COMPLEXITY OF APPROXIMATING #CSPS

by

Amir Hedayaty

B.Sc., Shahid Beheshti University, 2005

M.Sc., Sharif University of Technology, 2008

A THESIS SUBMITTED IN PARTIAL FULFILLMENT
OF THE REQUIREMENTS FOR THE DEGREE OF
MASTER OF SCIENCE
in the
School of Computing Science
Faculty of Applied Sciences

APPROVAL

Name:	Amir Hedayaty
Degree:	Master of Science
Title of thesis:	Complexity of Approximating #CSPs

Examining Committee: Dr. David Mitchell
Chair

Dr. Pavol Hell, Professor, Computing Science
Simon Fraser University
Senior Supervisor

Dr. Funda Ergun, Associate Professor, Computing Science
Simon Fraser University
Supervisor

Dr. Valentine Kabanets, Associate Professor, Computing Science
Simon Fraser University
SFU Examiner

Date Approved: October 30th 2012

Abstract

Constraint satisfactions is a framework to express combinatorial problems. #CSP is the problem of finding the number of solutions for a constraint satisfaction problem instance. In this work, we study complexity of approximately solving the #CSP. We provide several techniques for approximation preserving reductions among counting problems. Most of this work focuses on reduction to #BIS, the problem of finding the number of independent sets in a bipartite graph.

We prove that approximately solving #CSP(Γ) over relations which we call *monotone*, is not harder than #BIS. We also prove that approximately solving #Hom(H) for reflexive oriented graphs is not easier than #BIS. Finally, we investigate monotone reflexive graphs.

Keywords: #CSP, Approximation, AP-reduction, FPRAS

Acknowledgments

First of all, I want to thank my family for always supporting me even from far far away. I feel so lucky to be raised them. I am very grateful to have Dr. Hell as my senior-supervisor. I do not know how to thank him for his supervision, support, help, and patience. This thesis would have been is it is today without his dedication. I do not also know how to thank my dear friends which have been really supportive in my hardest days away from home. I also thank my previous supervisor Dr. Bulatov for his contributions in this work and introducing the area to me.

Contents

List of Figures

Chapter 1

Introduction

Constraint Satisfaction is a powerful framework to express many combinatorial problems. The Constraint Satisfaction Problem (CSP) is the problem of deciding if it is possible to assign values to some variables such that given constraints are satisfied. #CSP is the problem of finding the number of possible assignments for a set of constraints.

Many computational problems in mathematics, economics, and statistical mechanics can be expressed by #CSP. Graph homomorphism, sampling, and partition functions are among notions in mathematics that are closely related to #CSP. Computation of partition function plays a key role in dealing with problems involving Ising and Potts models. These models are commonly used in statistical mechanics and game theory.

In real world applications, there are many problems that are not efficiently solvable; however, usually an approximate solution is satisfactory for them. In order to find out which problems can be dealt with efficiently, we classify them by complexity of approximately solving them.

Generally solving the #CSP is a hard problem, however if we restrict the #CSP to a set of relations Γ, which is denoted by #CSP(Γ), depending on the Γ the complexity of the problem may vary. In this work we study the complexity of finding approximate solutions for the problem #CSP(Γ). We will introduce some of methods that can be used to approximate counting problems.

Bulatov [2] has proved a dichotomy for complexity of the problem #CSP(Γ). This dichotomy implies that for some Γ, the problem #CSP(Γ) can be solved in polynomial time and for the rest if one of them can be solved in polynomial time all of them can be solved in polynomial time. Feder and Vardi [19] have conjectured that #CSP(Γ) also exhibits of dichotomy of this sort. This conjecture has motivated many researchers [6, 7, 36]. With respect to approximation, more complexity classes are expected for the problem #CSP(Γ). The problem of finding the number of 2-Colorings of a graph can be solved in polynomial time. There is the problem of finding the number of independent sets in a bipartite graph often referred as #BIS. The problem of finding the number of independent sets in a general graphs often referred as the #IS is among a set of problems which if one of them can be approximately solved in polynomial time then for all Γ, the problem #CSP(Γ) can be approximately solved in polynomial time. Goldberg et al. [15] have proved that for Boolean Γ, there are three complexity classes for #CSP(Γ). These three classes are: problems that can be approximately solved in polynomial time, such as finding the number of 2-Colorings in a graph; problems as hard as the problem #BIS; and problems that are hard to approximate, such as the problem #IS.

In this work, we introduce some useful techniques to find approximation preserving reductions among counting problems. We will introduce *monotone* relations and show that the problem #CSP(Γ) over these families of relation is not harder that the problem #BIS. We will also show that the problem #CSP(Γ) for reflexive oriented graphs is at least as hard as the problem #BIS. We will also investigate reflexive monotone graphs.

1.1 Thesis Organization

In Chapter 2 we formally define CSP and #CSP and their applications. In Chapter 3 we define reductions for counting problems and mention major results on complexity of the problem #CSP(Γ). In Chapter 4 we define approximation preserving reductions and the known classes for approximated counting. We mention several problems from each class; we also mention several methods used for approximate counting problems. In Chapter 5 we introduce our own methods used for approximation preserving reductions. In Chapter 6 we mention our results on the complexity of approximating the problem #BIS. In Chapter 7 we investigate reflexive monotone graphs.

Chapter 2

CSP and #CSP

The Constraint Satisfaction Problem (CSP) is the problem of deciding if it is possible to assign values to some variables such that given constraints are satisfied. #CSP is the problem of finding the number of possible assignments for a set of constraints. Consider the following example:

Example 2.0.1 (3-Coloring)**.** The problem 3-Coloring is the problem of coloring vertices of the input graph G with colors $\{r, g, b\}$ such that no two adjacent vertices are colored the same color. This problem can be formulated with a set of constraints as follows. Let $V = \{v_1, v_2, \ldots, v_n\}$ be a set of variables where each variable corresponds to a vertex of the input graph G. The goal is to find a function $f : V \to \{r, g, b\}$ such that for each edge (u, v) of G, $f(u) \neq f(v)$ holds.

In the above example, the CSP is the problem of deciding if there is a function satisfying all the given constraints. #CSP is the problem of finding the number of such functions. Note that any CSP can then be expressed as deciding if the answer for #CSP with the same set of constraints is greater zero.

Let D be a set of elements; any subset of D^k is a *relation* of arity k with domain D. A *constraint language* with domain D is a set of relations with domain D. In this work domains and relations are always finite.

A CSP instance $\mathcal{P}$ with constraint language Γ is a tuple $(D, V, \mathcal{C})$ where:

- D is a set of values which is the domain of Γ,
- V is a set of variables,

- $\mathcal{C}$ is a collection of constraints where each constraint consists of a scope ϱ, which is a tuple of variables from V and a relation R from Γ of the same arity as ϱ. A constraint is represented as $\langle R, \varrho \rangle$.

An assignment for $\mathcal{P}$ is a function φ from V to D. The assignment φ is satisfying if the scope of each constraint is mapped to a tuple of the corresponding relation.

Definition 2.0.2 (CSP(Γ))**.** For a fixed constraint language Γ, CSP(Γ) is the problem of deciding whether a CSP instance $\mathcal{P}$ with constraint language Γ has a satisfying assignment.

For a constraint language Γ that consists of only a single relation R, we use CSP(R) instead of CSP(Γ) for simplicity.

Example 2.0.3 (3-Coloring)**.** We continue with the problem 3-Coloring from Example 2.0.1. Let G be the input graph for the problem 3-Coloring. One way to formulate problem 3-Coloring using a CSP(Γ) is as follows. Let $D = \{r, g, b\}$ be the domain and NEQ_{rgb}, the binary dis-equality relation on domain D, consists of all tuples from D^2 except the ones whose elements are equal, that is

$$NEQ_{rgb} = \{(r, g), (r, b), (g, r), (g, b), (b, r), (b, g)\}$$

Let φ be an assignment which is a function from vertices of G to D. For each edge (u, v) of G $\varphi(u) \neq \varphi(v)$ holds which is equivalent to $NEQ_{rgb}(u, v)$.

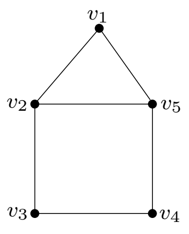
Let G be the graph in Figure 2.1. The instance of CSP for 3-Coloring on G is as follows. The set of variables is the set of vertices of G, and for each edge of G a constraint with relation NEQ_{rgb} is placed.

$$\begin{aligned}\mathcal{P} = (D = \{1, 2, 3\}, V &= \{v_1, v_2, v_3, v_4, v_5\},\\ \mathcal{C} &= \{\langle (v_1, v_2), NEQ_{rgb} \rangle, \langle (v_2, v_3), NEQ_{rgb} \rangle, \langle (v_3, v_4), NEQ_{rgb} \rangle,\\ &\langle (v_4, v_5), NEQ_{rgb} \rangle, \langle (v_5, v_1), NEQ_{rgb} \rangle, \langle (v_2, v_3), NEQ_{rgb} \rangle\})\end{aligned}$$

Example 2.0.4 (3-SAT)**.** The problem of deciding if there exists a truth assignment for a given Boolean CNF (Conjunctive Normal Form) formula where each clause contains exactly 3 literals is called 3-SAT.

Let $\Gamma_1 = \{R_1, R_2, \ldots, R_8\}$ be a constraint language where:

- $R_1 = \{(x, y, z) \mid x, y, z \in \{T, F\}, x \vee y \vee z = T\}$,



G

Figure 2.1: Graph G used in Example 2.0.3

- $R_2 = \{(x, y, z) \mid x, y, z \in \{T, F\}, \bar{x} \vee y \vee z = T\}$,
- $R_3 = \{(x, y, z) \mid x, y, z \in \{T, F\}, x \vee \bar{y} \vee z = T\}$,
- $R_4 = \{(x, y, z) \mid x, y, z \in \{T, F\}, x \vee y \vee \bar{z} = T\}$,
- $R_5 = \{(x, y, z) \mid x, y, z \in \{T, F\}, \bar{x} \vee \bar{y} \vee z = T\}$,
- $R_6 = \{(x, y, z) \mid x, y, z \in \{T, F\}, \bar{x} \vee y \vee \bar{z} = T\}$,
- $R_7 = \{(x, y, z) \mid x, y, z \in \{T, F\}, x \vee \bar{y} \vee \bar{z} = T\}$,
- $R_8 = \{(x, y, z) \mid x, y, z \in \{T, F\}, \bar{x} \vee \bar{y} \vee \bar{z} = T\}$

Every clause in a 3-SAT instance can be expressed by a constraint using an R_i and vice-versa. Hence, there is a one-to-one correspondence between constraints in a CSP(Γ_1) instance and a 3-SAT instance. Hence, the problems CSP(Γ_1) and the 3-SAT are the same.

Another type of problems defined by constraint satisfaction are counting problems which involve finding the number of solutions for a CSP instance. The number of solutions of a CSP instance can be used to express many computational problems such as partition functions which are mentioned in Section 2.1.

Definition 2.0.5 (#CSP(Γ))**.** For a constraint language Γ, #CSP(Γ) is the problem of finding the number of satisfying assignments for a given CSP instance with Γ as the constraint language.

Here are two examples of counting problems which can be expressed as #CSP(Γ) for some constraint language Γ.

Example 2.0.6 (#3-Coloring)**.** We continue with the NEQ_{rgb} relation from Example 2.0.3. The problem of finding the number of 3-Colorings of a given graph denoted by #3-Coloring, can be expressed as the problem #CSP(NEQ_{rgb}).

Example 2.0.7. The problem of finding the number of satisfying truth assignments for a given Boolean CNF (Conjunctive Normal Form) formula where each clause contains exactly 3 literals is called #3-SAT.

It is easy to see that with the constraint language Γ_1 from Example 2.0.4 the problems #3-SAT and the #CSP(Γ_1) are the same.

2.1 Applications of #CSP

In this section we will show several applications of #CSP and how computational problems can be formulated with #CSP. First, we will describe a graph homomorphism problem which can be viewed as a special case of CSP; next, we will relate the number of homomorphisms between two graphs to the graph isomorphism problem. We will also describe partition functions and give examples of partition functions in two common models in statistical physics.

Graph Homomorphism

Let G and H be two graphs. A homomorphism from G to H is a mapping h of vertices of G to vertices of H such that for every edge (u, v) of G, $(h(u), h(v))$ is an edge of H. If there is homomorphism from G to H we shall write $G \to H$, and $G \not\to H$ means that there is no homomorphism from G to H.

Example 2.1.1. Let G and H be the graphs shown in Figure 2.2. The mapping h defined as $h(a) = 1, h(b) = 2, h(c) = 3, h(d) = 1$ is not a homomorphism because (a, d) is mapped to $(1, 1)$ which is not an edge of H but the mapping h' defined as $h'(a) = 1, h'(b) = 2, h'(c) = 1, h'(d) = 2$ is a homomorphism because all the edges of G are mapped to edges of H. Note that a homomorphism is not necessarily surjective.

Definition 2.1.2 (Hom(H))**.** For a fixed graph H, Hom(H) is the problem of deciding whether a given graph G is homomorphic to H.

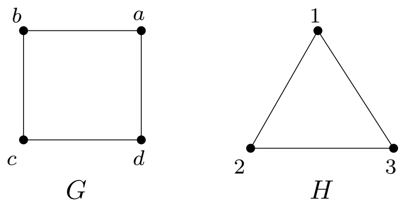


Figure 2.2: Graph m and H used in Example 2.1.1

Example 2.1.3 (3-Coloring)**.** We continue with the problem 3-Coloring from Example 2.0.1; however, in this example we express it as a graph homomorphism problem. As usual we denote the complete graph on k vertices by K_k. For all complete graphs, (u, v) is an edge of K_k if an only if $u \neq v$. This shows that $Hom(K_k)$ is the same as the problem k-coloring. As a special case, $\#Hom(K_3)$ is the same as problem #3-Coloring.

Let NoRB-3-Coloring be the problem 3-Coloring with an additional restriction that the vertices colored blue are not allowed to be connected to the vertices colored red. In order to formulate this problem with graph homomorphisms, obtain H from K_3 as follows. Label the vertices with $\{r, g, b\}$. Delete the edge between r and b. The problem Hom(H) is the same as the problem N0RB-3-Coloring. This example illustrates that the graph homomorphism problem generalizes the graph coloring problem. This is why the problem Hom(H) is also called the problem H-Coloring. More details and examples on graph homomorphism can be found in the book [25] by Hell and Nešetřil.

Observation 2.1.4. *Let G and H be two graphs. If H contains a loop (reflexive vertex) then $G \to H$. If G is bipartite then $G \to H$ if and only if H is has at least one edge. If H is bipartite then $G \to H$ if and only if G is bipartite.*

Observation 2.1.4 shows that if the graph H has a loop or is bipartite, the problem Hom(H) is easy; however, for other graphs the problem is of higher complexity.

Theorem 2.1.5 (Hell and Nešetřil 1990 [24])**.** *For an undirected graph H, the problem Hom(H) is polynomial time solvable if H contains a loop or H is bipartite; otherwise, it is NP-complete.*

Definition 2.1.6 (#Hom(H))**.** For a fixed graph H, #Hom(H) is the problem of finding the number of homomorphisms from a given input graph G to H.

Dyer and Greenhill proved the following dichotomy for the problem #Hom(H).

Theorem 2.1.7 (Dyer and Greenhill 2000 [16])**.** *For a graph H, if each component of H is either a reflexive complete graph or a irreflexive complete bipartite graph then the problem #Hom(H) is polynomial time solvable; otherwise, it is #P-complete.*

Hom(H) and #Hom(H) are special cases of CSP(Γ) and #CSP(Γ), respectively. In both cases, Γ consists of a single binary relation which is the edge set of H with vertex set of H as the domain. Hence, we may use Hom and #Hom instead of the corresponding CSP and #CSP, respectively.

Lovász Vectors

The number of homomorphisms from G to H is denoted by $\hom(G, H)$; this function can be used to define the *Lovász vector* which is used to characterize graphs. Lovasz [32] proved that two graphs H_1 and H_2 are isomorphic if and only if for any graph G, $\hom(G, H_1) = \hom(G, H_2)$. For an enumeration of all non-isomorphic graphs $(G_1, G_2, \dots)$, the Lovasz vector of a graph H is the infinite sequence consisting of the number of homomorphisms from graphs in the sequence to H, i.e., $(\hom(G_1, H), \hom(G_2, H), \dots)$. In other words Lovasz's theorem indicates two graphs are isomorphic if and only if they have the same Lovasz vector.

Sampling

In this section, we use the model from [30] for definition of classes of problems. Let Σ be the finite alphabet used to encode input and output of a problem. We can express the assignment of a solution to an instance with a relation $R \subseteq \Sigma^* \times \Sigma^*$.

A *uniform sampling* problem is a problem that for an instance x, generates a uniformly random solution y such that $(x, y) \in R$. An *almost uniform sampling* is a problem that for an instance x, generates a random solution y such that $(x, y) \in R$ and for any other y' that $(x, y') \in R$ the probabilities of choosing y and y' are approximately the same. Jerrum et al. [30] explained that for practical purposes, it is impossible to distinguish almost uniform sampling from uniform sampling by experiments running in polynomial time. Hence, we may use them interchangeably in this work.

In this model a *counting* problem is a problem that for an instance x, finds $|\{y \mid (x, y) \in$

$R\}|$ and an approximate counting problem is a problem that for an instance x, approximately finds $|\{y \mid (x, y) \in R\}|$.

Intuitively, a relation is said to be *self-reducible* if solutions for an instance can be expressed in terms of solutions for a number of smaller instances of the same relation. Many of the relation such as the relation between a graph and a matching in that graph and relation between CNF (Conjunctive Normal Form) or DNF (Disjunctive Normal Form) formula and a satisfying truth assignment of that formula are self-reducible.

Theorem 2.1.8 (Jerrum et al. 1986 [30])**.** *For any self-reducible relation R, the almost uniform sampling problem and the approximate counting problem have the same complexity.*

Here we provide a very informal proof for the problem of finding the number of truth assignment for a CNF. The general idea of the proof for Theorem 2.1.8 is the same. Let C be a CNF. For a variable x used in C, let C_x denote the CNF derived from C if x is assigned true; analogously, let $C_{\bar{x}}$ denote the CNF derived from C if x is assigned false. Clearly, the number of solutions for C is the sum of the number of solutions for C_x and the number of solutions for $C_{\bar{x}}$.

In order to generate an almost uniform solution for C using an approximate counter, we first approximately find the number of solutions for C_x and $C_{\bar{x}}$. Next, we assign true or false to x with probability proportional to number of solutions for C and $C_{\bar{x}}$. Then, we continue with C_x or $C_{\bar{x}}$ regarding the choice in previous step.

In order to solve approximate counting using an almost uniform generator, we first generate some number of random solutions. The number of solution we generate depends on the expected approximation ratio and the number of variables. Let $p(x)$ be the ratio of the number of solutions where x is true to the total number of generated solutions; analogously, let $p(\bar{x})$ be the ratio of number of solutions where x is false to the total number of generated solutions. Without loss of generality, suppose that $p(x) \geq p(\bar{x})$. By recursion, estimate the number of solutions for C_x. Finally, estimate the number of solutions for C by $p(x)$ and the estimation for the number of solutions for C_x.

Partition Functions

Let H be a graph.V_H. For a weight function $w : V_H \cup E_H \to \mathbb{R}$, the weight of the homomorphism h is defined as

$$w(h) = \prod_{uv \in E_G} w(h(u)h(v)) \prod_{v \in V_G} w(h(v))$$

The partition function $Z_H(G, w)$ is the sum of the weights for all the homomorphisms:

$$Z_H(G, w) = \sum_{h:G \to H} w(h)$$

Note that $\hom(G, H) = Z_H(G, w)$ if $w(v) = w(e) = 1$ for all $v, e \in H$ and $w(e) = 0$ for all $e \notin H$. The problem H-Partition is defined follows.
Instance: A graph G and a weight function w
Output: The value of partition function $Z_H(G, w)$

Given a weight function and the partition function, the *Gibbs distribution* on homomorphisms from G to H is the distribution that each homomorphism h has a probability

$$\pi_{H,G,w}(h) = \frac{w(h)}{Z_H(G, w)}$$

The problem H-GibbsSample is defined as:
Instance: A graph G
Output: An H-coloring of G chosen from distribution $\pi_{H,G,w}$

Dyer et al. [14] have proved that if H-GibbsSample can be approximately sampled in polynomial time then H-Partition can also be approximated by a randomized algorithm in polynomial time.

Statistical Mechanics

Statistical Mechanics is a branch of physics that applies probability theory to predict the behaviour of a system at a given temperature. Usually there is a complex system consisting of many microscopic elements. A state or configuration σ is an assignment of parameters of the microscopic elements. The *Hamiltonian* $H(\sigma)$ is the energy of the system in state σ. Let

β be the inverse (one over) temperature. The *Partition Function* at a given temperature is defined as

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

Given a model of the system and the temperature, many of the important properties of the system such as the free energy, the entropy, the specific heat, and the location of phase transition can be evaluate using the the partition function.

One of the commonly used models in Statistical Mechanics is the Ising model. In the Ising mode, the system is defined by a graph $G(V, E)$. Each edge of the graph $(u, v) \in E$ has an *interaction strength* $J_{u,v}$. Each vertex has an associated *local external magnetic field* l_v. A configuration of the system is an assignment $\sigma : V \to \{-1, +1\}$ of vertices of G to spins. Each configuration is associated with energy:

$$H(\sigma) = -\sum_{(u,v)\in E} J_{u,v} \cdot \sigma(u) \cdot \sigma(v) - \sum_{v \in V} l_v \cdot \sigma(v)$$

and the partition function for the Ising model is:

$$\begin{aligned} Z(G, \beta, J, l) &= \sum_{\sigma:V\to\{-1,+1\}} e^{-\beta H(\sigma)} \\ &= \sum_{\sigma:V\to\{-1,+1\}} \prod_{(u,v)\in E} e^{\beta J_{u,v}\sigma(u)\sigma(v)} \prod_{v\in V} e^{\beta l_v \sigma(v)} \end{aligned}$$

To avoid exponentials in the notation, let $\lambda_{u,v} = e^{2\beta J_{u,v}}$ and $\mu_v = e^{2\beta l_v}$. The partition function will be:

$$\begin{aligned} Z(G, \lambda, \mu) &= \sum_{\sigma:V\to\{-1,+1\}} \prod_{(u,v)\in E} \lambda_{u,v}^{\frac{1}{2}\sigma(u)\sigma(v)} \prod_{v\in V} \mu_v^{\frac{1}{2}\sigma(v)} \\ &= \prod_{(u,v)\in E} \lambda_{u,v}^{-\frac{1}{2}} \prod_{v\in V} \mu_v^{-\frac{1}{2}} \sum_{\sigma:V\to\{-1,+1\}} \prod_{(u,v)\in E} \lambda_{u,v}^{\frac{1}{2}+\frac{1}{2}\sigma(u)\sigma(v)} \prod_{v\in V:} \mu_v^{\frac{1}{2}+\frac{1}{2}\sigma(v)} \\ &= \prod_{(u,v)\in E} \lambda_{u,v}^{-\frac{1}{2}} \prod_{v\in V} \mu_v^{-\frac{1}{2}} \sum_{\sigma:V\to\{-1,+1\}} \prod_{\substack{(u,v)\in E \\ \sigma(u)=\sigma(v)}} \lambda_{u,v} \prod_{v\in V:\sigma(v)=+1} \mu_v \end{aligned}$$

The system is *ferromagnetic* if each interaction energy $J_{u,v}$ is non-negative; which implies for all u, v in V, we have $\lambda_{u,v} \geq 1$. A system is *consistent* if either for all v, $\mu_v \geq 1$ or for all v, $\mu_v \leq 1$. We will mention results on complexity of these cases in Chapter 4.

The Potts model generalizes the Ising model to q possible spins. In the Potts model there is an underlying graph $G(V, E)$ and each configuration is an assignment $\sigma : V \to \{1, 2, \ldots, q\}$. Each edge of the graph $(u, v) \in E$ has an interaction strength $J_{u,v}$ and each vertex v at state c is associated with an external field $h_{v,c}$. The energy of a configuration σ is:

$$H(\sigma) = -\sum_{(u,v)\in E} J_{u,v}\chi(\sigma(u), \sigma(v)) - \sum_{v\in V} h_{v,\sigma(v)}$$

where

$$\chi(s, s') = \begin{cases} +1, & s = s' \\ -1, & otherwise \end{cases}$$

As before, let $\lambda_{u,v} = e^{2\beta J_{u,v}}$ and $\mu_{u,c} = e^{\beta h_{v,c}}$. The partition function for the Potts model is:

$$Z(G, \lambda, \mu) = \prod_{(u,v)\in E} \lambda_{u,v}^{-\frac{1}{2}} \sum_{\sigma:V\to\{1,2,\ldots,q\}} \prod_{(u,v)\in E:\sigma(u)=\sigma(v)} \lambda_{u,v} \prod_{v\in V} \mu_{v,\sigma(v)}$$

Problems of computing partition functions in the Ising and Potts models can be reduced to a #CSP(Γ) for some constraint language Γ. In Chapter 4, we will mention results involving reductions from problems of computing partition functions in the Ising and Potts models to problem #CSP(Γ).

Chapter 3

Reductions and Complexity of Counting

In this chapter, we will mention several complexity classes for computational functions. We will define clones and polymorphisms and specify how they are related to the complexity of the problem CSP(Γ). We will mention the major results on the complexity of problem #CSP(Γ). We will also define partial clones and specify how they are related to the complexity of the problem #CSP(Γ).

Two important classes of functions we are interested in are FP and #P. FP is the class of functions that are computable in polynomial time by a deterministic Turing machine and #P is the class of function that can be expressed as the number of accepting paths of a non-deterministic polynomial time Turing machine.

Let $f, g : \Sigma^* \to \mathbb{N}$ be two functions. A *parsimonious reduction* from f to g is a polynomial time computable function $\sigma : \Sigma^* \to \Sigma^*$ such that $f(x) = g(\sigma(x))$ holds. A Turing reduction from f to g is an polynomial time algorithm that computes f using an oracle of g.

Definition 3.0.9 (#P-Completeness)**.** A problem f is #P-complete if it is a member of #P and every problem in #P is Turing reducible to f in polynomial time.

Consider the problems SAT and #SAT defined as follows. Note that the problems SAT and the #SAT are generalizations of the problems 3-SAT and the #3-SAT mentioned in Examples 2.0.4 and 2.0.7.

Name: SAT

Instance: A CNF formula φ

Output: Existence of a truth assignment that satisfies φ

Name: #SAT
Instance: A CNF formula φ
Output: The number of truth assignments that satisfy φ

In the same way that Cook [8] proved that SAT is NP-complete, Valiant [37] showed that #SAT is #P-complete. The proof is easy; Valiant observed that the Cook's reduction from a non-deterministic Turing machine to a SAT instance is a parsimonious reduction; hence, it works for counting problems, as well.

3.1 Clones

In instances of CSP, constraints can be expressed either explicitly or implicitly by interactions of other constraints. Consider the following example.

Example 3.1.1. Let $NEQ_{01} = \{(0,1),(1,0)\}$ be a binary relation over the set $\{0,1\}$. Consider the following instance of CSP(NEQ_{01}) :

$$\mathcal{P} = (D = \{0,1\}, V = \{x,y,z\}, C = \{\langle NEQ_{01},(x,z)\rangle, \langle NEQ_{01}(y,z)\rangle\}$$

There is no explicit constraint with scope (x,y); however, there is the implicit constraint $\langle EQV_{01},(x,y)\rangle$ where $EQV_{01} = \{(0,0),(1,1)\}$.

Definition 3.1.2 (pp-definition)**.** Let Γ be a constraint language with domain D and let R be a relation with the same domain. Relation R is said to be primitive positive (pp)-definable by Γ if R can be expressed as a predicate using relations from Γ, the relation EQV_D which is the binary equality relation over D, conjunctions, and existential quantification.

Example 3.1.3. Let Γ be a constraint language with domain $D = \{a,b,c\}$, consisting of a single relation $R = \{(a,a,a),(b,b,b),(a,b,a),(a,b,c)\}$. Relations $R_1 = \{a,b\}$, $R_2 = \{(a,a),(b,b),(a,b)\}$, and $R_3 = \{(a,a,a),(a,a,c),(b,b,b)\}$ are pp-definable by Γ as follows:

$$R_1(x) = \exists z.R(x,x,z)$$
$$R_2(x,y) = \exists z.R(x,y,z)$$
$$R_3(x,y,z) = \exists s,t.R(x,t,z) \wedge R(z,s,y)$$

The notion of pp-definition is equivalent to elementary operators used in [34, 1, 31], and strict implementations used in [10, 15].

A set of relations closed under pp-definition is called a *co-clone*. For a constraint language Γ, $\langle\langle\Gamma\rangle\rangle$ is the the least co-clone containing Γ, also called the *co-clone generated by* Γ.

A function f is said to be a *projection* function if $f(x_1, \ldots, x_l) = x_l$ for some l. A set of functions is said to be a *clone* if it contains all projection functions and is closed under superpositions (compositions). For a set of functions C, $\langle\langle C\rangle\rangle$ is the least clone containing C also called the clone generated by C.

A *Galois connection* between sets A and B is a pair of functions $\sigma : 2^A \to 2^B$ and $\tau : 2^B \to 2^A$ which maintain *antitony*, that is, for every $X, X' \subseteq A$ and every $Y, Y' \subseteq B$ we have

$$X \subseteq X' \implies \sigma(X) \supseteq \sigma(X')$$
$$Y \subseteq Y' \implies \tau(Y) \supseteq \tau(Y')$$

and *extensivity*, that is, for every $X \subseteq A$ and every $Y \subseteq B$ we have

$$X \subseteq \tau(\sigma(X))$$
$$Y \subseteq \sigma(\tau(Y))$$

Let R be a relation over domain D and $f : D^n \to D$ a function of arity n over the same domain. Function f *preserves* R or is a *polymorphism* of R if for any n tuples $\mathbf{a}_1, \mathbf{a}_2, \ldots, \mathbf{a}_n$ in R, the tuple $f(\mathbf{a}_1, \mathbf{a}_2, \ldots, \mathbf{a}_n)$ obtained by component-wise application of f, is in R. The relation R in this case is said to be *invariant* with respect to f. The set of all functions preserving a constraint language Γ is denoted by $Pol(\Gamma)$, the set of all relations invariant with respect to a set of functions C is denoted by $Inv(C)$.

Functions Inv and Pol satisfy antitony and extensivity; hence, they form a *Galois connection* between the sets of functions and the set of relations, namely, we have the following theorem.

Theorem 3.1.4 (Geiger [21], Romov et al. [1])**.** *For any constraint language Γ and any set of functions C, we have $Inv(Pol(\Gamma)) = \langle\langle\Gamma\rangle\rangle$ and $Pol(Inv(C)) = \langle\langle C\rangle\rangle$.*

Remark. *Since the Pol and Inv functions form a Galois connection, for any constraint language Γ and any set of functions C, $Pol(\Gamma)$ and $Inv(C)$ are co-clones and clones, respectively.*

Corollary 3.1.5. *Let Γ and Γ' be two constraint languages with the same domain. If $Pol(\Gamma) \subseteq Pol(\Gamma')$ then every relation from Γ' is pp-definable in Γ.*

Theorem 3.1.6 (Jeavons et al. 1997 [26])**.** *For a constraint language Γ and a relation R with the same domain, if R is pp-definable in Γ then CSP(Γ) is polynomial-time equivalent to CSP($\Gamma \cup \{R\}$).*

In order to reduce CSP($\Gamma \cup \{R\}$) to CSP(Γ), for each instance $\mathcal{P} = (D, V, C)$ of CSP($\Gamma \cup \{R\}$), create an instance $\mathcal{P}' = (D, V', C')$ of CSP(Γ) as follows. Include all variables of V in V'. For all relations R' in Γ, include constraints $\langle R', \varrho \rangle$ from C in C'. For all constraints $\langle R, \varrho \rangle$ in C, we use ψ the pp-definition of R in Γ to replace R in $\mathcal{P}'$; for each existential quantifier in ψ, add the quantified variables to V'; for each $EQV_D(x, y)$ relation in ψ, replace all occurrences of y with x; the rest of ψ consists of conjunction of clauses in the form $R'(\varrho)$ where R' is a relation in Γ and ϱ is a scope of variables from V'; for each clause $R'(\varrho)$ in ψ, add a constraint $\langle R', \varrho \rangle$ to C'. For each satisfying assignment φ of $\mathcal{P}'$, the restriction of φ to V is a satisfying assignment for $\mathcal{P}$, and for each satisfying assignment φ of $\mathcal{P}$, there is at least one extension of φ to V' which is a satisfying assignment for $\mathcal{P}'$.

Note that this reduction preserves the existence of an satisfying assignment but does not necessarily preserve the number of satisfying assignments.

Corollary 3.1.7. *For any Γ' a finite subset of $\langle\langle \Gamma \rangle\rangle$, the problem CSP$(\Gamma')$ is polynomial time reducible to the problem CSP(Γ); consequently, the complexity of the problem CSP(Γ) only depends on the $Pol(\Gamma)$.*

Conjecture 3.1.8 (Feder and Vardi 1998 [19])**.** For any constraint language Γ, CSP(Γ) is either polynomial time solvable or NP-complete.

Bulatov and Valeriote [7] have a survey of results on the relation between complexity of CSP(Γ) and $Pol(\Gamma)$. There are more results in [6, 36]; however, the CSP dichotomy still remains an open problem.

3.2 Exact Counting

In the previous section we mentioned the relation between polymorphisms and complexity of the problem CSP(Γ); in this section we will establish a similar connection between polymorphisms and complexity of #CSP(Γ) and mention results on the complexity of #CSP(Γ).

Consider the following problem:
Name: #2-Coloring
Instance: A graph G
Output: The number of 2-Colorings of G

The problem #2-Coloring can be solved in polynomial time as follows. For an input graph G, if G is not bipartite then the answer will be 0; otherwise, if G is connected then the answer will be 2 and if G is not connected there are 2 choices for each component; hence, if G has m connected component the answer will be 2^m.

The problem #2-Coloring can be expressed as #CSP(NEQ_{01}), where NEQ_{01} is the inequality relation over a two element set, that is, $NEQ_{01} = \{(0,1),(1,0)\}$. The problem #2-Coloring is easy to solve because the NEQ relation belongs to a family of relations called *affine* relations. A relation is affine if it is expressible by a system of linear equations over a finite field. The relation NEQ_{01} is an affine relation because $NEQ_{01} = \{(x,y) \mid x \oplus y \equiv 1 \pmod 2\}$.

Example 3.2.1. Relation R defined as $R = \{(0,1),(1,0),(2,2)\}$ is affine because it can be expressed as $\{(x,y) \mid x \oplus y \equiv 1 \pmod 3\}$.

For any affine relation R, the problem #CSP(R) can be solved in polynomial time as follows. If the system of linear equations is inconsistent then there is no solutions; otherwise, there are k^m solutions where m is the dimension of the solution space of the system of linear equations and k is the size of the finite field used to express R.

The simplest type of constraint languages are those over a domain of size two. They are usually referred to as *Boolean* constraint languages. #CSP(Γ) for Boolean Γ is often referred as Boolean #CSP(Γ). Creignou and Hermann proved a dichotomy for the Boolean problem #CSP(Γ).

Theorem 3.2.2 (Creignou and Hermann 1996 [9])**.** *For a Boolean constraint language Γ, if Γ is affine then #CSP(Γ) is polynomial time solvable; otherwise, it is #P-complete.*

Previously, we mentioned the dichotomy theorem for complexity of #CSP in Section 2.1.

Theorem 3.2.3 (Dyer and Greenhill 2000 [16])**.** *For a graph H, if each component of H is a complete reflexive graph or a complete irreflexive bipartite graph then the problem #Hom(H) is polynomial time solvable; otherwise, it is #P-complete.*

Creignou and Hermann [9] proved that a Boolean constraint language Γ is affine if and only if every relation R in Γ is closed under the function $f(x, y, z) = x \oplus y \oplus z$. It is also easy to see that a graph G is complete reflexive if and only if the edge set of G is closed under any polymorphism. Similarly, a bipartite graph G is complete if and only if the orientation of edges of G from one part to other part is closed under any polymorphism.

Hence, polymorphisms can also express criteria used in both theorems. This suggests that $Pol(\Gamma)$ can express the complexity of the problem #CSP(Γ). Suppose Γ_1 and Γ_2 are two constraint languages such that every relation in Γ_2 is pp-definable in Γ_1; the pp-definition suggests a reduction from CSP(Γ_2) to CSP(Γ_1). However, this reduction is not usable for counting problems. Bulatov and Dalmau provided a reduction that relates the complexity of the problem #CSP(Γ) to $Pol(\Gamma)$.

Theorem 3.2.4 (Bulatov, Dalmau 2007 [3]). *For a constraint language Γ and a relation R with the same domain, if R is pp-definable in Γ then #CSP(Γ) is polynomial-time equivalent to #CSP($\Gamma \cup \{R\}$).*

This theorem is analogous to Theorem 3.1.6. In the same manner Jeavons's theorem links complexity of CSP(Γ) to clones, this theorem links complexity of #CSP(Γ) to clones, as well. This connection led to a dichotomy for the problem #CSP(Γ).

Let Γ be a constraint language with domain D and let R be a k-ary relation on D pp-definable in Γ. A *congruence* of R is a $2k$-ary relation Q on D which is also pp-definable in Γ and satisfying the following conditions: (a) Q can be viewed as a binary relation on R, i.e., $Q \subseteq R^2$; (b) Q viewed as a binary relation on R is an equivalence relation.

Now, let Q, Q_1, Q_2 be congruences of R such that $Q \subseteq Q_1, Q_2$. Let $A_1, \ldots, A_m$ and $B_1, \ldots, B_n$ be the equivalence classes of Q_1 and Q_2, respectively. $M(R; Q_1, Q_2; Q)$ denotes an $m \times n$ matrix where M_{ij} is the number of Q-classes in $A_i \cap B_j$.

A constraint language Γ is said to be *congruence singular* if for any pp-definable relation R in Γ and any congruences Q, Q_1, Q_2 of R with $Q \subseteq Q_1, Q_2$, the *row rank* of matrix $M(R; Q_1, Q_2; Q)$ equals the number of classes of the smallest equivalence relation containing both Q_1 and Q_2.

Theorem 3.2.5 (Bulatov 2008 [2]). *For a constraint language Γ, the problem #CSP(Γ) is polynomial time solvable if Γ is congruence singular; otherwise, it is #P-complete.*

Bulatov proved a dichotomy for all Γ but the decidability of being congruence singular remained open until Dyer and Richerby [17] proved that being congruence singular can be verified in polynomial time.

Theorem 3.2.6 (Dyer and Richerby 2010 [17])**.** *For a constraint language* Γ*, checking if* Γ *is congruence singular is in NP.*

3.3 Weak Co-clones

In the previous section, we showed that the complexity of the problems CSP(Γ) and #CSP(Γ) depends on $Pol(\Gamma)$. However, it seems unlikely that a similar relation occurs for approximating #CSP(Γ); in fact, pp-definitions with existential quantifiers allowed, as in Definition 3.1.2, do not guarantee a reduction suitable for approximation.

For a domain D, any set of relations closed under conjunctions and containing EQV_D, the binary equality relation over D, is a weak co-clone. For a constraint language Γ, $\langle\Gamma\rangle$ is the least weak co-clone containing Γ, also called the weak co-clone generated by Γ.

For a partial function f on the domain D, the set of all tuples from D^n on which f is defined is called the *domain* of f and denoted by $\mathsf{dom}(f)$. A set of partial functions C is said to be *down-closed* if for every function f in C, C contains any function f' such that $\mathsf{dom}(f') \subseteq \mathsf{dom}(f)$ and $f'(a_1, a_2, \dots, a_n) = f'(a_1, a_2, \dots, a_n)$ for every tuple $(a_1, a_2, \dots, a_n) \in \mathsf{dom}(f')$. A down-closed set of partial functions, containing all projections and closed under superpositions is called a *partial clone*.

Let R be a relation with domain D and $f : D^n \to D$ be a partial function of arity n over the same domain. Function f is a *partial polymorphism* for R, if for any n tuples $\mathbf{a}_1, \mathbf{a}_2, \dots, \mathbf{a}_n$ in R if $(\mathbf{a}_1, \mathbf{a}_2, \dots, \mathbf{a}_n) \in dom(f)$ then $f(\mathbf{a}_1, \mathbf{a}_2, \dots, \mathbf{a}_n)$ is also in R. Relation R in this case is said to be *invariant* with respect to f. The set of all partial polymorphisms of a constraint language Γ is denoted by $pPol(\Gamma)$ and the set of all relations invariant under a set of partial functions C is denoted by $Inv(C)$. Note that the Inv function for partial functions is an extension of the Inv function for total functions.

Theorem 3.3.1 (Creignou et al. [10])**.** *Let* Γ_1,Γ_2 *be two sets of relations over the same domain; if* Γ_2 *is a finite subset of* $\langle\Gamma_1\rangle$ *then #CSP(*Γ_2*) is parsimoniously reducible to #CSP(*Γ_1*).*

Fleischer and Rosenberg [20] proved that for any constraint language Γ and any set of functions C, we have $Inv(pPol(\Gamma)) = \langle\Gamma\rangle$ and $pPol(Inv(C)) = \langle C\rangle$. On the other hand, $pPol$ and Inv functions maintain extensivity and antitony; hence, they form Galois connection between sets of relations and partial functions. Due to properties of Galois connections, for any constraint language Γ and any set of functions C, the sets $Inv(C)$ and $pPol(\Gamma)$ are a weak co-clone and a partial clone, respectively.

Chapter 4

Approximate Counting

In this chapter we formally define the complexity class which is considered as the efficient computational model for counting problems. We define AP-reductions and give a classification of computational problems with respect to AP-reductions. We use the definition of FPRAS from [13].

A *randomized approximation scheme* (RAS) for a function $f : \Sigma^* \to \mathbb{N}$ is a probabilistic algorithm that for an input $(x, \epsilon) \in \Sigma^* \times (0, 1)$ where x is an instance of f and ϵ is the error tolerance, produces an integer random variable z such that

$$Pr\left(\left|\frac{z - f(x)}{f(x)}\right| \leq \epsilon\right) \geq \frac{3}{4}.$$

A randomized approximation scheme is said to be *fully polynomial* if it runs in time $poly(|x|, \frac{1}{\epsilon})$. The phrase "fully polynomial randomized approximation scheme" is usually abbreviated to *FPRAS*. The complexity class FPRAS is referred to problems which have FPRAS.

Note that there is no significance in the constant $\frac{3}{4}$ in the definition, other than being in $(\frac{1}{2}, 1)$ interval. Jerrum et al. [30] proved that any success probability greater than $\frac{1}{2}$ can be improved to $1 - \delta$ for any desired δ by $O(\log \delta^{-1})$ trials of algorithm and taking the median of the results.

APX is another complexity class that is regarded as an efficient computational model for optimization problems. For a function $f : \Sigma^* \to \mathbb{N}$ and a constant factor α, f has an α-APX if there is an algorithm that takes $x \in \Sigma^*$ as an input and in time $poly(|x|)$ produces an integer z such that $\frac{1}{\alpha} \leq \frac{z}{f(x)} \leq \alpha$. In the same manner, poly-APX and log-APX are algorithms that given an input x, in time $poly(|x|)$ find a solution z such that

$\frac{1}{|x|^c} \leq \frac{z}{f(x)} \leq |x|^c$ for some constant c and $\frac{1}{\log|x|} \leq \frac{z}{f(x)} \leq \log|x|$, respectively. There are no results on #CSP(Γ) with APX schemas. With the next lemma, I explain why APX schemas are not used for problem #CSP(Γ) .

Lemma 4.0.2. *For a constraint language* Γ*, if there is a poly-APX for the problem #CSP(*Γ*) then there is an FPRAS it, as well.*

Proof. Let Alg be an algorithm and T be a polynomial function such that for any instance $\mathcal{P} = (D, V, \mathcal{C})$ of #CSP(Γ), we have

$$\frac{1}{T(|\mathcal{P}|)} \leq \frac{Alg(\mathcal{P})}{\#\mathcal{P}} \leq T(|\mathcal{P}|).$$

Choose k a sufficiently large number whose value will be determined later. Let $\mathcal{P}' = (D, V', \mathcal{C}')$ be k copies of $\mathcal{P}$ as

$$V' = \{x^i \mid 1 \leq i \leq k, x \in V\}$$

and

$$\mathcal{C}' = \{\langle (x^i_{l_1}, \dots, x^i_{l_t}), R\rangle \mid 1 \leq i \leq k, \langle (x_{l_1}, \dots, x_{l_t}), R\rangle \in \mathcal{C}\}.$$

By multiplication principle, $\#\mathcal{P}' = \#\mathcal{P}^k$; $\sqrt[k]{Alg(\mathcal{P}')}$ is an $(1+\epsilon)$-approximation for $\mathcal{P}$ if $1+\epsilon > \sqrt[k]{T(|\mathcal{P}| \cdot k)}$. For that, it is sufficient for k to be greater than $\frac{1}{\epsilon} \cdot \log T(|\mathcal{P}| \cdot k)$. Since the function T is polynomial in $|\mathcal{P}|$, k is also polynomially bounded by $|\mathcal{P}|$ and $\frac{1}{\epsilon}$. □

Definition 4.0.3 (AP-reduction)**.** For any two functions f and g, an *approximation-preserving reduction*(AP-reduction for short) from f to g is a probabilistic algorithm Alg which uses a solver Alg' for g and for any input $(x, \epsilon) \in \Sigma^* \times (0, 1)$ where x is an instance of f and ϵ is an error tolerance, produces an integer random variable z satisfying the following conditions:

- Alg' takes an input in the form $(w, \delta) \in \Sigma^* \times (0, 1)$ where w is an instance of g and δ is the error tolerance,
- if Alg' meets the specification for being a RAS for g then Alg meets the specifications for being a RAS for f,
- Alg runs in polynomial time in terms of $|x|$ and $\frac{1}{\epsilon}$

If an approximation-preserving reduction from f to g exists we write $f \leq_{AP} g$ and say f is *AP-reducible* to g. If $f \leq_{AP} g$ and $g \leq_{AP} f$ then we say that f and g are *AP-interreducible* and write $f \equiv_{AP} g$.

In this chapter we will study three major classes of counting problem with respect to complexity of approximate counting. We will study some of the important counting problem in FPRAS in Section 4.1, the problems that are hard to approximate in Section 4.2, and the problem #BIS and problems AP-interreducible with it in Section 4.3. In Section 4.4, we will mention some problems that not known to fit in this classification.

4.1 Polynomial Time Solvable Problems

Few non-trivial combinatorial problems involving counting can be solved in polynomial time. Some of these problems are reducible to determinant such as the problem of finding the number of spanning trees of a graph and the problem of finding the number of perfect matchings in a planar graph. Bulatov [2] showed that the problem #CSP(Γ) is polynomial time solvable for congruence singular relations Γ and otherwise it is #P-complete. In this section we show several examples of counting problems for which an approximate solution can be obtained in polynomial time.

There are some artificial examples of problem #CSP(Γ) that can be approximated in polynomial time.

Example 4.1.1. For the graph H shown in Figure 4.1, we show that the problem #Hom(H) is in FPRAS. Suppose the input is (G, ϵ) where G is a connected graph with n vertices and ϵ is the error tolerance. We know that $4^n \leq hom(G, H) \leq 4^n + 3^n$. For $\epsilon \geq (\frac{3}{4})^n$, take 4^n as an approximation, the error ratio will be

$$\frac{\hom(G,H) - 4^n}{\hom(G,H)} \leq \frac{3^n}{\hom(G,H)} \leq (\frac{3}{4})^n \leq \epsilon$$

For $\epsilon < (\frac{3}{4})^n$, we have $\frac{1}{\epsilon} > (\frac{4}{3})^n$; hence, $(\frac{1}{\epsilon})^{\log_{\frac{4}{3}} 3} > 3^n$; consequently, 3^n is polynomial in terms of $\frac{1}{\epsilon}$; which means the number of 3-colorings of G can be evaluated in $poly(n, \frac{1}{\epsilon})$ time. Note that $hom(G, H)$ is 4^n plus the number of 3-colorings of G.

The construction in Example 4.1.1 shows that for any graph G, there exists a graph H such that G is an induced subgraph of H and the problem #Hom(H) is in FPRAS. In other

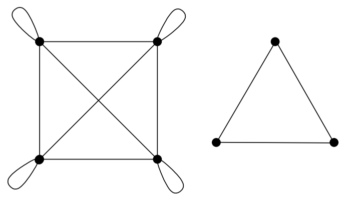


Figure 4.1: Graph G used in Example 4.1.1

words, although G is an induced subgraph of H, approximating the problem #Hom(H) may be easier than approximating the problem #Hom(G).

Another interesting counting problem which can be approximated in polynomial time is the problem #DNF-SAT defined as

Instance: A Boolean DNF (Disjunctive Normal Form) formula φ

Output: The number of satisfying assignments for φ

There are different algorithms to solve this problem. Luby and Veličkovic [33] provided a deterministic algorithm for approximating the probability of a random assignment being satisfying. Madras et al. [35] provided a Monte-Carlo algorithm for this problem. Jerrum et al. [30] provided an algorithm,depicted in Algorithm 1, for sampling the problem DNF-SAT.

Algorithm 1 An algorithm for sampling DNF-SAT

```
while true do
  Select a clause $C$ randomly with probability proportional to the size of the clause
  Select an assignment $A$ satisfying $C$ uniformly at random
  Let $N$ be the number of clauses satisfied by $A$
  With probability $\frac{1}{N}$: output $A$ and halt
end while
```

Since the problem #DNF-SAT is self-reducible, Theorem 2.1.8 implies that Algorithm 1 can be used to find an approximate solution for DNF-SAT.

Other common method used to approximate counting problems is using the Markov chain Monte-Carlo algorithms. Consider the following problems:

Name: #Match

Instance: A graph G

Output: The number of matchings in G

Name: #LowDegree-k-Coloring
Instance: A graph G such that $2\Delta(G) + 1 \leq k$, where $\Delta(G)$ is the maximum degree in G
Output: The number of proper k-Colorings of G

Jerrum and Sinclair [29] used Markov chain Monte-Carlo method to approximate the problem #Match; Jerrum [27] used the same method to approximate the problem #LowDegree-k-Coloring for graphs with bounded degree. We give a brief description of the Markov chain Monte Carlo method and show how these two problems are solved using this method.

The Markov chain Monte Carlo method solves a sampling problem as follows.

a Markov chain is finite automaton in which the transition between the states are labeled with probabilities of transition between the states. X_t is used to denote the state of the automaton at step t. Consider a Markov chain with state space Ω and stationary distribution π. Intuitively, a Markov chain is said to be *ergodic* if regardless of the initial state, the probability distribution over Ω asymptotically converges to π. In order to sample, start from an arbitrary state in Ω; simulate the Markov chain for T steps; finally output the final step. The number T should be chosen sufficiently large so that after T steps, the distribution of state is arbitrarily close to the desired distribution π. The number of steps required for the algorithm to become close enough to π is called the *mixing time*. Loosely, if the mixing time for a Markov chain is polynomial in terms of the size of the input and the desired approximation ratio, then the Markov chain is *rapidly mixing*.

The problem #LowDegree-k-Coloring is solved as follows. Let G be the input graph with n vertices and m edges. Consider a sequence of subgraphs of G such that $G_m = G$ and G_i is obtained from G_{i+1} by removing an arbitrary edge. For any graph G, let $\Omega_k(G)$ denote the set of k-colorings of G. The general idea is using the following formula to estimate $|\Omega_k(G)|$:

$$|\Omega_k(G)| = \frac{|\Omega_k(G_m)|}{|\Omega_k(G_{m-1})|} \times \frac{|\Omega_k(G_{m-1})|}{|\Omega_k(G_{m-2})|} \times \cdots \times \frac{|\Omega_k(G_1)|}{|\Omega_k(G_0)|} \times |\Omega_k(G_0)|$$

Trivially, $|\Omega_k(G_0)| = k^n$. All that remains is estimating the ratios ϱ_i defined as

$$\varrho_i = \frac{|\Omega_k(G_i)|}{|\Omega_k(G_{i-1})|}$$

for all values of i in the range $1 \leq i \leq m$.

Let $M(G_i, k)$ denote a Markov chain whose state space is $\Omega_k(G_i)$. The transition probabilities from the state X_t are modeled as

i) choose a vertex v and a color c uniformly at random

ii) recolor the vertex v with the color c; if the resulting coloring X' is proper then let $X_{t+1} = X'$; otherwise, let $X_{t+1} = X_t$.

Jerrum [27] showed that for $k \geq \Delta(G) + 2$, the $M(G_i, k)$ is ergodic and rapidly mixing with a uniform stationary distribution. For $X \in \Omega_k(G_{i-1})$, let $Z_i(X)$ be 1 if $X \in \Omega_k(G_i)$; otherwise, 0. Jerum also proved that by a high probability the expected value of $Z_i(X)$ over polynomial number of samples is an acceptable estimation for ϱ_i.

The general idea to solve the problem #Match is similar. Jerrum and Sinclair [29] solved a weighted version of the problem #Match which is called the problem Monomer-Dimer in statistical physics. It is defined as follows.
Name: Monomer-Dimer
Instance: A graph G and a positive real number λ
Output: The partition function $Z_G(\lambda)$ which is the sum of $\lambda^{|M|}$ for all matchings M of G

Note that $Z_G(1)$ is the number of matchings in G and $Z_G(0) = 1$ because for the empty matching M, $\lim_{\lambda \to 0^+} \lambda^{|M|} = 1$. The technique used to solve this problem is the same as the problem #LowDegree-k-Coloring; however, instead of removing edges from the graph, the sequence is obtained by reducing the value of λ. We will not go through details about this algorithm.

We defined the Ising model in Section 2.1. Jerrum and Sinclair [28] have proved that there exists an FPRAS for the problem of computing the partition function in Ising model if the model is consistent.

4.2 Problems Hard To Approximate

In this section we study the set of counting problems that are the hardest problems in #P for the complexity of approximation. In other words every problem in #P is AP-reducible to them. Existence of an FPRAS for any of these problems implies NP = RP where a problem is in RP if there is a randomized algorithm that runs in polynomial time and if the answer is NO always rejects and if the answer is YES accepts with probability of at least $\frac{3}{4}$. The following theorem points us to the first set of problems in this complexity class.

Theorem 4.2.1 (Goldberg et al. [12]). *For any NP-complete decision problem, the corresponding counting problem is complete for #P with respect to AP-reducibility.*

Corollary 4.2.2. *#SAT, #3-SAT, and* #3-Coloring *are complete for #P with respect to AP-reducibility.*

Another important problem AP-interreducible with the problem #SAT is the problem #IS defined as:
Instance: A graph G
Output: The number of independent sets in G

The decision problem IS is a trivial problem; every graph has at least an empty independent set. However, the counting problem #IS is proved to be AP-interreducible with the problem #SAT by Goldberg et al.[12]. Finding AP-reductions from many problems from the problem #IS is more straightforward than finding reductions from the problems that the decision version is NP-complete.

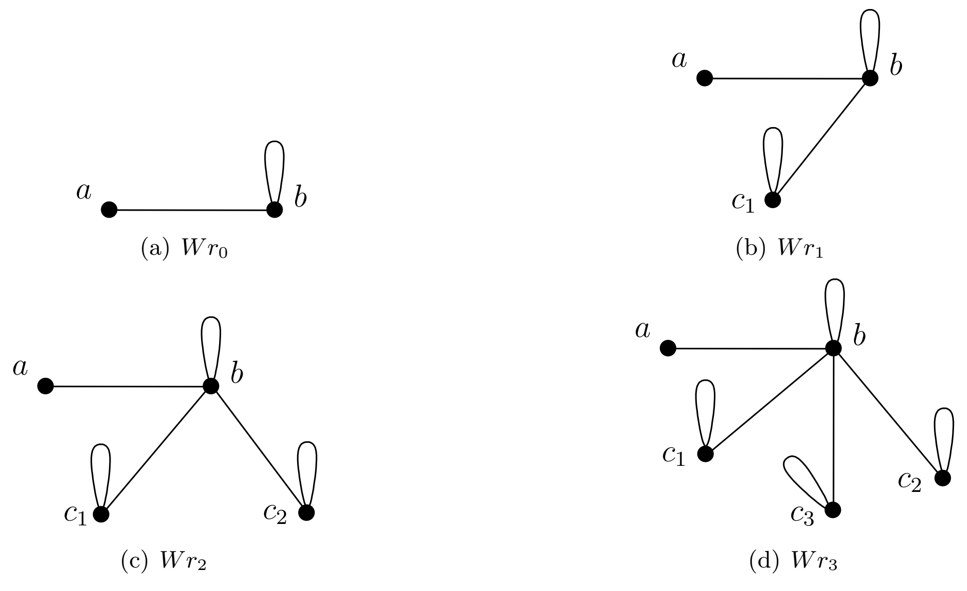


Figure 4.2: Wrench graphs Family

Example 4.2.3. Consider the binary relations OR and NAND on domain $\{0,1\}$ defined as $\text{OR} = \{(0,1),(1,0),(1,1)\}$ and $\text{NAND} = \{(0,0),(0,1),(1,0)\}$. For an instance $G = (V, E)$ of the problem #IS, consider an instance $\mathcal{P} = (\{0,1\}, V, \mathcal{C})$ of the problem #CSP(OR) where

$$\mathcal{C} = \{\langle (x,y), \text{OR}\rangle \mid xy \in E\}.$$

Every independent set I in G, corresponds to an assignment φ for $\mathcal{P}$ where $\varphi(x) = 0$ if $x \in I$ and $\varphi(x) = 1$ if $x \notin I$. Similarly, the problem #IS can be expressed by the #CSP(NAND) as well; hence, the problems #CSP(OR) and #CSP(NAND) are both AP-interreducible with the problem #SAT.

Although the problem 2-SAT (the decision problem) is polynomial time solvable, the problem #2-SAT can express #CSP(OR) and #CSP(NAND); hence, the problem #2-SAT is also AP-interreducible with the problem #SAT.

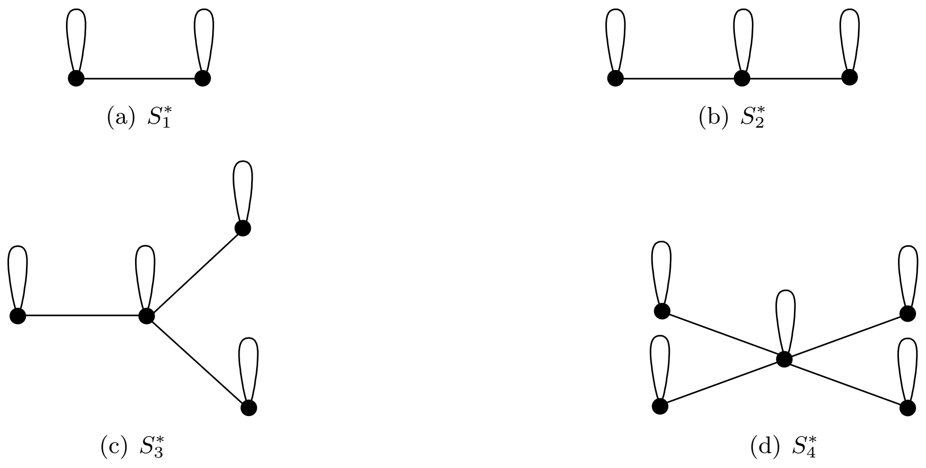
Example 4.2.4 (#Wrench-Col)**.** The Wrench graphs family are denoted by $Wr_q = (V_q, E_q)$ where $V_q = \{a, b, c_1, c_2, \ldots, c_q\}$ and $E_q = \{\{a,b\},\{b,b\}\} \cup \{\{b,c_i\},\{c_i,c_i\} : 1 \le i \le q\}$. Graphs Wr_0, Wr_1, Wr_2, and Wr_3 are shown in Figure 4.2. The problem #Hom(Wr_q) is referred to as the problem #q-Wrench-Col. Goldberg et al. [12] proved that for $q \neq 2$, the problem#q-Wrench-Col is AP-interreducible with the problem #SAT.

Example 4.2.5 (#q-Particle-WR-Configs)**.** For $q \geq 1$, homomorphisms to S_q^* where S_q^* is a q-leaf star with loops at all the $q+1$ vertices are configuration in the q-particle Widom-Rowlinson model. The problem #q-Particle-WR-Configs is defined as the problem #Hom(S_q^*). Graphs S_1^*, S_2^*, S_3^*, and S_4^* are shown in Figure 4.3.

The problem #Hom(S_1^*) is polynomial time solvable. We will consider the problems #Hom(S_2^*) and #Hom(S_3^*) later; Goldberg et al. [12] have proved that for $q \geq 4$, the problem #Hom(S_q^*) is AP-interreducible with the problem #SAT.

We defined the Ising and Potts models in Section 2.1. Jerrum and Sinclair [28] showed that the problem of computing the partition function in Ising model is AP-interreducible with the #SAT problem if the model is not consistent.

Jerrum and Goldberg [22] showed that the problem of computing the partition function in Potts model with more that two states is AP-interreducible with the #SAT problem. They also showed that the problem of computing the partition function in Ising model is AP-interreducible with the problem #SAT if the model is not ferromagnetic.



(a) S_1^* (b) S_2^* (c) S_3^* (d) S_4^*

Figure 4.3: Particle-WR-Configs

4.3 Problems AP-interreducible With #BIS

The proof used in [12] to show that the #SAT problem is AP-reducible to the problem #IS does not work if the input for the problem #IS is limited to bipartite graphs. The version of the problem #IS with input limited to bipartite graphs is called the problem #BIS and is defined as follows:

Instance: A bipartite graph G

Output: The number of independent sets in G

Note that by Theorem 3.2.3 exactly solving the problem #BIS is #P-complete. Goldberg et al. [12] proved that the following problems are AP-interreducible with #BIS.

Name: #DownSets

Instance: A partial order $(P, \preceq)$

Output: The number of down-sets in P

In a partial order $P = (X, \preceq)$, a down-set (also called lower-set) is a set D such that for every $x \in D$ if there exist $y \preceq x$ then $y \in D$, as well. Note that the number of down-sets in a partial order equals the number of anti-chains in the same partial order.

Name: #1P1N-SAT

Instance: A Boolean CNF formula φ with clauses of size one, or size two such that there is one negative and one positive literal per clause

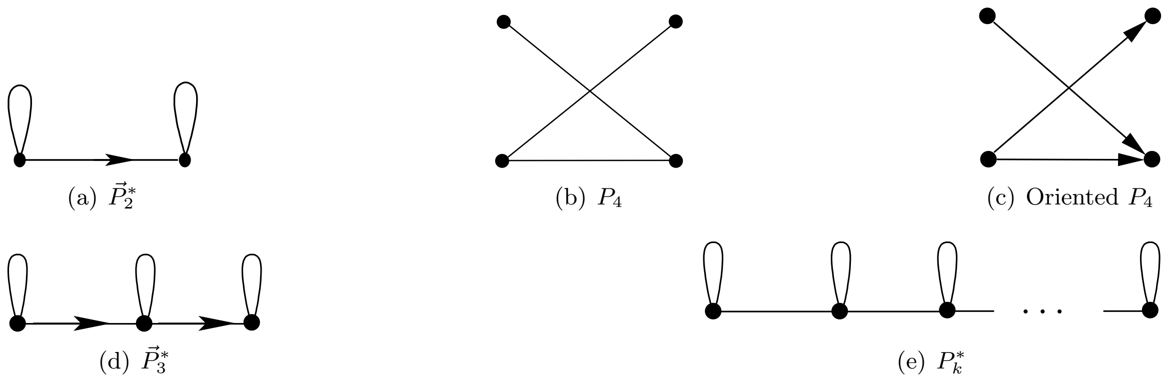


Figure 4.4: Several graphs AP-interreducible with #BIS

Output: The number of satisfying assignments of φ

For many graphs, the problem #Hom(H) is also AP-interreducible with the problem #BIS. For example, the problem #Hom(H) is AP-interreducible with the problem #BIS if H is any of the graphs $\vec{P}_2^*$, P_4, $\vec{P}_4$, $\vec{P}_3^*$, or $P_k^*(k \geq 3)$, which are shown in Figure 4.4.

There are also problems from statistical physics that are AP-interreducible with the problem #BIS. The Ising model is described in Section 2.1. The problems #2-Wrench-Coloring and #2-Particles-WR-Configs are described in Examples 4.2.4 and 4.2.5, respectively.

Name: Ferromagnetic Ising
Instance: A graph G, inverse temperature β, interaction strengths J such that $J_{u,v} > 0$, local external magnetic field l
Output: The partition function $Z(G, \beta, J, l)$

Name: #2-Wrench-Coloring
Instance: A graph G
Output: The number of homomorphisms from G to WR_2

Name: #2-Particles-WR-Configs
Instance: A graph G
Output: The number of homomorphisms from G to S_2^*

Goldberg and Jerrum [22] proved that the problem Ferromagnetic Ising is AP-interreducible with the problem #BIS and Goldberg et al. [12] proved that the problems #2-Wrench-Coloring and the #2-Particles-WR-Configs are AP-interreducible with the problem #BIS.

Here we prove that the two characterizing problems in this class, i.e., #BIS and #DownSets, are AP-interreducible.

Lemma 4.3.1. *#DownSets $\leq_{AP}$ #BIS.*

We use the proof from [13]. We denote the number of down-sets in a partial order P by #DS(P) and the number of independent sets in a graph G by #IS(G).

Proof. Let $P = (X = [n], \preceq)$ be an instance of the problem #DownSets. Let $B = (U, V, E)$ be a bipartite graph defined as follows. For i in X, let U_i and V_i be a collection of disjoints sets of size $2n$. Then, take $U = \bigcup_{1\leq i\leq n} U_i$, $V = \bigcup_{1\leq i\leq n} V_i$, and

$$E = \{uv \mid u \in U_i \wedge v \in V_j \wedge i \preceq j\}.$$

An independent set I in B is said to be *full* if for all $1 \leq i \leq n$ the set $I \cap (U_i \cup V_i)$ is nonempty. Every full independent set I in B corresponds to the down-set $D = \{i \mid I \cap V_i \neq \emptyset\}$ in P. In the same manner, every down-set D in P gives exactly $(2^{2n} - 1)^n$ full independent sets in B. Note that $2^{2n} - 1$ is the number of nonempty subsets of $U_i \cap V_i$.

On the other hand the number of non-full independent sets in B is less than $3^n(2^{2n} - 1)^{n-1}$. Thus, for $n \geq 5$ we have

$$\#\text{DS}(P) = \left\lfloor \frac{\#\text{IS}(P)}{(2^{2n} - 1)^n} \right\rfloor.$$

□

Lemma 4.3.2. *#BIS $\leq_{AP}$ #DownSets.*

The proof used in [13] involves reduction to several other problem. Here we present our own proof which is the special case of Theorems we will be using in Chapter 6.

Proof. Let $B = (U, V, E)$ be a bipartite graph. We will create a partial order $P = (X, \preceq)$ such that #IS(B) = #DS(P). Take $X = U \cup V$ and $u \preceq v$ if and only if $uv \in E$. We claim that for every down-set D in P, $I = (U \cap D) \cup (V - D)$ is an independent set and for any independent set I in B the set $D = (U \cap I) \cup (V - I)$ is a down-set in P.

Let D be a down-set in P. Since $U \cap D$ and $V - D$ are subsets of U and V respectively, they are both independent sets. There is no edge from V to U and since D is a DownSet all the neighbors of $U \cap D$ rely in $V \cap D$ and none of them are in $V - D$; hence, the set $I = (U \cap D) \cup (V - D)$ is an independent set in B. On the other hand, let I be an independent set in B. There is no out-going edge from any vertex in V and since I is an independent set out-neighbors of $U \cap I$ are limited to $V - I$; hence, the set $D = (U \cap I) \cup (V - I)$ is a down-set in P.

Hence, we have a parsimonious reduction from the problem #BIS to the problem #DownSets. □

4.4 Other Difficult Problems

Goldberg et al. [15] proved an approximation trichotomy for the Boolean problem #CSP(Γ). A Boolean relation R is *monotone* if R is closed under $\wedge$ and $\vee$ operators; a Boolean constraint languages Γ is monotone if every relation in Γ is monotone.

Theorem 4.4.1 (Dyer, Goldberg, Jerrum, 2007 [15])**.** *For a constraint language Γ, if Γ is affine then #CSP(Γ) is polynomial time solvable; otherwise, if Γ is monotone then it is AP-interreducible to the problem* #BIS*; otherwise, it is AP-interreducible with the problem #SAT.*

Despite the approximation trichotomy for the Boolean problem #CSP(Γ), the non-Boolean problem #CSP(Γ) seems to have more complexity classes. So far, our knowledge on complexity classes of the problem #CSP(Γ) is very limited. We are not aware if the approximation complexity classes for this problem are finite or even countable. We know a few problems that are only proven to be harder than the problem #BIS [12] and currently, we have no knowledge on how hard these problems are.

Here are some of these problems. #3-Particles-WR-Configs was introduced in Example 4.2.5.

Name: #3-Particles-WR-Configs
Instance: A graph G
Output: The number of homomorphisms from G to Wr_3

Name: #Bipartite q-Coloring for $q \geq 3$
Instance: A bipartite graph G

Output: The number of q-Colorings of G

Chapter 5

Techniques for AP-Reductions

In this chapter we provide a number of lemmas which can be used for obtaining AP-reductions. These results have been communicated in [4], [5]. First, we show that we can limit the input problem to connected structures. Next, we show that AP-reductions are possible when solutions for the two problems are related by a linear transformation. Next, we introduce a technique called pinning , which can be also used to derive AP-reductions. Finally, we introduce max-definability, which is our best tool for deriving AP-reductions. Some of these techniques will be used in subsequent chapters.

5.1 Connected graphs

First, we show that by restricting the input to connected structures the complexity of #CSP(Γ) does not change. We extend the definition of connectedness from graphs to general structures.

Let $\mathcal{P} = (D, V, \mathcal{C})$ be an instance of #CSP(Γ). Let H($\mathcal{P}$) be a graph with the vertex set V, that contains an edge uv if and only if u and v appear in the same scope for a constraint in $\mathcal{C}$. We say an instance $\mathcal{P}$ is *connected* if the graph $H(\mathcal{P})$ is connected. Let #CSP$_c$(Γ) denote the problem #CSP(Γ) limited to connected instances.

Lemma 5.1.1. *For any constraint language* Γ *the problem #CSP(*Γ*) is AP-interreducible with #CSP*$_c$*(*Γ*).*

Proof. The reduction of #CSP$_c$(Γ) to #CSP(Γ) is trivial. Now, let $\mathcal{P}$ be an instance of #CSP(Γ), and let $\mathcal{P}_1, \ldots, \mathcal{P}_r$ be its connected components. Take $\epsilon > 0$ and set $\delta = \frac{\epsilon}{2r}$.

Given an instance $(\mathcal{P}, \epsilon)$, our reduction calls the algorithm for $\#\mathrm{CSP}_c(\Gamma)$ on instances $(\mathcal{P}_1, \delta), \ldots, (\mathcal{P}_r, \delta)$ and get outputs $N = N_1 \cdots N_r$, where N_i is the answer given by the oracle on $(\mathcal{P}_i, \delta)$.

We claim that the above reduction is an AP-reduction. First of all, observe that it is polynomial time, and the instances it produces satisfy the conditions of AP-reductions. It remains to show that if the oracle approximates the solutions with relative error δ then the reduction provides approximation within ϵ.

Since we can assume ϵ is small, we have $(1-\delta)^r \geq 1-2r\delta = 1-\epsilon$ and $(1+\delta)^r \leq 1+2r\delta = 1+\epsilon$. If the actual solutions to $\mathcal{P}, \mathcal{P}_1, \ldots, \mathcal{P}_r$ are $N', N'_1, \ldots, N'_r$, then we obviously have $N' = N'_1 \cdots N'_r$. The rest of the proof goes as follows.

$$\begin{array}{rcl}
 & \left|\frac{N'_i - N_i}{N_i}\right| & \leq \delta \\
 & \left|\frac{N'_i}{N_i} - 1\right| & \leq \delta \\
1-\delta \leq & \frac{N'_i}{N_i} & \leq 1+\delta \\
(1-\delta)^r \leq & \frac{N'_1 \cdot N'_2 \cdots N'_r}{N_1 \cdot N_2 \cdots N_r} & \leq (1+\delta)^r \\
1-\epsilon \leq & \frac{N'}{N} & \leq 1+\epsilon \\
 & \left|\frac{N'}{N} - 1\right| & \leq \epsilon \\
 & \left|\frac{N'-N}{N}\right| & \leq \epsilon.
\end{array}$$

□

Hence, in the later proofs we can restrict the input structures to connected structures.

5.2 Linear Transformations

Next, we show that if the solutions of $\#\mathrm{CSP}(\Gamma)$ and $\#\mathrm{CSP}(\Gamma')$ are linearly related on all inputs, then they are AP-interreducible.

Lemma 5.2.1. *Let φ be a polynomial time computable function that maps every instance of a counting problem A to an instance of a counting problem problem B in such a way that there are constants $d > 0$ and c (not necessarily positive) such that for any A-instance $\mathcal{P}$, we have $\#\mathcal{P} = d \cdot \#\varphi(\mathcal{P}) + c$. If every instance of A has at least one solution (in case $c < 0$), or every instance of B has at least one solution (in case $c > 0$), then there is an AP-reduction from A to B.*

Proof. The AP-reduction works as follows. On an instance $\mathcal{P}$ of A and $\epsilon > 0$ it makes an oracle call $(\varphi(\mathcal{P}), \delta)$ to B, where $\delta = \frac{\epsilon}{p}$ for $c > 0$, $p = 1$ otherwise, $p = \frac{1}{-}c/d$ and if the oracle's reply is N, it returns $dN + c$. Clearly the algorithm make polynomially many steps and oracle calls, and the oracle request is of the correct form. Thus, it suffices to show that if the oracle's solution is within relative error of δ then the algorithm gives an $1 + \epsilon$-approximation of $\#\mathcal{P}$.

Let the exact and approximation solutions for $\varphi(\mathcal{P})$ be N' and N, respectively; then the exact and approximation solutions for $\mathcal{P}$ are $dN' + c$ and $dN + c$, respectively. Since $\mathcal{P} > 0$, we can assume that $dN + c, dN' + c, N, N' > 0$. With the choice of p, it is trivial that $\frac{dN}{dN+c} \leq p$. Thus we have $\frac{dN}{dN+c}\delta \leq \epsilon$. All we need to show is that if $\left|\frac{N-N'}{N}\right| < \delta$ holds then $\left|\frac{dN-dN'}{dN+c}\right| < \epsilon$ holds as well. This is seen as follows:

$$\begin{aligned} \frac{dN}{dN+c}\delta &\leq \epsilon \\ 1 + \frac{dN}{dN+c}\delta &\leq 1 + \epsilon \\ \frac{dN(1+\delta)+c}{dN+c} &\leq \epsilon. \end{aligned}$$

The inequality $\epsilon \leq \frac{dN(1-\delta)+c}{dN+c}$ is similar. Combining these with the assumption that $1 - \delta \leq \frac{N'}{N} \leq 1 + \delta$, we get $1 - \epsilon \leq \frac{dN'+c}{dN+c} \leq 1 + \epsilon$. We continue with the proof as follows:

$$\begin{aligned} 1 - \epsilon \leq \frac{dN'+c}{dN+c} &\leq 1 + \epsilon \\ \left|1 + \frac{dN'+c}{dN+c}\right| &\leq \epsilon \\ \left|\frac{dN - dN'}{dN+c}\right| &\leq \epsilon. \end{aligned}$$

This completes the proof. □

Lemma 5.2.2. *Let φ be a polynomial time computable function that maps every instance of a counting problem A to an instance of a counting problem problem B in such a way that there is a constant $m > 0$ such that for any A-instance $\mathcal{P}$ we have $\#\mathcal{P} = (\#\varphi(\mathcal{P}))^m$. Then there is an AP-reduction from A to B.*

Proof. The AP-reduction works as follows: on an instance $\mathcal{P}$ of A and $\epsilon > 0$ it makes an oracle call $(\varphi(\mathcal{P}), \delta)$ to B, where $\delta = \frac{\epsilon}{m}$, and and if the oracle's reply is N, it returns N^m.

Clearly the algorithm make polynomially many steps and oracle calls, and the oracle request is of the correct form. Thus, it suffices to show that if the oracle's solution is within relative error of δ then the algorithm gives an $1+\epsilon$-approximation of $\#\mathcal{P}$.

Let the exact and approximation solutions for $\varphi(\mathcal{P})$ be N' and N, respectively; then the exact and approximation solutions for $\mathcal{P}$ are N'^m and N^m, respectively. All we need to show is that if $\left|\frac{N-N'}{N}\right| < \delta$ holds then $\left|\frac{N^m-N'^m}{N^m}\right| < \epsilon$ holds as well. Without loss of generality we assume $N > N'$, and proceed as follows:

$$\begin{array}{rcl} \frac{|N-N'|}{N} & \leq & \delta \\ \frac{N'}{N} & \leq & 1+\delta \\ \frac{N'^m}{N^m} & \leq & (1+\delta)^m \\ \frac{N'^m}{N^m} & \leq & 1+m\cdot\delta \\ \frac{N'^m}{N^m} & \leq & 1+\epsilon \\ \frac{N^m-N'^m}{N^m} & \leq & \epsilon. \end{array}$$

The inequality $\epsilon \leq \frac{N'^m-N^m}{N^m}$ is similar. This completes the proof. □

5.3 Projection

Let R be a k-ary relation and $S = \{i_1,\ldots,i_l\} \subseteq \{1,\ldots,k\}$. By $\pi_S R$ we denote the *projection* of R onto the set S of its coordinate positions, that is, the relation $\{(a_{i_1},\ldots,a_{i_l}) \mid (a_1,\ldots,a_k) \in R\}$. Observe that $\pi_S R$ is pp-definable in R by quantifying away all coordinate positions of R except for those in S. Although existential quantification is not known to give rise to AP-reducible problems, in some cases it does.

Lemma 5.3.1. *Let Γ be a constraint language over the domain D and let R be some k-ary relation in Γ. If there is a set $S \subset D$ such that $|\pi_S R| = |R|$ then #CSP($\Gamma \cup \{\pi_S R\}$) $\leq_{AP}$ #CSP(Γ).*

Proof. The AP-reduction is constructed as follows: Given an instance $\mathcal{P} = (D, V, \mathcal{C})$ of #CSP($\Gamma \cup \{\pi_S R\}$), we define an instance $\mathcal{P}' = (D, V', \mathcal{C}')$ of $\#CSP(\Gamma)$ with the same number of solutions. Let $l = |S|$. V' includes all the variables of V and $\mathcal{C}'$ includes all constraints from $\mathcal{C}$ except for those with $\pi_S R$. For each constraint $C = \langle \pi_S R, (v_1, \ldots, v_l) \rangle$ in $\mathcal{C}$ we include a new constraint $C' = \langle R, (w_1, \ldots, w_k) \rangle$ in $\mathcal{C}'$, where $w_i = v_{i_j}$ if $i \in S$ and $i = i_j$, otherwise a fresh variable.

Clearly, the restriction of any solution of $\mathcal{P}'$ onto V is a solution of $\mathcal{P}$. Furthermore, the condition $|\pi_S R| = |R|$ implies that any solution of $\mathcal{P}$ can be extended to a solution of $\mathcal{P}'$ in a unique way. $\square$

5.4 Pinning

Theorem 3.3.1 implies that if relation R can be expressed as conjunctions of relations from Γ, #CSP($\Gamma \cup \{R\}$) $\leq_{AP}$ #CSP(Γ). The conditions for this theorem, even for parsimonious reductions, are too strict. In the rest of this chapter we provide several more lenient conditions.

The ability to tie certain CSP variables to specific values in hardness proofs is *pinning*. This idea was introduced by Creignou and Hermann [9]. The idea is also used in many other proofs [3, 16, 15, 10]. Pinning can be viewed as showing that for a constraint language Γ and a set S, #CSP($\Gamma \cup \{S\}$) $\leq_{AP}$ #CSP(Γ) holds.

Lemma 5.4.1 (Pinning and reflexive elements). *Let Γ be a set of relations on a set D, and let for a certain subset $S \subset D$ there be a relation R such that $x \in S$ if and only if $(x, x, \ldots, x) \in R$. If such a relation R exists in Γ then #CSP($\Gamma \cup \{S\}$) $\leq_{AP}$ #CSP(Γ).*

Proof. S can be viewed as a unary relation and can be expressed as a predicate $\varphi(x) = R(x, x, \ldots, x)$. By Theorem 3.3.1 #CSP($\Gamma \cup \{S\}$) $\leq_{AP}$ #CSP(Γ). $\square$

Next lemma generalizes the Pinning Lemma from [15]. This lemma allows one to add an extra unary relation to the constraint language. The proof of Lemma 5.4.2 also follows closely the proof in [15].

Lemma 5.4.2 (Extended Pinning). *Let Γ be a constraint language over the set D of size k, and let for a certain subset $S \subset D$ there be an l-ary relation $R \in \Gamma$ and a coordinate position j, $1 \leq j \leq l$, such that for any $a \in S$ the relation R has more tuples $\boldsymbol{a}$ with $\boldsymbol{a}[j] = a$ than tuples $\boldsymbol{b}$ with $\boldsymbol{b}[j] \notin S$. Then $\#CSP(\Gamma \cup \{S\}) \leq_{AP} \#CSP(\Gamma)$.*

Proof. Fix an l-ary relation $R \in \Gamma$, and a coordinate position j such that R and j satisfy the conditions of the lemma. Let also w be the minimal (over elements $a \in S$) number of tuples $\mathbf{a}$ such that $\mathbf{a}[j] = a$, and let w' be the number of tuples $\mathbf{b}$ with $\mathbf{a}[j] \notin S$. By the conditions of the lemma $w' < w$.

Consider an instance $\mathcal{P} = (D, V, \mathcal{C})$ of #CSP($\Gamma \cup \{S\}$) with n variables. Let N_S be the set of variables which occur in the scope of constraints of $\mathcal{C}$ with relation S. Set $n_S = |N_S|$ and $m = \lceil \frac{n+2}{\lg \frac{w}{w'}} \rceil$. Construct an instance $\mathcal{P}' = (D, V', \mathcal{C}')$ of $\#CSP(\Gamma)$ as follows:

- V' includes all variables from V, and also, for each variable $x \in N_S$, any $u \in \{1, 2, \ldots, m\}$, and any $v \in \{0, 1, \ldots, k\} - \{j\}$ a fresh variable $x_{u,v}$.
- $\mathcal{C}'$ includes all constraints from $\mathcal{C}$ other than those involving S.
- For each constraint $C = \langle S, (x)\rangle$ from $\mathcal{C}$ include m constraints whose relation is R, variable x occupies the jth position in the scope, and the variable $x_{u,v}$ is in the vth position of the uth constraint.

Now any solution of $\mathcal{P}$ can be extended in at least w^{mn_S} ways to a solutions of $\mathcal{P}'$, provided all variables from N_S take values from S. On the other hand, every assignment that does not satisfy this condition can be extended in at most $w^{m(n_S-1)}w'^m$ ways. There exist no more than k^n such solutions. Therefore if N and N' denote the number of solutions to $\mathcal{P}$ and $\mathcal{P}'$, respectively, then

$$N \cdot w^{mn_S} \leq N' \leq N w^{mn_S} + k^n w^{m(n_s-1)} w'^m.$$

So, for a properly chosen m,

$$N \leq \frac{N'}{w^{mn_S}} \leq N + \frac{1}{4},$$

By Lemma 5.2.1 the linear transformations preserve AP-reduction and this completes the proof. □

Example 5.4.3. Let $R = \{(1,2),(1,3),(1,4),(2,1),(2,2),(2,3),(3,1),(3,4)\}$ be a relation over domain $D = \{1, 2, 3, 4\}$ and let $S = \{1, 2\}$.

The number of tuples $(1, x) \in R$ is 3, the number of tuples $(2, x) \in R$ is also 3; however, the number of tuple $(x, y) \in R$ where $x \in \{3, 4\}$ is 2 which less than 3. By Lemma 5.4.2, #CSP({R, S}) is AP-reducible to #CSP({R}).

As an application, we can now use this fact to deduce

$$\#\text{IS} \leq_{AP} \#\text{CSP}(\{R\}).$$

We first show that #IS$\leq_{AP}$#CSP($\{R, S\}$). For any input graph $G = (V, E)$ for the problem #IS, take $\mathcal{P} = (\{1, 2, 3, 4\}, V, \mathcal{C})$ as an instance of #CSP($\{R, S\}$) where

$$\mathcal{C} = \{\langle R, (x, y)\rangle \mid xy \in E\} \cup \{\langle S, (x)\rangle \mid x \in V\}.$$

Let φ be a solution of $\mathcal{P}$. For any vertex $x \in V$, $\varphi(x) \in \{1, 2\}$. Thus, for any edge xy in G, $(\varphi(x), \varphi(y)) \in \{(1, 2), (2, 1), (2, 2)\}$. Hence, there is a one-to-one correspondence between s independent sets G and solutions of $\mathcal{P}$. Thus,

$$\#\text{IS} \leq_{AP} \#\text{CSP}(\{R, S\}) \leq_{AP} \#\text{CSP}(\{R\})$$

5.5 Maximization

Maximization is a powerful tool to prove hardness results among many problems #CSP(Γ). We believe this technique may play a role in proving a classification theorem for the approximability of the problems #CSP(Γ).

Definition 5.5.1 (Max-implementation)**.** Let Γ be a set of relations over the domain D, and R be an n-ary relation over the same domain. Let $\mathcal{P}$ be in an instance of $\#CSP(\Gamma)$ over the set of variables consisting of $V = V_x \cup V_y$, where $V_x = \{x_1, x_2, \ldots, x_n\}$ and $V_y = \{y_1, y_2, \ldots, y_m\}$. For any assignment $\varphi : V_x \to D$, let $\#\varphi$ be the number of assignments $\psi : V_y \to D$ such that $\varphi \cup \psi$ satisfy $\mathcal{P}$. Let M be the maximum value of $\#\varphi$ among all assignments of V_x. The instance $\mathcal{P}$ is said to be a *max-implementation* of R if a tuple φ is in R if and only if $\#\varphi = M$.

If R has a max-implementation over Γ, we say it is *max-implementable* by Γ.

Example 5.5.2. Let $\mathcal{P} = (\{1, 2, 3\}, \{x_1, x_2, y_1, y_2, y_3\}, \mathcal{C})$ be an instance of #CSP(Γ). Suppose the numbers of solutions of $\mathcal{P}$ for each possible assignment of x_1, x_2 are according to the Table 5.1. Here, the maximum possible value for the solutions, M in the definition, is 7. Let R be the set of values for x_1, x_2 such that the number of solutions of $\mathcal{P}$ is 7, that is

$$R = \{(1, 1), (1, 2), (1, 3), (2, 2), (2, 3), (3, 3)\}.$$

Here, $\mathcal{P}$ is a max-implementation of the relation R by the constraint language Γ.

Theorem 5.5.3 (Maximization)**.** *Let Γ be a set of relations over the domain D, and R be an n-ary relation on the same domain. If R is max-implementable by Γ then #CSP($\Gamma \cup \{R\}$)$\leq_{AP}$#CSP(Γ).*

x_1	1	1	1	2	2	2	3	3	3
x_2	1	2	3	1	2	3	1	2	3
$\#\mathcal{P}$	7	7	7	5	7	7	4	3	7

Table 5.1: Number of solutions of $\mathcal{P}$ for each possible assignment of x_1, x_2

Proof. Let $\mathcal{P}$ be a max-implementation of R (see Definition 5.5.1), For any instance $\mathcal{P}_1 = (D, V_1, \mathcal{C}_1)$ of #CSP($\Gamma \cup R$) we construct in instance $\mathcal{P}_2 = (D, V_2, \mathcal{C}_2)$ of #CSP(Γ) as follows:

- Choose a sufficiently large integer m (to be determined later)
- Let $C_1, C_2, \ldots, C_l \in \mathcal{C}$ be constraints from $\mathcal{P}_1$ involving R, say $C_i = \langle \varrho_i, R \rangle$. Set $V_2 = V_1 \bigcup_{i=1}^{l} (V_1^i \cup \cdots V_m^i)$, where each V_j^i is a separate copy of V_y from Definition 5.5.1.
- Let $\mathcal{C}$ be the set of constraints of $\mathcal{P}$. Set $\mathcal{C}_2 = (\mathcal{C}_1 - \{C_1, \ldots, C_l\}) \cup \bigcup_{i=1}^{l} (C_1^i \cup \cdots C_m^i)$, where each C_j^i is a copy of $\mathcal{C}$ defined as follows. For each $\langle \varrho, Q \rangle \in \mathcal{C}$, we include $\langle \varrho_j^i, Q \rangle$ into C_j^i where ϱ_j^i is obtained from ϱ by replacing every variable from V_y with its copy from V_j^i.

Now, it can be easily seen that every solution of $\mathcal{P}_1$ can be extended to a solution of $\mathcal{P}_2$ in M^{lm} ways. Observe that sometimes the restriction of a solution ψ of $\mathcal{P}_2$ to V_1 is not a solution of $\mathcal{P}_1$. Indeed, it may happen that although ψ satisfies every copy of $\mathcal{C}_j^i$, its restriction to ϱ_j^i does not belong to R; however, this restriction does not have sufficiently many extensions to solutions of $\mathcal{P}_1$. On the other hand, any assignment to V_1 that is not a solution to $\mathcal{P}_1$ can be extended to a solution of $\mathcal{P}_2$ in at most $(M-1)^m \cdot M^{(l-1)m}$ ways. Hence,

$$M^{lm} \cdot \#\mathcal{P}_1 \leq \#\mathcal{P}_2 \leq M^{lm} \cdot \#\mathcal{P}_1 + |V|^{|D|} \cdot (M-1)^m \cdot M^{(l-1)m}.$$

The we output $\lfloor N \rfloor$, where $N = \#\mathcal{P}_2 / M^{lm}$.

We want to choose m large enough such that the following holds.

$$\begin{aligned}
|V|^{|D|} \cdot \left(\frac{M-1}{M}\right)^m &\leq 1 \\
\log(|V|^{|D|}) + \log\left(\left(1 - \frac{1}{M}\right)^m\right) &\leq 0 \\
|D| \cdot \log|V| + m \cdot \log\left(1 - \frac{1}{M}\right) &\leq 0 \\
|D| \cdot \log|V| &\leq m \cdot - \log\left(1 - \frac{1}{M}\right) \\
\frac{|D| \cdot \log|V|}{-\log\left(1 - \frac{1}{M}\right)} &\leq m.
\end{aligned}$$

For any $0 < x < 1$ we have $log(1-x) > x$; hence,

$$\frac{|D| \cdot \log|V|}{-\log\left(1 - \frac{1}{M}\right)} \leq M \cdot |D| \cdot \log|V|$$

This implies for $m \geq M \cdot |D| \cdot \log|V|$, we have $\#P = \lfloor N \rfloor$. □

As an example of the applications of the theorem we mention a corollary bellow.

Let R be a relation of arity l over D. For $x \in D$, we denote by $s_{R,i}(x)$ the number of tuples in R in which x is in position i.

Corollary 5.5.4. *If there exists a set $S \subseteq D$ and real numbers $a_1, a_2, \ldots, a_l, M$, such that for every $x \in S$ we have*

$$\sum_{i=1}^{l} a_i \cdot s_{R,i}(x) = M,$$

and for every $x \notin S$ we have

$$\sum_{i=1}^{l} a_i \cdot s_{R,i}(x) < M,$$

then $\#CSP(\{R, S\}) \leq_{AP} \#CSP(R)$.

Chapter 6

Results on Complexity of #BIS

In this chapter and the next one, after discussing general results on #CSP(Γ) over monotone Γ, we use #BIS as a yardstick to classify some #Hom(H) as "not too hard" or "not too easy". We first show that for monotone relations, there is an AP-reduction from the problem #CSP(Γ) to the problem #BIS. Then, we show that there is an AP-reduction from the problem #BIS to the problem #Hom(H) if H is a reflexive oriented graph.

For constraint languages consisting of a binary relation, the problem #CSP(Γ) is the same as the problem #Hom(H); hence, we use them interchangeably depending on the on the context.

6.1 Monotone Relations

In this section, we extend the definition of *monotone* constraint languages defined in [15] from Boolean domains to general domains and show that for any monotone relation Γ there is an AP-reduction from the problem #CSP(Γ) to the problem #BIS.

Definition 6.1.1 (Distributive Lattice). A *distributive lattice* $L = (X, \wedge, \vee)$ is a set X with a pair of binary operators $\wedge$ and $\vee$ on X that are

- idempotent: $x \wedge x = x$ and $x \vee x = x$,
- commutative: $x \wedge y = y \wedge x$ and $x \vee y = y \vee x$,
- associative: $x \wedge (y \wedge z) = (x \wedge y) \wedge z$ and $x \vee (y \vee z) = (x \vee y) \vee z$,
- absorptive: $x \wedge (x \vee y) = x$ and $x \vee (x \wedge y) = x$,

- distributive: $x \wedge (y \vee z) = (x \wedge y) \vee (x \wedge z)$ and $x \vee (y \wedge z) = (x \vee y) \wedge (x \vee z)$

Definition 6.1.2 (Monotone relation)**.** Let Γ be a constraint language over the domain D; Γ is *monotone* if there exists distributive lattice order $L = (D, \wedge, \vee)$ such that every relation in Γ is closed under operations $\vee$ and $\wedge$.

Theorem 6.1.3 is based on an unpublished manuscript by Hedayaty and Bulatov(2009).

Theorem 6.1.3. *Let* Γ *be a constraint language over a domain* D*, if* Γ *is monotone then #CSP(*Γ*)* $\leq_{AP}$#BIS*.*

Proof. We reduce the #CSP(Γ) from the general domain to Boolean domain. The proof is easy. Let Γ be a monotone constraint language with distributive lattice $L = (D, \wedge, \vee$. By the Birkhoff's Representation Theorem, every finite distributive lattice is isomorphic to a lattice ordered Boolean vector space of dimension k. Denote the isomorphism functions by *cube* : $D \rightarrow \{0,1\}^k$ and $cube^{-1} : \{0,1\}^k \rightarrow D$. The definition of the *cube* function is extended to tuples, as $cube(x_1, \ldots, x_l) = (cube(x_1), \ldots, cube(x_l))$. The definition of *cube* is further extended to relations as $cube(R) = \{cube(\varrho) \mid \varrho \in R\}$.

Take $\Gamma' = \{cube(R) \mid R \in \Gamma\}$. The functions *cube* and $cube^{-1}$ form a bijection between the solutions for any CSP(Γ) instance and the corresponding CSP(Γ') instance. Hence, there is a parsimonious reduction between the problem #CSP(Γ) and the problem #CSP(Γ'). Now, we prove that Γ' is monotone. For any relation $R' \in \Gamma'$ we have

$$\mathbf{x} \in R \Leftrightarrow cube(\mathbf{x}) \in R'$$

We show that for any two tuples $\mathbf{x}, \mathbf{y} \in R'$ we have $\mathbf{x} \vee \mathbf{y}, \mathbf{x} \wedge \mathbf{y} \in R'$. Namely,

$$\begin{aligned} \mathbf{x}, \mathbf{y} \in R' \\ cube^{-1}(\mathbf{x}), cube^{-1}(\mathbf{y}) \in R \\ cube^{-1}(\mathbf{x}) \vee cube^{-1}(\mathbf{y}) \in R \\ cube(cube^{-1}(\mathbf{x}) \wedge cube^{-1}(\mathbf{y})) \in R' \\ \mathbf{x} \wedge \mathbf{y} \in R' \end{aligned}$$

Similarly $\mathbf{x} \vee \mathbf{y} \in R'$. Γ' is monotone and by Theorem 4.4.1, we have #CSP(Γ')$\leq_{AP}$ #BIS. Consequently, #CSP(Γ) $\leq_{AP}$ #BIS. □

6.2 Bipartite Monotone Graphs

In this part, we extend the results from the previous section to bipartite graphs and define monotone bipartite graphs.

Theorem 6.2.1 (Bipartite Orientation)**.** *Let H be a bipartite graph and let A and B be an arbitrary bipartitions of H. For H' the directed graph obtained from H by orienting the edges from A to B, we have #Hom(H) $\leq_{AP}$#Hom(H').*

Proof. Let G be an input graph for the problem #Hom(H). If G is not bipartite then $\hom(G, H) = 0$. Restrict G to bipartite graphs also restrict G to connected graphs by Lemma 5.1.1. Let X and Y be bipartition of G and let G_1 and G_2 be the graphs obtained from G by orienting all edges from X to Y and from Y to X, receptively. We claim that $\hom(G, H) = \hom(G_1, H') + \hom(G_2, H')$. Every homomorphism from G to H has to either map all the vertices in X to A and all the vertices in Y to B or vice-versa. The homomorphisms that map X to A and Y to B are exactly the same as homomorphisms from G_1 to H' and similarly the homomorphisms that map Y to A and X to B are exactly the same as homomorphisms from G_2 to H'. This implies that #Hom(H) $\leq_{AP}$ #Hom(H'). □

Typically bipartite graphs are not monotone; however, with the latter orientation we can extend the definition of monotone graphs to bipartite graphs. we may get monotone graph.

Definition 6.2.2 (Bipartite Monotone Graphs)**.** Let H be a bipartite graph and let A and B be an arbitrary bipartitions of H. Let H' be the directed graph obtained from H by orienting the edges from A to B. The graph H is said to be *monotone* if H' is monotone.

Note that, the choice for bipartition of H is not significant for H' being monotone or not.

Theorem 6.2.3 (Bipartite Monotone Graphs)**.** *For any bipartite monotone graph H, we have #Hom(H) $\leq_{AP}$#*BIS*.*

Proof. For any bipartite monotone graph H, by Theorem 6.2.1 there is a directed monotone graph H' such that #Hom(H) $\leq_{AP}$#Hom(H'). By Theorem 6.1.3, #Hom(H') $\leq_{AP}$#BIS. Thus, #Hom(H) $\leq_{AP}$#BIS. □

6.3 Reflexive Oriented Graphs

An *oriented graph* is a digraph such that if uv and vu are edges then $u = v$. In this section, we show that for any reflexive oriented graph H, we have #BIS$\leq_{AP}$#Hom(H).

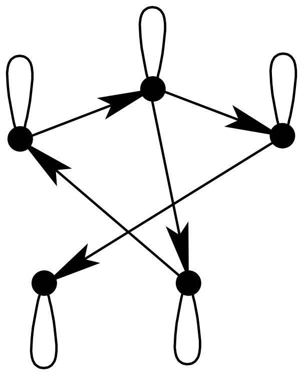


Figure 6.1: A reflexive oriented graph

Let $N_H(a,b)$ be the set of vertices that are both in out-neighbours of a and in-neighbours of b. Note that if H is reflexive and $(a,b) \in E(H)$, both a and b are in $N_H(a,b)$. Let $H_{a,b}$ be the subgraph of H induced by $N_H(a,b)$. *Magnitude* of H represented as $k(H)$ is defined as $\max_{(a,b)\in E(H)}\{|N_H(a,b)|\}$.

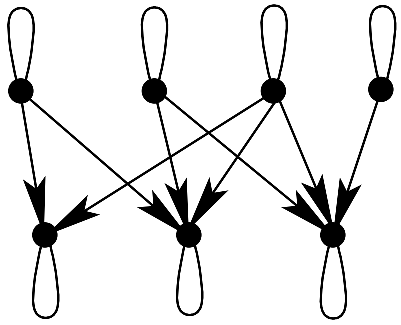


Figure 6.2: A reflexive oriented graph H such that $k(H) = 2$

Lemma 6.3.1. *For any reflexive oriented graph H, if $k(H) = 2$ then #Hom(H) $\geq_{AP}$ #BIS.*

Proof. Let H be a a reflexive oriented graph such that $k(H) = 2$. We reduce the problem

#DownSets to the problem #Hom(H). For a given partial order $P = (V, \preceq)$, let $G = (V, E)$ be a reflexive oriented graph graph in which $uv$is an edge if and only if $a \preceq b$. Take $G' = (V', E')$ where $V' = V \cup \{u, v\}$ and $E' = E \cup \{(u, v)\} \cup \{(u, x), (x, v) \mid x \in V\}$.

For any homomorphism $h : G' \to H$, if $h(u) = h(v) = s$, then $h(x) = s$ for all $x \in V$; otherwise, $(h(u), h(v)) \in E(H)$ and for all $x \in V$, we have either $h(x) = h(u)$ or $h(x) = h(v)$; thus, h corresponds to a downset in $P = (V, \preceq)$.

The number of homomorphisms with $h(u) = h(v)$ is equal to $|V(H)|$ and the number of homomorphisms with $h(u) \neq h(v)$ is equal to $|E(H)|$ times the number of downsets in P, because $k(H) = 2$. Note, we denote by $\hom(G, H)$ the number of homomorphisms from G to H and by #DS(P) the number of downsets in P. Hence,

$$\hom(G', H) = |V(H)| + |E(H)| \cdot \#\mathrm{DS}(P).$$

In other words

$$\#\mathrm{DS}(P) = \frac{\hom(G', H) - |V(H)|}{|E(H)|}.$$

By Lemma 5.2.1 we have #Hom(H) $\geq_{AP}$ #DownSets. □

Definition 6.3.2 (Polar Graph)**.** Let H be a reflexive oriented graph. We say H is *polar* if there are vertices $a, b \in V(H)$, such that $(a, x), (x, b) \in E(H)$ for all vertices $x \in V(H)$ (including a and b).

Note that for a polar graph H, we have $N_H(a, b) = V(H)$ and $k(H) = |V(H)|$.

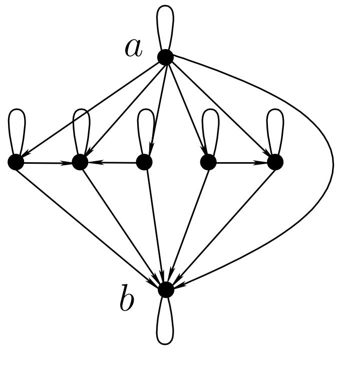


Figure 6.3: A polar graph

Lemma 6.3.3. *For any reflexive oriented graph H that $k(H) > 2$, there is a reflexive oriented graph H' such that each connected component of H' is a polar graph, $k(H) = k(H')$, and #Hom(H) $\geq_{AP}$ #Hom(H').*

Proof. Take the graph H' as union of $H_{a,b}$ for all $(a,b) \in E(H)$, and not that $k(H_{a,b}) = k(H)$. Note also that, each component of H' is $H_{a,b}$ for some a and b; thus, each component of H' is a polar graph. We show that #Hom(H') $\leq_{AP}$#Hom(H).

Let $G = (V, E)$ be the input graph for the problem #Hom(H') and let Y be a set of t fresh vertices (the value of t will be determined later). Take graph $G' = (V', E')$ where $V' = V \cup Y \cup \{u, v\}$ and $E' = E \cup \{(u, v)\} \cup \{(u, x), (x, v) \mid x \in V \cup Y\}$.

Consider any homomorphism $h : G' \to H$. For any $a, b \in V(H)$ that $h(u) = a$ and $h(v) = b$, we have $(a, b) \in E(H)$; also for any vertex x in V or Y, since (u, x) and (x, v) are edges in G', x has to be mapped to $N_H(a, b)$. Note that any vertex in Y can be freely mapped to any vertex in $N_H(a, b)$; however, mapping of the vertices of V should preserve the edges of G. Thus, we have

$$\begin{aligned}\hom(G', H) &= \sum_{(a,b)\in E(H)} |N_H(a,b)|^t \cdot \hom(G, H_{a,b}) \\ &= \sum_{\substack{(a,b)\in E(H)\\ k(H_{a,b})=k(H)}} k^t(H) \cdot \hom(G, H_{a,b}) + \sum_{\substack{(a,b)\in E(H)\\ k(H_{a,b})<k(H)}} k^t(H_{a,b}) \cdot \hom(G, H_{a,b})\end{aligned}$$

Divide both sides of the formula by $k^t(H)$. We have

$$\begin{aligned}\frac{\#Hom(G', H)}{k^t(H)} &= \sum_{\substack{(a,b)\in E(H)\\ k(H_{a,b})=k(H)}} \hom(G, H_{a,b}) + \sum_{\substack{(a,b)\in E(H)\\ k(H_{a,b})<k(H)}} \left(\frac{k(H_{a,b})}{k(H)}\right)^t \cdot \hom(G, H_{a,b}) \\ \frac{\#Hom(G', H)}{k^t(H)} &= \hom(G, H') + \sum_{\substack{(a,b)\in E(H)\\ k(H_{a,b})<k(H)}} \left(\frac{k(H_{a,b})}{k(H)}\right)^t \cdot \hom(G, H_{a,b})\end{aligned}$$

Let $n = |V|$ and let $m = |V(H)|$. We have $\frac{k(H_{a,b})}{k(H)} \leq \frac{m-1}{m}$, $E(H) \leq n^2$, and $\hom(G, H_{a,b}) \leq n^m$. We can simplify the formulas as $(\frac{m-1}{m})^t \approx e^{\frac{-t}{m}}$, $n^m = e^{m \log n}$, and $n^2 = e^{2 \log n}$. For $t > (m^2 + 2m) \log n$, the second part of the summation is less the 1. Thus, by Lemma 5.2.1 #Hom(H) $\geq_{AP}$#Hom(H'). □

Lemma 6.3.4. *Let H be a reflexive oriented graph such that each connected component of H is a polar graph. There is connected component H' of H, such that #Hom(H') $\leq_{AP}$#Hom(H).*

Proof. There are two cases:

First case, all connected components of H are isomorphic. Let H' be a connected component of H and m be the number of connected components of H. For any graph G as input for the problem #Hom(H), $\hom(G, H) = \hom^m(G, H')$. By Lemma 5.2.2, #Hom(H) $\geq_{AP}$#Hom(H').

Second case, there are two connected components of H, H_1 and H_2 that are not isomorphic. By Lovász's Theorem, there is a connected graph Z such that $\hom(Z, H_1) \neq \hom(Z, H_2)$. Let $Z_1, \dots, Z_t$ be t separate copies of Z, where t is a large number to be determined later. Take the graph $G = (V', E')$ where $V' = V \cup V(Z_1) \cup \dots \cup V(Z_t) \cup \{u, v\}$ and $E' = E \cup \{(u, v)\} \cup \{(u, x), (x, v) \mid x \in V \cup Z_1 \cup \dots \cup Z_t\} \cup E(Z_1) \cup \dots \cup E(Z_t)$. Take

$$q = \max_{(a,b)\in E(H)} \hom(Z, H_{a,b})$$

also take

$$H' = \bigcup_{\substack{(a,b)\in E(H)\\ \hom(G,H_{a,b})=q}} H_{a,b}$$

Consider all homomorphisms $h : G' \to H$. For any $a, b \in V(H)$ that $h(u) = a$ and $h(v) = b$, we have $(a, b) \in E(H)$. For any vertex x in V or Z_i, since (u, x) and (x, v) are edges in G', x has to be mapped to $N_H(a, b)$. Thus, we have

$$\begin{aligned}
\hom(G', H) &= \sum_{(a,b)\in E(H)} \hom(G, H_{a,b}) \cdot \hom^t(Z, H_{a,b}) \\
&= \sum_{\substack{(a,b)\in E(H)\\ \hom(Z,H_{a,b})=q}} \hom(G, H_{a,b}) \cdot q^t + \sum_{\substack{(a,b)\in E(H)\\ \hom(Z,H_{a,b})<q}} \hom(G, H_{a,b}) \cdot \hom^t(Z, H_{a,b}) \\
&= \hom(G, H') \cdot q^t + \sum_{\substack{(a,b)\in E(H)\\ \hom(Z,H_{a,b})<q}} \hom(G, H_{a,b}) \cdot \hom^t(Z, H_{a,b}).
\end{aligned}$$

Divide both sides by q^t, and let $n = |V|$ and $m = k(H)$. We have

$$\frac{\hom(G', H)}{q^t} = \hom(G, H') + \sum_{\substack{(a,b)\in E(H)\\ \hom(Z,H_{a,b})<q}} \hom(G, H_{a,b}) \cdot \hom^t(Z, H_{a,b}) \cdot (\frac{1}{q})^t.$$

The second part of right hand side can be bounded as

$$\begin{aligned}\sum_{\substack{(a,b)\in E(H)\\ \text{hom}(Z,H_{a,b})<q}} \text{hom}(G,H_{a,b}) \cdot \text{hom}^t(Z,H_{a,b}) \cdot (\frac{1}{q})^t &\leq n^2 \cdot n^m \cdot \left(\frac{q-1}{q}\right)^t \\ &\approx e^{2\log n} \cdot e^{m\log n} \cdot e^{-\frac{t}{q}}.\end{aligned}$$

For $t > q \cdot (m+2)\log n$, it is less then one; hence, we have $hom(G,H') = \frac{\text{hom}(G',H)}{q^t}$. By Lemma 5.2.1, we have #Hom$(H) \geq_{AP}$ #Hom(H').

H' is a subgraph of H, with $k(H') = k(H)$ and connected components of H' are proper subsets of components of H. After considering the cases multiple times, H' will consist of only a single component. □

Using Lemmas 6.3.4 and 6.3.3 we show that we can reduce the $k(H)$ function without making the problem easier.

Lemma 6.3.5. *Let H be a reflexive oriented graph, if $k(H) > 2$, then there is a reflexive oriented graph H' such that $k(H') < k(H)$ and #Hom(H) $\geq_{AP}$ #Hom(H').*

Proof. By Lemma 6.3.3, there is a reflexive oriented graph H_1 such that every component of H_1 is a polar graph, $k(H_1) = k(H)$, and #Hom$(H_1) \leq_{AP}$#Hom(H).

By Lemma 6.3.4, there is a connected component H_2 of H_1 such that, $k(H_2) = k(H_1)$ and #Hom$(H_2) \leq_{AP}$#Hom(H_1).

Consider the graph H_2 as relation R over the domain D. R is a polar relation, so there is an element $a \in D$ such that for all $x \in D$ we have $R(a,x)$. Let $S = D \setminus \{a\}$. We have $R(x,a)$ implies $x = a$ and for any $y \in S$, we have $R(a,y), R(y,y)$. Thus, by Lemma 5.4.2, #CSP({R,S}) $\leq_{AP}$#CSP(R).

For $H' = H_2 - a$, $k(H') < k(H)$ and

$$\#Hom(H') \leq_{AP} \#CSP(\{R,S\}) \leq_{AP} \#CSP(R) \leq_{AP} \#Hom(H_2) \leq_{AP} \#Hom(H).$$

□

Theorem 6.3.6. *For every non-empty reflexive oriented graph H, #Hom(H) $\geq_{AP}$*#BIS.

Proof. By contradiction, consider a reflexive oriented graph H with minimum $k(H)$ such that the problem #Hom(H) is not AP-reducible to the problem #BIS. By Lemma 6.3.1, $k(H) > 2$. By Lemma 6.3.5, there exists a reflexive oriented graph H' such that #Hom(H') $\leq_{AP}$#Hom(H) and $k(H) < k(H')$. This contradicts with H having the minimum $k(H)$. $\square$

Chapter 7

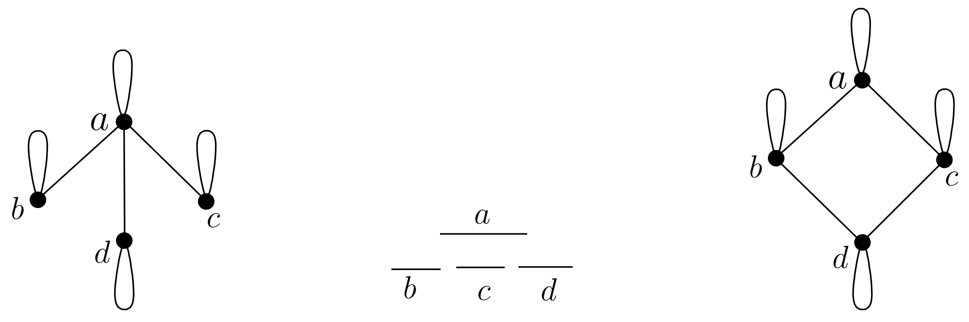
Monotone Reflexive Graphs

Here we discuss a class of reflexive graphs H for which the problem of approximating #Hom(H) is not harder than the problem #BIS. In Chapter 6, we showed that for monotone relations and monotone bipartite graphs the problem #Hom(H) is AP-reducible to the problem #BIS. In this chapter, we will investigate monotone reflexive graphs; although the arguments are analogous for bipartite graphs we will not cover bipartite graphs here. The notation $u \sim v$ indicates there is an edge between vertices u and v; the notation $u \not\sim v$ indicates there is not an edge between vertices u and v.

Interval graphs and *proper interval graphs* are two interesting families of reflexive graphs that play a key role in complexity of list homomorphism and minimum cost homomorphism problems. We are interested in these two families because of their connection with monotone graphs. An interval graph is defined as follows:

Definition 7.0.7. A graph $G = (V, E)$ is said be to an *interval graph* if G can be represented by a family of closed intervals such that each vertex $v \in V$ corresponds to an interval I_v and $u \sim v$ if and only if the intervals I_u and I_v intersect.

The graph G_1 in Figure 7.1(a), called *claw*, is an interval graph and $\mathcal{I}_1$ in Figure 7.1(b) is a possible family of intervals representing G_1. The graph G_2 in Figure 7.1(c) is not an interval graph. In order to show that G_2 is not an interval graph, suppose $\{I_a, I_b, I_c, I_d\}$ is a family of intervals representing G_2. Without loss of generality, suppose I_a is the interval with the smallest starting point. Since $a \not\sim d$, I_d must appear after I_a. Since $b \sim a$, $b \sim d$, $c \sim a$, and $c \sim d$, I_b and I_c must both intersect I_a and I_d; for that, they must both include the interval between I_a and I_d; this implies that $b \sim c$ which is a contradiction.



(a) G_1 (b) I_1 (c) G_2

Figure 7.1: Interval graph G_1, intervals $\mathcal{I}_1$, and non-interval graph G_2

One of the problems that relate homomorphisms to interval graphs is the list homomorphism problem. The list homomorphism problem is defined as follows:

Definition 7.0.8 (L-Hom)**.** Let H be a fixed graph. Given an input graph G, and for each vertex v of G, a *list* $L(v) \subseteq V(H)$, the *list homomorphism* problem for H, denoted as L-Hom(H), is the problem of deciding whether or not there exists a homomorphism $h : G \to H$ such that for each vertex v of G we have $h(v) \in L(v)$.

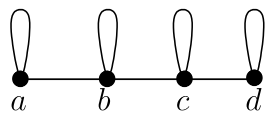
Theorem 7.0.9 shows a connection between the list homomorphism problem and interval graphs.

Theorem 7.0.9 (Feder and Hell 1996 [18])**.** *For any fixed reflexive graph H, the problem L-Hom(H) is polynomial time solvable if H is an interval graph, and otherwise it is NP-complete.*

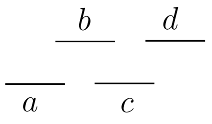
In addition to interval graphs, we shall also focus on proper interval graphs. Proper interval graphs are defined as follows:

Definition 7.0.10. A graph $G = (V, E)$ is said be to a *proper interval graph* if G can be represented by a family of closed intervals such that each vertex $v \in V$ corresponds to an interval I_v, we have $I_u \not\subseteq I_v$ for any two vertices such that $u \neq v$, and $u \sim v$ if and only if the intervals I_u and I_v intersect.

The graph G_3 in Figure 7.2(a) is a proper interval graph and $\mathcal{I}_2$ in Figure 7.2(b) is a possible set of intervals representing G_3. The graph G_1 in Figure 7.1(a) is not a proper interval graph. In order to show that G_1 is not a proper interval graph, suppose $\{I_a, I_b, I_c, I_d\}$



(a) G_3



(b) I_2

Figure 7.2: Proper interval graph G_2 and intervals $\mathcal{I}_2$

is a family of intervals representing G_1. Intervals I_b, I_c, and I_d must intersect I_a and not intersect each other; two of them can intersect I_a in the beginning and the end of I_a, the third interval can not intersect I_a without intersecting the other two; hence, we have a contradiction.

One of the problems that relate homomorphisms to proper interval graphs is the minimum cost homomorphism problem. The minimum cost homomorphism problem is an optimization problem defined as follows:

Definition 7.0.11 (MinHom)**.** Let H be a fixed graph. Given an input graph G and a *cost* function $c : V(G) \times V(H) \to \mathbb{R}^+$ the *min-cost homomorphism* problem, denoted as MinHom(H), is the problem of finding a homomorphism $h : G \to H$ such that $\sum_{v \in G} c(v, h(v))$ is minimized.

The value $c(u, v)$ represents the cost of assigning the vertex u of G to the vertex v of H. Theorem 7.0.12 shows a connection between the min-cost homomorphism problem and proper interval graphs.

Theorem 7.0.12 (Gutin et al. 2008 [23])**.** *For any fixed reflexive graph H, the problem MinHom(H) is polynomial time solvable if H is a proper interval graph, and otherwise it is NP-complete.*

We repeat the definition of *monotone* relations from Chapter 6 for reflexive graphs. Before monotone graphs, we repeat the definition of distributive lattice from Chapter 6.

Definition 7.0.13 (Distributive Lattice)**.** A *distributive lattice* $L = (X, \wedge, \vee)$ is set X with a pair of binary operators $\wedge$ and $\vee$ on X that are:

- idempotent: $x \wedge x = x$ and $x \vee x = x$,
- commutative: $x \wedge y = y \wedge x$ and $x \vee y = y \vee x$,

- associative: $x \wedge (y \wedge z) = (x \wedge y) \wedge z$ and $x \vee (y \vee z) = (x \vee y) \vee z$,
- absorptive: $x \wedge (x \vee y) = x$ and $x \vee (x \wedge y) = x$,
- distributive: $x \wedge (y \vee z) = (x \wedge y) \vee (x \wedge z)$ and $x \vee (y \wedge z) = (x \vee y) \wedge (x \vee z)$

A lattice $L = (X, \wedge, \vee)$ can be viewed as a partial order $P = (X, \preceq)$;

$$a \wedge b \preceq a, b, \preceq a \vee b$$

$\perp$ and $\top$ denote the smallest and largest elements in L, respectively; in other words:

$$x \preceq \perp \Rightarrow x = \perp$$

$$\top \preceq x \Rightarrow x = \top$$

alternately,

$$a \wedge \perp = \perp, a \vee \perp = a$$

$$a \vee \top = a, a \vee \top = \top$$

a *cover* is a pair (a, b) such that $a \neq b$ and:

$$a \preceq x \preceq b \Rightarrow a = x \text{ or } b = x$$

A *cover graph* of a partial order is a directed graph such that there is an edge from a to b if (a, b) is a cover in the partial order.

Definition 7.0.14 (Monotone reflexive graph)**.** Let $G = (V, E)$ be a reflexive graph; G is *monotone* if there exists distributive lattice $L = (V, \wedge, \vee)$ such that when $a \sim b$ and $c \sim d$, we have $(a \wedge c) \sim (b \wedge d)$ and $(a \vee c) \sim (b \vee d)$.

Let $G = (V, E)$ be a graph and let $L = (V, \wedge, \vee)$ be a distributive lattice; for any two edges $e_1 = ab, e_2 = cd \in E$, we denote by $e_1 \wedge e_2$ the vertex pair $(a \wedge c)(b \wedge d)$ and by $e_1 \vee e_2$ the vertex pair $(a \vee c)(b \vee d)$. Note that $e_1 \wedge e_2$ and $e_1 \vee e_2$ are edges of G if G is monotone.

We also provide an alternate characterization for interval and proper interval graphs. These characterizations show a relation between proper interval graphs and monotone relations. A *total order* is a partial order $\preceq$ such that for every a, b either $a \preceq b$ or $b \preceq a$ holds. For every total order, we can define $\wedge$ and $\vee$ operators as:

$$a \preceq b \Rightarrow a \wedge b = a, a \vee b = b$$

A function f that $f(x_1, x_2, \ldots, x_k) \in \{x_1, x_2, \ldots, x_k\}$ is called a *conservative* function. Hence, the conservative operators $\wedge$ and $\vee$ represent a total order.

The following two results are reformulations of known characterizations of interval graphs and proper interval graphs in [23, 25].

Theorem 7.0.15. *A reflexive graph $G = (V, E)$ is an interval graph if and only if there exists a total order $T = (V, \wedge, \vee)$ such that when $a \sim b$ and $c \sim d$, we have $(a \wedge c) \sim (b \wedge d)$.*

Proof. Let $G = (V, E)$ be a graph and let $T = (V, \wedge, \vee)$ be a total order such that when $a \sim b$ and $c \sim d$, we have $(a \wedge c) \sim (b \wedge d)$. We show that G is an interval graph. For $x \in V$, let $o(x) = |\{y | y \preceq x\}|$.

Take $\mathcal{I}$ as follows:

$$I_v = [o(v), \max_{y \sim v}\{o(y)\}]$$

Let $a, b \in V$ such that $a \preceq b$. If $a \sim b$, we have

$$o(a) \leq o(b) \Rightarrow o(b) \in [o(a), o(b)] \subseteq I_a \Rightarrow I_a \cap I_b \neq \emptyset$$

and if $I_a \cap I_b \neq \emptyset$, we have

$$o(a) \leq o(b) \Rightarrow \exists x : a \sim x, o(b) \leq o(x) \Rightarrow b \sim b, a \sim x \Rightarrow (a \wedge b) \sim (b \sim x) \Rightarrow a \sim b.$$

Hence, $\mathcal{I}$ represent G.

Now, let $G = (V, E)$ be a graph and let $\mathcal{I}$ be a family of intervals such that each vertex $v \in V$ corresponds to an interval I_v and $u \sim v$ if and only if the intervals I_u and I_v intersect. We show that there is a total order $T = (V, \wedge, \vee)$ such that when $a \sim b$ and $c \sim d$, we have $(a \wedge c) \sim (b \wedge d)$.

Take $a \preceq b$ if I_a starts *before* I_b or if I_a and I_b start together and I_a ends *after* I_b. Let $a, b, c, d \in V$ such that $a \sim b$, $c \sim d$ holds. Without loss of generality assume $a \preceq b$, $c \preceq d$, and $a \preceq c$. If $b \preceq d$, we have

$$a \wedge c = a, b \wedge d = b, a \sim b \Rightarrow (a \wedge c) \sim (b \wedge d).$$

If $d \preceq b$, then

$$o(a) \leq o(c) \leq o(d), o(d) \leq o(b), I_a \cap I_b \neq \emptyset \Rightarrow a \sim d;$$

and

$$a \wedge c = a, b \wedge d = d, a \sim d \Rightarrow (a \wedge c) \sim (b \wedge d).$$

□

Theorem 7.0.16. *A reflexive graph $G = (V, E)$ is a proper interval graph if and only if there exists a total order $T = (V, \wedge, \vee)$ such that when $a \sim b$ and $c \sim d$, we have $(a \wedge c) \sim (b \wedge d)$ and $(a \vee c) \sim (b \vee d)$.*

Since a total order is a distributive lattice, we have the following fact:

Corollary 7.0.17. *Every reflexive proper interval graph is a monotone graph.*

Theorems 7.0.15 and 7.0.16 indicate similarities between monotone graphs and (proper) interval graphs. However, monotone graphs are not equal to (proper) interval graphs.

Example 7.0.18. The graph G_1 in Figure 7.1(a) is an interval graph; however, we show that G_1 is not a monotone graph. Suppose that G_1 is monotone. If $\top = a$, then

$$a \sim b, c \sim a \Rightarrow (a \wedge c) \sim (b \wedge a) \Rightarrow c \sim b.$$

Analogously, if $\bot = a$, then $b \sim c$. Hence, a can not be $\top$ to $\bot$. Without loss of generality suppose $\top = b$ and $\bot = c$; if $a \preceq d$

$$a \sim b, d \sim a \Rightarrow (a \vee d) \sim (b \vee a) \Rightarrow d \sim b.$$

Analogously, if $d \preceq a$, the $d \sim c$. The last possible case is $a \wedge d = c$ and $a \vee d = b$. In this case

$$a \sim d, d \sim d \Rightarrow (a \wedge d) \sim (d \wedge d) \Rightarrow c \sim d.$$

Hence, G_1 is not monotone.

Now consider the graph H_1 in Figure 7.3; H_1 is not a proper interval graph since the vertices a, b, c, and d form a claw; H_1 is not an interval graph either since the vertices e, f, g, and h induce a subgraph isomorphic to the graph G_2 from Figure 7.1(c); however, in Example 7.1.10, we show that H_1 is a monotone graph.

For any (proper) interval graph H, each induced subgraph of H is a (proper) interval graph. However, some monotone graph have induced subgraphs that are not monotone. In Example 7.0.18, we show that H_1 is a monotone graph while G_1 and G_2, induced subgraphs of H_1, are not monotone.

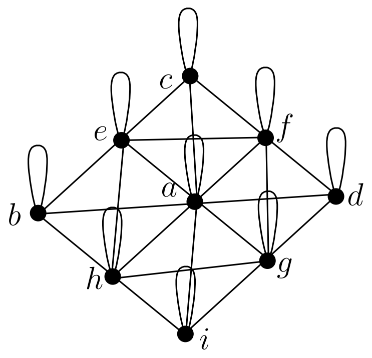


Figure 7.3: Graph H_1

7.1 Generating monotone graphs

In this section, we provide several ways to generate monotone reflexive graphs. We attempt to provide an algorithm to generate all reflexive monotone graphs; however, there exist monotone graphs that can not be generated by the our algorithm, we provide an example of such graphs. This section is based on an unpublished manuscript by P. Hell and M. Siggers (2009). In this work, I have modified the statements of the Lemmas and provided alternating proofs to be more consistent with rest of the thesis. The ideas for the proofs are mostly the same.

By Corollary 7.0.17, every proper interval graph is a monotone graph. Hence, proper interval graphs can be used as a basis to generate monotone graphs. We first show that in order to find out if a graph is monotone, we can investigate individual connected components.

Lemma 7.1.1. *A reflexive graph G is monotone if and only if each connected component of G is monotone.*

Proof. Let G be a graph with connected components $G_1, G_2, \ldots, G_k$; if each G_i is a monotone graph with lattice L_i, then G is monotone with any lattice L based on the union of the L_is, with an arbitrary total ordering between the L_is. For any two edges e_1 and e_2 from the same component, by hypothesis $e_1 \wedge e_2$ and $e_1 \vee e_2$ are edges of the same component. For any two edge e_1 and e_2 from different components, $e_1 \wedge e_2$ and $e_1 \vee e_2$ are both loops and since G is reflexive, they are edges of G.

Let $G = (V, E)$ be a monotone graph with lattice $L = (V, \wedge, \vee)$ and connected components $G_1, G_2, \ldots, G_k$; we show that each G_i is a monotone graph. Before that, we show that each G_i is closed under L. In other words, we have

$$x, y \in G_i \Rightarrow x \wedge y, x \vee y \in G_i.$$

Let $x, y \in G_i$; since x, y are in the same connected component of G, there is a path $P = v_0 = x, v_1, v_2, \ldots, v_k = y$ from x to y in G. Let $v'_0 = v_0$ and $v'_j = v'_{j-1} \wedge v_j$.

$$v'_0 \sim v'_0, v_0 \sim v_1 \Rightarrow (v'_0 \wedge v_0) \sim (v'_0 \wedge v_1) \Rightarrow v'_0 \sim v'_1;$$

by induction we have

$$v'_j \sim v'_{j-1}, v_{j+1} \sim v_j \Rightarrow (v'_j \wedge v_{j+1}) \sim (v'_{j-1} \wedge v_j) \Rightarrow v'_{j+1} \sim v'_j.$$

This shows that there is a vertex $z' = v'_{k+1}$ in G_i such that $x, y \preceq z'$. Let $z = a \wedge b$; if $z' \prec z$, then there is some j such that $v'_j \prec z \prec v'_{j+1}$ and

$$v'_j \sim v'_{j+1}, z \sim z \Rightarrow (v'_j \wedge z) \sim (v'_{j+1} \wedge z) \Rightarrow z \sim v'_{j+1} \Rightarrow z \in G_i.$$

Hence, G_i is closed under the operator $\wedge$; similarly, G_i is also closed under the operator $\vee$. For any two edges $e_1, e_2 \in G_i$, we have $e_1 \wedge e_2 \in G_i$ and $e_1 \vee e_2 \in G_i$; hence, G_i is a monotone graph. □

Lemma 7.1.1 indicates that any monotone graph can be generated from its connected components. From here, all the monotone graphs are assumed to be connected and reflexive, unless explicitly specified.

Lemma 7.1.2. *Let G be a monotone graph with lattice L. For every vertex v, there is descending path from v to $\perp$.*

Proof. Since G is connected, there a path $u_0 = u, u_1, u_2, \ldots, u_k = \perp$ from u to $\perp$ in G. Let $u'_0 = u_0$ and $u'_i = u_i \wedge u'_{i-1}$. Since G is reflexive, $u \sim u \Rightarrow u'_0 \sim u'_0$. Moreover,

$$u'_0 \sim u'_0, u_0 \sim u_1 \Rightarrow u'_0 \sim u'_1$$

and by induction we have

$$u'_{i-1} \sim u'_i, u_i \sim u_{i+1} \Rightarrow u'_i \sim u'_{i+1}.$$

This implies that $W = u'_0 = u, u'_1, \ldots, u'_k = \perp$ is a descending walk from u to $\perp$. The walk W may contain loops (but no other cycles because W is descending); by removing the loops, we get a descending path from u to $\perp$. □

Corollary 7.1.3. *Let $G = (V, E)$ be a monotone graph with lattice $L = (V, \preceq)$; For every element $u \neq \bot$, there is an element $x \preceq u$ such that $u \sim x$.*

Lemma 7.1.4. *Let G be a monotone graph with lattice L, then every cover of L is an edge of G.*

Proof. Let a, b be a cover in L such that $a \preceq b$.

$$a \neq b, a \preceq b \Rightarrow b \neq \bot \Rightarrow \exists x \preceq b : b \sim x$$

If $x = a$, the proof is complete; otherwise, since (a, b) is a cover, we have $x \preceq a$.

$$b \sim x, a \sim a \Rightarrow (b \vee a) \sim (x \vee a) \Rightarrow b \sim a$$

□

Corollary 7.1.5. *Let $G = (V, E)$ be a monotone graph with lattice $L = (V, \preceq)$. For every pair of vertices u, v such that $u \preceq v$, there is descending path from v to u in G.*

Lemma 7.1.6. *Let G be a monotone graph and let a, b, c, and d be vertices of G such that $b \wedge c = d$ and $b \vee c = a$; if $b \sim d, c \sim d$ or $a \sim b, a \sim c$, all edges between a, b, c, d are present in G.*

Proof. If $b \sim d, c \sim d$ we have

$$b \sim d, c \sim d \Rightarrow (b \vee c) \sim (d \vee d) \Rightarrow a \sim d$$
$$b \sim d, d \sim c \Rightarrow (b \vee d) \sim (d \vee c) \Rightarrow b \sim c$$
$$b \sim c, c \sim d \Rightarrow (b \vee c) \sim (c \vee d) \Rightarrow a \sim c$$
$$b \sim c, d \sim b \Rightarrow (b \vee d) \sim (c \vee b) \Rightarrow b \sim a.$$

Analogously, if $a \sim b, a \sim c$, all the edges above must be present in G. □

Next, we show that monotone graphs are closed under Cartesian product. The Cartesian product on graphs is defined as follows:

Definition 7.1.7 (Cartesian product of graphs)**.** For graphs $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$,the graph $G = G_1 \times G_2$ is the *Cartesian product* of G_1 and G_2 if $G = (V, E)$ such that $V = V_1 \times V_2$ and $(u_1, u_2) \sim (v_1, v_2)$ in E if and only if $u_1 \sim v_1$ in E_1 and $u_2 \sim v_2$ in E_2.

We need the Cartesian product of lattices, as well; since lattices are partial orders we simply define the Cartesian product for partial orders.

Definition 7.1.8 (Cartesian product of partial orders)**.** For partial orders $P_1 = (X_1, \preceq)$ and $P_2 = (X_2, \preceq)$, the partial order $P = P_1 \times P_2$ is the *Cartesian product* of P_1 and P_2 if $P = (X, \preceq)$ such that $X = X_1 \times X_2$ and $(x_1, x_2) \preceq (y_1, y_2)$ if and only if $x_1 \preceq y_1$ and $x_2 \preceq y_2$.

Remark. *Let $L_1 = (X_1, \wedge, \vee)$ and $L_2 = (X_2, \wedge, \vee)$ be two lattices. For $x_1, y_1 \in X_1$ and $x_2, y_2 \in X_2$, we have*

$$(x_1, x_2) \wedge (y_1, y_2) = (x_1 \wedge y_1, x_2 \wedge y_2)$$

and

$$(x_1, x_2) \vee (y_1, y_2) = (x_1 \vee y_1, x_2 \vee y_2).$$

Lemma 7.1.9 will show that we can generate monotone graphs from products of other monotone graphs.

Lemma 7.1.9. *For any monotone graphs G_1 and G_2, the product $G_1 \times G_2$ is a monotone graph.*

Proof. For any two edges $e = (u_1, u_2)(v_1, v_2)$ and $e' = (u_1', u_2')(v_1', v_2')$ of $G_1 \times G_2$, the edges u_1v_1 and $u_1'v_1'$ are edges of G_1 and edges u_2v_2 and $u_2'v_2'$ are edges of G_2. Since G_1 and G_2 are monotone,

$$e_1 = u_1v_1 \wedge u_1'v_1'$$

and

$$e_2 = u_2v_2 \wedge u_2'v_2'$$

are edges of G_1 and G_2, respectively. Then

$$\begin{aligned} e \wedge e' &= ((u_1, u_2) \wedge (u_1', u_2'))((v_1, v_2) \wedge (v_1', v_2')) \\ &= (u_1 \wedge u_1', u_2 \wedge u_2')(v_1 \wedge v_1', v_2 \wedge v_2') \\ &= (u_1 \wedge u_1')(v_1 \wedge v_1') \times (u_2 \wedge u_2')(v_2 \wedge v_2') \\ &= e_1 \times e_2 \\ &\in G_1 \times G_2. \end{aligned}$$

Analogously, $e \vee e'$ is also an edge of $G_1 \times G_2$. Hence, $G_1 \times G_2$ is a monotone graph. □

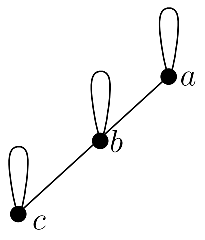


Figure 7.4: Graph H_2

Example 7.1.10. The graph H_2 in Figure 7.4 is a proper interval graph because it is closed under the operators $\wedge$ and $\vee$ defined by the order $a \preceq b \preceq c$. By Corollary 7.0.17, every proper interval graph is monotone; hence, H_2 is a monotone graph. The graph H_1 in Figure 7.3 is $H_2 \times H_2$; thus, H_1 is a monotone graph.

Another operation that preserves monotonicity is retraction. Retraction is defined as follow.

Definition 7.1.11 (Retraction)**.** Let H be a subgraph of G; a *retraction* of G to H is a homomorphism $r : G \to H$ such that for every $v \in V(H)$ we have $r(v) = v$.

When there is a *retraction* of H to G, equivalently we say H is a *retract* of G, and G *retracts* to H. Note that, a retraction of G is an induced subgraph of G. Now, we show that retractions preserve monotone relations.

Lemma 7.1.12. *Let G be a monotone graph; for any retraction r over G, the graph $r(G)$ is monotone.*

Proof. Let $G = (V, E)$ be a monotone graph with lattice $L = (V, \wedge, \vee)$ and let r be a retraction from G to $H = (U, E')$. We claim H is monotone with lattice $M = (U, \wedge_M, \vee_M)$ where $\wedge_M$ and $\vee_M$ are defined as:

$$\begin{aligned} u \wedge_M v &= r(u \wedge v) \\ u \vee_M v &= r(u \vee v) \end{aligned}$$

For any two edges $u_1 \sim v_1$ and $u_2 \sim v_2$ in H, we have $u_1 \wedge_M u_2 = r(u_1 \wedge u_2)$ and $v_1 \wedge_M v_2 = r(v_1 \wedge v_2)$; since G is monotone, $(u_1 \wedge u_2) \sim (v_1 \wedge v_2)$ in G; since r is a retraction, $r(u_1 \wedge u_2) \sim r(v_1 \wedge v_2)$ in H. $\square$

So far we presented several ways in which we can generate monotone graphs. Algorithm 2 combines them to generate more monotone graphs.

Algorithm 2 Generating monotone relations

```
1. Take $m$ proper interval graphs $G_1, G_2, \ldots, G_m$.
2. Let $G' = G_1 \times G_2 \times \cdots G_m$.
3. Use a retraction $r : G' \to G^r$ to generate $G^r$
4. Output $G^r$.
```

By the Lemmas 7.1.9 and 7.1.17, Algorithm 2 generates a monotone graph.

Corollary 7.1.13. *Every retract of a product of proper interval graphs is a monotone graph.*

Corollary 7.1.14. *If H is a retract of a product of proper interval graphs then $\#Hom(H) \leq_{AP} \#\mathrm{BIS}$.*

Next, we try to find out if Algorithm 2 generates all monotone graphs.

We define more parameters for partial orders. The *length* $l(P)$ of a partial order P is the length of a maximum chain in the partial order. The *width* $w(P)$ of a partial order P is the size of a maximum antichain the partial order. Note that a proper interval graph is a monotone graph with a lattice of width 1.

We will use the Theorem of Duffus and Rival to decompose a distributive lattice into a product of chains.

Theorem 7.1.15 (Duffus and Rival 1983 [11])**.** *Let L be a sub-lattice of a distributive lattice L' with $l(L) = l(L')$. Then the cover graph of L is a retract of the cover graph of L'.*

Corollary 7.1.16. *Every lattice of width m is a result of a retraction of a product of m chains.*

Lemma 7.1.17. *Let G be a monotone graph with lattice L. For a retraction r of the cover graph of L, we have that $r(G)$ is a monotone graph.*

Proof. Let $G = (V, E)$ be a monotone graph with lattice $L = (V, \wedge, \vee)$ and let r be a retraction from L to $M = (U, \wedge, \vee)$. Take H as the subgraph of G induced by U. We claim that H is a monotone graph with lattice M. For every two vertices $x, y \in U$, $x \wedge y, x \vee y \in U$,

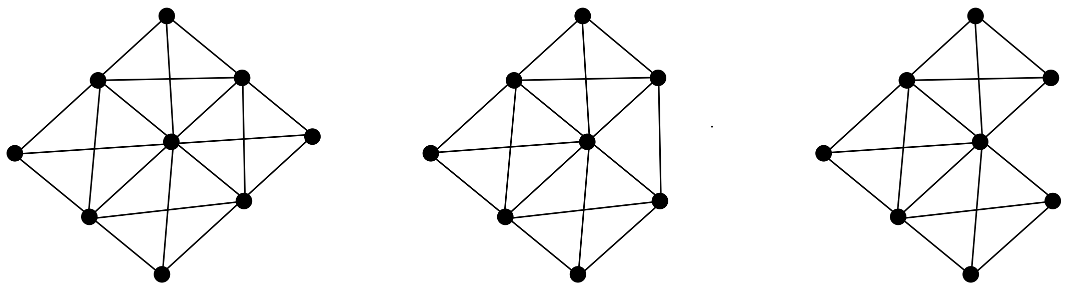

(a) G' (b) G^r (c) G

Figure 7.5: Graphs G', G^r, and G

because M is a lattice. For every two edges e_1 and e_2 of H, $e_1 \wedge e_2, e_1 \vee e_2 \in G$, because G is monotone and $e_1 \wedge e_2, e_1 \vee e_2 \in H$, because H is an induced subgraph of G. □

Now, we want generate a given monotone graph G with lattice L, using Algorithm 2. Let m be the width of the L. Take m chains $C_1, C_2, \ldots, C_m$ of the $l(L)$. Let G_i be a proper interval graph with total order C_i. Let L' be the product of the C_is and let G' be the product of G_is.

By Theorem 7.1.15, there is a retraction r from the cover graph of L' to the cover graph of L. Apply the retraction r over G' and generate $G^r = r(G')$. It would be very nice if G^r was isomorphic with G. Unfortunately, G is a subset of G^r.

For some graphs G^r, removing some of the edges from G^r results a connected monotone graph with the same lattice as G^r. For example, consider the graph G' in Figure 7.5(a). By applying a retraction, we get the graph G^r in Figure 7.5(b). The graph G in Figure 7.5(c) is also a monotone graph with the same lattice as G^r. Thus, Algorithm 2 can not generate G.

The only remaining challenge is given G^r and L, finding the set of edges E_1 of G^r such that $G^r - E$ is connected and monotone with lattice L.

Chapter 8

Conclusion and Future Work

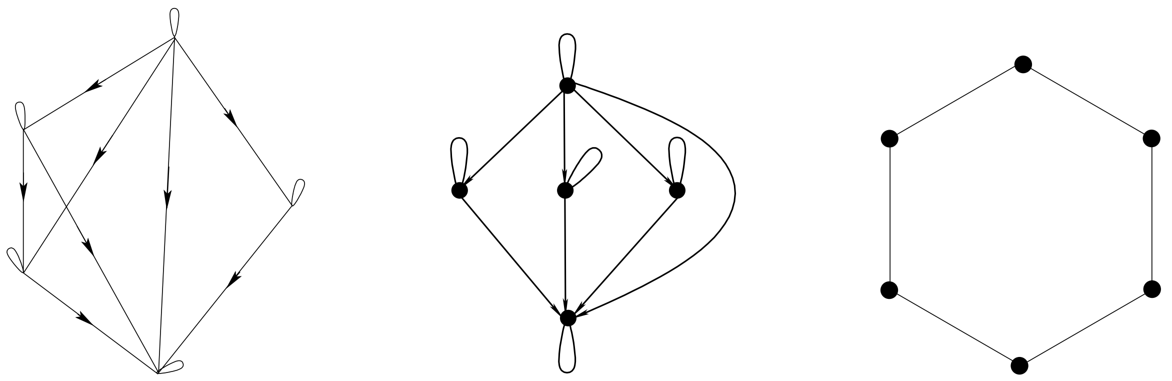
8.1 Conclusion

We formally defined the problem #CSP(Γ), mentioned some of its applications; mentioned the major results on complexity of the problem #CSP(Γ). We defined the class FPRAS which is considered the efficient computational model for finding approximate solutions for counting problems. We mentioned FPRAS algorithms for several problems such as simple #CSP(Γ), #DNF-SAT,#MATCH, and #LOWDEGREE-k-COLORING. The problem #DNF-SAT is solved by sampling and the problems #MATCH and #LOWDEGREE-k-COLORING are solved by Markov chain Monte-Carlo method.

We defined AP-reductions which are used to classify approximation counting problems. We introduced the problem #BIS and the class of the problems AP-interreducible with the problem #BIS. We mentioned several well-known examples of this class such as the problems #DOWNSETS and #1P1N-SAT. Computation of the partition function for the Ising model, where the model is ferromagnetic, also belongs to this class.

We also mentioned that some of the problems are hard to approximate. There are no polynomial time approximation algorithm for any of the problems in this class, unless $RP = NP$. Note that although according to current knowledge these problems are not efficiently solvable, approximating these problems is still easier than exactly solving them. Some of the well known problems in this class are the problems #2-SAT, #3-SAT, #SAT, #IS, and #3-COLORING. Computation of the partition function for the Ising model, where the model is anti-ferromagnetic, also belongs to this class.

We showed that by limiting the input for the problems #CSP(Γ) to connected structures,



(a) N_5 (b) M_5 (c) C_6

Figure 8.1: Example graphs for which complexity of the #Hom(H) is unknown

the problems do not become easier. We showed that if there is linear transformation between the number of solutions for two counting problems and both problems always have solutions, then those two problems are AP-interreducible. We generalized the pinning theorem from [15].

We introduced the maximization technique for AP-reductions. The sets of relations closed under maximization form sets of relations similar to relational clones.

We extended the notion of *monotone* relations from Boolean domains to general domains. We proved that for any monotone constraint language Γ, the problem #CSP(Γ) is AP-reducible to the problem #BIS. We also extended the notion of monotone relations to bipartite graphs and proved that for any monotone bipartite graph H, the problem #Hom(H) is also AP-reducible to to the problem #BIS.

We proved that for any reflexive oriented graph H, the problem #BIS is AP-reducible to the problem #Hom(H). Despite these results, finding a necessary and sufficient condition for Γ such that the problem #CSP(Γ) is AP-interreducible with the problem #BIS remains open.

We investigated monotone reflexive graphs. We proved that a retract of a product of proper interval graphs is a monotone graph.

We expect that there exists a trichotomy for the problem #CSP(Γ) over 3-element sets similar to the trichotomy for the problem #CSP(Γ) over 2-element sets. However, the common belief is that there is no such a trichotomy in general for the problem #CSP(Γ). For example, consider the graphs N_5, M_5, and C_6 shown in Figure 8.1. The problems

#Hom(N_5), #Hom(M_5), and #Hom(C_6) are harder than the problem #BIS; however, they are not expected to be as hard as the problem #SAT.

8.2 Future Work

We studied the complexity of approximately solving the problem #CSP(Γ) for maximal partial clones on 3-element sets [5], we also studied complexity of approximately solving the problem #Hom(H) for graphs with 3 vertices. However, in both categories, there are several problems whose complexity of approximately solving them is still unknown.

Jerrum et al. [30] proved that with a SAT oracle, there is FPRAS for any problem in #P. With the recent growth of practical usage of SAT-solvers, this theorem potentially provides a practical approach to approximately solve the problem #CSP(Γ).

The Markov chain Monte-Carlo method is used approximately solve several counting problems. This method may approximately solve the problem #CSP(Γ) for some Γ.

We defined the monotone constraint languages as the constraint languages that are closed under the meet and join operators of a distributive lattice. N_5 and M_5, shown in Figure 8.1, appear on non-distributive lattices. Proving that the problems #Hom(N_5) and #Hom(M_5) are AP-reducible to the problem #BIS may extend our result to constraint languages closed under meet and join operators of a general lattice.

We proved that for any reflexive oriented graph H, the problem #BIS is AP-reducible to the problem #Hom(H). We can also prove that for some oriented graph with possible loop, the problem #BIS is AP-reducible to the problem #Hom(H). We believe for a more general family of digraphs with possible loops, the problem #BIS is AP-reducible to the problem #Hom(H).

Bibliography

[1] V. G. Bodnarchuk, L. A. Kaluzhnin, V. N. Kotov, and B. A. Romov. Galois theory for Post algebras. I. *Cybernetics and Systems Analysis*, 5:243–252, 1969. 10.1007/BF01070906.

[2] Andrei A. Bulatov. The complexity of the counting constraint satisfaction problem. In *ICALP (1)*, pages 646–661, 2008.

[3] Andrei A. Bulatov and Víctor Dalmau. Towards a dichotomy theorem for the counting constraint satisfaction problem. *Inf. Comput.*, 205(5):651–678, 2007.

[4] Andrei A. Bulatov and Amir Hedayaty. Counting predicates, subset surjective functions, and counting csps. In *ISMVL*, pages 331–336, 2012.

[5] Andrei A. Bulatov and Amir Hedayaty. Counting problems and clones of functions. *Multiple-Valued Logic and Soft Computing*, 18(2):117–138, 2012.

[6] Andrei A. Bulatov, Peter Jeavons, and Andrei Krokhin. Classifying the complexity of constraints using finite algebras. *SIAM journal on computing.*, 34(3):720–742, April 2005.

[7] Andrei A. Bulatov and Matthew Valeriote. Recent results on the algebraic approach to the CSP. In *Complexity of Constraints*, pages 68–92, 2008.

[8] Stephen A. Cook. The complexity of theorem proving procedures. In *Proceedings of the Third Annual ACM Symposium*, pages 151–158, New York, 1971. ACM.

[9] Nadia Creignou and Miki Hermann. Complexity of generalized satisfiability counting problems. *Inf. Comput.*, 125(1):1–12, 1996.

[10] Nadia Creignou, Sanjeev Khanna, and Madhu Sudan. *Complexity Classifications of Boolean Constraint Satisfaction Problems.* SIAM, Philadephia, PA, 2001.

[11] Dwight Duffus and Ivan Rival. Graphs orientable as distributive lattices. *Proceedings of the American Mathematical Society*, 88(2):pp. 197–200, 1983.

[12] Martin E. Dyer, Leslie A. Goldberg, Catherine S. Greenhill, and Mark Jerrum. On the relative complexity of approximate counting problems. In Klaus Jansen and Samir

Khuller, editors, *Approximation Algorithms for Combinatorial Optimization*, volume 1913 of *LNCS*, pages 557–570. Springer Berlin / Heidelberg, 2000. 10.1007/3-540-44436-X_12.

[13] Martin E. Dyer, Leslie A. Goldberg, Catherine S. Greenhill, and Mark Jerrum. The relative complexity of approximate counting problems. *Algorithmica*, 38(3):471–500, 2003.

[14] Martin E. Dyer, Leslie A. Goldberg, and Mark Jerrum. Counting and sampling H-colourings. *Inf. Comput.*, 189:1–16, February 2004.

[15] Martin E. Dyer, Leslie A. Goldberg, and Mark Jerrum. An approximation trichotomy for Boolean #CSP. *J. Comput. Syst. Sci.*, 76:267–277, May 2010.

[16] Martin E. Dyer and Catherine S. Greenhill. The complexity of counting graph homomorphisms. In *Proceedings of the eleventh annual ACM-SIAM symposium on Discrete algorithms*, SODA '00, pages 246–255, Philadelphia, PA, USA, 2000. Society for Industrial and Applied Mathematics.

[17] Martin E. Dyer and David Richerby. On the complexity of #CSP. In *STOC*, pages 725–734, 2010.

[18] Tomás Feder and Pavol Hell. List homomorphisms to reflexive graphs. *J. Comb. Theory, Ser. B*, 72(2):236–250, 1998.

[19] Tomás Feder and Moshe Y. Vardi. The computational structure of monotone monadic snp and constraint satisfaction: A study through datalog and group theory. *SIAM J. Comput.*, 28(1):57–104, 1998.

[20] Isidore Fleischer and Ivo G. Rosenberg. The Galois connection between partial functions and relations. *Pacific J. Math*, 79(1):93–97, 1978.

[21] David Geiger. Closed Systems of Functions and Predicates. *Pacific Journal of Mathematics*, 27:95–100, 1968.

[22] Leslie A. Goldberg and Mark Jerrum. The complexity of ferromagnetic Ising with local fields. *Combinatorics, Probability and Computing*, 16(01):43–61, August 2007.

[23] Gregory Gutin, Pavol Hell, Arash Rafiey, and Anders Yeo. A dichotomy for minimum cost graph homomorphisms. *Eur. J. Comb.*, 29(4):900–911, 2008.

[24] Pavol Hell and Jaroslav Nešetřil. On the complexity of H-coloring. *J. Comb. Theory, Ser. B*, 48(1):92–110, 1990.

[25] Pavol Hell and Jaroslav Nešetřil. *Graphs and homomorphisms.* Oxford lecture series in mathematics and its applications. Oxford University Press, 2004.

[26] Peter Jeavons, David A. Cohen, and Marc Gyssens. Closure properties of constraints. *J. ACM*, 44(4):527–548, 1997.

[27] Mark Jerrum. A very simple algorithm for estimating the number of k-colorings of a low-degree graph. *Random Struct. Algorithms*, 7:157–165, September 1995.

[28] Mark Jerrum and Alistair Sinclair. Polynomial-time approximation algorithms for the Ising model. *SIAM Journal on Computing*, 22:1087–1116, 1993.

[29] Mark Jerrum and Alistair Sinclair. The Markov chain Monte Carlo method: an approach to approximate counting and integration. In *Approximation Algorithms for NP-hard Problems*, pages 482–520. PWS Publishing, 1996.

[30] Mark Jerrum, Leslie G. Valiant, and Vijay V. Vazirani. Random generation of combinatorial structures from a uniform distribution. *Theor. Comput. Sci.*, 43:169–188, 1986.

[31] Dietlinde Lau. *Function Algebras on Finite Sets: Basic Course on Many-Valued Logic and Clone Theory (Springer Monographs in Mathematics)*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2006.

[32] László Lovász. Operations with structures. *Acta Mathematica Hungarica*, 18(3-4):321–328, 1967.

[33] Michael Luby and Boban Veličkovic. On deterministic approximation of dnf. In *Proceedings of the twenty-third annual ACM symposium on Theory of computing*, STOC '91, pages 430–438, New York, NY, USA, 1991. ACM.

[34] Emil L. Post. *The two-valued iterative systems of mathematical logic*, volume 5 of *Annals Mathematical Studies*. Princeton University Press, 1941.

[35] Neal Madras Richard M. Karp, Michael Luby. Monte-Carlo approximation algorithms for enumeration problems. *Journal of Algorithms*, 10(3):429–448, 1989.

[36] Matthew Valeriote. A subalgebra intersection property for congruence distributive varieties. *Canad. J. Math.*, 61:451–464, 2009.

[37] Leslie G. Valiant. The Complexity of Enumeration and Reliability Problems. *SIAM Journal on Computing*, 8(3):410–421, 1979.