation. Solvers using dynamic programming (e.g., `DPMC` [21] and `TensorOrder` [16]) exploit the clause structure of the input formula to break the counting task down to simpler computations. The problem breakdown and subproblem recall is achieved by using graph-decomposition algorithms in the constraint graph representation of the formula.

In addition to exact solvers, there is a pool of approximate model counters that seek to rigorously trade precision (up to an admissible error and a level of confidence specified by the user) for speed [49]. The numerical experiments in the next section use representative solvers from the three groups of exact solvers outlined, as well as an approximate solver.

C. Numerical experiments

We empirically demonstrate the utility of weighted model counters in Ising model partition function computations by comparison with two state-of-the-art approximate tensor network contraction tools often used to compute partition functions in physics. Note that contracting tensor networks can be reduced to WMC and vice versa, so these physics tools can be applied to the same range of problems as model counters. The first tool comes from Ref. [25]. There, it was shown that tensor network contraction and a matrix product state calculus outperform other strategies in computational physics to obtain machine-precision approximations of the free energy, denoted as F. The free energy and the partition function are related as $F = -(1/\beta)\ln Z_\beta$. Hereafter, we refer to the publicly available implementation of their method as `CATN` [50]. The latter can outperform various competitive

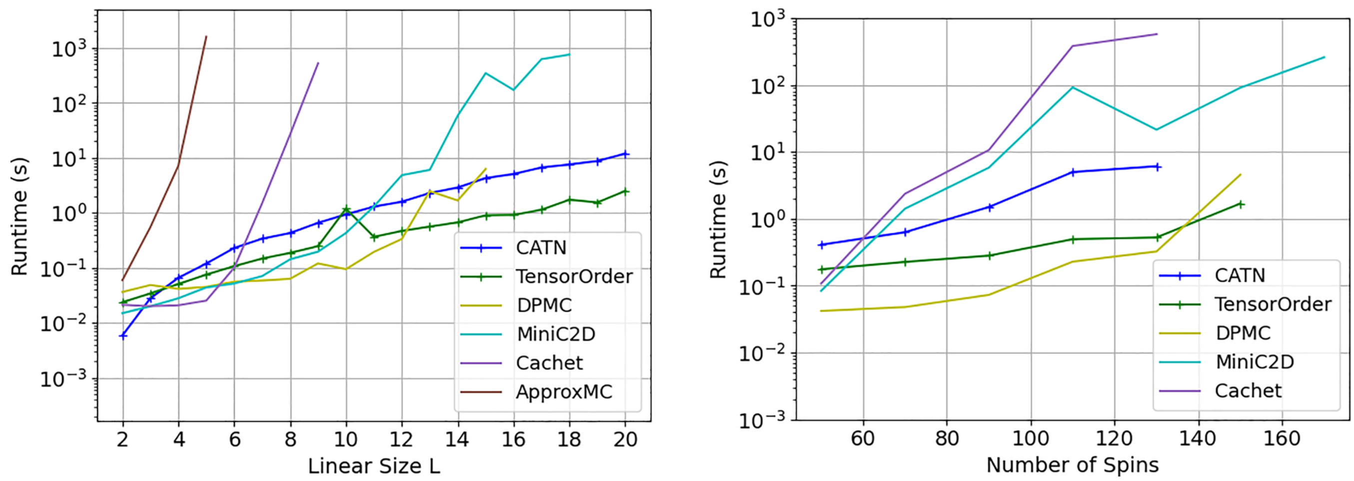

FIG. 1. Running time of the partition function computation for the exact weighted model counters, `miniC2D` [18], `Cachet` [48], `TensorOrder` [16], and `DPMC` [21]; the approximate model counter, `ApproxMC` [52]; and the approximate reference tool from physics, `CATN` [25]. Left: Two-dimensional square lattice with linear size L at $\beta = 1$. Right: Random graphs with average degree three at $\beta = 1$.

mean-field methods in physics with a small runtime overhead, so we take `CATN` as a baseline for comparison in our weighted counting-based experiments. It is worth highlighting that, unlike `CATN`, model counters are exact or have accuracy guarantees. In practice `CATN` produces machine-precision results, but it cannot promise to always do so.

The second state-of-the-art approximate tool from the physics community that we compare against is `Cotengra` [26]. This tensor contraction tool uses a variety of techniques including hypergraph partitioning and Bayesian optimization to construct efficient contraction trees. These contraction trees are then approximated via a "compressed contraction" strategy. The comparisons of Ref. [51] suggest `Cotengra` outperforms `CATN` on square and cubic lattices in terms of FLOPs (number of floating point operations), but our results suggest this is not borne out in the actual runtime. However, `Cotengra` does demonstrate a runtime advantage on random graphs.

Our computational evaluation includes random graphs of average degree three and two-dimensional $L \times L$ lattices, where L is referred to as the linear size. Square lattices are a standard type of model used to benchmark performance in the Ising context [25]. Random graphs are also considered because they challenges solvers' abilities to exploit irregular structure. We expect WMC solvers to perform well on random graphs because they are designed to find any exploitable structure even on random substrates.

Each experiment in Fig. 1 was run in a high-performance cluster using a single 2.60 GHz core of an Intel Xeon CPU E5-2650 and 32 GB RAM. Experiments in Fig. 2 were run on a high-performance cluster using a single 2.60 GHz core of an Intel Xeon Gold 6126 CPU and 32 GB RAM. Each implementation was run once on each benchmark. An experiment is deemed to be successful when there are no insufficient memory errors or timeouts.

The results of the experiments and comparisons across multiple tools are summarized in Fig. 1, with a larger set of instances and the two competitive physics-based tools in Fig. 2. Significantly, for every square lattice with linear size less than 25, there is an exact weighted model counter that outperforms the state-of-the-art approximate tool (`CATN`). In particular, `TensorOrder` was consistently faster than `CATN` in all shown midsized instances with $L > 10$ and on random graphs, as well as faster than `Cotengra` for all random graphs and square lattices with $L > 11$. We stress that partition function computations by model counters are exact or have accuracy guarantees. Overall, for these instances, the dynamic-programming solvers (`DPMC` and `TensorOrder`) displayed the best performance, especially for random graphs. We attribute `TensorOrder`'s advantage on all random benchmarks and most square lattices to the power of dynamic-programming model counters, which not only excel at finding optimal solution strategies in structured instances, like the square lattice, but also display excellent performance for randomly generated topologies given their sequential planning and execution phases that identify anything exploitable in the problem instances while deferring long-range interactions as much as possible. Also, `miniC2D` (knowledge compilation) seems to be better suited than `Cachet` (direct reasoning), though neither performed especially well for this Ising task.

Regarding `ApproxMC`, the approximate model counter used, it fails on all benchmarks except those of extremely small size because it requires that weighted model counting problems be encoded as unweighted problems, greatly increasing the number of variables and clauses in the formulas it operates over.

Overall `CATN`, `Cotengra`, and `TensorOrder` were the only solvers that did not timeout quickly. In addition, on all benchmarks except large square lattices, `TensorOrder`'s exact calculations were generally faster than both `Cotengra`'s and `CATN`'s approximate computations, at times an order of magnitude faster. We note that all three tools are tensor network contraction-based; however, `TensorOrder` is a dynamic programming solver that uses state-of-the-art graph decomposition techniques to select contraction orderings. We take the better performance of `TensorOrder` on the midsized bench-

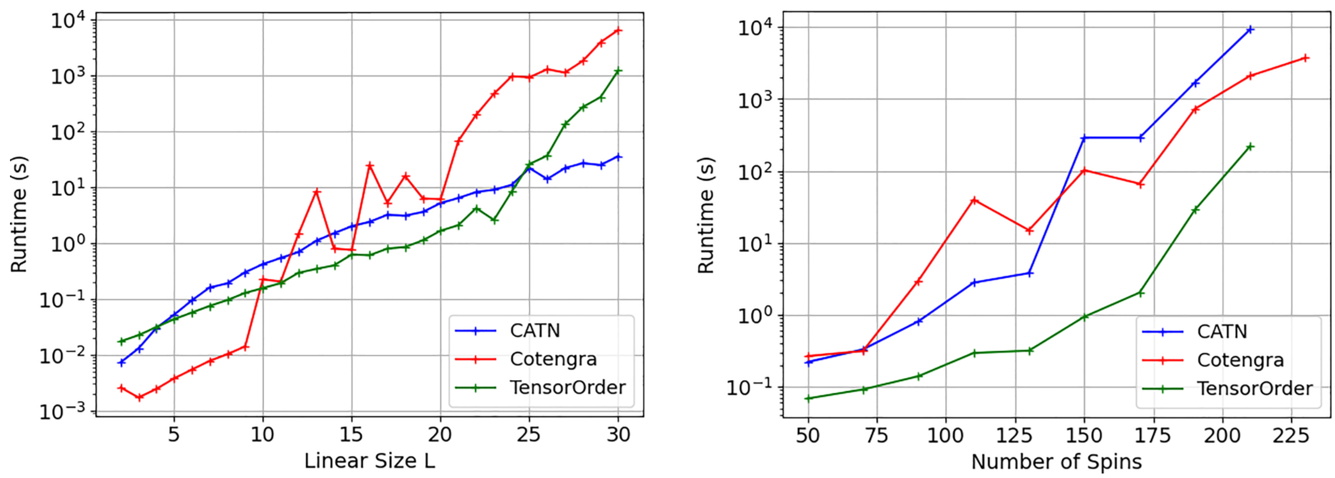

FIG. 2. Running time of the partition function computation for the exact weighted model counter `TensorOrder` [16] and the approximate reference tools from physics, `CATN` [25] and `Cotengra` [26]. Note that `TensorOrder` outperforms both `Cotengra` and `CATN` for midsize lattices. For the sizes we were able run, `TensorOrder` maintains a strong advantage on random graphs (210 spins with treewidth 31). Left: Two-dimensional square lattice with linear size L at $\beta = 1$. Right: Random graphs of average degree 3 at $\beta = 1$.

marks and especially on random graphs, as evidence that its contraction orderings are more robust than those used by `CATN` and `Cotengra`. On the large square lattice benchmarks ($L > 25$), on the other hand, the running time of `TensorOrder` increases since it spends more time looking for better graph decompositions; the runtime of `TensorOrder` could be improved by incorporating graph decomposition techniques that are optimized for such highly structured graphs. For these large benchmarks, the tree widths of `TensorOrder`'s decompositions ranged from 35 at $L = 25$ to 42 at $L = 30$.

Regarding the inverse temperature parameter in our experiments, it is fixed at $\beta = 1$, as we consider the same Ising model instances in the work of Pan *et al.* [25]. Typically, weighted counting solvers use fixed-point arithmetic that may lead to numerical errors due to decimal truncation. We did not detect significant numerical errors for the Ising model instances in this study; however, we expect numerical errors may amplify for large inverse temperature values of β. In our preliminary experiments, solvers' runtimes were not found to vary with β, supporting the choice to fix β.

Taking all results together, we believe that the computational advantage of weighted counters over state-of-the-art tools in this study is partly due to their ability to detect and take advantage of treelike structures of Ising model instances. Thus, the computational advantage of weighted counters will hold in graphs where the tree width stays fixed as the number of spins increases across a variatey of interaction topologies. In contrast, for graphs where the tree width increases as the number of spins increases, like the square lattice, weighted counters will face the same challenges of every other tool in combinatorics, exact or approximate. In sum, our evaluation shows that weighted model counters (especially `TensorOrder`) outperform state-of-the-art tools in the physics community in computing partition functions for some Ising instances. In particular, model counters have an advantage on instances with disordered topologies. This suggests that `TensorOrder` would be of use in a portfolio of partition function solvers. Additionally, exact model counters are more robust than alternatives such as `Cotengra` and `CATN`, which are approximate tools lacking guarantees of accuracy, although these two approximate tools give machine-precision results in practice.

IV. HARDNESS AND RELATIONSHIP TO WEIGHTED CONSTRAINT SATISFACTION

Weighted model counting of CNF formulas, the problem to which we reduced partition function computation in Sec. III, is #P-complete [53] meaning that WMC is likely to be computationally intractable. It is also known that computing the partition functions of Ising models is itself #P-hard [54]. This all means that the problem of computing Ising model partition functions is at least as hard as WMC. In this section we demonstrate how this #P-hardness can be easily derived from a formulation of partition-function computation as a w#CSP (Definition 5). While the hardness of computing Ising-model partition functions is well known, most proofs are extremely complex. The method we discuss below is a straightforward dichotomy test that can be directly applied to many similar problems without the difficulties of a more classical proof (i.e., selecting an appropriate #P-hard problem to reduce from, and constructing a clever reduction). Furthermore, we fold the discussion of hardness of computing the Ising-model partition functions into a much broader discussion of evaluating w#CSPs, and from this we gain a better understanding of what makes partition functions so hard to compute—the difficulty in relating the change of a single lattice site's spin to a change in probability across configurations.

In Appendix A we show that any problem expressible as a w#CSP can be reduced to WMC. As in Sec. III, we encode both variables' values and satisfaction of constraints as Boolean variables, and we introduce formulas so that all valid assignments correctly relate constraints to configurations. This generalizes the approach of Sec. III to arbitrary w#CSPs.

A. Introduction to computational complexity

We provide here a high-level overview of computational-complexity theory—complexity theory for short—for counting problems. Readers interested in a more rigorous treatment of complexity theory might refer to Sipser's *Introduction to the Theory of Computation* [55].

One use of a general problem format such as w#CSP is to be able to assess the computational complexity of problems representable in the format. A problem is a mapping from problem instances to their associated outputs. When discussing problems, it is always important to be explicit about the set of instances over which a problem is defined (often called a problem's "domain"). Hardness of a problem, as we see below, is a property of a problem's domain, which consists of an infinite class of instances.

A problem's computational (time) complexity describes how the time (more formally the number of operations) required to solve a problem grows with the size of the input. Algorithms with low computational complexity are called *scalable*, while algorithms with high computational complexity are not considered to be scalable. Thus, there is often interest in knowing whether an algorithm with low computational complexity exists for a given problem [56].

A theoretical distinction is often made between problems that can be solved in polynomial time and those that are not believed to be solvable in polynomial time. There are many counting problems whose solvability in polynomial time is an open problem. The most famous example of such a problem is propositional model counting (#SAT), which counts the number of assignments that satisfy a given formula in Boolean logic. Other examples include computing the number of maximum size cut sets of a graph (#MAXCUT), computing matrix permanents for Boolean matrices, and counting the number of perfect matchings of a bipartite graph [14]. In the study of counting problems, we often discuss problems in the complexity class FP (Function Polynomial-Time) and problems which are $\#P$-hard. Problems in FP are known to be solvable in polynomial time, and problems which are $\#P$-hard are believed (but not provably known) to be impossible to solve in polynomial time [14].

A useful tool in establishing membership of a problem in FP or $\#P$-hard is the idea of polynomial equivalence. Problems that can be converted to one another in polynomial time are said to be polynomially equivalent. If two problems are polynomially equivalent, a polynomial-time algorithm for one problem can be used to construct a polynomial-time algorithm for the other, and vice versa. If a problem is polynomially equivalent to another problem in FP, both problems are in FP. Similarly, if a problem is polynomially equivalent to another problem that is $\#P$-hard, both problems are $\#P$-hard.

B. The w#CSP Dichotomy Theorem

A landmark result in complexity theory is the following Dichotomy Theorem, which gives criteria for determining the complexity class of a w#CSP [8]. Every w#CSP is either in FP, meaning that it can be solved in polynomial time, or $\#P$-hard, which for our purposes means it is not believed to be polynomially solvable. While the existence of this dichotomy is itself theoretically significant in that it helps us understand how computational hardness arises, this paper focuses on applying the theorem to determine the hardness of computing partition functions. Note also that the Dichotomy Theorem applies only to problems with finite constraint languages (Definition 4). The conditions referenced in the following theorem are discussed below and in greater detail in Appendixes B and C.

Theorem 1 (Dichotomy Theorem for w#CSPs [8]). Let $\mathcal{F}$ be a finite constraint language with algebraic complex weights. Then $\#CSP(\mathcal{F})$ is polynomially solvable if the Block-Orthogonality Condition (Definition 13), Mal'tsev Condition (Definition 23), and Type Partition Condition hold (Definition 26). Otherwise, $\#CSP(\mathcal{F})$ is $\#P$-hard.

The statement of the w#CSP Dichotomy Theorem and its conditions is complex, so it is useful to include some motivation. The three criteria given by the theorem follow naturally when we attempt to construct a polynomial-time algorithm for w#CSPs. One reason that w#CSPs are often computationally expensive to solve is that the effect of a single assignment to a given variable is hard to capture. In particular, given a problem instance I and its associated n-ary formula $F_I(\boldsymbol{x})$, it is challenging to find a general rule that relates $F_I(x_1, x_2, \ldots, x_n)$ to $F_I(x_1', x_2, \ldots, x_n)$. For this reason, current computational approaches require consideration of a set of assignments whose size is exponential in n. The algorithm driving the w#CSP Dichotomy Theorem demands that changing the assignment of a particular variable changes the value of F_I in a predictable way. The criteria of the Dichotomy Theorem are necessary and sufficient conditions for these assumptions to hold.

A detailed treatment of the conditions and the intuition the dichotomy criteria capture can be found in the appendices. Appendix B discusses the Block Orthogonality Condition, and Appendix C explains the Mal'tsev and Type Partition Conditions. Here we are as brief as possible while including enough information to follow Sec. IV C, which invokes Theorem 1 on Ising model partition function computation.

In the definitions below, we take t to be an arbitrary member of $\{1, \ldots, n\}$.

Definition 8 ($F_I^{[t]}$). Let $F_I^{[t]} : D^t \to \mathbb{C}$ be defined as

$$F_I^{[t]}(y_1, \ldots, y_t) = \sum_{y_{t+1}, \ldots, y_n \in D} F_I(y_1, \ldots, y_n). \tag{11}$$

$F_I^{[t]}$ describes the impact on $Z(I)$ of all assignments agreeing with a given partial assignment to $x_1, \ldots, x_t$.

Definition 9 ($F_I^{[t]}(\boldsymbol{y}, \cdot)$). If we regard $F_I^{[t]}$ as a $|D|^{t-1} \times |D|$ matrix, we may refer to its rows by $F_I^{[t]}(\boldsymbol{y}, \cdot)$ as below. Here $\boldsymbol{y} \in D^{t-1}$ and $D = \{d_1, \ldots, d_{|D|}\}$. Thus

$$F_I^{[t]}(\boldsymbol{y}, \cdot) = \left[\sum_{\boldsymbol{w} \in D^{n-t}} F_I(\boldsymbol{y}, d_1, \boldsymbol{w}), \quad \ldots, \quad \sum_{\boldsymbol{w} \in D^{n-t}} F_I(\boldsymbol{y}, d_{|D|}, \boldsymbol{w})\right]. \tag{12}$$

We can group similar partial inputs $\boldsymbol{y}$ by linear dependence of the rows $F_I^{[t]}(\boldsymbol{y}, \cdot)$. We formalize that as follows.

Definition 10 ($\boldsymbol{v}^{\boldsymbol{y}}$). Given $\boldsymbol{y} \in D^{t-1}$ and a total order on D, we define

$$\boldsymbol{v}^{\boldsymbol{y}} = \frac{F_I^{[t]}(\boldsymbol{y}, \cdot)}{F_I^{[t]}(\boldsymbol{y}, a_t)} \tag{13}$$

for the least $a_t \in D$ so that $F_I^{[t]}(\boldsymbol{y}, a_t) \neq 0$. We say $\boldsymbol{v}^{\boldsymbol{y}} = 0$ if no such a_t exists.

Definition 11 ($S_{[t,j]}$). Given $\boldsymbol{y}, \boldsymbol{y}' \in D^{t-1}$, we say that $\boldsymbol{y}$ and $\boldsymbol{y}'$ are equivalent, denoted $\boldsymbol{y} \equiv_t \boldsymbol{y}'$ if $\boldsymbol{v}^{\boldsymbol{y}} = \boldsymbol{v}^{\boldsymbol{y}'}$. We denote the equivalence classes induced by this equivalence relation $S_{[t,1]}, \ldots, S_{[t,m_t]}$.

We often use $\equiv$ instead of $\equiv_t$, when t is clear from context.

In general, this allows us to write

$$Z(I) = F_I(a_1, \ldots, a_n) \prod_{t \in \{1,\ldots,n\}} \left(\sum_{b \in D} \boldsymbol{v}^{a_1,\ldots,a_{t-1}}[b] \right). \tag{14}$$

If our equivalence classes are well structured and few in number, we can compute this efficiently.

We are now nearly ready to state the Block Orthogonality Condition. Note that while the complete formulation of the Dichotomy Theorem for w#CSPs is for complex-valued constraints, we simplify the statements here to cover only the real case. Readers can refer to [8] for coverage of the general complex case or to [57] for the simpler non-negative case.

Definition 12 (Block-Orthogonal). Consider two vectors $\boldsymbol{a}$ and $\boldsymbol{b} \in \mathbb{R}^k$, and define $|\boldsymbol{a}| = (|a_1|, \ldots, |a_k|)$ and $|\boldsymbol{b}| = (|b_1|, \ldots, |b_k|)$. Then $\boldsymbol{a}$ and $\boldsymbol{b}$ are said to be block orthogonal if the following hold:

$|\boldsymbol{a}|$ and $|\boldsymbol{b}|$ are linearly dependent (i.e. they are scalar multiples of one another).

For every distinct value $a \in \{|a_1|, \ldots, |a_k|\}$, letting $T_a = \{j \in [k] : |a|_j = a\}$ be the set of all indices j on which $|a|_j = a$, we have $\sum_{j \in T_a} a_j b_j = 0$.

Note that if two vectors $\boldsymbol{a}$ and $\boldsymbol{b} \in \mathbb{R}^k$ are block orthogonal, then $|\boldsymbol{a}|$ and $|\boldsymbol{b}|$ are linearly dependent, but $\boldsymbol{a}$ and $\boldsymbol{b}$ need not be.

Definition 13 (Block-Orthogonality Condition). We say that a constraint language $\mathcal{F}$ satisfies the block orthogonality condition if, for every function $F \in \mathcal{F}$, $t \in [n]$, and $\boldsymbol{y}, \boldsymbol{z} \in D^{t-1}$, the row vectors $F^{[t]}(\boldsymbol{y}, \cdot)$ and $F^{[t]}(\boldsymbol{z}, \cdot)$ are either block orthogonal or linearly dependent.

The Dichotomy Theorem for w#CSPs (Theorem 1) requires two other criteria: the Mal'tsev Condition and the Type-Partition Condition. They are reviewed in Appendix C and more thoroughly in [8], but they are unnecessary for the remainder of the paper.

When possible, representing an arbitrary weighted counting problem as a w#CSP (such as Ising partition function computation), allows us to more easily assess its hardness. This analysis can produce a polynomial-time algorithm to solve a problem when the problem is found to be in *FP*. Note that the question of determining whether or not a problem with complex (or even real-valued) weights meets or fails these dichotomy criteria is not known to be decidable [8]. While the Block-Orthogonality Condition is easy to check, the Mal'tsev and Type-Partition Conditions, which impose requirements on all F_I's, are not. In the non-negative real case, there is a separate but analogous dichotomy theorem, which is known to be decidable [57]. Regardless, in many cases it is not hard to determine manually whether a given problem instance satisfies the criteria. An example in the case of the Ising problem can be found in Lemma 2.

C. Ising as a w#CSP

For Ising models, it is possible to reduce computation of the partition function to a w#CSP. In doing so, however, we must be explicit about the instances over which such a formulation is defined. The Ising problem corresponding to #*CSP*(Ising) includes all instances whose topologies are (finite) simple graphs at any temperature. Note that there is no requirement that $h = 0$.

Definition 14 (#CSP(Ising)). Let A_1 be the set of functions $\{f : \{-1, 1\} \to \mathbb{R} \mid f(a) = \lambda^a \mid \lambda > 0\}$, and let A_2 be the set of functions $\{f : \{-1, 1\}^2 \to \mathbb{R} \mid f(a, b) = \lambda^{ab} \mid \lambda > 0\}$. Take $\mathcal{F} = A_1 \cup A_2$. Then define #*CSP*(Ising) = #*CSP*($\mathcal{F}$).

The functions in A_1 correspond to the effect of the external field (h) on the lattice sites, and the functions in A_2 correspond to interactions between lattice sites.

Often we restrict our constraint language to only include rational-valued functions. This makes all the values we deal with finitely representable, which is required for our problem to be computable. For simplicity, we omit this detail from our discussion of #*CSP*(Ising). In real-world computation, values are expressed as floating points [58], and we accept the small inaccuracies that entails. Thus, the instances we are interested in in practice are rational-valued. Readers interested in attempts to eliminate floating point inaccuracies might refer to Schwarz [59].

Given the structural similarities between computation of an Ising partition function and #*CSP*(Ising), the equivalence of these problems should not be surprising. Recall that polynomial equivalence is discussed at the end of Sec. IV A.

Lemma 1. #*CSP*(Ising) is polynomially equivalent to the following problem: Given an inverse temperature and an Ising model whose topology is a (finite) simple graph, compute the associated partition function.

Proof. Given an Ising Model (Λ, J, h) and an inverse temperature $\beta \geqslant 0$, we set

$F_{i,j} \in \mathcal{F}$ so that $F_{i,j}(1, 1) = F_{i,j}(-1, -1) = e^{\beta J_{i,j}}$ and $F_{i,j}(1, -1) = F_{i,j}(-1, 1) = e^{-\beta J_{i,j}}$,

$F_i \in \mathcal{F}$ so that $F_i(1) = e^{\beta h_i}$ and $F_i(-1) = e^{-\beta h_i}$.

Then we have an instance (I, n) of #*CSP*(Ising) where $n = |\Lambda|$ and $I = \{F_{i,j}(i, j) : i, j \in \Lambda\} \cup \{h_i(i) : i \in \Lambda\}$. See that $Z(I) = Z_\beta$, the solution to the instance of #*CSP*(Ising), is equal to the partition function at the inverse temperature β.

Similarly, see that any instance of #*CSP*(Ising) can be converted (in polynomial time) to an instance of partition function computation of an Ising model by the reverse construction. ■

Thus #*CSP*(Ising) and computing partition functions of Ising models are polynomially equivalent problems. So we can easily determine the hardness of the problem of computing the partition function of the Ising model by applying the Dichotomy Theorem to #CSP(Ising). However, the Dichotomy Theorem for w#CSP applies only to finite constraint languages. We have defined #CSP(Ising) with an infinite constraint language, so we cannot directly apply the Dichotomy Theorem to #CSP(Ising).

To discuss hardness in a meaningful way, we consider a simple finite subproblem of #CSP(Ising) (i.e., a w#CSP whose constraint language is a finite subset of that of #CSP(Ising)). If we show that this subproblem is #P-hard, then #CSP(Ising) must also be #P-hard. Intuitively, if #CSP(Ising) is believed to have no polynomial time solution on a subset of its valid instances, then there should be no solution for #CSP(Ising) that is polynomial time on the entire set of valid problem instances. In general, if a subproblem of a problem is #P-hard, then the problem itself is #P-hard as well.

For example, one can consider weighted model counting of Boolean formulas in CNF (Definition 6) as a subproblem of WMC (Definition 7). Weighted model counting of CNF formulas can be shown to be #P-hard, and from this fact we can determine that WMC is #P-hard as well. Note that the converse does not hold; not all subproblems of a #P-hard problem are guaranteed to be #P-hard (e.g., partition functions of planar Ising models can be computed in polynomial time even though the general case is #P-hard [1]).

The instances of our subproblem of #CSP(Ising) correspond to Ising models with no external field ($h = 0$) whose interactions J are integer multiples of some positive $a \neq 1$. We construct this subproblem as follows: Let $a' \neq 1$ be a positive rational number, and define $a = \log a'$. Define $\mathcal{F}_a = \{f_a\}$, where $f_a : \{-1, 1\}^2 \to \mathbb{R}$ is

$$f_a(x, y) = \begin{cases} e^a & x = y \\ e^{-a} & x \neq y \end{cases}. \quad (15)$$

Clearly, $\mathcal{F}_a$ is a subset of the constraint language used to define #CSP(Ising), so #$CSP(\mathcal{F}_a)$ is a subproblem of #CSP(Ising).

Lemma 2. #$CSP(\mathcal{F}_a)$ does not satisfy the Block-Orthogonality Condition

Proof. We will show that #$CSP(\mathcal{F}_a)$ does not satisfy the Block-Orthogonality Condition, and thus that it is #P-hard. Consider $F^{[2]}$ as a matrix,

$$F^{[2]} = \begin{bmatrix} e^a & e^{-a} \\ e^{-a} & e^a \end{bmatrix}. \quad (16)$$

Clearly, the rows of $F^{[2]}$ are not linearly dependent ($\frac{e^a}{e^{-a}} = e^{2a} \neq e^{-2a} = \frac{e^{-a}}{e^a}$). Furthermore, $|F^{[2]}(0)|$ and $|F^{[2]}(1)|$ are not linearly dependent, so the rows of $F^{[2]}$ are not block-orthogonal. We conclude that #$CSP(\mathcal{F}_a)$ does not satisfy the Block-Orthogonality Condition. ■

Since #$CSP(\mathcal{F}_a)$ does not satisfy the Block-Orthogonality Condition, it is #P-hard by our earlier Dichotomy Theorem. The next theorem follows.

Theorem 2. #CSP(Ising) is #P-hard.

To summarize, we first saw that computing the partition function of Ising models whose interactions are integer multiples of some positive rational $a \neq 1$ is #P-hard. We then determined that computing the partition function of Ising models in general is #P-hard. These results followed immediately from a straightforward application of the Dichotomy Theorem for w#CSPs rather than a laborious bespoke reduction. Furthermore, our application of the Dichotomy Theorem provides some intuition on where the difficulty of Ising partition function computation comes from (i.e., the large number of $S_{t,\cdot}$ equivalence classes, corresponding to the difficulty in relating the change of a single lattice site's spin to a change in probability across configurations). This connection is made more apparent by the explanation of the Block-Orthogonality Condition in Appendix B.

Insofar as hardness of subproblems is concerned, our w#CSP formulation does not cover all cases. We might also consider the case where we restrict Ising models' interactions to take on values of either a or 0. In many physical systems of interest, only a single type of interaction with a fixed strength can occur (e.g., in models of ferromagnetism with only nearest-neighbor interactions) [60]. This formulation of the Ising problem is unfortunately not easily expressible as a w#CSP, since constraint functions can be applied arbitrarily many times to the same inputs. While the w#CSP approach is not applicable here, it is known that even in this restricted case computing the Ising partition function is #P-hard. This follows from a reduction of #MAXCUT to polynomially many instances of Ising model partition function computation [54]. This reduction from #MAXCUT gives a stronger statement than our application of the w#CSP Dichotomy Theorem, but it requires much more work than applying an out-of-the-box theorem. Specifically, the instances of Ising partition function computation are used to approximate the integer-valued solution to #MAXCUT, and the solution to #MAXCUT is recovered once the error of the approximation is sufficiently small (<0.5).

There is also often interest in hardness when we restrict our attention to planar Ising models [1]. This case too is not easily represented as a w#CSP, since w#CSPs do not give us control over problem topologies. However, it is known that the partition functions of planar Ising models (with no external field) are polynomially computable [1]. The typical approach is to reduce the problem of computing the partition function to a weighted perfect matching problem and apply the Fisher-Kasteleyn-Temperley (FKT) Algorithm. This solution is given in great detail in [1].

The inability of w#CSP to capture these Ising subproblems demonstrates the limitations of the w#CSP framework and opportunities to expand beyond them. For example, there has been recent work on *Holant Problems*, a class of problems broader than w#CSP, which adds the ability to control how many times variables may be reused [61]. The Holant framework still does not allow the degree of specificity needed to easily capture some Ising formulations, although it is perhaps a step in the right direction. While these problem classes and their accompanying dichotomy theorems are intended to answer deep theoretical questions about the source of computational hardness, applying them to real-world problems gives very immediate and useful hardness results. Application to real-world problems also suggests where the expressiveness of problem classes like w#CSP can be improved. As the expressiveness of these problem classes increases, so does our ability to determine the hardness of precisely defined subproblems [61].

V. CONCLUSIONS

The principal aim of our work has been to demonstrate the utility of casting Ising-model partition-function computation to standard forms in the field of computer science (i.e., w#CSP and WMC). Because these standard forms are well studied, they provide immediate theoretical and computational results without the need for laborious, handcrafted techniques as in Jerrum [54] and Pan [25].

By representing partition-function computation as a w#CSP, we were able to determine the computational complexity of computing Ising model partition functions with minimal effort. While lacking the flexibility of bespoke reductions, the w#CSP framework serves as a very easy, out-of-the-box way to determine the hardness of some problems for Ising models. This w#CSP representation also provided some intuition on where the difficulty of Ising partition-function computation comes from (i.e., the difficulty in relating the change of a lattice site's spin to a change in probability across configurations).

From our #wCSP formulation, we were inspired to develop a reduction from Ising-model partition-function computation to weighted model counting (WMC), a well-studied problem for which many off-the-shelf solvers exist. Some exact WMC solvers, most notably `TensorOrder`, were able to outperform state-of-the-art approximate tools used for computing partition functions on midsized and disordered benchmarks, running in a tenth of the time in many cases. `TensorOrder`'s success on disordered topologies makes previously challenging Ising instances easier in practice. `TensorOrder` (and weighted model counters more generally) shows promise as part of a portfolio of Ising parition function solvers. Moreover, `Tensororder` is capable of handling more general constraints and higher rank tensors than required to compute Ising model partition functions, suggesting its use for more complex models (e.g., Potts models). Furthermore, as weighted model counters continue to improve, Ising-model partition-function computation performed using weighted model counters will improve in lockstep.

While our work offers effective strategies for partition function computation in most cases, the zero-temperature limit ($\beta \to \infty$) remains challenging for both exact and approximate tools. Nonetheless, Liu *et al.* [2] recently showed that if one replaces the usual sum and product binary operators for ordinary real numbers with the max and sum operators respectively (i.e., if one operates over the tropical algebra), then partition function computations is feasible in the zero-temperature limit. We believe that the computational complexity analyses and Ising reductions to weighted model counters offered in our study for finite β in partition function computations may pave the way for future tropical algebra counters, especially as they rely on the common language of tensor network contractions [15,16].

While we have focused primarily on the Ising model in this paper, the same type of analysis is easily done for Potts models and other related models. Our hope is that by demonstrating the utility of the w#CSP and WMC frameworks in this setting, readers will be encouraged to apply them to other settings and problems of interest in statistical physics, engineering reliability, or probabilistic inference.

Furthermore, we believe that weighted counters could be applied in the computation of the partition function of non classical models such as the quantum Ising model. The main justification is that computing the ground state degeneracy and density of states for both classical and quantum Hamiltonians is computationally equivalent [62]. A systematic study of the computational complexity of the partition function of non-classical models and the promise of using existing weighted counters in this context is left as a promising avenue for future research.

In addition, the approach of Sec. III may complement emerging strategies in quantum simulation. Many techniques in this area are limited to restricted topologies (e.g., Tindall [63] is effective only over short-range interactions), so the application of weighted model counters may support broader simulation in the future.

We provide all code, benchmarks, and detailed data from benchmark runs at [64].

ACKNOWLEDGMENTS

This work was supported in part by the Big-Data Private Cloud Research Cyberinfrastructure MRI award funded by the NSF under Grant No. CNS-1338099 and a Civil Infrastructure Systems award also funded by the NSF through Grant No. CMMI-2037545. We also acknowledge the use of computational resources from Rice University's Center for Research Computing (CRC). The authors are also grateful for discussions with members of the Quantum Algorithms Group at Rice University, including Professors Kaden Hazzard and Guido Pagano (Physics and Astronomy), as well as Anastasios Kyrillidis (Computer Science).

APPENDIX A: REDUCING W#CSPS TO WEIGHTED MODEL COUNTING

Every instance of a weighted counting constraint-satisfaction problem (w#CSP, Definition 5) can be reduced to an equivalent instance of weighted model counting (WMC, Definition 7) expressed in conjunctive normal form (CNF, Definition 6). One particular reduction for computing the partition function of the Ising model is given in Sec. III. In this Appendix, we give a more general reduction from an aribtrary w#CSP to WMC.

Lemma 3. For every constraint language $\mathcal{F}$, there is a polynomial-time reduction reduction from $\#CSP(\mathcal{F})$ to WMC.

Proof. Consider a constraint language $\mathcal{F}$ with a domain $D = \{0, 1, \dots, |D| - 1\}$, and the associated problem $\#CSP(\mathcal{F})$. Consider an instance (I, n) over the variables $x_1, \dots, x_n$. Let $\overline{D} = \lceil \log_2(|D|) \rceil$. We reduce (I, n) to a WMC instance as follows.

We first define the set X of Boolean variables in the WMC instance.

$x_{j,1},\ldots,x_{j,\overline{D}}$: For each of the n variables x_j in the input instance, we add $\overline{D}$ corresponding Boolean variables $x_{j,1},\ldots,x_{j,\overline{D}}$ to X. These variables encode the value of each x_j. In particular, the value of x_j corresponds to the binary number that results from concatenating the values of Boolean variables: $x_{j,\overline{D}}\cdots x_{j,1}$.

$x_{(i_1,\ldots,i_k),\boldsymbol{d}}$: For each formula $F(x_{i_1},\ldots,x_{i_k})$ in I, we add $|D|^k$ Boolean variables $x_{(i_1,\ldots,i_k),\boldsymbol{d}}$, one for each $\boldsymbol{d}\in D^k$. These variables encode the value of the input variables $(x_{i_1},\ldots,x_{i_k})$ of each formula $F(x_{i_1},\ldots,x_{i_k})$ in I.

We now introduce the following formulas.

First, we must restrict the allowed values of the variables $x_{j,1},\ldots,x_{j,\overline{D}}$ so that the value of x_j is in D. Specifically, we want to disallow the binary number $x_{j,\overline{D}}\cdots x_{j,2}x_{j,1}$ from exceeding $|D|-1$. The most straightforward way to express this condition is that $x_{j,\overline{D}}\cdots x_{j,2}x_{j,1}$ should not be greater than the binary representation of $|D|-1$, written $B_{\overline{D}}\cdots B_2B_1$, with respect to the following strict lexicographic order ($<$) on binary strings of equal length:

$0<1$

$0\omega_a<1\omega_b$ for arbitrary binary strings ω_a and ω_b of equal length

$0\omega_a<0\omega_b$ and $1\omega_a<1\omega_b$ if $\omega_a<\omega_b$ for arbitrary binary strings ω_a and ω_b of equal length.

For $a,b\in\{0,1\}$ and binary strings of equal length ω_a and ω_b, we have that $a\omega_a<b\omega_b$ if $a<b$ or if $a=b$ and $\omega_a<\omega_b$. We now construct inductively a formula $\phi(a_m\cdots a_1,b_m\cdots b_1)$ that checks whether $a_m\cdots a_1<b_m\cdots b_1$, defined as follows:

$$\phi(a_m\cdots a_1,b_m\cdots b_1) \tag{A1}$$

$$=\begin{cases}(a_m<b_m) & m>1\\ \quad\vee((a_m=b_m)\wedge\phi(a_{m-1}\cdots a_1,b_{m-1}\cdots b_1)) & \\ (a_m<b_m) & m=1\end{cases} \tag{A2}$$

$$\begin{aligned}=&(a_m<b_m)\\ &\vee((a_m=b_m)\wedge(a_{m-1}<b_{m-1}))\\ &\vee((a_m=b_m)\wedge(a_{m-1}=b_{m-1})\wedge(a_{m-2}<b_{m-2}))\\ &\vee\cdots\end{aligned} \tag{A3}$$

$$=\bigvee_{p\in[m]}\left(\left(\bigwedge_{q\in[m],q>p}(a_q=b_q)\right)\wedge(a_p<b_p)\right) \tag{A4}$$

$$=\bigvee_{p\in[m],a_p=0}\left(\left(\bigwedge_{q\in[m],q>p}(a_q=b_q)\right)\wedge(a_p<b_p)\right) \tag{A5}$$

$$=\bigvee_{p\in[m],a_p=0}\left(\left(\bigwedge_{q\in[m],q>p}(a_q=b_q)\right)\wedge(b_p=1)\right). \tag{A6}$$

The last two simplifications come from the fact that $a_p<b_p$ can only be true if $a_p=0$ and $b_p=1$.

Next, we construct a Boolean formula to capture $\neg\phi(B_{\overline{D}}\cdots B_2B_1,x_{j,\overline{D}}\cdots x_{j,2}x_{j,1})$.

Now define

$$\phi_j=\neg\phi\big(B_{\overline{D}}\cdots B_2B_1,x_{j,\overline{D}}\cdots x_{j,2}x_{j,1}\big), \tag{A7}$$

and note that terms in ϕ_j checking equality can be replaced with positive and negative literals over $x_{j,\overline{D}},\ldots,x_{j,2},x_{j,1}$ determined by the values of $B_{\overline{D}},\ldots,B_2,B_1$. As the negation of a DNF formula, ϕ_j is a CNF formula.

We now have a formula ϕ_j that is satisfied exactly when $x_{j,\overline{D}}\cdots x_{j,2}x_{j,1}\leqslant|D|-1$.

Next, we want to relate the variables that encode the inputs to the constraint functions (i.e. $x_{(i_1,\ldots,i_k),\boldsymbol{d}}$) to the variables encoding the value of x_j (i.e. $x_{j,1},\ldots,x_{j,\overline{D}}$). To do this, we introduce the following shorthand:

$$\phi_{j,d}=\bigwedge_{q\in[\overline{D}]}l_{j,q,d}, \tag{A8}$$

where $d\in D$ with the binary representation $d_{\overline{D}}\cdots d_1$, and $l_{j,q,d}=\{\begin{smallmatrix}x_{j,q} & d_q=1\\ \neg x_{j,q} & d_q=0\end{smallmatrix}$.

Evidently, $\phi_{j,d}$ is true exactly when $x_j=d$. We also introduce the following formulas:

$$\psi_{F(x_{i_1},\ldots,x_{i_k})}=\bigwedge_{\boldsymbol{d}\in D^k}\left(x_{(i_1,\ldots,i_k),\boldsymbol{d}}\Rightarrow\bigwedge_{p\in[k]}\phi_{i_p,\boldsymbol{d}_p}\right) \tag{A9}$$

$$= \bigwedge_{\boldsymbol{d}\in D^k} \left(\bigwedge_{p\in[k]} \left(\neg x_{(i_1,\ldots,i_k),\boldsymbol{d}} \vee \phi_{i_p,\boldsymbol{d}_p}\right) \right) \tag{A10}$$

$$= \bigwedge_{\boldsymbol{d}\in D^k, p\in[k]} \left(\neg x_{(i_1,\ldots,i_k),\boldsymbol{d}} \vee \bigwedge_{q\in[\overline{D}]} l_{j,q,\boldsymbol{d}_p} \right) \tag{A11}$$

$$= \bigwedge_{\boldsymbol{d}\in D^k, p\in[k]} \left(\bigwedge_{q\in[\overline{D}]} \left(\neg x_{(i_1,\ldots,i_k),\boldsymbol{d}} \vee l_{j,q,\boldsymbol{d}_p}\right) \right) \tag{A12}$$

$$= \bigwedge_{\boldsymbol{d}\in D^k, p\in[k], q\in[\overline{D}]} \left(\neg x_{(i_1,\ldots,i_k),\boldsymbol{d}} \vee l_{j,q,\boldsymbol{d}_p}\right) \tag{A13}$$

and

$$\gamma_{F(x_{i_1},\ldots,x_{i_k})} = \bigvee_{\boldsymbol{d}\in D^k} x_{(i_1,\ldots,i_k),\boldsymbol{d}}, \tag{A14}$$

which together force $x_{(i_1,\ldots,i_k),\boldsymbol{d}}$ to be true exactly when the value assigned to $(x_{i_1},\ldots,x_{i_k})$ is $\boldsymbol{d}$. Specifically, $\psi_{F(x_{i_1},\ldots,x_{i_k})}$ ensures that if $x_{(i_1,\ldots,i_k),\boldsymbol{d}}$ is true, then the value assigned to $(x_{i_1},\ldots,x_{i_k})$ is $\boldsymbol{d}$. Since the value assigned $(x_{i_1},\ldots,x_{i_k})$ will be equal to exactly one $\boldsymbol{d}\in D^k$, $x_{(i_1,\ldots,i_k),\boldsymbol{d}}$ can be true for at most one $\boldsymbol{d}\in D^k$. Moreover, from $\gamma_{F(x_{i_1},\ldots,x_{i_k})}$, we have that $x_{(i_1,\ldots,i_k),\boldsymbol{d}}$ is true for at least one $\boldsymbol{d}\in D^k$. Thus, $x_{(i_1,\ldots,i_k),\boldsymbol{d}}$ will be true exactly when the value assigned to $(x_{i_1},\ldots,x_{i_k})$ is $\boldsymbol{d}$.

All together, we write

$$\Phi = \left(\bigwedge_{j\in[n]} \phi_j\right) \wedge \bigwedge_{F(x_{i_1},\ldots,x_{i_k})\in I} \left(\psi_{F(x_{i_1},\ldots,x_{i_k})} \wedge \gamma_{F(x_{i_1},\ldots,x_{i_k})}\right). \tag{A15}$$

This formula Φ is the formula whose weighted count our WMC reduction of the (I,n) instance will compute.

Finally, we need a weight function W. For each $x_{j,q}$, $W(x_{j,q},1)=W(x_{j,q},0)=1$. For each $x_{(i_1,\ldots,i_k),\boldsymbol{d}}$, $W(x_{(i_1,\ldots,i_k),\boldsymbol{d}},0)=1$ and $W(x_{(i_1,\ldots,i_k),\boldsymbol{d}},1)=F(\boldsymbol{d})$.

All together, X, Φ, and W constitute the WMC instance produced by the reduction. Correctness of the reduction follows by construction. The variables $x_{j,\overline{D}},\ldots,x_{j,1}$ give a binary encoding of the value of x_j, thanks to ϕ_j. For each formula $F(x_{i_1},\ldots,x_{i_k})\in I$, the literals $x_{(i_1,\ldots,i_k),\boldsymbol{d}}$ indicate the value of the arguments of F, and the weight assigned to each literal $x_{(i_1,\ldots,i_k),\boldsymbol{d}}$ is exactly the value of $F(\boldsymbol{d})$. The total weight of an assignment to the variables of Φ is a product of these weights, which is precisely the product of the values taken by the formulas in I for the corresponding assignment to $x_1,\ldots,x_n$. So the weight W on each assignment over the Boolean variables is equal to the value of F_I on the corresponding assignment to $x_1,\ldots,x_n$. Since there is a bijection between assignments to the Boolean variables satisfying Φ and assignments to $x_1,\ldots,x_n$, we can show $W(\Phi)=Z(I)$. Thus the WMC instance that the reduction produces is equivalent to the $\#CSP(\mathcal{F})$ instance (I,n) as desired. ■

Note that we have $\overline{D}*n$ variables of the form $x_{j,q}$ and at most $|D|^K*|I|$ variables of the form $x_{(i_1,\ldots,i_k),\boldsymbol{d}}$, where K is the maximum arity of the constraints used in I. There are n formulas of the form ϕ_j in Φ, each with $\overline{D}$ literals. There are $|I|$ formulas of the form $\psi_{F(x_{i_1},\ldots,x_{i_k})}$ in Φ, each with at most $|D|^K*K*\overline{D}$ CNF clauses, each clause having two literals. Finally, there are $|I|$ formulas of the form $\gamma_{F(x_{i_1},\ldots,x_{i_k})}$, each with at most $|D|^K$ literals. The reduction of (I,n) is exponential only in K, the maximum arity of the functions appearing in I.

If the arities of the constraints comprising $\mathcal{F}$ are bounded, then the reduction is polynomial. Any finite constraint language $\mathcal{F}$ has bounded arity, but many useful infinite constraint languages have bounded arity as well. In the case of $\#CSP$(Ising), the functions in the associated (infinite) constraint language have arity at most 2.

For many particular problems, a more compact reduction can be found. For example, when there is symmetry in the constraints F in $\mathcal{F}$ (i.e., when the constraints are not injective), more efficient encodings of the constraints' inputs can be used.

APPENDIX B: HARDNESS AND RELATIONSHIP TO WEIGHTED CONSTRAINT SATISFACTION

In Sec. IV B we took a very brief look at the Dichotomy Theorem (Theorem 1), but we omitted much of the intuition behind its criteria. In this Appendix we develop the Block Orthogonality Condition more carefully, giving insight into the reason for the criterion. The Mal'tsev and Type Partition Conditions are handled in Appendix C.

As previously mentioned, the three criteria given by the Dichotomy Theorem follow naturally when we attempt to construct a polynomial-time algorithm for w#CSPs. One reason that w#CSPs are often computationally expensive to solve is that the effect of a single assignment to a given variable is hard to capture. In particular, given a problem instance I and its associated n-ary formula $F_I(\boldsymbol{x})$, it is challenging to find a general rule that relates $F_I(x_1,x_2,\ldots,x_n)$ to $F_I(x'_1,x_2,\ldots,x_n)$. For this reason, current computational approaches require consideration of a set of assignments whose size is exponential in n. The algorithm driving the w#CSP Dichotomy Theorem demands that changing the assignment of a particular variable changes the value of F_I in a predictable way. The criteria of the Dichotomy Theorem are necessary and sufficient conditions for these assumptions to hold.

We now establish some definitions to help us describe the effect of an assignment on the value of F_I. Recall the definition of an instance of a w#CSP (Definition 5). An instance of $\#CSP(\mathcal{F})$ is written as (I, n) with n a positive integer and I a finite collection of formulas each of the form $F(x_{i_1}, \ldots, x_{i_k})$. Here each $F \in \mathcal{F}$ is a k-ary function, $i_1, \ldots, i_k \in [n]$, and each variable x_{i_j} ranges over D. Note that k and $i_1, \ldots, i_k$ may differ in each formula in I. Given an instance (I, n), we define $F_I : D^n \to \mathbb{C}$ as follows, where $\boldsymbol{y} \in D^n$:

$$F_I(\boldsymbol{y}) = \prod_{F(x_{i_1},\ldots,x_{i_k})\in I} F(y[i_1], \ldots, y[i_k]). \tag{B1}$$

So F_I is a mapping that takes an assignment $\boldsymbol{y}$ to the variables $(\boldsymbol{x})$ and returns a value in $\mathbb{C}$, where the value assigned to each x_{i_j} is given by $y[i_j]$. The output of the instance (I, n) is the sum over all assignments $Z(I) = \sum_{\boldsymbol{y}\in D^n} F_I(\boldsymbol{y})$, as in Definition 5.

Note the similarity between this sum of products and our Ising partition function (Definition 2).

Our first step is to break up the computation of $Z(I)$ so that we can understand the effect each individual variable assignment has on its value. We would like to be able to split $Z(I)$ into a sum of smaller sums, each restricted to a particular partial assignment. Then we can consider how differences between these partial assignments affect their contributions to $Z(I)$.

In the development below, we take t to be an arbitrary member of $\{1, \ldots, n\}$.

Definition 15 ($F_I^{[t]}$). Let $F_I^{[t]} : D^t \to \mathbb{C}$ be defined as

$$F_I^{[t]}(y_1, \ldots, y_t) = \sum_{y_{t+1},\ldots,y_n\in D} F_I(y_1, \ldots, y_n). \tag{11}$$

So $F_I^{[t]}$ describes the impact on $Z(I)$ of all assignments agreeing with a given partial assignment to $x_1, \ldots, x_t$. We can decompose $Z(I)$ into a sum of such terms. Thus $Z(I) = \sum_{a\in D} F_I^{[1]}(a)$, and in general, we have that $Z(I) = \sum_{\boldsymbol{y}\in D^t} F_I^{[t]}(\boldsymbol{y})$.

We now consider $F_I^{[t]}$ as a $|D|^{t-1} \times |D|$ matrix, where $F_I^{[t]}(\boldsymbol{y}, d) = F_I^{[t]}(y_1, \ldots, y_{t-1}, d)$ for $\boldsymbol{y} \in D^{t-1}$ and $d \in D$:

$$F_I^{[t]} = \begin{bmatrix} F_I^{[t]}(\boldsymbol{y}^1, d_1) & \cdots & F_I^{[t]}(\boldsymbol{y}^1, d_{|D|}) \\ \vdots & \vdots & \vdots \\ F_I^{[t]}(\boldsymbol{y}^{|D|^{t-1}}, d_1) & \cdots & F_I^{[t]}(\boldsymbol{y}^{|D|^{t-1}}, d_{|D|}) \end{bmatrix} \tag{B2}$$

$$= \begin{bmatrix} \sum_{\boldsymbol{w}\in D^{n-t}} F_I(\boldsymbol{y}^1, d_1, \boldsymbol{w}) & \cdots & \sum_{\boldsymbol{w}\in D^{n-t}} F_I(\boldsymbol{y}^1, d_{|D|}, \boldsymbol{w}) \\ \vdots & \vdots & \vdots \\ \sum_{\boldsymbol{w}\in D^{n-t}} F_I(\boldsymbol{y}^{|D|^{t-1}}, d_1, \boldsymbol{w}) & \cdots & \sum_{\boldsymbol{w}\in D^{n-t}} F_I(\boldsymbol{y}^{|D|^{t-1}}, d_{|D|}, \boldsymbol{w}) \end{bmatrix}. \tag{B3}$$

Writing $F_I^{[t]}$ as a matrix invites us to consider the following question: How does changing the assignment of a single variable change the contribution of a partial assignment to the instance's output $Z(I)$? In our notation, how does changing d affect the value of $F_I^{[t]}(\boldsymbol{y}, d)$ for a given $\boldsymbol{y}$?

Definition 16 ($F_I^{[t]}(\boldsymbol{y}, \cdot)$). If we regard $F_I^{[t]}$ as a $|D|^{t-1} \times |D|$ matrix, we may refer to its rows by $F_I^{[t]}(\boldsymbol{y}, \cdot)$ as below. Here, $\boldsymbol{y} \in D^{t-1}$ and $D = \{d_1, \ldots, d_{|D|}\}$. Thus

$$F_I^{[t]}(\boldsymbol{y}, \cdot) = \left[\sum_{\boldsymbol{w}\in D^{n-t}} F_I(\boldsymbol{y}, d_1, \boldsymbol{w}), \quad \ldots, \quad \sum_{\boldsymbol{w}\in D^{n-t}} F_I(\boldsymbol{y}, d_{|D|}, \boldsymbol{w})\right]. \tag{12}$$

We now have a way to relate similar partial assignments. In particular, for each partial assignment $\boldsymbol{y}$, the vector $F_I^{[t]}(\boldsymbol{y}, \cdot)$ captures the contributions to $Z(I)$ of partial assignments to the first t variables that agree with $\boldsymbol{y}$ on the first $t-1$ variables. These vectors are now our objects of interest.

The set of such vectors $F_I^{[t]}(\boldsymbol{y}, \cdot)$ is quite large, since there are $|D|^{t-1}$ choices for $\boldsymbol{y}$. We would like to reduce the computation and information needed to manage this set. We make the observation that if two vectors $F_I^{[t]}(\boldsymbol{y}, \cdot)$ and $F_I^{[t]}(\boldsymbol{y}', \cdot)$ are scalar multiples of one another, then if we know $F_I^{[t]}(\boldsymbol{y}, \cdot)$ and a nonzero entry of $F_I^{[t]}(\boldsymbol{y}', \cdot)$, we can compute $F_I^{[t]}(\boldsymbol{y}', \cdot)$ easily. This motivates us to introduce an equivalence relation on our set of partial assignments $\boldsymbol{y}$. Given $\boldsymbol{y}, \boldsymbol{y}' \in D^{t-1}$, we say that $\boldsymbol{y} \equiv \boldsymbol{y}'$ if the vectors $F_I^{[t]}(\boldsymbol{y}, \cdot)$ and $F_I^{[t]}(\boldsymbol{y}', \cdot)$ are scalar multiples of one another. This notion is formalized below in Definition 12.

If we have a small number of equivalence classes and if it is easy to determine to which equivalence class a partial assignment $\boldsymbol{y} \in D^{t-1}$ belongs to, then using this equivalence relation would make it easier to compute and manage the $F_I^{[t]}(\boldsymbol{y}, \cdot)$ vectors. The dichotomy criteria of Theorem 1 guarantee both these desiderata.

If we want our partition to be useful, we need to assign to each equivalence class a representative element. Then for each $\boldsymbol{y}$ in the equivalence class, we can compute $F_I^{[t]}(\boldsymbol{y}, \cdot)$ by scaling this representative. We construct such a vector of dimension $|D|$ as follows.

Definition 17 ($\boldsymbol{v}^{\boldsymbol{y}}$). Given $\boldsymbol{y} \in D^{t-1}$ and a total order on D, we define

$$\boldsymbol{v}^{\boldsymbol{y}} = \frac{F_I^{[t]}(\boldsymbol{y}, \cdot)}{F_I^{[t]}(\boldsymbol{y}, a_t)} \tag{13}$$

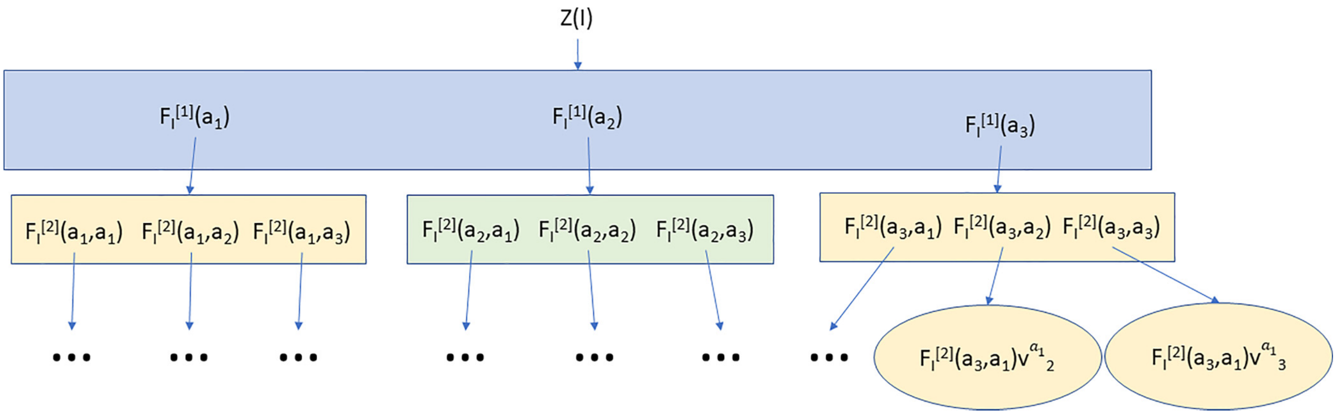

FIG. 3. Diagram of contributions of partial assignments to $Z(I)$. The colors of each vector denote its equivalence class, thus $a_1 \equiv a_3$. When computing $F_I^{[2]}(a_3, a_2)$, we can use $\boldsymbol{v}^{a_1}$, which we already know from $F_I^{[2]}(a_1, \cdot)$. In this way, we need only check one partial assignment to two variables (a_3, a_1) to determine $F_I^{[2]}(a_3)$ when we otherwise would have needed to check three. As we work further down the tree, repeated applications of this shortcut result in an exponential speedup (assuming we have sufficiently few equivalence classes, guaranteed by the criteria of Theorem 1).

for the least $a_t \in D$ so that $F_I^{[t]}(\boldsymbol{y}, a_t) \neq 0$. We say $\boldsymbol{v}^{\boldsymbol{y}} = 0$ if no such a_t exists.

Observe that for an arbitrary $\boldsymbol{y} \in D^{t-1}$, we have that $F_I^{[t]}(\boldsymbol{y}, \cdot) = F_I^{[t]}(\boldsymbol{y}, a_t) \cdot \boldsymbol{v}^{\boldsymbol{y}}$. In addition, given $\boldsymbol{y}, \boldsymbol{y}' \in D^{t-1}$, if $\boldsymbol{y} \equiv \boldsymbol{y}'$ as discussed earlier, then $\boldsymbol{v}^{\boldsymbol{y}} = \boldsymbol{v}^{\boldsymbol{y}'}$. This motivates the following formalization of the equivalence relation discussed above.

Definition 18 ($S_{[t,j]}$). Given $\boldsymbol{y}, \boldsymbol{y}' \in D^{t-1}$, we say that $\boldsymbol{y}$ and $\boldsymbol{y}'$ are equivalent, denoted $\boldsymbol{y} \equiv_t \boldsymbol{y}'$ if $\boldsymbol{v}^{\boldsymbol{y}} = \boldsymbol{v}^{\boldsymbol{y}'}$. We denote the equivalence classes induced by this equivalence relation $S_{[t,1]}, \dots, S_{[t,m_t]}$.

We often use $\equiv$ instead of $\equiv_t$, when t is clear from context. We now have a way to discuss symmetries between partial assignments in D^{t-1}. We can consider computing $Z(I)$ in this framework to understand what advantages we have gained. This computation can be visualized as in Fig. 3. We regard $Z(I)$ as $F_I^{[0]}()$ and compute it recursively, traversing the tree in Fig. 3 in a depth-first traversal.

Consider the vector $\boldsymbol{v} = \frac{F_I^{[1]}(\cdot)}{F_I^{[1]}(a_1)} \in D^{|D|}$, where a_1 is the least element of D such that $F_I^{[1]}(a_1) \neq 0$ as in Definition 11. We denote entries of $\boldsymbol{v}$ as $\boldsymbol{v}[a] = \frac{F_I^{[1]}(a)}{F_I^{[1]}(a_1)}$. Observe

$$Z(I) = F_I^{[0]}() = \sum_{a \in D} F_I^{[1]}(a) \tag{B4}$$

$$= \sum_{a \in D} F_I^{[1]}(a_1) \frac{F_I^{[1]}(a)}{F_I^{[1]}(a_1)} \tag{B5}$$

$$= \sum_{a \in D} F_I^{[1]}(a_1) \boldsymbol{v}[a] \tag{B6}$$

$$= F_I^{[1]}(a_1) \sum_{a \in D} \boldsymbol{v}[a]. \tag{B7}$$

So to compute $Z(I)$, one needs to compute $F_I^{[1]}(a_1)$ and $\boldsymbol{v}$. We first compute $F_I^{[1]}(a_1)$. As per Definition 11, consider the least $a_2 \in D$ so that $F_I^{[2]}(a_1, a_2) \neq 0$ and write $\boldsymbol{v}^{a_1} = \frac{F_I^{[2]}(a_1, \cdot)}{F_I^{[2]}(a_1, a_2)}$. Then

$$F_I^{[1]}(a_1) = \sum_{b \in D} F_I^{[2]}(a_1, b) \tag{B8}$$

$$= \sum_{b \in D} F_I^{[2]}(a_1, a_2) \frac{F_I^{[2]}(a_1, b)}{F_I^{[2]}(a_1, a_2)} \tag{B9}$$

$$= \sum_{b \in D} F_I^{[2]}(a_1, a_2)(\boldsymbol{v}^{a_1}[b]) \tag{B10}$$

$$= F_I^{[2]}(a_1, a_2) \sum_{b \in D} \boldsymbol{v}^{a_1}[b]. \tag{B11}$$

In order to finish computing $F_I^{[1]}(a_1)$, we must compute $F_I^{[2]}(a_1, a_2)$ and $\boldsymbol{v}^{a_1}$. We continue walking down our tree in this way until we reach the leaves where $F_I^{[n]} = F_I$ is easily computable. When we compute our $\boldsymbol{v}^{\boldsymbol{y}}$ vectors, we find each entry as per Definition 11 by visiting unexplored branches of the tree in a depth-first fashion. However, we can apply the equivalence relation $\equiv$ to avoid computing $\boldsymbol{v}^{\boldsymbol{y}}$ when it is known from earlier computation of an equivalent partial assignment. This speedup is explained next in the context of the top level of our tree.

After computing $F_I^{[1]}(a_1)$, we must compute $\boldsymbol{v}$. For each $a_1' \in D$ such that $a_1 \equiv a_1'$, we have that

$$\boldsymbol{v}[a_1'] = \frac{F_I^{[1]}(a_1')}{F_I^{[1]}(a_1)} \tag{B12}$$

$$= \frac{1}{F_I^{[1]}(a_1)} F_I^{[2]}(a_1', a_2) \sum_{b \in D} (\boldsymbol{v}^{a_1'}[b]) \tag{B13}$$

$$= \frac{1}{F_I^{[1]}(a_1)} F_I^{[2]}(a_1', a_2) \sum_{b \in D} (\boldsymbol{v}^{a_1}[b]). \tag{B14}$$

Since we have already computed $\boldsymbol{v}^{a_1}$ in our computation of $F_I^{[1]}(a_1)$, we need only compute $F_I^{[2]}(a_1', a_2)$ to determine $\boldsymbol{v}[a_1']$. This reduces the number of partial assignments of length 2 we must explore to determine $\boldsymbol{v}[a_1']$ by a factor of n (see Fig. 3). As we proceed with our computation of various $\boldsymbol{v}^{\boldsymbol{y}}$ and $F_I^{[t]}(\cdot)$, the ability to reuse previously computed $\boldsymbol{v}$'s substantially reduces the search space we must explore (supposing the number m_t of equivalence classes of $\equiv_t$ is small enough $[m_t << |D|^{t-1}$ for each $t \in \{1, \ldots, n\}]$).

More generally, we may write

$$Z(I) = F_I(a_1, \ldots, a_n) \prod_{t\in\{1,\ldots,n\}} \left(\sum_{b\in D} \boldsymbol{v}^{a_1,\ldots,a_{t-1}}[b] \right), \quad \text{(B15)}$$

and as we have just seen, the computation of each $\boldsymbol{v}^{a_1,\ldots,a_{t-1}}$ is made easier each time we have $(a_1, \ldots, a_{t-1}, d) \equiv (a_1, \ldots, a_{t-1}, d')$ for distinct $d, d' \in D$.

For this approach to be useful, we must be able to determine in polynomial time to which equivalence class $S_{[t,j]}$ a given $\boldsymbol{y}$ belongs (so that we can reuse $\boldsymbol{v}^{\boldsymbol{y}}$). The Dichotomy Theorem criteria are precisely the conditions required for the collection of these data to be determined in polynomial time.

While the complete formulation of the Dichotomy Theorem for w#CSPs is for complex-valued constraints, we simplify the statements here to cover only the real case. Interested readers can refer to [8] for coverage of the general complex case or to [57] for the simpler non-negative case. First, we go over the three criteria of the Dichotomy Theorem.

The *Block-Orthogonality Condition* guarantees that the number m_t of equivalence classes of $\equiv_t$ is not too large.

Definition 19 (Block-Orthogonal). Consider two vectors $\boldsymbol{a}$ and $\boldsymbol{b} \in \mathbb{R}^k$, and define $|\boldsymbol{a}| = (|a_1|, \ldots, |a_k|)$ and $|\boldsymbol{b}| = (|b_1|, \ldots, |b_k|)$. Then $\boldsymbol{a}$ and $\boldsymbol{b}$ are said to be block orthogonal if the following hold:

$|\boldsymbol{a}|$ and $|\boldsymbol{b}|$ are linearly dependent (i.e. they are scalar multiples of one another).

For every distinct value $a \in \{|a_1|, \ldots, |a_k|\}$, letting $T_a = \{j \in [k] : |a|_j = a\}$ be the set of all indices j on which $|a|_j = a$, we have $\sum_{j\in T_a} a_j b_j = 0$.

Note that if two vectors $\boldsymbol{a}$ and $\boldsymbol{b} \in \mathbb{R}^k$ are block orthogonal, then $|\boldsymbol{a}|$ and $|\boldsymbol{b}|$ are linearly dependent, but $\boldsymbol{a}$ and $\boldsymbol{b}$ need not be.

Definition 20 (Block-Orthogonality Condition). We say that a constraint language $\mathcal{F}$ satisfies the block orthogonality condition if, for every function $F \in \mathcal{F}$, $t \in [n]$, and $\boldsymbol{y}, \boldsymbol{z} \in D^{t-1}$, the row vectors $F^{[t]}(\boldsymbol{y}, \cdot)$ and $F^{[t]}(\boldsymbol{z}, \cdot)$ are either block orthogonal or linearly dependent.

The Dichotomy Theorem for w#CSPs (Theorem 1) requires two other criteria: the Mal'tsev Condition and the Type-Partition Condition. These conditions make computing membership in $S_{[t,j]}$ tractable. They are reviewed in Appendix C and more thoroughly in [8].

APPENDIX C: THE MAL'TSEV AND TYPE PARTITION CONDITIONS

Recall the discussion from Sec. IV B. We consider there a problem instance (I, n) and the associated instance function F_I. For each $t < n$, we partition the partial assignments D^{t-1} into equivalence classes $S_{[t,j]}$ based on the contributions of each partial assignment to $Z(I)$. We take m_t to be the number of these equivalence classes for each t. The discussion then proceeds with an analysis of these objects to determine conditions under which a problem is polynomially solvable. Section IV B introduces the Block-Orthogonality Condition. Here we review the remaining two conditions: the Mal'tsev Condition and the Type-Partition Condition.

The Block-Orthogonality Condition given in Sec. IV B gives us that m_t is reasonably small. This means that the speedup we get from using the equivalence relation will be substantial, provided we are able to easily identify equivalent partial assignments.

To make this identification, we require that membership in $S_{[t,j]}$ is computable in polynomial time. This requirement is guaranteed by the *Mal'tsev Condition*. In particular, the Mal'tsev Condition, by requiring the existence of Mal'tsev polymorphisms (defined below) for each $S_{[t,j]}$, guarantees the existence of a witness function for each $S_{[t,j]}$ whose evaluation time is linear in t. For our purposes, a witness function for a set is a function that verifies whether a given input is an element of that set. Further details about such witness functions and their construction is given in [8].

Understanding the Mal'tsev Condition requires us to define polymorphisms and Mal'tsev polymorphisms.

Definition 21 ((Cubic) Polymorphism). A cubic polymorphism (polymorp-hism, for short) of $\Phi \subset D^t$ is a function $\phi : D^3 \to D$ such that, for all $\boldsymbol{u}, \boldsymbol{v}, \boldsymbol{w} \in \Phi$, $(\phi(u_1, v_1, w_1), \ldots, \phi(u_t, v_t, w_t)) \in \Phi$.

Definition 22 (Mal'tsev Polymorphism). A Mal'tsev polymorphism of $\Phi \subset D^t$ is a polymorphism $\phi : D^3 \to D$ of Φ such that, for all $a, b \in D$, we have that $\phi(a, a, b) = \phi(b, a, a) = a$.

Definition 23 (Mal'tsev Condition). We say that a constraint language $\mathcal{F}$ satisfies the Malt'sev Condition if, for every instance function F_I of $\#CSP(\mathcal{F})$, all the equivalence classes $S_{[t,j]}$ associated with F_I share a common Mal'tsev polymorphism.

With the Mal'tsev Condition, we know that there exists a tractable witness function for each $S_{[t,j]}$. Given sufficient information about each $S_{[t,j]}$, we might hope to compute such witness functions. However, we will in general not know m_t, nor will we know much about each $S_{[t,j]}$. The existence of a *shared* Mal'tsev polymorphism, along with the *Type-Partition Condition* below, allows us to overcome this lack of information. When these conditions are satisfied, we can determine each m_t and construct witness functions for each $S_{[t,j]}$ without requiring prior knowledge or construction of each $S_{[t,j]}$.

We present the Type-Partition Condition below. A complete coverage can be found in [8].

Definition 24 (Prefix). Let $n, m \in \mathbb{Z}_+$ with $n \geqslant m$ be given. Let $\boldsymbol{v} = (v_1, \ldots, v_n) \in \mathbb{R}^n$ be given. We say a vector $\boldsymbol{w} \in \mathbb{R}^m$ is a prefix of $\boldsymbol{v}$ if, for all $j \leqslant m$, $w_j = v_j$. Thus $\boldsymbol{v} = (w_1, \ldots, w_m, v_{m+1}, \ldots, v_n)$, so $\boldsymbol{w}$ appears as a prefix of $\boldsymbol{v}$.

Definition 25 ($type_F$). Let a function $F : D^n \to \mathbb{C}$ with $n \geqslant 2$ be given. Let $P([m])$ denote the power set of $\{1, \ldots, m\}$. We define the map $type_F : \bigcup_{k\in[n]} D^k \to P([m])$ so that for each $\boldsymbol{y} \in \bigcup_{k\in[n]} D^k$, we have $type_F(\boldsymbol{y}) = \{j \in [m] :$ $\boldsymbol{y}$ is a prefix of an element of $S_{[n,j]}\}$.

Definition 26 (Type-Partition Condition). We say that a constraint language $\mathcal{F}$ satisfies the Type-Partition Condition if, for all instance functions F_I in $\#CSP(\mathcal{F})$ with arity

$n \geqslant 2$, for all $t \in [n]$, $l \in [t-1]$, and $\mathbf{y}, \mathbf{z} \in D^l$, we have that the sets $type_{F_l^{[t]}}(\mathbf{y})$ and $type_{F_l^{[t]}}(\mathbf{z})$ are either equal or disjoint.

All together, the criteria for the dichotomy theorem give us the ability to make real use of our equivalence relation $\equiv_t$. The Block-Orthogonality Condition guarantees that we do not have too many equivalence classes. The Mal'tsev and Type-Partition Conditions together let us construct witness functions that we use to identify which equivalence class a given partial assignment belongs to. With a guarantee that our equivalence relation is useful and the ability to determine elements' equivalence classes, we are able to use the methodology described in Sec. IV B to determine $Z(I)$ in polynomial time.

This provides some intuition that the dichotomy theorem criteria are sufficient for a problem to be in FP. For a proof that w#CSPs that do not obey the dichotomy theorem criteria are #P-hard, refer to [8].

[1] F. Barahona, J. Phys. A: Math. Gen. **15**, 3241 (1982).
[2] J.-G. Liu, L. Wang, and P. Zhang, Phys. Rev. Lett. **126**, 090506 (2021).
[3] F. Tanaka and S. Edwards, J. Phys. F **10**, 2769 (1980).
[4] E. Eaton and R. Mansbach, in *Proceedings of the AAAI Conference on Artificial Intelligence* (Association for the Advancement of Artificial Intelligence, Toronto, Canada, 2021), Vol. 26, pp. 900–906.
[5] F. Bacchus, S. Dalmao, and T. Pitassi, in *Proceedings of the 44th Annual IEEE Symposium on Foundations of Computer Science, Cambridge, Massachusetts* (IEEE Computer Society, Piscataway, NJ, 2003), pp. 340–351.
[6] C. Domshlak and J. Hoffmann, J. Artif. Intell. Res. **30**, 565 (2007).
[7] Y. Naveh, M. Rimon, I. Jaeger, Y. Katz, M. Vinov, E. Marcus, and G. Shurek, AI Mag. **28**, 13 (2007).
[8] J.-Y. Cai, in *Proceedings of the 44th Symposium on Theory of Computing Conference* (Association for Computing Machinery, New York, 2011).
[9] C. P. Gomes, A. Sabharwal, and B. Selman, in *Handbook of Satisfiability*, edited by A. Biere, M. Heule, H. van Maaren, and T. Walsh (IOS Press BV, 2009), pp. 633–654.
[10] M. Chavira and A. Darwiche, Artif. Intell. **172**, 772 (2008).
[11] D. Koller and N. Friedman, *Probabilistic Graphical Models: Principles and Techniques* (MIT Press, Cambridge, MA, 2009).
[12] S. Chakraborty, D. Fried, K. S. Meel, and M. Y. Vardi, in *Proceedings of the 24th International Conference on Artificial Intelligenc, IJCAI'15*, edited by Q. Yang and M. Wooldridge (AAAI Press, Buenos Aires, Argentina, 2015), pp. 689–695.
[13] J. Dudek, D. Fried, and K. S. Meel, in *Advances in Neural Information Processing Systems*, edited by H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan, and H. Lin (Curran Associates, Inc., Vancouver, Canada, 2020), pp. 1071–1082.
[14] L. G. Valiant, SIAM J. Comput. **8**, 410 (1979).
[15] J. M. Dudek, L. Dueñas-Osorio, and M. Y. Vardi, arXiv:1908.04381.
[16] J. M. Dudek and M. Y. Vardi, in *MCW* (2020), https://api.semanticscholar.org/CorpusID:220250670.
[17] T. Sang, P. Beame, and H. Kautz, in *Theory and Applications of Satisfiability Testing*, edited by F. Bacchus and T. Walsh (Springer, Berlin, Heidelberg, 2005), pp. 226–240.
[18] U. Oztok and A. Darwiche, in *Proceedings of the 24th International Conference on Artificial Intelligence, IJCAI'15*, edited by Q. Yang and M. Wooldridge (AAAI Press, Buenos Aires, Argentina, 2015), pp. 3141–3148.
[19] J.-M. Lagniez and P. Marquis, in *Proceedings of the 26th International Joint Conference on Artificial Intelligence, IJCAI'17*, edited by C. Sierra (AAAI Press, Melbourne, Australia, 2017), pp. 667–673.
[20] J. M. Dudek, V. H. N. Phan, and M. Y. Vardi, in *Proceedings of the AAAI Conference on Artificial Intelligence* (AAAI Press, New York, NY, 2020), Vol. 34, pp. 1468–1476.
[21] J. M. Dudek, V. H. N. Phan, and M. Y. Vardi, in *Principles and Practice of Constraint Programming: 26th International Conference, CP 2020, Louvain-La-Neuve, Belgium, September 7–11, 2020, Proceedings*, edited by H. Simonis (Springer-Verlag, Berlin, Heidelberg, 2020), pp. 211–230.
[22] J. K. Fichte, M. Hecher, S. Woltran, and M. Zisser, in *26th Annual European Symposium on Algorithms (ESA 2018)*, Leibniz International Proceedings in Informatics (LIPIcs), edited by A. Yossi, B. Hannah, and H. Grzegorz (Schloss Dagstuhl-Leibniz-Zentrum für Informatik, Dagstuhl, Germany, 2018), Vol. 112, pp. 28:1–28:16.
[23] J. K. Fichte, M. Hecher, and M. Zisser, in *Principles and Practice of Constraint Programming: 25th International Conference, CP 2019, Stamford, CT, USA, September 30 - October 4, 2019, Proceedings*, edited by T. Schiex and S. de Givry (Springer-Verlag, Berlin, Heidelberg, 2019), pp. 491–509.
[24] J. K. Fichte, M. Hecher, and F. Hamiti, ACM J. Exp. Algorithmics **26**, 13 (2021).
[25] F. Pan, P. Zhou, S. Li, and P. Zhang, Phys. Rev. Lett. **125**, 060503 (2020).
[26] J. Gray and S. Kourtis, Quantum **5**, 410 (2021).
[27] M. Hamann and B. Strasser, J. Exp. Algorithmics (JEA) **23**, 1 (2018).
[28] T. E. Oliphant, *A Guide to NumPy* (Trelgol Publishing, USA, 2006), Vol. 1.
[29] A. Lucas, Front. Phys. **2**, 5 (2014).
[30] Z. Zhang, Mathematics **11**, 237 (2023).
[31] O. Gableske, in *Theory and Applications of Satisfiability Testing—SAT 2014*, edited by C. Sinz and U. Egly (Springer International Publishing, Cham, 2014), pp. 367–383.
[32] W. Huang, D. A. Kitchaev, S. T. Dacek, Z. Rong, A. Urban, S. Cao, C. Luo, and G. Ceder, Phys. Rev. B **94**, 134424 (2016).
[33] T. Jagielski, R. Manohar, and J. Roychowdhury, arXiv:2306.01569.
[34] N. Mohseni, P. L. McMahon, and T. Byrnes, Ising machines as hardware solvers of combinatorial optimization problems, Nat. Rev. Phys. **4**, 363 (2022).
[35] K. Tanahashi, S. Takayanagi, T. Motohashi, and S. Tanaka, J. Phys. Soc. Jpn. **88**, 061010 (2019).

[36] M. W. Johnson, M. H. Amin, S. Gildert, T. Lanting, F. Hamze, N. Dickson, R. Harris, A. J. Berkley, J. Johansson, P. Bunyk *et al.*, Nature (London) **473**, 194 (2011).
[37] M. Yamaoka, C. Yoshimura, M. Hayashi, T. Okuyama, H. Aoki, and H. Mizuno, Hitachi Review 65, 156 (2016).
[38] Y. Ren, S. Eubank, and M. Nath, Phys. Rev. E **94**, 042125 (2016).
[39] D. Achlioptas, A. Coja-Oghlan, M. Hahn-Klimroth, J. Lee, N. Müller, M. Penschuck, and G. Zhou, Random Struct. Algorithms **58**, 609 (2021).
[40] A. Braunstein, M. Mézard, and R. Zecchina, Random Struct. Algorithms **27**, 201 (2005).
[41] M. Mézard and R. Zecchina, Phys. Rev. E **66**, 056126 (2002).
[42] T. Sang, P. Beame, and H. A. Kautz, in *7th International Conference on Theory and Applications of Satisfiability Testing, Vancouver (BC)* (Lecture Notes in Computer Science (LNCS), Canada, 2004).
[43] L. Duenas-Osorio, K. S. Meel, R. Paredes, and M. Y. Vardi, in *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence* (AAAI Press, San Francisco, California, 2017), pp. 4488–4494.
[44] D. Agrawal, Y. Pote, and K. S. Meel, in *Proceedings of the Thirtieth International Joint Conference on Artificial Intelligence, IJCAI-21*, edited by Z.-H. Zhou (International Joint Conferences on Artificial Intelligence Organization, Montreal, Quebec, Canada, 2021), pp. 4276–4285.
[45] M. Chavira and A. Darwiche, in *Proceedings of the 20th International Joint Conference on Artifical Intelligence, IJCAI'07*, edited by R. Sangal, H. Mehta, and R. K. Bagg (Morgan Kaufmann Publishers Inc., Hyderabad, India, 2007), pp. 2443–2449.
[46] C. Viricel, D. Simoncini, S. Barbe, and T. Schiex, in *Principles and Practice of Constraint Programming*, edited by M. Rueher (Springer International Publishing, Cham, 2016), pp. 733–750.
[47] E. Birnbaum and E. L. Lozinskii, J. Artif. Intell. Res. **10**, 457 (1999).
[48] T. Sang, F. Bacchus, P. Beame, H. A. Kautz, and T. Pitassi, *SAT: The Seventh International Conference on Theory and Applications of Satisfiability Testing* (Springer Science + Business Media via LNCS (Lecture Notes in Computer Sciences, Vancouver, Canada, 2004), Vol. 4, pp. 20–28.
[49] S. Chakraborty, K. S. Meel, and M. Y. Vardi, in *Handbook of Satisfiability*, edited by A. Biere, M. Heule, H. van Maaren, and T. Walsh, 2nd ed. (IOS Press, 2021), pp. 1015–1045.
[50] https://github.com/panzhang83/catn.
[51] J. Gray and G. K.-L. Chan, Phys. Rev. X **14**, 011009 (2024).
[52] S. Chakraborty, K. S. Meel, and M. Y. Vardi, in *Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence, IJCAI'16*, edited by G. Brewka (AAAI Press, New York, NY, 2016), pp. 3569–3576.
[53] D. Roth, Artif. Intell. **82**, 273 (1996).
[54] M. Jerrum and A. Sinclair, SIAM J. Comput. **22**, 1087 (1993).
[55] M. Sipser, ACM SIGACT News **27**, 27 (2012).
[56] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms*, 3rd ed. (MIT Press, Cambridge, MA, 2009).
[57] J.-Y. Cai, X. Chen, and P. Lu, SIAM J. Comput. **45**, 2177 (2016).
[58] D. Goldberg, ACM Comput. Surv. **23**, 5 (1991).
[59] J. Schwarz, in *Proceedings of 9th Symposium on Computer Arithmetic, Santa Monica* (IEEE, Piscataway, NJ, 1989), pp. 10–17.
[60] G. F. Newell and E. W. Montroll, Rev. Mod. Phys. **25**, 353 (1953).
[61] S. Shao and J.-Y. Cai, in *2020 IEEE 61st Annual Symposium on Foundations of Computer Science (FOCS), Durham, NC* (IEEE, Piscataway, NJ, 2020), pp. 1091–1102.
[62] B. Brown, S. T. Flammia, and N. Schuch, Phys. Rev. Lett. **107**, 040501 (2011).
[63] J. Tindall, M. Fishman, E. M. Stoudenmire, and D. Sels, PRX Quantum **5**, 010308 (2024).
[64] https://github.com/vardigroup/Ising.