Harnessing tractability in constraint satisfaction problems

Clément Carbonnel

THÈSE

En vue de l'obtention du

DOCTORAT DE L'UNIVERSITÉ DE TOULOUSE

Délivré par :

Institut National Polytechnique de Toulouse (INP Toulouse)

Discipline ou spécialité :

Intelligence Artificielle

Présentée et soutenue par :

M. CLÉMENT CARBONNEL

le mercredi 7 décembre 2016

Titre :

Harnessing tractability in constraint satisfaction problems

Ecole doctorale :

Mathématiques, Informatique, Télécommunications de Toulouse (MITT)

Unité de recherche :

Institut de Recherche en Informatique de Toulouse (I.R.I.T.)

Directeur(s) de Thèse :

M. MARTIN COOPER

M. EMMANUEL HEBRARD

Rapporteurs :

M. STANISLAV ZIVNY, UNIVERSITY OF OXFORD

M. STEFAN SZEIDER, WIEN UNIVERSITAT

Membre(s) du jury :

Mme NADIA CREIGNOU, AIX-MARSEILLE UNIVERSITE, Président

M. EMMANUEL HEBRARD, LAAS TOULOUSE, Membre

M. MARTIN COOPER, UNIVERSITE PAUL SABATIER, Membre

M. PHILIPPE JEGOU, AIX-MARSEILLE UNIVERSITE, Membre

First and foremost, I would like to thank my supervisor Emmanuel Hebrard. Emmanuel was always there to guide me, answer my questions, comment on my research and support me in the difficult moments; he really took his supervision role to heart and worked hard to ensure I do my Ph.D. in the best possible conditions. Through countless discussions, he has shaped my scientific mind and prepared me for the world of academic research. It has been a pleasure and an honour working with him.

I would also like to thank my co-supervisor Martin C. Cooper for his invaluable scientific advice and the opportunities he gave me. His numerous and detailed comments on my ideas and manuscripts have been of great help.

I thank the members of the ROC team at LAAS-CNRS for these three years spent together. It was a great experience, both from the scientific and personal point of view. I am particularly indebted to Christian Artigues, who battled to obtain funding and make this thesis happen.

I thank every person who took the time to read my manuscript, including my referees Stanislav Živný and Stefan Szeider as well as the other members of my committee, Nadia Creignou and Philippe Jégou.

I thank those who have played a role (sometimes indirectly) in shaping the path that ultimately led me to write this dissertation - that includes among others Gilles Trombettoni, Philippe Vismara and many of my teachers and friends from Lycée Champollion, Ensimag and Université Joseph Fourier.

Finally, I thank my family for their continued and unwavering support.

Contents

Chapter 1

Introduction

A considerable part of the current research in theoretical computer science finds its roots in Cobham's thesis that problems solvable in practice are those with polynomial-time algorithms [40]. This has certainly proved to be a powerful insight: polynomial-time problems are usually easier to solve than NP-hard ones, and the study of complexity classes closed under polynomial-time reductions has raised deep questions about the structure of combinatorial problems. When the limitations of Cobham's thesis are discussed, the same argument is often heard: there are impractical polynomial-time algorithms. Recently, the spectacular performance improvement of SAT solvers, which sometimes handle instances with millions of variables and prove nontrivial theorems on their own [87], suggests that there may exist reasonably practical exponential-time algorithms as well.

Ironically, the scientific community that studies the Constraint Satisfaction Problem (CSP) can be partitioned into two groups that are in perfect contradiction with Cobham's thesis. The design of exponential-time algorithms is left into the hands of researchers whose main goal is practical solving, while in the meantime complexity theorists have built an impressive catalog of subproblems with sophisticated polynomial-time algorithms and zero applications.

The present work is an attempt to bring this literature on tractable subproblems closer to being useful. The apparent lack of ambition of this statement is better understood with a measure of the obstacles that must be overcome. First, many of these tractable classes are defined by elusive algebraic properties and more often than not the complexity of testing for them is unknown. Second, hoping that practical CSP instances would fall readily into these highly structured classes on a regular basis is probably unrealistic, so mechanisms must be designed in order to make the framework more adaptable. Third, some of those polynomial-time algorithms were not designed with performance in mind and have critical flaws, such as an exponential dependency on the domain size. Fourth and last, the definition of CSP used in most theoretical works does not leave room for one of the most

widely used feature of constraint solvers, global constraints.

There has been remarkably little research on these issues, but we find they are fertile grounds for both theoretical and practical investigation. In this thesis we tend to stay on the theoretical side, but draw our motivation from practical considerations and even perform preliminary experiments to illustrate our results.

Outline of the Thesis

The necessary technical background is given in Chapter 2. The next three chapters present our contributions:

Chapter 3 addresses the issue of CSP instances that do not quite fit in any well-studied tractable class. We investigate the possibility of computing efficiently a decomposition of a given instance into a reasonable number of subproblems that belong to a fixed "target" tractable class using the framework of *strong backdoors* [135]. We observe that computing optimal decompositions is almost always NP-hard, but a refined analysis through parameterized complexity shows that for certain tractable classes good decompositions can be found efficiently if they exist. We also introduce *partition backdoors*, which are generally easier to compute but may provide suboptimal decompositions. The content of this chapter has been published in [11] and [33], although this manuscript contains slightly different definitions and a minor additional result.

Chapter 4 presents novel and highly non-trivial polynomial-time algorithms for testing membership in tractable classes that have the property of being *conservative*. These classes have the double advantage of being very well understood and working nicely with the partition backdoor approach developed in Chapter 3. Our main result is a proof that the conservative tractability criterion defined by Bulatov [21] is polynomially decidable. This is without a doubt the most significant contribution of this thesis, as it both answers a difficult theoretical question and sows the seeds for a potential efficient algorithm in the future (our algorithm being at the same time polynomial-time and *extremely inefficient*). We get as a byproduct of our methods an improved ("uniform") algorithm for one of the most important conservative tractable classes, CSP over languages admitting conservative edge polymorphisms. This chapter has been published in [31] and [30].

Chapter 5 approaches the subject of harnessing tractability in CSP from a very different angle. It is commonplace for CSP solvers to allow modelling with predicates: for instance, the user can state "at most k distinct values can be used to assign these n variables". Some predicates encapsulate NP-hard problems, and

making meaningful deductions from these constraints is very difficult for the solver. This time, we explore the possibility of using the (parameterized) tractability results *for the encapsulated problems* in order to speed up the resolution of CSP instances. For this, we define formally *loss-less kernelization*, a new variant of kernelization dedicated to the context of constraint reasoning. We use the Vertex Cover problem as a case study to showcase our ideas. The experimental part of this chapter has been published in [34].

Chapter 2

Background

This chapter introduces the fundamentals of constraint satisfaction and parameterized complexity, which form the technical core of the present thesis. We will on purpose present more material than is absolutely necessary in order to help the reader understand the broader scientific context. This background will be complemented by small sections spread between our contributions which will introduce auxiliary concepts, notation and results as they are needed. We will only assume from the reader familiarity with elementary notions of mathematics and computational complexity.

This chapter is organized as follows:

- Section 2.1 defines the Constraint Satisfaction Problem and introduces basic notation.

- Section 2.2 is a survey of tractable classes for CSP based on restrictions of the constraint language. This section is given special emphasis because its content will serve as a basis for our own contributions presented in Chapters 3 and 4.

- Section 2.3 complements this survey by a quick overview of known tractable classes obtained by imposing restrictions on the constraint (hyper)graph. Knowledge of these tractable classes is not required to understand our results or their significance, but we feel that omitting them would give the reader an excessively truncated view of tractability in CSP.

- Finally, Section 2.4 introduces parameterized complexity, which will be an integral part of our arsenal in our attempt to harness the tractable classes presented in Section 2.2.

We note that tractability results that do not fall within the scopes of Sections 2.2 and 2.3 exist. These tractable classes are called *hybrid*. However, they are quite scattered and tend to be relatively small in general. For detailed information on these classes, we direct the reader to the recent surveys on the complexity of CSP [43][32].

2.1 Constraint Satisfaction Problems

The Constraint Satisfaction Problem (CSP) is a common formalism for combinatorial problems that can be expressed as deciding if there exists an assignment to a set of finite-domain variables that satisfies a set of *constraints*. For example, solving a system of equations over a finite field is a problem of this kind; in this case the constraints are equations that must hold. In 3-SAT, the variables are Boolean and constraints are ternary clauses that must be satisfied. CSP generalizes both by only assuming that constraints are *relations* imposed on subsets of variables. The first explicit mention of this problem is found in a 1974 paper by Montanari [114].

Definition 1. A **CSP instance** is a triple $(\mathcal{X}, \mathcal{D}, \mathcal{C})$, where

- $\mathcal{X}$ is a finite set of variables;
- $\mathcal{D}$ is a finite set of values;
- $\mathcal{C}$ is a finite set of constraints, that is, pairs (S_C, R_C) where S_C is a tuple of variables (the *scope* of C) and $R_C \subseteq \mathcal{D}^{|S_C|}$ is a relation over $\mathcal{D}$.

A *solution* to a CSP instance is an assignment $\phi : \mathcal{X} \to \mathcal{D}$ such that $\forall (S_C, R_C) \in \mathcal{C}$, $\boldsymbol{\phi}(S_C) \in R_C$ (where $\boldsymbol{\phi}$ is the operation on tuples of variables obtained by componentwise application of ϕ). The problem is to decide if a solution exists, which is NP-complete [98]. In this definition, variables do not come with individual domains: variable-specific domain restrictions must be enforced using unary constraints. Unless explicitly stated otherwise, we will assume that all relations are given in input as tables of tuples.

Remark 1. This last assumption has very important implications. For instance, it implies that SAT (with unbounded clause arity) is not a subproblem of CSP. More generally, in many areas of mathematics equations are never represented as explicit lists of solutions. Doing so may change the complexity drastically by inflating exponentially the input size. However, for other problems in which the arity of the relations is naturally bounded, such as Graph k-Coloring, this hypothesis is harmless. In the more practice-oriented literature, it is common to

represent a relation R as the set of tuples that is accepted by a certain polynomial-time algorithm δ_R. This gives more latitude to CSP solvers for handling internally high-arity constraints, and makes the modelling phase noticeably easier by allowing the use of predicates. Families of relations represented intentionally are usually called *global constraints*. However, with this framework for relations the problem becomes much less interesting from a complexity-theoretic point of view since every NP problem becomes technically a CSP with a single constraint. These two definitions of CSP coexist peacefully in the literature because the constraint programming community is strongly polarized; the choice of encoding for relations is usually clear from the research topic. Unfortunately, the scope of this thesis is quite wide and will cover both settings. We will try to reduce ambiguity as much as possible.

Example 1. The input of the Betweenness problem is a set U of n elements and a set T of ordered triples of distinct elements of U. The question is: is it possible to find a total ordering $>_U$ of U such that for each triple $(a, b, c) \in T$, either $a >_U b >_U c$ or $c >_U b >_U a$? Betweenness is NP-complete [119], and can be easily encoded in CSP as follows. The domain is $\{1, \ldots, n\}$ and we have one variable x_u for each $u \in U$. Between any two distinct variables x_u, x_v we add the constraint $x_u \neq x_v$. This ensures that solutions will always be bijections between U and $\{1, \ldots, n\}$. Then, for each triple $(a, b, c) \in T$ we add on (x_a, x_b, x_c) a constraint with predicate $R(x_a, x_b, x_c) \iff (x_a > x_b > x_c) \vee (x_c > x_b > x_a)$, where $>$ is the usual ordering on $\mathbb{N}$. Note that these constraints have $O(n^3)$ tuples, so the resulting instance has polynomial size. It is easy to see that an assignment ϕ is a solution to the CSP instance if and only if the ordering $>_U$ such that $a >_U b \iff \phi(x_a) > \phi(x_b)$ is a solution to the instance of Betweenness.

Equivalent formulations. CSP is known under multiple names. In logic, CSP corresponds to special types of first-order formulas that allow only existential quantification and conjunctions of relational predicates. These formulas are called *primitive positive*. In database theory, CSP instances correspond to *conjunctive queries*. CSP can also be formalized as a homomorphism problem as follows. A *signature* is a finite set σ of relation symbols R_i, each with a specified arity k_i. A *relational structure* $\mathbb{A}$ over σ, or σ-structure, is a finite universe A together with one relation $R_i^{\mathbb{A}} \subseteq A^{k_i}$ for each symbol $R_i \in \sigma$. If $I = (\mathcal{X}, \mathcal{D}, \mathcal{C})$ is a CSP instance and $\sigma_I = \{R_1, \ldots, R_m\}$ is the set of all relation symbols that appear in its constraints, the *left-hand side structure* of I is the σ_I-structure $\mathbb{A}^L$ with universe $\mathcal{X}$ and for each i, the tuples of $R_i^{\mathbb{A}^L}$ are the scopes of constraints whose relation is R_i. The *right-hand side structure* of I is the σ_I-structure $\mathbb{A}^R$ with universe $\mathcal{D}$ and for each i, $R_i^{\mathbb{A}^R}$ is the relation R_i. The solutions to I are then exactly the mappings $\phi : \mathcal{X} \to \mathcal{D}$ such that $\phi(R_i^{\mathbb{A}^L}) \subseteq R_i^{\mathbb{A}^R}$ for each $R_i \in \sigma$, or equivalently,

the homomorphisms from $\mathbb{A}^L$ to $\mathbb{A}^R$.

CSP solving. CSP is a popular choice for encoding real-world problems in artificial intelligence and operational research. The reason is twofold: modelling is intuitive (much more so than Integer Linear Programming or SAT), and CSP solvers perform quite well in practice despite the obvious exponential worst-case. For readers completely unfamiliar with practical CSP solving, let us recall the basic skeleton of most CSP solvers.

> *Backtracking search.* The search space is the set of all possible assignments to the variables. The typical solver will explore depth-first a search tree, picking a variable-value pair (x, d) at each node and branching on the two cases $x \leftarrow v$ and $x \neq v$. When the algorithm detects that the succession of decisions leading to a node makes the instance unsatisfiable, it reverses the last decision (it *backtracks*) and continues the exploration from there. The heuristic that chooses the value (x, v) to branch on is critical as even a single poor decision may have devastating consequences. The design of such heuristics is one of the motivations for our results in Chapter 3.
>
> *Propagation.* At each node of the search tree, the solver uses a polynomial-time algorithm to shrink the problem by identifying inconsistent assignments. Those rules are typically variants of local consistency methods (such as enforcing generalized arc-consistency, described in Section 2.2.3, Remark 2). Propagation algorithms are delicate tradeoffs between pruning quality and time complexity; an overzealous algorithm might backfire by spending a considerable amount of time to prune assignments that would be much more easily ruled out after a few additional decisions. The main contribution of Chapter 5 is a novel theoretical framework for the conception of such algorithms.

In modern implementations these components are often complemented with *learning* [52][99], which infers new valid constraints by analyzing past failures in the search tree and has been hugely successful in SAT solvers [115]. Other features commonly sighted include (but are not limited to) symmetry-breaking [75], local search procedures [97] and alternative exploration strategies such as breadth-first search and limited discrepancy search [82].

Notation. In the subsequent chapters we will manipulate CSP instances in many ways, and for this we need to introduce some elementary notations.

> *Tuples and scopes.* Throughout the thesis, tuples will be delimited by parentheses and symbols for tuples of domain values will be written in boldface.

Given a tuple $\mathbf{t}$ and $I \subseteq \{1, \ldots, |\mathbf{t}|\}$, we denote by $\mathbf{t}[I]$ the projection of $\mathbf{t}$ onto I. We will give special treatment to constraint scopes and sometimes treat them as sets of variable occurrences rather than tuples. For instance, if S_C is a scope and we pick $x \in S_C$, the operation $S_C \backslash x$ will remove x from S_C but not every occurrence of the variable pointed by x. If $C = (S_C, R_C)$ is a constraint, $\mathbf{t} \in R_C$ and $x \in S_C$, we will use the shorthand $\mathbf{t}[x]$ for "$\mathbf{t}[i]$ with i such that $S[i] = x$". This will greatly alleviate the notational burden by reducing the use of integer indexes.

Operations on CSP *instances.* Given a constraint $C = (S_C, R_C)$ and a subset of variables $X_1 \subseteq \mathcal{X}$, we denote by $C[X_1]$ the projection of C onto X_1 (which is null if S_C does not contain any variable in X_1). The projection of a CSP instance $I = (\mathcal{X}, \mathcal{D}, \mathcal{C})$ onto $X_1 \subseteq \mathcal{X}$ is the instance $I_{|X_1} = (X_1, \mathcal{D}, \mathcal{C}^*)$ with $\mathcal{C}^* = \{C[X_1] \mid C \in \mathcal{C}\}$. A *partial solution* to I is a solution to some projection of I. Given a subset $X_1 \subseteq \mathcal{X}$ and an assignment $\psi : X_1 \to \mathcal{D}$, we denote by $I[X_1 \leftarrow \psi(X_1)]$ the instance obtained from I by removing from each constraint relation the tuples $\mathbf{t}$ such that $\mathbf{t}[x] \neq \psi(x)$ for at least one $x \in X_1$, and then projecting I onto $\mathcal{X} \backslash X_1$. We will say that $I[X_1 \leftarrow \psi(X_1)]$ is the *residual instance* after application of the *partial assignment* ψ to X_1.

Constraint languages. We will use $\mathcal{R}(.)$ and $\mathcal{S}(.)$ as operators that return respectively the relation and the scope of a constraint. A *constraint language* over a set $\mathcal{D}$ is a set of relations over $\mathcal{D}$, and the language $\mathcal{L}(I)$ of a CSP instance $I = (\mathcal{X}, \mathcal{D}, \mathcal{C})$ is the set $\{\mathcal{R}(C) \mid C \in \mathcal{C}\}$. We will use $D(\Gamma)$ to denote the domain of a constraint languages Γ, that is, the set of all values that appear in at least one tuple. Because constraint languages are sets of relations and relations are sets of tuples, we will use square brackets as delimiters for relations to avoid confusion.

2.2 Language-based Tractable Classes

A natural way to study the fine-grained complexity of CSP is to impose a fixed catalog of relations that are available for modelling instances and see how that affects the complexity. This type of restriction is often referred to as *non-uniform CSP*, and the rich theory that has been built around it will be the background for many of our own contributions.

Definition 2. Let Γ be a constraint language. We denote by CSP(Γ) the restriction of CSP to instances I such that $\mathcal{L}(I) \subseteq \Gamma$.

Well-known problems which are naturally of this form are legion. For instance, Graph k-Coloring is exactly CSP($\{\neq_k\}$) where $\neq_k$ is the disequality over a k-

element domain, and k-XORSAT is CSP($\Gamma_\oplus$) where $\Gamma_\oplus$ contains all XOR-clauses of arity at most k. Because of the sheer diversity of possible constraint languages Γ, one could expect a rich complexity landscape where, depending on Γ, CSP(Γ) can be in P, NP-complete, or in one of the infinitely many intermediate complexity classes whose existence is proven by Ladner's Theorem [106] assuming P $\neq$ NP. The starting point for much of the current research is the observation that non-uniform CSP is, from a logic point of view, the largest subclass of NP that might contain only P and NP-complete problems [62]; in the same paper Feder and Vardi conjectured that it is the case, at least for finite languages.

Conjecture 1 (The Dichotomy Conjecture [62])**.** *For every finite constraint language Γ, CSP(Γ) is either polynomial-time or NP-complete.*

More that twenty years have passed and this conjecture is still open. Much of the initial research on non-uniform CSP consisted in disparate tractability or hardness results, with no unified framework to relate these results to each other [128][133][95]. This issue was solved by the *algebraic approach* to non-uniform CSP, formally introduced in a seminal 1997 paper by Cohen, Gyssens and Jeavons [94] but already suggested in Feder and Vardi's original work [62]. This approach has been the foundation of most of the recent successes [21][19][8]. The central idea is to relate the complexity of non-uniform CSP to the identities satisfied by certain closure operations called *polymorphisms*.

2.2.1 Expressive Power of Constraint Languages

Let Γ be a fixed, finite constraint language. The textbook approach to prove that CSP(Γ) is NP-complete is to pick another language Γ^* known to be NP-complete and patiently build gadgets for each $R \in \Gamma^*$ using only relations from Γ. The strategy of the algebraic approach to CSP is to study the complexity of constraint languages not by looking directly at the relations, but at the gadgets that can or cannot be made from them.

What we call *gadget* is generally a CSP instance whose solution set, when projected onto a specific subset of variables, is the desired relation. This notion of "CSP plus projections" is captured by formulas called *pp-definitions*.

Definition 3. A relation R has a **pp-definition** in a constraint language Γ if it can be defined via a formula using only relations from Γ, conjunction, existential quantification and the equality relation.

We also say that R is *expressible* over Γ. By extension, a language has a pp-definition in Γ if and only if each of its relations has one.

Definition 4. Let Γ be a constraint language. The **relational clone** of Γ, denoted by $\langle\Gamma\rangle$, is the set of all relations pp-definable in Γ.

If $\Gamma^* \subseteq \langle\Gamma\rangle$, then CSP($\Gamma^*$) is (logspace) reducible to CSP(Γ) because we can turn every relation in Γ^* into a gadget over Γ [94]. This improvement alone already simplifies greatly the task of classifying the complexity of all constraint languages, because it implies that we need not distinguish two languages that generate the same relational clone. The next step is to find a way to characterize nicely which relations are found in $\langle\Gamma\rangle$.

2.2.2 Polymorphisms

Polymorphisms are componentwise closure operations that can have the powerful property of witnessing that a given relation R does not belong to $\langle\Gamma\rangle$.

Definition 5. Let Γ be a constraint language over a domain $\mathcal{D}$ and $k > 0$ be a natural number. An operation $f : \mathcal{D}^k \to \mathcal{D}$ is a **polymorphism** of Γ if and only if

$$\forall R \in \Gamma,\ \forall \mathbf{t_1}, \ldots, \mathbf{t_k} \in R,\ \mathbf{f}(\mathbf{t_1}, \ldots, \mathbf{t_k}) \in R$$

where $\mathbf{f}$ is the operation on tuples obtained by componentwise application of f.

The set of polymorphisms of a given constraint language contains every projection over its domain and is closed under composition. It is therefore a (concrete) *clone*. The link with relational clones is made explicit in the next proposition by Geiger, which dates back to 1968.

Proposition 1 ([74])**.** *If Γ and Γ^* are constraint languages over the same domain $\mathcal{D}$, then $\Gamma^* \subseteq \langle\Gamma\rangle$ if and only if every polymorphism of Γ is a polymorphism of Γ^*.*

This means in particular that in order to show that a constraint language Γ^* is *not* expressible over Γ, we need only produce a single polymorphism of Γ that does not preserve Γ^*. Unfortunately, in the worst case the size of this polymorphism can be exponentially large and deciding expressibility is co-NEXPTIME-hard in general [134][132].

Intuitively, the computational hardness of a language Γ can only increase as $\langle\Gamma\rangle$ gets bigger. Thus, if Γ has very nontrivial polymorphisms (i.e. operations that are polymorphisms of few relations) then $\langle\Gamma\rangle$ will be severely restricted and Γ has a chance to be tractable. This relationship between relational clones and clones of polymorphisms is quite tight as a Galois connection can be established between the two [92].

Example 2. Let us consider the language $\Gamma_{\text{3-LIN}_p}$ of all 3-variables linear equations over GF(p) for a given prime natural number $p \geq 2$. CSP($\Gamma_{\text{3-LIN}_p}$) is easily seen as polynomial-time via Gaussian elimination, and thus it must admit nontrivial polymorphisms. Let $R(x, y, z) \iff ax + by + cz = d$ be an equation over GF(p), and let f be such that $f(d_1, d_2, d_3) = d_1 - d_2 + d_3$. Observe that if $(\mathbf{t_1}, \mathbf{t_2}, \mathbf{t_3}) = ((x_1, y_1, z_1), (x_2, y_2, z_2), (x_3, y_3, z_3))$ are tuples of R then

$$\begin{aligned} a\, f(x_1, x_2, x_3) + b\, f(y_1, y_2, y_3) + c\, f(z_1, z_2, z_3) \\ &= a\,(x_1 - x_2 + x_3) + b\,(y_1 - y_2 + y_3) + c\,(z_1 - z_2 + z_3) \\ &= d \end{aligned}$$

and hence $\mathbf{f}(\mathbf{t_1}, \mathbf{t_2}, \mathbf{t_3}) \in R$. This means that f is a polymorphism of $\Gamma_{\text{3-LIN}_p}$; this type of group-theoretic polymorphism is highly nontrivial and is a particular case of *Mal'tsev polymorphisms*, which will be discussed in Section 2.2.4.

Now, we can start asking the important question: what makes a polymorphism desirable? We aim to identify polymorphisms that imply the tractability of any language they preserve, so they must enforce a strong structural property to the relational clone or, equivalently, to the clone of polymorphisms. The study of such properties is the main topic of universal algebra, and they are usually presented in the form of *identities* satisfied by the operations (here, polymorphisms), where identities are universally quantified equations over operation symbols and domain variables.

Throughout the thesis we will make use of a number of different properties for polymorphisms, and many of them will appear in multiple sections or chapters. For simplicity, we have gathered them in the next definition. The wavy equality sign $\approx$ signifies that the variables (x, y, z, etc.) are universally quantified over the domain.

Definition 6. Let f be an operation of arity k. We say that f is

- **idempotent** if
$$f(x, \ldots, x) \approx x$$
- **conservative** if $\forall (x_1, \ldots, x_k)$,
$$f(x_1, \ldots, x_k) \in \{x_1, \ldots, x_k\}$$
- a **Siggers operation** if $k = 4$ and
$$f(y, x, y, z) \approx f(x, y, z, x)$$

- a **weak near-unanimity (WNU) operation** if $k \geq 3$ and

$$f(y, x, x, \ldots, x, x) \approx f(x, y, x, \ldots, x, x) \approx \ldots \approx f(x, x, x, \ldots, x, y)$$

- a **near-unanimity (NU) operation** if $k \geq 3$ and

$$f(y, x, x, \ldots, x, x) \approx f(x, y, x, \ldots, x, x) \approx \ldots \approx f(x, x, x, \ldots, x, y) \approx x$$

- a **majority operation** if $k = 3$ and

$$f(x, x, y) \approx f(x, y, x) \approx f(y, x, x) \approx x$$

- a **minority operation** if $k = 3$ and

$$f(x, x, y) \approx f(x, y, x) \approx f(y, x, x) \approx y$$

- a **Mal'tsev operation** if $k = 3$ and

$$f(x, x, y) \approx f(y, x, x) \approx y$$

- a **generalized majority-minority (GMM) operation** if $k \geq 3$, and $\forall(a, b)$ either

$$\forall(x, y) \in \{a, b\},\ f(y, x, \ldots, x) = f(x, y, \ldots, x) = \ldots = f(x, x, \ldots, y) = x$$

or

$$\forall(x, y) \in \{a, b\},\ f(y, x, \ldots, x) = f(x, x, \ldots, y) = y$$

- a **(k-1)-edge operation** if $k \geq 3$ and

$$\begin{gathered} f(x, x, y, y, y, \ldots, y, y) \approx y \\ f(x, y, x, y, y, \ldots, y, y) \approx y \\ f(x, y, y, x, y, \ldots, y, y) \approx y \\ f(x, y, y, y, x, \ldots, y, y) \approx y \\ \ldots \\ f(x, y, y, y, y, \ldots, x, y) \approx y \\ f(x, y, y, y, y, \ldots, y, x) \approx y \end{gathered}$$

- a **totally symmetric (TS) operation** (or **set operation**) if $\forall(x_1, \ldots, x_k, y_1, \ldots, y_k)$,

$$\{x_1, \ldots, x_k\} = \{y_1, \ldots, y_k\} \Rightarrow f(x_1, \ldots, x_k) = f(y_1, \ldots, y_k)$$

- a **semilattice operation** if $k = 2$ and

$$f(x,x) \approx x, \quad f(x,y) \approx f(y,x), \quad f(x,f(y,z)) \approx f(f(x,y),z)$$

- a **2-semilattice operation** if $k = 2$ and

$$f(x,x) \approx x, \quad f(x,y) \approx f(y,x), \quad f(x,f(x,y)) \approx f(x,y)$$

These identities will allow us to present concise definitions of the most important tractable classes of languages found in the literature. Polymorphisms have proved to be very valuable in capturing the deep computational properties of constraint languages; in particular they have been used to make a more specific version of Conjecture 1 that suggests a characterization of all tractable constraint languages.

Conjecture 2 ([26])**.** *Let* Γ *be a constraint language.* Γ *is tractable if and only if it has a Siggers polymorphism.*

This conjecture is the algebraic equivalent of the statement "a language is tractable if and only if it cannot simulate Positive 1-in-3-SAT", where the "simulation" is done using a slightly improved version of pp-definability called *pp-interpretability*. It can be made even more specific by assuming without loss of generality that Γ is a digraph [62] and a *core* [26], that is, a language whose unary polymorphisms (endomorphisms) are all surjective. The conjecture has been confirmed in the case of 2- and 3-element domains [128][19], languages that contain all possible unary relations over their domain (which are called *conservative* and will be the main focus of Chapter 4) [21], and digraphs with no sinks and no sources [8] (generalizing the Hell-Nešetřil theorem for undirected graphs [85]).

Now that we know what polymorphisms are and why they are important, we can finally survey the major tractability results for non-uniform CSP. We will partition them into two groups, one for each of the two main algorithmic techniques.

2.2.3 Tractability via Local Consistency

Local consistency techniques are the most common incomplete polynomial-time algorithms for CSP in practical works. The idea is as follows: instead of looking at the instance as a whole, focus on subproblems with a bounded number of variables and infer as much information as you can from them. This information will be embodied by new constraints, which may in turn be used to refine the reasoning made earlier on adjacent subproblems, and so on. This fundamental process is generally referred to as *constraint propagation*. If a contradiction is derived on a

subproblem then the instance has no solution. For certain constraint languages, the converse is true.

The ideas behind local consistency fit quite well the reasoning most humans would do when solving a CSP. Let us illustrate this with sudoku solving. What everybody does first is to create a list of potential values for each cell, and then review the constraints involving that cell (line, column and subtable) to see which values can be pruned. Whenever a list has been reduced, we check if the change can be used to prune additional values elsewhere, and so on. This is one of the simplest form of local consistency, where we derive new constraints (lists) on size-1 subproblems (cells).

Definition 7. Let k, l be fixed natural numbers such that $1 \leq k \leq l$. A CSP instance is (k, l)-minimal if

- Every subset of l variables is contained in the scope of a constraint, and
- For every subset X of k variables, the projections onto X of any two constraints whose scopes contain X are identical.

For every fixed k, l a CSP instance I can be turned into an equivalent instance I' that is (k, l)-minimal in polynomial time. The algorithm adds new "allow-all" constraints on every size-l subset of variables, and then removes tuples from the constraints until all projections on k variables agree. Given a language Γ, we say that (k, l)-minimality *decides* CSP(Γ) if the above algorithm always derives a contradiction on unsatisfiable instances of CSP(Γ). In this case, we say that Γ has *width* (k, l). We will use *width* k as a shorthand for width (k, k), *k-minimality* for (k, k)-minimality and say that Γ has *bounded width* if it has width (k, l) for some k, l.

For SAT, unit propagation is equivalent to enforcing 1-minimality, so Horn clauses are a simple example of width 1 language [128]. The idea was later generalized to show that all constraint languages with a semilattice polymorphism have width 1 [94]. A full characterization of width 1 languages can be found in Feder and Vardi's work [62], which has been subsequently reformulated in algebraic terms by Dalmau and Pearson [49].

Theorem 1 ([49])**.** *A constraint language Γ has width 1 if and only if it has totally symmetric polymorphisms of all arities.*

An equivalent formulation requires a single totally symmetric polymorphism of arity $D(\Gamma)$ [49].

Remark 2. A CSP instance is *generalized arc consistent* (GAC) if there is one unary constraint $D(x)$ on each variable $x \in \mathcal{X}$, and for every $d \in D(x)$ and $C \in \mathcal{C}$ such that $x \in \mathcal{S}(C)$ there exists $\mathbf{t} \in \mathcal{R}(C)$ such that $\forall y$, $\mathbf{t}[y] \in D(y)$ and

$\mathbf{t}[x] = d$ [111][112]. The tuple $\mathbf{t}$ is called a *support* for the pair (x, d). 1-minimal instances are GAC, and if I is GAC then removing from every constraint every tuple that is not a support yields a 1-minimal instance, so GAC and 1-minimality are almost the same property. However, enforcing GAC does not involve removing tuples from constraints of arity greater than 1 and as such this operation is much easier to perform on instances involving intentionally-represented relations; all that is needed is an algorithm δ_R^+ for each relation R that computes in polynomial time the variable-value pairs that have supports in R. This algorithm δ_R^+ is called a *propagator*, and the design of such algorithms for frequently used global constraints is a very active research topic [123][113][14]. Propagators will be the main motivation for the contributions presented in Chapter 5.

The set of all bijunctive clauses, defined as conjunctions of binary clauses, is an example of language that has width $(2, 3)$ but not width 1 [128]. On the CSP front, *connected row-convex* constraints have been shown to have width 3 [55]. In both cases, the languages have actually a stronger property than bounded width called *bounded strict width* in which every consistent assignment to k variables (for some finite k) can be extended greedily to a solution of the whole instance. The polymorphisms characterizing bounded strict width were already identified by Feder and Vardi (although the terminology used was different) [62], and Jeavons, Cohen and Cooper gave a refined characterization [93]. We say that a relation is *k-decomposable* if it is equivalent to the conjunction of its projections on k variables.

Theorem 2 ([93])**.** *Let Γ be a constraint language and $k \geq 3$. The following statements are equivalent:*

- *Γ has strict width k;*
- *Γ has a near-unanimity polymorphism of arity k;*
- *Every $R \in \langle\Gamma\rangle$ is $(k-1)$-decomposable.*

While all width 1 and bounded strict width languages were identified at the early times of the algebraic era, progress on other languages on the bounded width hierarchy has been quite slow (aside from Bulatov's proof that languages with a 2-semilattice polymorphism have width 3 [18]). In Feder and Vardi's original work, a conjecture was formulated: every core constraint language Γ has either bounded width or the *ability to count*, which is informally the ability to simulate linear equations over a finite field [62]. More than fifteen years later, this conjecture was confirmed by Barto and Kozik [7] and a simple polymorphism-based characterization was given in [101].

Theorem 3 ([7][101])**.** *A constraint language* Γ *has bounded width if and only if it has two weak near-unanimity polymorphisms* f, g *of respective arities* $3, 4$ *satisfying*

$$f(x, x, y) \approx g(x, x, x, y)$$

Note that [7] only establish this result for core constraint languages, but it can easily be extended to the general case (see e.g. Lemma 6.4 of [36]). This condition is equivalent to having weak near-unanimity polymorphisms of all arities greater than 3. The tractable class of all languages of bounded width is very general, but at first sight it could suffer from a scalability problem: enforcing (k, l)-minimality is exponential-time in l, which can potentially be arbitrarily large depending on Γ. A surprising result, obtained independently by Bulatov and Barto, shows that every language of bounded width has width $(2, 3)$ [6][20][23]. Combining this result with [47] we can draw the following picture.

Theorem 4 ([6][47])**.** *Let* Γ *be a constraint language. Exactly one of the following statements is true:*

- Γ *has width* 1*;*
- Γ *has width* $(2, 3)$*, but not width* 2 *nor width* $(1, l)$ *for any* $l \geq 1$*;*
- Γ *does not have bounded width.*

As it is, the class of languages of bounded width seems reasonably practical since enforcing $(2, 3)$-minimality is cubic time. In a very recent development, it was shown that enforcing *singleton linear arc consistency* (SLAC), a consistency notion weaker than $(2, 3)$-minimality, is an alternative decision procedure for languages of bounded width [100]. Enforcing SLAC is quadratic time. This series of highly nontrivial results is an edifying example of the impressive progress made in the understanding of tractable classes during the past decades.

2.2.4 Tractability via Few Subpowers

We have seen in the last section that the only obstruction to solvability by local consistency methods is the ability to simulate linear equations over a prime field. This leads us directly to the second major algorithmic technique, which can handle those equations as it generalizes Gaussian elimination. This type of algorithm differs fundamentally from local consistency methods as it uses polynomial-sized representations of all solutions to reason directly on the whole set of variables. For instance, the solution set of a system of homogeneous linear equations is a vector space, which can be represented very efficiently by a basis and the table of the field operations. In the few subpowers algorithm, the encoding of relations is very

similar: a small subset of tuples is kept (a *generating set*), and polymorphisms are used to combine them.

Initially, this generating set approach was borrowed from group theory [62] but since then it has been extended to more general algebras. The first major breakthrough of the approach was Bulatov's proof that the existence of a Mal'tsev polymorphism implies tractability.

Theorem 5 ([17])**.** *Every constraint language that admits a Mal'tsev polymorphism is tractable.*

The original proof was very involved, but it has been greatly simplified over the years [24][60]. The most recent algorithm encodes relations (i.e. solution sets) using *frames*, which are generating sets whose size is linear in both the domain size and the relation arity, and whose closure under the Mal'tsev polymorphism is the original relation [60]. The algorithm for solving CSP instances assuming a Mal'tsev polymorphism starts from a frame of the solution set of an instance with no constraints (i.e. $\mathcal{D}^n$) and then adds the constraints one by one, updating the frame at each step. The final frame is empty if and only if the instance does not have a solution.

The first notable generalization of Bulatov's result was obtained by Dalmau.

Theorem 6 ([46])**.** *Every constraint language that admits a generalized majority-minority polymorphism is tractable.*

Because near-unanimity polymorphisms are also generalized majority-minority polymorphisms, the languages they preserve can be solved by both local consistency and algorithms based on polynomial-sized representations.

The conclusion of this line of work was reached in [90] and [10] with a necessary and sufficient condition for this algorithmic technique to be applicable: the existence of an edge polymorphism.

Theorem 7 ([90])**.** *Every constraint language that admits an edge polymorphism is tractable.*

The algorithm for CSP over a language with an edge polymorphism is usually referred to as the *few subpowers algorithm* (hence the title of this section). If a language Γ does not have an edge polymorphism, the number of distinct relations of arity r in $\langle\Gamma\rangle$ grows too fast with r for compact representations to exist [10], so this result determines precisely the reach of methods based on generating sets.

2.3 Structural Tractable Classes

In the relational structure homomorphism formulation of CSP seen in Section 2.1, non-uniform CSP corresponds to the case where the right-hand side structure is

fixed. A class of CSP is said to be *structural* if the left-hand side structure is restricted instead (but not fixed, because it would be trivially polynomial-time). None of our contributions will involve directly structural classes of CSP, but we feel that a quick overview would help the reader understand the big picture.

A *hypergraph* is a pair (V, E) where V is a finite set of *vertices*, and E is a collection of non-empty subsets of V called *hyperedges*. A *graph* is an hypergraph with only size-2 hyperedges, which are called *edges*.

Definition 8. The **constraint hypergraph** of a CSP instance $(\mathcal{X}, \mathcal{D}, \mathcal{C})$ is the hypergraph $(\mathcal{X}, \{\mathcal{S}(C) \mid C \in \mathcal{C}\})$.

Definition 9. The **primal graph** of a CSP instance $(\mathcal{X}, \mathcal{D}, \mathcal{C})$ is the graph $(\mathcal{X}, E)$ such that $(x, y) \in E$ if and only if there exists $C \in \mathcal{C}$ such that $\{x, y\} \subseteq \mathcal{S}(C)$.

Like many other combinatorial problems, CSP is hard because of feedback effects. Restricting the primal graph to be acyclic turns arc-consistency into a complete decision procedure for CSP, and thus this fragment is tractable [53]. More generally, this property of having limited feedback can be extended using tree decompositions.

Definition 10. Let $G = (V, E)$ be a graph. A **tree decomposition** of G is a tree $T = (\{V_1, \ldots, V_n\}, E_T)$ whose vertices are subsets of V and such that

(i) $\cup_{i \in \{1,\ldots,n\}} V_i = V$;

(ii) $\forall e \in E$, there exists i such that $e \subseteq V_i$;

(iii) $\forall v \in V$, the sets V_i that contain v form a connected subtree of T.

The *width* of a tree decomposition is the size of the largest set V_i minus one, and the *treewidth* of a graph is the minimum width possible over all its tree decompositions. The following theorem by Grohe shows the full strength of tree decompositions.

Theorem 8 ([79])**.** *Let $\mathcal{G}$ be a recursively enumerable class of graphs. Assuming FPT $\neq$ W[1][1], the restriction of* CSP *to instances whose primal graph is in $\mathcal{G}$ is polynomial-time if and only if $\mathcal{G}$ has bounded treewidth.*

The straightforward algorithm for solving a CSP instance assuming a tree decomposition works in a dynamic programming fashion, by computing full solution sets of the subinstances corresponding to the set V_i starting from the leaves. The bound on the size of each V_i ensures that the number of solutions is polynomial.

[1] A common complexity-theoretic assumption; additional information on these classes can be found in Section 2.4

An alternative is the k-consistency algorithm, which becomes a decision procedure when the primal graph has treewidth at most k [65].

When considering restrictions on the constraint hypergraphs, the complexity landscape becomes a bit more varied. If the restriction entails bounded arity, every tractable case falls in the scope of Theorem 8 [79]. However, it is clear that this cannot be the case in general. For example, if the restriction only imposes the existence of an hyperedge that covers every vertex, the treewidth of the primal graph is not bounded but the associated CSP is polynomial-time: this hyperedge translates into a giant constraint that covers the whole instance, and solutions can only be found among its tuples.

Fractional edge covers assign to each hyperedge a weight in such manner that for each vertex v, the weights of the hyperedges that contain v sum up to 1. The *fractional edge cover number* is the minimum total weight over all fractional edge covers. Grohe and Marx [80] showed that if a restriction on constraint hypergraphs entails a bounded fractional edge cover, not only the associated CSP is tractable but every instance has polynomially many solutions. Recall that the dynamic programming approach on (hyper)tree decompositions works well because each bag of vertices has few solutions; if the two approaches are mixed by defining the size of each bag as its fractional edge cover number rather than its number of vertices, we obtain the tractable restriction of CSP instances whose constraint hypergraphs have bounded *fractional hypertree width*. This is the most general structural tractability result known to date, but there is no matching hardness result.

Theorem 9 ([80])**.** *Let $\mathcal{H}$ be a class of hypergraphs with bounded fractional hypertree width. The restriction of* CSP *to instances whose constraint hypergraph is in $\mathcal{H}$ is polynomial-time.*

There is a certain asymmetry between the results we presented for left- and right-hand side restrictions: in this section, we considered constraint hypergraphs but systematically ignored the relation symbols that label each hyperedge. With these labels taken into account the picture is not much different, as Theorem 8 carries over if $\mathcal{H}$ is only required to have bounded treewidth modulo homomorphic equivalence [79].

2.4 Parameterized Complexity

It is reasonable to say that an algorithm is efficient if it scales well when the instances become more difficult. Classical complexity theory is a direct application of this statement, in which the input size is implicitly regarded as a universal measure for the intrinsic hardness of a given instance. This is quite coarse-grained;

maybe the algorithm performs very well on some distributions of inputs and terribly on others. When this situation arises, one can only wonder: is there a better measure of hardness than the input size?

Parameterized complexity aims to answer that question by measuring the efficiency of an algorithm as a multivariate function of both the input size and an auxiliary measure, the *parameter*. The parameter will typically attempt to measure some structural property of the instance; in graphs problems, it can for example measure how tree-like the input graph is. If the time complexity of an algorithm scales well when the parameter k is fixed but the input size increases, we can narrow down the source of hardness to the structural property measured by k. Just like classical complexity, this multivariate analysis of algorithms can then be used to study problems.

A problem is *parameterized* if each instance x is paired with a nonnegative integer k called the *parameter*.

Definition 11. The class **XP** consists of all parameterized problems that can be solved in time $O(f(k)|x|^{g(k)})$ for some computable functions f and g.

This class of problems is an immediate application of the discussion above: if the parameter k is fixed, the complexity of the algorithm witnessing membership in XP grows polynomially with the input size. These parameterized problems are called *slice-wise polynomial*. However, XP is not what parameterized complexity is famous for, since the vast majority of the literature is concerned with a more ambitious class of problems.

Definition 12. The class **FPT** consists of all parameterized problems that can be solved in time $O(f(k)|x|^{O(1)})$ for some computable function f.

The problems in FPT are said to be *fixed-parameter tractable*. FPT exhibits the same property as XP: if the parameter is fixed, an FPT algorithm becomes polynomial-time. The major difference is that k now only contributes to the constant factor, and no longer to the degree of the polynomial. This distinction yields a huge difference in scalability because even a polynomial of degree 10 is largely impractical.

Example 3. The Vertex Cover problem is a classical NP-complete problem that will be discussed in depth in Chapter 5. Vertex Cover takes as input a graph $G = (V, E)$ and an integer p, and asks if there exists $S \subseteq V$ of at most p vertices that intersects every edge in E. We will use p as the parameter. Membership in XP is straightforward since we can simply enumerate all $|V|^p$ subsets of at most p vertices.

To see that Vertex Cover parameterized by p is in FPT as well, let us consider the following algorithm $\mathcal{A}$ which takes as input G and a subset L of

vertices. If L intersects every edge in E, the algorithm returns `true`. Then, if $|L| \geq p$ we return `false` and otherwise we pick an edge $e = (u, v)$ that does not intersect with L and return $\mathcal{A}(G, L \cup \{u\}) \vee \mathcal{A}(G, L \cup \{v\})$. Now, let us see what happens when we invoke $\mathcal{A}(G, \emptyset)$. If $\mathcal{A}(G, \emptyset)$ returns `true`, at least one recursion call to $\mathcal{A}$ has been made with an input L of size at most p that intersects every edge, so the instance of Vertex Cover is satisfiable. Conversely, if the instance has a solution S then invoking $\mathcal{A}$ with $L \subset S$ as input will make two recursion calls with respectively $L \cup \{u\}$ and $L \cup \{v\}$ for some edge $\{u, v\}$. Because S must intersect $\{u, v\}$, at least one of these calls will be with a set $L' \subseteq S$ such that $|L'| = |L| + 1$ and we can repeat the same argument inductively. Eventually, a recursion call will be made with $L = S$ and $\mathcal{A}$ will return `true`. Therefore, because $\emptyset \subseteq S$, $\mathcal{A}(G, \emptyset)$ will always return `true` if the instance is satisfiable.

The recursion tree of $\mathcal{A}$ has a branching factor of 2, its depth cannot exceed p and the amount of work at each node is polynomial, so $\mathcal{A}(G, \emptyset)$ gives its answer in time $O(2^p|G|^{O(1)})$. This is a simple illustration of the bounded search tree technique which will be used in Chapter 3.

Beyond Vertex Cover a number of problems have been shown to be FPT for parameters much better than the input size. The classes FPT and XP are nested, with FPT known to be a strict subset of XP [59]. Between FPT and XP, Downey and Fellows proposed a full hierarchy of intermediate parameterized classes [57]:

$$\text{FPT} \subseteq \text{W[1]} \subseteq \text{W[2]} \subseteq \ldots \subseteq \text{XP}$$

where for each t, W[t] is believed to be a strict subset of W[t+1]; for instance, we know that FPT $\neq$ W[1] unless the Exponential Time Hypothesis fails [37]. This hierarchy is called the *W-hierarchy*, or *weft hierarchy*. Each of these classes is closed under *fpt-reductions*, which are the parameterized counterparts to polynomial-time many-one reductions and defined as follows.

Definition 13. Let $\mathcal{P}$ and $\mathcal{Q}$ be parameterized problems. An **fpt-reduction** of $\mathcal{P}$ to $\mathcal{Q}$ is an operation ϕ that maps every instance (x, k) of $\mathcal{P}$ to an instance (x', k') of $\mathcal{Q}$ such that

- x is a yes-instance if and only if x' is,
- ϕ can be computed in time $O(f(k)|x|^{O(1)})$ for some computable function f, and
- $k' \leq g(k)$ for some computable function g.

Fpt-reductions differ from polynomial-time many-one reductions in two ways: they are allowed to be exponential-time as long as they are FPT, but they must

preserve the parameter. The two types of reductions are in general incomparable. Fpt-reductions provide a well-defined completeness theory for these classes, and through this notion they give the ability to show that a problem is fixed-parameter *intractable* unless there is a collapse somewhere in the W-hierarchy. For instance, the k-Clique problem parameterized by k is W[1]-complete [58], and Dominating Set parameterized with solution size is W[2]-complete [57].

Over the years, the considerable scientific attention directed towards parameterized complexity has resulted in a wealth of algorithmic techniques tailored for FPT algorithm design. Notable examples include bounded search trees (as in Example 3), iterative compression [122], color coding [3], and LP-branching [91]; a detailed survey of these techniques (and more) can be found in [45]. In Chapter 5 we will be particularly concerned with another technique, *kernelization*.

Definition 14. Let $\mathcal{P}$ be a parameterized problem. A **kernelization** of $\mathcal{P}$ is an operation $\mathcal{K}$ that maps every parameterized instance (x, k) of $\mathcal{P}$ to a new instance (x', k') of the same problem such that

- $\mathcal{K}$ can be computed in polynomial time,
- $k' \leq g(k)$ for some computable function g, and
- $|x'| \leq f(k)$ for some computable function f.

The instance (x', k') is the *kernel* of (x, k). Kernelization is in essence a polynomial-time preprocessing routine whose performance in reducing the problem size is guaranteed in terms of the parameter. The philosophy is identical to that of the class FPT, as it captures algorithms that work well when the parameter is small. The idea is simply applied to polynomial-time preprocessing rather than exponential-time exact algorithms. Most kernelization algorithms work using a list of small, easily applicable reduction rules - the main technical challenge is often to prove that when these rules are no longer applicable, the size of the instance is bounded by a function of the parameter. Kernelization has been the subject of extensive research, with very small kernels obtained for a variety of problems (e.g. at most $2k$ vertices for Vertex Cover [117], or k^6 clauses for Almost-2-SAT [103]), and new tools have been developed to prove kernelization lower bounds [15][86].

If a kernelization exists for a given parameterized problem, an FPT algorithm exists as well because employing bruteforce on the kernel is FPT. An important fact about kernelization is that the converse is true as well: if a problem is FPT, then it can be kernelized. To see this, assume that you have an algorithm with running time $O(f(k)|x|^{O(1)})$ that solves your problem. Then, if the instance has size at least $f(k)$, this algorithm is actually polynomial-time and we can use it

to compute a constant-sized kernel, which only depends on the satisfiability of the instance. Otherwise, we can leave the instance untouched since $|x| \leq f(k)$. Therefore, a kernel of size $f(k)$ can be derived from the FPT algorithm. However, because $f(k)$ will typically be exponential in k, this kernel is very poor; the true challenge is to find kernels that are polynomial-sized and as small as possible.

Chapter 3

Backdoors into Tractable Constraint Languages

In general, tractable classes that can be solved by an algorithm whose complexity is a reasonable polynomial tend to be quite small, in the sense that they contain only highly structured instances. Furthermore, even assuming that every constraint language that is not currently known to be NP-hard is tractable (which is as hopeful as one can be), the probability that a language comprised of a single loop-free random relation is tractable tends to zero as either the domain size or the maximum arity tends to infinity [110]. This effect is even more pronounced for random languages because they are at least as hard as each of their relations. In practice, even though industrial CSP instances are all but random it is quite unlikely that they fit nicely into one of the few well-studied tractable classes presented in Chapter 2.

The viewpoint adopted in this chapter is that it is possible to make broader use of the theoretical work on tractable classes by defining a proper notion of *distance* between a given instance and an efficiently solvable restriction of CSP. In this setting, this notion of distance to a tractable class $\mathcal{H}$ must be algorithmically meaningful: the instances at distance k should be easier to solve than instances at distance $k+1$ (at least in terms of worst-case complexity), and there must exist a reasonably fast algorithm for solving instances k-distant from $\mathcal{H}$ if k is small. In the context of CSP and SAT, a natural measure of distance that satisfies our requirements is the *strong backdoor size*. Informally, a *strong backdoor* into a tractable class $\mathcal{H}$ is a subset of variables whose complete instantiation always yields a residual instance in $\mathcal{H}$. If an instance has a size-k strong backdoor B into a class $\mathcal{H}$, then it can be decomposed into at most $|\mathcal{D}|^k$ instances in $\mathcal{H}$ by branching exhaustively on the possible assignments to B. The computation of the set B itself can be done either by enumerating all k-subsets of variables or by a more sophisticated method. Strong backdoors were introduced in [135] with the

original goal of explaining the performance of CSP and SAT solvers, and have attracted significant attention in the following years [125][56][72].

When approaching strong backdoors, one must make a clear distinction between the solving phase, where the backdoor is known, and the computation of the backdoor. If a nontrivial backdoor B is already known then regardless of its size it is a useful information for a constraint solver, which can invoke a dedicated algorithm whenever the backtracking search procedure has assigned every variable in B (we omit small additional technical requirements here; variables outside B may have been assigned as well and the instance may have been reduced by auxiliary inference, such as consistency algorithms). In fact, if the backdoor size k is quite large this approach is likely to be more efficient in practice than a straight decomposition of the instance into $|\mathcal{D}|^k$ subproblems. For example, if a CSP instance has 150 variables a backdoor of size 40 provides an uncompetitive decomposition but can still allow a general backtracking algorithm to prune a large number of branches, even if the backdoor is not used to guide the branching heuristic. As a direct consequence, improving the worst-case complexity of the solving phase is important but not critical for the usefulness of the framework.

The computation of the backdoor itself is a whole different matter. As k grows, deciding if a size-k strong backdoor exists becomes increasingly difficult while the information it would provide becomes less and less computationally relevant. The critical point is the (approximate) value of k for which the backdoor approach no longer yield any performance gain overall. If the backdoor detection algorithm scales poorly with k, the threshold will be reached very quickly and the applicability of the whole framework will be severely restricted. Therefore, it makes sense to focus on the detection phase (instead of the solving phase) and more precisely on algorithms that scale very favorably with the backdoor size k. To that endeavor the use of parameterized complexity theory is natural and k is the parameter of choice.

In this chapter we shall attempt to identify which properties of the tractable class $\mathcal{H}$ affect the parameterized complexity of detecting strong backdoors into $\mathcal{H}$ for the parameters k and $k + r$ (backdoor size plus maximum arity). In a complementary fashion we also describe *partition backdoors*, a relaxation of strong backdoors that is noticeably easier to detect but may also be much larger.

3.1 Strong Backdoors

In this section we shall define formally all necessary notions and give a comprehensive overview of existing results on strong backdoor theory. While strong backdoors were originally defined in terms of *subsolvers* (in this case the target property is to be solvable by a given algorithm), we opt for another definition that only requires

the subproblems to belong to a given *set of instances*. This is roughly equivalent since we can always define the target set of instances to be exactly those solvable by the subsolver, or design the subsolver so that it only accepts the designated instances.

Definition 15. Let $\mathcal{I} = (\mathcal{X}, \mathcal{D}, \mathcal{C})$ be a CSP instance and $\mathcal{H}$ be a fixed set of CSP instances. A **strong backdoor** of $\mathcal{I}$ into $\mathcal{H}$ is a subset of variables $B \subseteq \mathcal{X}$ such that $\forall\, \phi : B \to \mathcal{D}$, $\mathcal{I}[B \leftarrow \phi(B)] \in \mathcal{H}$.

For the associated detection problem, we can either make $\mathcal{H}$ part of the input or define a detection problem for every fixed $\mathcal{H}$. The latter is more adapted to our ultimate purpose, which is to identify which properties of $\mathcal{H}$ can make strong backdoors into $\mathcal{H}$ easier to detect.

Strong $\mathcal{H}$-Backdoor Detection
Input: A CSP instance $I = (\mathcal{X}, \mathcal{D}, \mathcal{C})$, an integer k
Question: Does I have a strong backdoor into $\mathcal{H}$ of size at most k?

We note that strong backdoors could easily be strengthened with additional inference after the backdoor variables are assigned. Our definition requires that every subproblem must belong to $\mathcal{H}$, even those that are trivially unsatisfiable. While we have no doubt that extra layers of inference (such as arc consistency) could have a dramatic effect on the backdoor size, the backdoor is also likely to become unrealistically hard to compute. For instance, in the context of SAT even the very straightforward feature of *empty clause detection* (which adds to $\mathcal{H}$ every instance that contains an empty clause) makes backdoor detection substantially more difficult [56]. We believe that finding a form of inference that improves sufficiently the backdoor size to counterbalance the increased computational cost of detection is quite challenging. In this thesis we will only focus on the standard definition of strong backdoor and leave these considerations for future work.

Example 4 (Root sets)**.** Following [51], given a binary CSP instance $I = (\mathcal{X}, \mathcal{D}, \mathcal{C})$ and a constraint $C = ((x, y), R) \in \mathcal{C}$ we write $x \to y$ (resp. $y \to x$) if there exists a function $f : \mathcal{D} \to \mathcal{D}$ such that $(a, b) \in R \iff b = f(a)$ (resp. $(a, b) \in R \iff a = f(b)$). If either $x \to y$ or $y \to x$, C (and by extension R) is said to be *functional*. Using the $\to$ relation, one can associate every binary instance I with a directed graph G_I that describes the functional dependencies of the variables. One of the key ideas of [51] is that evaluating every complete instantiation of a *root set* of G_I (a subset of vertices B such that every vertex can be reached from B via a directed path) is sufficient to decide the satisfiability of I. Because of the functional dependencies, a root set is always a strong backdoor into the set of all instances that are solved by a single call to the arc consistency

procedure. The converse is false, but finding a minimum-size root set in a binary CSP has the advantage of being polynomial-time [51].

Because of the complete freedom left in the definition of the class $\mathcal{H}$, strong backdoor detection covers a wide variety of problems as particular cases. For instance, if $\mathcal{H}$ is the set of all CSP instances whose constraint graph has some fixed property $\mathcal{P}$ then detecting a minimum-size backdoor into $\mathcal{H}$ can be reformulated as finding in the constraint graph the largest induced subgraph with property $\mathcal{P}$ (or, equivalently, a minimum subset of vertices whose removal leaves a residual graph with property $\mathcal{P}$), a classical problem which has been extensively studied for many choices of $\mathcal{P}$ [109][81][89][29]. Orthogonally, some forms of symmetry breaking can be formulated in terms of strong backdoor detection. Given a CSP instance $I = (\mathcal{X}, \mathcal{D}, \mathcal{C})$ a variable $x \in \mathcal{X}$ is *irrelevant* [16] if for every solution $\phi : \mathcal{X} \to \mathcal{D}$ and $v \in \mathcal{D}$, the assignment $\phi' : \mathcal{X} \to \mathcal{D}$ such that $\phi'(y) = v$ if $y = x$ and $\phi'(y) = \phi(y)$ otherwise is also a solution. The set of all irrelevant variables of a CSP instance is exactly the complement of the minimum-size strong backdoor into the union of $\mathrm{CSP}(\{\mathcal{D}^k \mid k \in \mathbb{N}\})$ with the set of all unsatisfiable instances. An even more far-fetched example of strong backdoor detection would consider CSP as deciding if the input instance has an empty strong backdoor to the set of all satisfiable CSP instances. It is clear from these examples that if we wish to keep backdoor detection from being too wild we must restrict $\mathcal{H}$ to tractable classes of instances whose associated strong backdoor detection problem can be handled by relatively homogeneous algorithmic methods.

For CSP it is natural to focus on classes $\mathcal{H}$ of instances that are either structural (definable by restrictions on the constraint hypergraph) or language-based (definable by restrictions on the constraint language). This excludes degenerate classes while still allowing most types of backdoor that have a nonzero probability of being useful in practice.

If $\mathcal{H}$ is structural, the complexity of Strong $\mathcal{H}$-Backdoor Detection parameterized by k is a pure graph theoretic problem. In CSP, the most interesting case arises when $\mathcal{H}$ is the set of all CSP instances whose primal constraint graph has treewidth bounded by a constant c; assuming FPT $\neq$ W[1] and a bound on the constraint arities every tractable structural restriction of CSP must have bounded treewidth [79]. In this case Strong $\mathcal{H}$-Backdoor Detection is (non-uniform) FPT because the class $\mathrm{tw}_c + kv$ of all graphs obtained by adding at most k vertices to a graph with treewidth at most c is minor-closed and hence can be recognized in cubic time [124]. From a theoretical perspective this result essentially settles the complexity of detecting strong backdoors into (relevant) structural restrictions of CSP. It leaves however the practical questions completely unanswered since neither the FPT algorithm itself nor its exact complexity is known (only its existence is proven by [124]). The most notable exception is the case $c = 1$ (that is, back-

door to instances whose constraint graph is a forest) which admits a quite efficient backdoor detection algorithm [54]. This type of backdoor (sometimes referred to as a *cycle cutset* [53]) is well known to the CSP community but has yet to be exploited in practice.

Due to the broad attention attracted by the Feder-Vardi conjecture, the literature on the complexity of CSP leans heavily towards language-based tractable classes. However, aside from the content of this thesis the only notable results on the complexity of Strong $\mathcal{H}$-Backdoor Detection for CSP when $\mathcal{H}$ is language-based are presented in [69]. Their results can be summarized as follows: if $\mathcal{H}$ is characterized by a predicate on individual polymorphisms of bounded arity then Strong $\mathcal{H}$-Backdoor Detection is FPT with parameters $d + k + r$ (the combination of domain size, backdoor size, and constraint arity), W[2]-hard with parameter k if the polymorphisms are idempotent, and W[2]-hard with parameter k even on binary instances for a selection of common types of polymorphisms. There is a slight redundancy between their results and those presented in the present thesis due to the fact that they were obtained roughly at the same time and published independently.

Beyond CSP, there is a rich literature on Strong $\mathcal{H}$-Backdoor Detection for SAT, including a complexity classification of Strong $\mathcal{H}$-Backdoor Detection when $\mathcal{H}$ is a defined via a DPLL-based subsolver [131], one the base classes of Schaefer's Dichotomy Theorem [118], combinations of such classes [69] and several generalizations [72][70]. Backdoor theory has also been extended to valued CSP [66], answer-set programming [64], planning [104][105] and other AI problems beyond NP [63].

The main contribution of this chapter is a series of quite general theorems that aim to classify the parameterized complexity of Strong $\mathcal{H}$-Backdoor Detection for language-based classes $\mathcal{H}$. We opted to avoid using the domain size d as a parameter because it would restrict the applicability too severely (see [69] for a treatment of this case), and instead we focus on the parameters k and $k + r$.

3.2 Composite Classes

As stated in the previous section, we will focus on backdoors into language-based classes of instances. For simplicity, from this point on we will use $\mathcal{H}$ to denote classes of languages (instead of classes of instances, as we have done so far in this chapter) and write abusively Strong $\mathcal{H}$-Backdoor Detection to mean Strong CSP($\mathcal{H}$)-Backdoor Detection. In order to achieve a homogeneous treatment of existing tractable classes, we need a framework that is

(i) General enough to capture most tractable classes defined in the CSP literature;

(ii) Sufficiently restrictive to permit a uniform technical approach, and in particular avoid artificial or degenerate classes for which strong backdoor detection requires specific algorithmic techniques;

(iii) Capable of highlighting in a simple and clear fashion the key properties of a tractable class $\mathcal{H}$ that are relevant to the complexity of Strong $\mathcal{H}$-Backdoor Detection.

Given the nature of most published results, it is natural to focus on language-based classes defined by restrictions on the set of polymorphisms. The first idea that comes to mind is to suppose that we have some predicate $\mathbb{P}$ on the set of all polymorphisms of a language that holds true if and only if the language belongs to the class $\mathcal{H}$. This framework is extremely general and satisfies (i). However, to see that it fails to satisfy (ii) it is sufficient to observe that every reasonable tractable class of languages is closed by taking sublanguages, as any algorithm that solves CSP(Γ) can also solve CSP(Γ') for every $\Gamma' \subseteq \Gamma$. This simple property enforces a form of monotonicity, and by ensuring that the empty language is always in the class it also ensures that a strong backdoor exists. However, a general predicate on the set of all polymorphisms can for instance be designed to hold false whenever the language has a semilattice polymorphism, which yields an artificial tractable class that does not even contain the empty language and for which the backdoor detection problem has very unusual properties.

Instead of predicates on the set of polymorphisms, Gaspers et al. [69] opted for predicates on individual polymorphisms - a language Γ then belongs to the class if and only if the predicate holds true for at least one polymorphism of Γ. This solves the problem with (ii), but with a potential loss in expressibility since some classical tractable classes of languages are characterized by *combinations* of polymorphisms. Furthermore, our findings concerning the condition (iii) do not fit well within this framework.

For simplicity of presentation (and with the property (iii) in mind) we opted for a set-based approach instead of predicates. For the remainder of this chapter, we shall assume that every class $\mathcal{H}$ has a decidable membership problem and is associated with an infinite supply of domain values $D(\mathcal{H})$ such that the domain of each $\Gamma \in \mathcal{H}$ is a finite subset of $D(\mathcal{H})$. However, we do not require that every $d \in D(\mathcal{H})$ belongs to at least one $\Gamma \in \mathcal{H}$.

Definition 16. Let $\mathcal{H}$ be a class of constraint languages. We say that $\mathcal{H}$ is

- **simple** if $\forall\, \Gamma$, $(\Gamma \in \mathcal{H} \iff \forall\, R \in \Gamma, \{R\} \in \mathcal{H})$;
- **composite** if there exists a set $\mathcal{S}$ of simple classes such that $\mathcal{H} = \cup_{S \in \mathcal{S}} S$;
- **algebraic** if $\forall\, \Gamma \in \mathcal{H}$ and $\Gamma' \subseteq \langle \Gamma \rangle$, $\Gamma' \in \mathcal{H}$;

- **idempotent** if $\forall\, \Gamma \in \mathcal{H}$ and $a \in D(\mathcal{H})$, $\Gamma \cup \{[(a)]\} \in \mathcal{H}$;
- **conservative** if $\forall\, \Gamma \in \mathcal{H}$, $c \geq 1$ and $\mathbf{t} \in D(\mathcal{H})^c$, $\Gamma \cup \{[(\mathbf{t}[1]), \ldots, (\mathbf{t}[c])]\} \in \mathcal{H}$.

Note that by definition simple classes always contain the empty language. The set $\mathcal{S}$ is allowed to have infinitely many elements, which is critical for generality purpose, and using the distributivity of intersection over union together with the fact that any intersection of simple classes is simple it is easy to see that any class derived from simple classes using intersections and unions is composite. The following lemma is immediate.

Lemma 1. *A class $\mathcal{H}$ is composite if and only if $\forall\, \Gamma \in \mathcal{H}$ and $\Gamma' \subseteq \Gamma, \Gamma' \in \mathcal{H}$.*

Proof. Suppose that $\mathcal{H}$ is composite and let $\Gamma' \subseteq \Gamma \in \mathcal{H}$. By compositionality, there exists a simple class $F_\Gamma \subseteq \mathcal{H}$ such that $\Gamma \in F_\Gamma$, and it follows from the definition of simple classes that $\Gamma' \in F_\Gamma \subseteq \mathcal{H}$. For the converse implication, suppose that $\forall\, \Gamma \in \mathcal{H}$ and $\Gamma' \subseteq \Gamma, \Gamma' \in \mathcal{H}$ and define $F_\Gamma = \{\Gamma' | \Gamma' \subseteq \Gamma\}$ for every $\Gamma \in \mathcal{H}$. For a fixed Γ, $\emptyset$ always belongs to F_Γ and $\forall\, \Gamma' \neq \emptyset$, $\Gamma' \subseteq \Gamma \iff \forall R \in \Gamma', R \in \Gamma$. Therefore, each F_Γ is simple. Moreover $\mathcal{H} \subseteq \cup_{\Gamma \in \mathcal{H}} F_\Gamma$, which is composite, and by hypothesis we have $F_\Gamma \subseteq \mathcal{H}$ for every $\Gamma \in \mathcal{H}$, so $\cup_{\Gamma \in \mathcal{H}} F_\Gamma \subseteq \mathcal{H}$ and finally $\cup_{\Gamma \in \mathcal{H}} F_\Gamma = \mathcal{H}$. □

Corollary 1. *Every algebraic class is composite.*

Alternatively, composite classes can be defined directly using the property of Lemma 1. The choice of a constructive definition involving unions of simple classes will become clear from our results (Corollary 2 and Theorem 12 in particular). In general, we will assume algebraicity and idempotency for our hardness results but only compositionality for FPT algorithms.

Example 5. Let $\mathcal{H}_{\max}$ be the set of all languages over finite subsets of $\mathbb{N}$ that admit the polymorphism $\max(.,.)$ with respect to the usual ordering of $\mathbb{N}$. This class is tractable as any instance over a max-closed language can be solved by establishing generalised arc consistency. Using our terminology this class is simple, idempotent, conservative and algebraic. Now, consider instead the class $\mathcal{H}_{p\max}$ of all languages that are max-closed with respect to at least one ordering of $\mathbb{N}$. This class is still algebraic, idempotent and conservative, but no longer simple: $R_0 = [(0,1),(1,0),(0,0)]$ is max-closed with respect to all orderings $>_i$ such that $0 >_i 1$, $R_1 = [(0,1),(1,0),(1,1)]$ is max-closed with respect to all orderings $>_j$ such that $1 >_j 0$ but $\{R_0, R_1\}$ is not max-closed with respect to any ordering. More generally, every class that can be defined using disjunctions of predicates on individual polymorphisms is algebraic, and idempotent if the polymorphisms satisfying the predicate are.

3.3 Complexity of Strong Backdoor Detection

In general, FPT algorithms only make sense for problems that cannot be solved in polynomial time. Therefore, before we dive into the realm of parameterized complexity we must establish the classical hardness of Strong $\mathcal{H}$-Backdoor Detection.

Theorem 10. Strong $\mathcal{H}$-Backdoor Detection *is NP-hard for every idempotent algebraic tractable class* $\mathcal{H}$*, even for binary* CSP*.*

Proof. We reduce from Vertex Cover. Let $I = (G, k)$ be an instance of Vertex Cover and $D_3 = \{\alpha, \beta, \gamma\} \subset D(\mathcal{H})$ be a set of three distinct domain values. We consider two cases. First, suppose that $\Gamma = \{[(\alpha, \alpha), (\beta, \beta)], [(\beta, \beta), (\gamma, \gamma)], [(\alpha, \alpha), (\gamma, \gamma)]\} \notin \mathcal{H}$. We create a CSP with one variable per vertex in G, and if two variables correspond to adjacent vertices we add three constraints using respectively the three relations in Γ between them. Since $\Gamma \notin \mathcal{H}$, every strong backdoor into $\mathcal{H}$ corresponds to a vertex cover of G. Conversely, if we have a vertex cover then after any assignment of the corresponding variables we are left with unary constraints with at most one tuple and the resulting language is in $\mathcal{H}$ by idempotency. Now, suppose instead that $\Gamma \in \mathcal{H}$. For every edge (x, y) in G, we add the constraint $\neq_{D_3}(x, y)$ (the disequality on D_3). Since CSP($\neq_{D_3}$) is NP-hard and $\mathcal{H}$ is tractable, a strong backdoor must correspond to a vertex cover on G. Furthermore, once the variables corresponding to a vertex cover on G are assigned we are left with an instance over $\{[(\alpha), (\beta)], [(\beta), (\gamma)], [(\alpha), (\gamma)]\} \subseteq \langle\Gamma\rangle$. Therefore, any vertex cover is a strong backdoor into $\mathcal{H}$, which concludes the reduction. □

Remark 3. This theorem does not apply in general to tractable composite classes. For instance, let $\mathcal{H}$ be the set of all languages that contain only binary bijections between subsets of $D(\mathcal{H})$ or unary relations with a single tuple. This tractable class is simple (thus composite) and idempotent, but on binary CSP Strong $\mathcal{H}$-Backdoor Detection can be solved in polynomial time using these two observations:

(i) If there is a binary constraint R with scope (x, y) and α, β, γ such that $\{(\alpha, \beta), (\alpha, \gamma)\} \subseteq R$ and $\beta \neq \gamma$, then y must belong to every strong backdoor into $\mathcal{H}$;

(ii) If there is a unary constraint with scope (x) and strictly more than one tuple, then x must belong to every strong backdoor into $\mathcal{H}$.

These two observations define a sufficient condition for a variable to belong a minimum-size backdoor (i.e. the identified variables must belong to every strong backdoor into $\mathcal{H}$). We shall now prove that this condition is also necessary. Since

$\mathcal{H}$ is simple, we only need to show that every constraint in every residual instance belongs to $\mathcal{H}$. Suppose that we have identified a maximal set B of variables using (i) and (ii) and let $\mathcal{I}$ denote a residual instance obtained after some assignment ϕ to B. By rule (i), every binary constraint in $\mathcal{I}$ is a bijection so we only need to focus on unary constraints. By rule (ii), if $\mathcal{I}$ contains a unary constraint $C = ((x), R)$ with more than one tuple (say, $(\alpha), (\beta) \in R$) then it was produced from a binary constraint $C' = ((x, y), R')$ of the original instance in which $y \in B$ but $x \notin B$. Then, by hypothesis we had $(\alpha, \phi(y)), (\beta, \phi(y)) \in R'$, and thus applying rule (ii) should have also added x to B, a contradiction. Therefore, B is a strong backdoor into $\mathcal{H}$.

In the case of Boolean CSP, Theorem 10 cannot apply verbatim. This is due to the fact that every binary Boolean language is a special case of 2-SAT and is therefore tractable. Thus, a binary Boolean CSP has always a backdoor of size 0 to any class that is large enough to contain 2-SAT, and the minimum backdoor problem is trivial. The next proposition shows that this is the only case for which Strong $\mathcal{H}$-Backdoor Detection is not NP-hard under the idempotency condition. Note that looking for a strong backdoor in a binary Boolean CSP has no practical interest; however this case is considered for completeness.

Proposition 2. *On Boolean* CSP *with arity at most r,* Strong $\mathcal{H}$-Backdoor Detection *is NP-hard for every idempotent algebraic tractable class $\mathcal{H}$ if $r \geq 3$. For $r = 2$,* Strong $\mathcal{H}$-Backdoor Detection *is either trivial (if every binary Boolean language is in $\mathcal{H}$) or NP-hard.*

Proof. The proof is essentially identical to that of Theorem 10, only with more cases to examine. First, suppose that every binary Boolean language is in $\mathcal{H}$. Then, if $r = 2$, Strong $\mathcal{H}$-Backdoor Detection is trivial. If $r \geq 3$, we reduce from 3-Hitting Set. Let $I = (U, S, k)$ be an instance of 3-Hitting Set. We create a CSP instance with one variable per element in U, and for each $(u_i, u_j, u_l) \in S$, if $R_1 = [(1, 0), (0, 1)] \in \mathcal{H}$ we add the constraint $R_2 = [(1, 0, 0), (0, 1, 0), (0, 0, 1)]$ on the corresponding variables and $R_3 = [(1, 0, 0), (0, 1, 1)]$ otherwise. Observe that, since CSP($\{R_2\}$) is known to be NP-hard (by a reduction from Positive 1-in-3-SAT) and $R_3 \in \langle\{R_1\}\rangle$, in either case the added constraint does not belong to $\mathcal{H}$. Thus, if a backdoor of size at most k exists, then the corresponding subset of U must be a hitting set. Conversely, let B denote the CSP variables associated with a hitting set of size at most k. If we used the constraint R_2 in the reduction, then after a complete assignment to B the remaining constraints are a subset of $\{R_1, [(0)], [(1)], [(0, 0)]\}$ which is in $\mathcal{H}$ since $\mathcal{H}$ is idempotent, algebraic and $R_1 \in \mathcal{H}$. If the reduction was done using R_3, after any assignment we are left with a CSP instance whose language contains only relations with a single tuple, which always belong to $\mathcal{H}$ by idempotency and algebraicity. Finally, in both cases, B is a

backdoor of size k, which completes this part of the reduction. Now, suppose that there exists a binary Boolean language Γ that is not in $\mathcal{H}$. Once more, we reduce from Vertex Cover. Let (G, k) be an instance of Vertex Cover and $\Gamma_1 = \{[(0,0),(1,1)]\}$. We create a CSP instance with one variable per vertex in G. If two vertices are adjacent, we add between them the constraint of Γ_1 if $\Gamma_1 \notin \mathcal{H}$ and all constraints in Γ otherwise. By construction any strong $\mathcal{H}$-backdoor of size at most k must be a vertex cover, and the same line of reasoning as in the proof of Theorem 10 gives us that the variables corresponding to any vertex cover of size at most k is a strong backdoor into $\mathcal{H}$, which concludes the proof. □

The typical example of an algebraic tractable class that is not idempotent (and thus does not fall into the scope of Theorem 10) is that of constant-closed languages. We will explore strong backdoor detection for that particular case in Section 3.3.3. Finally, while these results are enough to justify the use of parameterized complexity, they do not quite settle the classical complexity of strong backdoor detection. This problem does not seem to be in NP in general as checking if a subset of variables is a strong backdoor can be difficult. However, we will show that Strong $\mathcal{H}$-Backdoor Detection is in NP for many composite classes $\mathcal{H}$ in Section 3.3.2.

3.3.1 Parameter k

We now consider the parameterized complexity of Strong $\mathcal{H}$-Backdoor Detection when the parameter is k (the size of the backdoor) and show that unfortunately no FPT algorithm exists unless FPT = W[2], at least when $\mathcal{H}$ is algebraic and idempotent. Furthermore, we establish this result under the very restrictive condition that the input CSP has a single constraint, which highlights the fact that Strong $\mathcal{H}$-Backdoor Detection is more that a pseudo-Hitting Set on the constraints that do not belong to $\mathcal{H}$.

For any triplet $(\alpha, \beta, m) \in D(\mathcal{H})^2 \times \mathbb{N}^+$ such that $\alpha \neq \beta$ and $m \geq 3$, we denote by $R_3^m(\alpha, \beta)$ the result of duplicating the last column of $[(\beta,\alpha,\alpha),(\alpha,\beta,\alpha),(\alpha,\alpha,\beta)]$ until the total arity becomes m. It is straightforward to see that $\mathrm{CSP}(\{R_3^m(\alpha,\beta)\})$ is NP-hard for every m, α, β by a reduction from Positive 1-in-3-SAT. In similar fashion, we define $R_2^m(\alpha, \beta)$ as an extension of $[(\beta,\alpha),(\alpha,\beta)]$ of arity m.

Theorem 11. Strong $\mathcal{H}$-Backdoor Detection *is W[2]-hard for every idempotent algebraic tractable class $\mathcal{H}$ when the parameter is the size of the backdoor, even if the* CSP *has a single constraint.*

Proof. The proof is an FPT-reduction from Hitting Set parameterized by solution size k. Let (p, Ω, S) be an instance of Hitting Set, where Ω is the universe ($|\Omega| = n$) and $S = \{S_i \mid i = 1..s\}$ is the collection of p-sets. We assume without

loss of generality that $p \geq 3$ (if needed we pad each set with unique elements), and S_s does not intersect with any other set in S. If it is not the case, we can just add a new set of p new unique elements and increment k by one. Let $(\alpha, \beta, \alpha_1, \ldots, \alpha_s)$ be $s+2$ distinct elements from $D(\mathcal{H})$. We build an n-ary relation R, where each column is associated with a value from Ω as follows.

For every $S_i \in S$, we consider two cases. If $\{R_2^2(\alpha, \beta)\} \notin \mathcal{H}$, we add two tuples $\mathbf{t_1}, \mathbf{t_2}$ to R such that the restriction of $[\mathbf{t_1}, \mathbf{t_2}]$ to the columns corresponding to the values appearing in S_i form the relation $R_2^p(\alpha, \beta)$, and the other columns are constant with value α_i. If $\{R_2^2(\alpha, \beta)\} \in \mathcal{H}$, we add 3 tuples $\mathbf{t_1}, \mathbf{t_2}, \mathbf{t_3}$ such that the restriction of t_1, t_2, t_3 to the columns corresponding to S_i form $R_3^p(\alpha, \beta)$, and the remaining columns are constant with value α_i. Once the relation is complete, we apply it to n variables to obtain an instance of our backdoor problem. See Figure 3.1 for an example of the construction.

$$
\begin{aligned}
\Omega &= (u_1, \ldots, u_{10}) \\
S_1 &= (u_3, u_4, u_5) \\
S_2 &= (u_2, u_5, u_6) \\
S_3 &= (u_1, u_3, u_7) \\
S_4 &= (u_8, u_9, u_{10})
\end{aligned}
$$

(a) 3-Hitting Set

u_1	u_2	u_3	u_4	u_5	u_6	u_7	u_8	u_9	u_{10}
α_1	α_1	β	α	α	α_1	α_1	α_1	α_1	α_1
α_1	α_1	α	β	α	α_1	α_1	α_1	α_1	α_1
α_1	α_1	α	α	β	α_1	α_1	α_1	α_1	α_1
α_2	β	α_2	α_2	α	α	α_2	α_2	α_2	α_2
α_2	α	α_2	α_2	β	α	α_2	α_2	α_2	α_2
α_2	α	α_2	α_2	α	β	α_2	α_2	α_2	α_2
β	α_3	α	α_3	α_3	α_3	α	α_3	α_3	α_3
α	α_3	β	α_3	α_3	α_3	α	α_3	α_3	α_3
α	α_3	α	α_3	α_3	α_3	β	α_3	α_3	α_3
α_4	α_4	α_4	α_4	α_4	α_4	α_4	β	α	α
α_4	α_4	α_4	α_4	α_4	α_4	α_4	α	β	α
α_4	α_4	α_4	α_4	α_4	α_4	α_4	α	α	β

(b) Constraint C

Figure 3.1: Example of reduction from a Hitting Set instance to the problem of finding a strong backdoor into a class $\mathcal{H}$ such that $\{R_2^2(\alpha, \beta)\} \in \mathcal{H}$. The reduction produces a single constraint C.

Suppose we have a backdoor of size at most k, and suppose there exists a set S_i such that none of the corresponding variables belong to the backdoor. Then, if we assign every variable in the backdoor to α_i, the reduced constraint must belong to $\mathcal{H}$. By idempotency and algebraicity, we can further assign every remaining variable outside of S_i to the value α_i and the resulting constraint must still be in $\mathcal{H}$. The reduced constraint becomes either $R_2^p(\alpha, \beta)$ if $\{R_2^2(\alpha, \beta)\} \notin \mathcal{H}$, or $R_3^p(\alpha, \beta)$, which is not in $\mathcal{H}$ since $\mathcal{H}$ is tractable and we assume P $\neq$ NP. In both cases, this constraint does not belong to $\mathcal{H}$, and we have a contradiction. Therefore, if there is a backdoor of size at most k, we also have a hitting set of size at most k.

Conversely, suppose we have a hitting set L of size at most k. We prove that the associated set of variables form a backdoor. Observe that because S_s has an empty intersection with every other set there is no set S_i such that $L \subseteq S_i$. Therefore,

every assignment to the variables corresponding to L that does not yield an empty constraint must assign at least one variable to a value that is neither α nor β. This implies that every residual constraint (after some assignment to L) is either empty or a subrelation of a single block (i.e. pair or triple) of tuples associated with some $S_i \in S$. The latter case yields two possibilities. If $\mathcal{H}$ does not contain $\{R_2^2(\alpha, \beta)\}$, then the block i must have been reduced to a single tuple, since the two initial tuples $\mathbf{t_1}, \mathbf{t_2}$ satisfy $\mathbf{t_1}[x_j] \neq \mathbf{t_2}[x_j]$ for all x_j associated with a value in S_i. Thus, by idempotency and algebraicity the resulting constraint is in $\mathcal{H}$. Now, if $\mathcal{H}$ contains $\{R_2^2(\alpha, \beta)\}$ the resulting constraint has at most two tuples (same argument as above), which can only happen if no variable is assigned to β. If we are in this situation, the new constraint must be an extension of $R_2^2(\alpha, \beta)$ (with the extra columns with constant value α_i) and hence is in $\mathcal{H}$ by algebraicity and idempotency. In both cases, L provides a strong backdoor into $\mathcal{H}$ of same size. □

A version of this theorem still holds on Boolean CSP if we allow multiple constraints in the target instance, even if these constraints are all the same relation.

Proposition 3. *On Boolean* CSP*,* Strong $\mathcal{H}$-Backdoor Detection *is W[2]-hard for every idempotent algebraic tractable class $\mathcal{H}$ when the parameter is the size of the backdoor, even if the* CSP *has a single type of constraint.*

Proof. The proof is similar to the non-Boolean case, but more straightforward since we are allowed multiple constraints. We FPT-reduce from Hitting Set parameterized by solution size. Let (p, Ω, S) be an instance of Hitting Set, where Ω is the universe ($|\Omega| = n$) and $S = \{S_i \mid i = 1..s\}$ is the collection of p-sets. We assume $p \geq 3$, as we did for Theorem 11. We create a CSP instance with one variable per element in Ω. For each set $S_i \in S$, if $\{R_2^p(0,1)\} \in \mathcal{H}$ we add the constraint $R_3^p(0,1)$ on the variables corresponding to the values in S_i, and $R_2^p(0,1)$ otherwise. By construction, the language of the instance only contains constraints outside $\mathcal{H}$, so a strong backdoor of size k must intersect every constraint and hence corresponds to a hitting set of (p, Ω, S). Conversely, the set of variables B corresponding to a hitting set of size at most k form a backdoor: after every assignment to B, at least one variable in each constraint is assigned, so the language is either formed of relations with a most one tuple (if $\{R_2^p(0,1)\} \notin \mathcal{H}$) or a collection of extensions of $R_2^p(0,1)$ ($\{R_2^p(0,1)\} \notin \mathcal{H}$) plus relations with at most one tuple. In either case, the resulting language is in $\mathcal{H}$, which concludes the proof. □

3.3.2 Combined Parameter k+r

As shown by Theorem 10 and Theorem 11, considering independently the maximum constraint arity r and the size of the backdoor k as parameters is unlikely

to yield an FPT algorithm. In this section, we study the complexity of Strong $\mathcal{H}$-Backdoor Detection for the *combined* parameter $k+r$ and show that FPT tractability ensues for tractable composite (but not necessarily algebraic or idempotent) classes $\mathcal{H}$ that exhibit a particular locality property.

In order to design an algorithm for Strong $\mathcal{H}$-Backdoor Detection that is FPT for $k+r$, it is important to have a procedure to *check* whether a subset of variables of size at most k is a strong backdoor into $\mathcal{H}$. The natural algorithm for this task runs in time $O(mrtd^k P_{\mathcal{H}}(\Gamma))$ (where m is the number of constraints, t the maximum number of tuples and $P_{\mathcal{H}}(\Gamma)$ the complexity of the membership problem of a language Γ in $\mathcal{H}$) by checking independently each of the d^k possible assignments to B. In our case this approach is not satisfactory: since d is not a parameter, the term d^k is problematic for the prospect of an algorithm FPT in $k+r$. The next lemma presents an alternative algorithm for the "backdoor verification" problem that is exponential only in the number of constraints m. Although it may seem impractical at first sight (as m is typically much larger than k), we will show that it can be exploited for many tractable classes.

Lemma 2. *Let $\mathcal{H}$ be a composite class recognizable in time $P_{\mathcal{H}}(\Gamma)$. Let $I = (\mathcal{X}, \mathcal{D}, \mathcal{C})$ be a* CSP *instance with m constraints of arity at most r and containing at most t tuples, and $B \subseteq \mathcal{X}$. It is possible to decide whether B is a strong backdoor into $\mathcal{H}$ in time $O(mrt^2 + m^2 r(2t)^m P_{\mathcal{H}}(\Gamma))$.*

Proof. We first focus on a single constraint (S, R). Let $B_S = B \cap S$. Observe that at most t different assignments of B_S can leave R nonempty, since the subrelations of R obtained with each assignment are pairwise disjoint and their union is R. To compute these assignments in polynomial time, one can explore a search tree. Starting from a node labelled R, we pick a nonfixed variable $v \in B_S$ and for every $d \in D(v)$ such that the subrelation $R_{v=d}$ is not empty we create a child node labelled with $R_{v=d}$. Applying this rule recursively, we obtain a tree of depth at most r and with no more than t leaves, so it has at most rt nodes. The time spent at each node is $O(t)$, so computing all leaves can be done in time $O(rt^2)$. Now, suppose that for each constraint $c = (S, R) \in C$ we have computed the set ϕ_c of all locally consistent assignments to B_S and stored the resulting subrelation. For every $\phi \in \prod_{c \in C} \phi_c$ and every possible subset C' of the constraints, we check if the restriction ϕ_s of ϕ to the constraints of C' is a consistent assignment (i.e., every variable is assigned to at most one value). If ϕ_s is consistent, we temporarily remove from the instance the constraints outside C', we apply the assignment ϕ_s and we check whether the language of the resulting instance is in $\mathcal{H}$. The algorithm returns that B is a backdoor if and only if each membership test in $\mathcal{H}$ is successful. To prove the correctness of the algorithm, suppose that ψ is an assignment of B such that the resulting language is not in $\mathcal{H}$. Then, at least one subset of the constraints have degraded into non-empty subrelations. For each of these constraints, the

restricted assignment ψ_R is consistent with the others, so the algorithm must have checked membership of the resulting language in $\mathcal{H}$ and concluded that B is not a strong backdoor. Conversely, if B is a strong backdoor, every complete assignment to B yields an instance in $\mathcal{H}$. In particular, if we consider only a subset of the constraints after each assignment, the language obtained is also in $\mathcal{H}$ since $\mathcal{H}$ is composite. Thus, none of the membership tests performed by the algorithm will fail. The complexity of the algorithm is $O(mrt^2 + m^2rt^m2^mP_{\mathcal{H}}(\Gamma))$. □

As a corollary, if a CSP instance has a bounded number of constraints then we can check in polynomial time if a given subset of variables is a strong backdoor into a polynomially recognizable class $\mathcal{H}$. We now need to identify large classes of constraint languages for which the backdoor verification problem on arbitrary CSP instances reduces to the case where the number of constraints is bounded by a constant. For the next definition, recall that $|\Gamma|$ denotes the number of distinct relations in Γ.

Definition 17. Let $h \geq 1$ be a fixed integer. A class $\mathcal{H}$ is **h-Helly** if and only if for every language Γ,

$$\Gamma \in \mathcal{H} \iff \forall \Gamma_h \subseteq \Gamma \text{ such that } |\Gamma_h| \leq h, \Gamma_h \in \mathcal{H}$$

The minimum integer h such that $\mathcal{H}$ is h-Helly is the **Helly number** of $\mathcal{H}$.

This property is analogous to its counterpart for set systems. By definition, simple classes are exactly those with Helly number 1. Intuitively, if $\mathcal{H}$ has a finite Helly number we can check if the language of a CSP instance with m constraints belongs to $\mathcal{H}$ by looking at its m^h possible projections on h constraints. Our FPT algorithm for Strong $\mathcal{H}$-Backdoor Detection parameterized by $k + r$ on classes with a finite Helly number mixes this idea with Lemma 2 and a standard bounded search tree.

Theorem 12. *For every fixed composite class $\mathcal{H}$ recognizable in polynomial time, if $\mathcal{H}$ has a finite Helly number then* Strong $\mathcal{H}$-Backdoor Detection *is FPT when the parameter is $k + r$, where k is the size of the backdoor and r is the maximum constraint arity.*

Proof. Let h denote the Helly number of $\mathcal{H}$. The algorithm is a bounded search tree that proceeds as follows. Each node is labelled by a subset of variables B. The root of the tree is labelled with the empty set. At each node, we examine every possible combination of h constraints and check if B is a strong backdoor for the subset in time $O(hrt^2 + h^2r(2t)^hP_{\mathcal{H}}(\Gamma))$ (where $P_{\mathcal{H}}(\Gamma)$ is the complexity of deciding the membership of a language Γ in $\mathcal{H}$) using Lemma 2. Suppose that B is a strong backdoor for every h-subset. Then, for any possible assignment to B,

each h-subset of the constraints of the resulting instance must be in $\mathcal{H}$: otherwise, B would not be a strong backdoor for the h original constraints that generated them. Since $\mathcal{H}$ is h-Helly, we can conclude that B is a valid strong backdoor for the whole instance. Now suppose that we have found a h-subset for which B is not a strong backdoor. For every variable x in the union of the scopes of the constraints in this subset that is not already in B (there are at most rh such variables x), we create a child node labelled with $B \cup \{x\}$. At each step we are guaranteed to add at least one variable to B, so we stop creating child nodes when we reach depth k. The algorithm returns 'YES' at the first node visited that corresponds to a strong backdoor, and 'NO' if no such node is found.

If no strong backdoor of size at most k exists, it is clear that the algorithm correctly returns 'NO'. Now suppose that a strong backdoor $\mathbf{B}$, $|\mathbf{B}| \leq k$, exists. Observe that if a node is labelled with $B \subset \mathbf{B}$ and B is not a backdoor for some h-subset of constraints, then $\mathbf{B}$ contains at least one more variable within this subset. Since the algorithm creates one child per variable that can be added and the root is labelled with a subset of $\mathbf{B}$, by induction there must be a path from the root to a node labelled with $\mathbf{B}$ and the algorithm returns 'YES'. The complexity of the procedure is $O\left((rh)^k m^h (hrt^2 + h^2 r(2t)^h P_{\mathcal{H}}(\Gamma))\right) = O\left(f(k+r)m^h(ht^2 + h^2(2t)^h P_{\mathcal{H}}(\Gamma))\right)$. □

In contrast to the previous hardness results, the target tractable class is not required to be algebraic or idempotent. However, the class must have a finite Helly number, which may seem restrictive. The following results aim to identify composite classes with this particular property.

Proposition 4. *Let h be a positive integer and $\mathcal{T}$ be a set of simple classes. Then, $\mathcal{H} = \{\Gamma \mid \Gamma$ belongs to every $T_i \in \mathcal{T}$ except at most $h\}$ is a $(h+1)$-Helly composite class.*

Proof. $\mathcal{H}$ is composite since it is the union of every possible intersection of all but h classes from $\mathcal{T}$ and any class derived from simple classes through any combination of intersections and unions is composite. We write $\mathcal{T} = \{T_i \mid i \in I\}$. Let Γ be a language such that every sublanguage of size at most $h+1$ is in $\mathcal{H}$. For each $R \in \Gamma$ we define $S(R) = \{T_i \mid \{R\} \notin T_i\}$. Simple classes are 1-Helly so $\Gamma \notin T_i \Leftrightarrow (\exists R \in \Gamma$ such that $\{R\} \notin T_i) \Leftrightarrow T_i \in \cup_{R\in\Gamma} S(R)$. So $\Gamma \in \mathcal{H}$ if and only if $|\cup_{R\in\Gamma} S(R)| \leq h$. We discard from Γ every relation R such that $|S(R)| = 0$ as they have no influence on the membership of Γ in $\mathcal{H}$. If that process leaves Γ empty, then it belongs to $\mathcal{H}$. Otherwise, let s_j denote the maximum size of $\cup_{R\in\Gamma_j} S(R)$ over all size-j subsets Γ_j of Γ. Since each sublanguage Γ_j of size $j \leq h+1$ is in $\mathcal{H}$, from the argument above we have $1 \leq s_1 \leq \ldots \leq s_{h+1} \leq h$, thus there exists $j < h+1$ such that $s_j = s_{j+1}$. Let $\Gamma_j \subseteq \Gamma$ denote a set of j relations such that $|\cup_{R\in\Gamma_j} S(R)| = s_j$. Suppose there exists $R_0 \in \Gamma$ such that $S(R_0) \not\subseteq \cup_{R\in\Gamma_j} S(R)$. Then, $|\cup_{R\in\Gamma_j\cup\{R_0\}} S(R)| > s_j = s_{j+1}$,

and we get a contradiction. So $\cup_{R\in\Gamma} S(R) \subseteq \cup_{R\in\Gamma_j} S(R)$, hence $|\cup_{R\in\Gamma} S(R)| \leq h$ and Γ is in $\mathcal{H}$. Therefore, $\mathcal{H}$ is $(h+1)$-Helly. □

In the particular case where $\mathcal{T}$ is finite and $h = |\mathcal{T}| - 1$, we get the following nice corollary. Recall that a composite class is any union of simple classes.

Corollary 2. *Any union of h simple classes is h-Helly.*

Example 6. Let $\mathcal{H} = \{\Gamma \mid \Gamma \text{ is either min-closed, max-closed or 0/1/all}\}$ on the universe $D(\mathcal{H}) = \mathbb{N}$. $\mathcal{H}$ is the union of 3 well-known tractable classes of languages. By definition, min-closed and max-closed constraints are respectively the languages that admit min(.,.) and max(.,.) as polymorphisms. Likewise, 0/1/all constraints have been shown to be exactly the languages that admit as polymorphism the majority operation [94]

$$f(x,y,z) = \begin{cases} y & \text{if } y = z \\ x & \text{otherwise} \end{cases}$$

Thus, $\mathcal{H}$ is the union of 3 simple classes and hence is 3-Helly by Corollary 2. Since $\mathcal{H}$ is also recognizable in polynomial time, by Theorem 12 Strong $\mathcal{H}$-Backdoor Detection is FPT when parameterized by backdoor size and maximum arity.

In the light of these results, it would be very interesting to show a dichotomy. Is Strong $\mathcal{H}$-Backdoor Detection parameterized by $k+r$ at least W[1]-hard for every tractable algebraic class $\mathcal{H}$ that does not have a finite Helly number? While we leave most of this question unanswered, we have identified generic sufficient conditions for W[2]-hardness when r is fixed and the parameter is k.

Given two subsets D_1, D_2 of $D(\mathcal{H})$ and a bijection $\phi : D_1 \to D_2$, we denote by R_ϕ the relation $[(d, \phi(d)), d \in D_1]$. Given a language Γ, a subdomain D_1 of $D(\Gamma)$ is said to be *conservative* if every polymorphism f of Γ satisfies $f(x_1, \ldots, x_m) \in D_1$ whenever $\{x_1, \ldots, x_m\} \subseteq D_1$. For instance, $D(\Gamma')$ is conservative for every $\Gamma' \subseteq \Gamma$, and for every column of some $R \in \Gamma$ the set of values that appear in that column is conservative.

Definition 18. A class of constraint languages $\mathcal{H}$ is **value-renamable** if for every $\Gamma \in \mathcal{H}$ and $\phi : D_1 \to D_2$, where D_1 is a conservative subdomain of Γ and $D_2 \cap D(\Gamma) = \emptyset$, $\Gamma \cup \{R_\phi\}$ is in $\mathcal{H}$.

Intuitively, the addition of R_ϕ to Γ gives the extra ability to make a copy of a variable and replace the values in the domain of the copy with new unique symbols. This feature does not change anything from the algorithmic point of view, and any tractable class can be generalized to one that is value-renamable.

Example 7. Let $D(\mathcal{H}) = \mathbb{N}$. The class $\mathcal{H}_{\max}$ of all max-closed languages is not value-renamable: even the simple bijective relation $R_\phi = [(0,3),(1,2)]$ is not max-closed, and thus adding R_ϕ to a Boolean language does not always preserve membership in $\mathcal{H}_{\max}$. However, the class of languages that are max-closed with respect to at least one ordering of the domain is value-renamable since we can define a polymorphism $\max_2$ of $\Gamma \cup R_\phi$ as follows:

$$\max_2(d_1,d_2) = \begin{cases} \max(d_1,d_2) & \text{if } \{d_1,d_2\} \subseteq D(\Gamma) \\ \phi(\max(\{\phi^{-1}(d_1),\phi^{-1}(d_2)\})) & \text{if } \{d_1,d_2\} \subseteq D_2 \\ d_2 & \text{if } (d_1,d_2) \in D(\Gamma) \times D_2 \\ d_1 & \text{if } (d_1,d_2) \in D_2 \times D(\Gamma) \end{cases}$$

Definition 19. A class of constraint languages $\mathcal{H}$ is **domain-decomposable** if for each pair of languages $\Gamma_1 \in \mathcal{H}$ and $\Gamma_2 \in \mathcal{H}$, $D(\Gamma_1) \cap D(\Gamma_2) = \emptyset$ implies $\Gamma_1 \cup \Gamma_2 \in \mathcal{H}$.

Note that constraints with relations from Γ_1 cannot share a variable with a constraint from Γ_2 unless the instance is trivially unsatisfiable, so again any tractable class can be generalized to one that is domain-decomposable.

Example 8. Let $\mathcal{H}_{maj}$ be the class of all languages with a majority polymorphism and $\mathcal{H}_{msv}$ be the class of all languages with a Mal'tsev polymorphism. Let $\mathcal{H}_{mm} = \mathcal{H}_{maj} \cup \mathcal{H}_{msv}$. Both $\mathcal{H}_{maj}$ and $\mathcal{H}_{msv}$ are domain-decomposable, but not $\mathcal{H}_{mm}$ since

$$\Gamma_1 = \{[(\alpha_1,\beta_1),(\beta_1,\alpha_1),(\beta_1,\beta_1)]\} \in \mathcal{H}_{maj} \subset \mathcal{H}_{mm}$$
$$\Gamma_2 = \{[(\alpha_2,\alpha_2,\beta_2),(\alpha_2,\beta_2,\alpha_2),(\beta_2,\alpha_2,\alpha_2),(\beta_2,\beta_2,\beta_2)]\} \in \mathcal{H}_{msv} \subset \mathcal{H}_{mm}$$

but $\Gamma_1 \cup \Gamma_2 \notin \mathcal{H}_{mm}$ if $\alpha_1,\beta_1,\alpha_2,\beta_2$ are all distinct.

Theorem 13. *For* CSP *with arity at most r, if $\mathcal{H}$ is an algebraic class that is*

- *idempotent*
- *not* 1*-Helly (i.e. not simple) for constraints of arity at most r*
- *value-renamable*
- *domain-decomposable*

then Strong $\mathcal{H}$-Backdoor Detection *is W[2]-hard when the parameter is k.*

Proof. Since $\mathcal{H}$ is not 1-Helly for constraints of arity at most r, there exists a language $\Gamma_m = \{R_i \mid i \in 1..l_m\}$ (of arity $r_m \leq r$ and over a domain D_m, $|D_m| = d_m$) such that $l_m > 1$ and every sublanguage of Γ_m is in $\mathcal{H}$ but Γ_m is not. Since $\mathcal{H}$ is

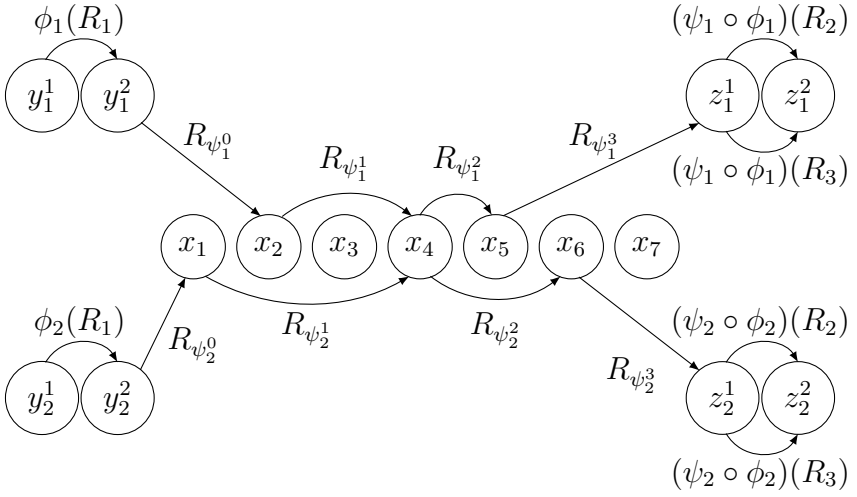

Figure 3.2: Example of the construction for $U = (u_1, \ldots, u_7)$, two sets $S_1 = (u_2, u_4, u_5)$, $S_2 = (u_1, u_4, u_6)$, $\Gamma_m = \{R_1, R_2, R_3\}$ and $r_m = 2$. Each arrow is a (binary) constraint. The upper part of the instance is constructed from S_1 and the lower part from S_2.

fixed, we shall consider that Γ_m is fixed as well and hence has constant size. We assume for simplicity of presentation that every $R \in \Gamma_m$ has arity r_m.

We perform an FPT-reduction from Hitting Set parameterized with solution size as follows. Let (p, U, S) be an instance of Hitting Set, with $S = \{S_1, \ldots, S_s\}$ and $U = \{u_1, \ldots, u_n\}$. For every $u_i \in U$, we associate a unique variable x_i. For every set $S_j = (u_{\sigma_j(1)}, \ldots, u_{\sigma_j(p)})$, we add $2r_m$ new variables $y_j^1, \ldots, y_j^{r_m}, z_j^1, \ldots, z_j^{r_m}$ and we create $p+2$ new disjoint domains $D_j^i \subset D(\mathcal{H})$, $i \in [0 \ldots p+1]$ of size d_m. Then, we pick a chain of $p+1$ bijections $\psi_j^i : D_j^i \to D_j^{i+1}$, $i \in [0 \ldots p]$ and we add a chain of constraints $R_{\psi_j^i}$ between the $p+2$ variables $(y_j^{r_m}, x_{\sigma_j(1)}, \ldots, x_{\sigma_j(p)}, z_j^1)$. Afterwards, we pick a bijection $\phi_j : D_m \to D_j^0$ and we apply $\phi_j(R_1)$ to $y_j^1, \ldots, y_j^{r_m}$. In the same fashion, if we denote by ψ_j the bijection from D_j^0 to D_j^{p+1} obtained by composition of all the ψ_j^i, we apply every constraint in $(\psi_j \circ \phi_j)(\Gamma_m \backslash \{R_1\})$ to the variables $z_j^1, \ldots, z_j^{r_m}$. The main idea behind the construction is that both $\Gamma_m \backslash \{R_1\}$ and $\{R_1\}$ are in $\mathcal{H}$ but Γ_m is not: by adding $\phi_j(R_1)$ on the variables y, $\psi_j \circ \phi_j(\Gamma_m \backslash \{R_1\})$ on the variables z and the chain of bijections $R_{\psi_j^i}$ on the x variables, we have a language that is not in $\mathcal{H}$ but assigning any value to x yields a residual language in $\mathcal{H}$ (the proof can be found below). We use this property to encode a Hitting Set instance. See Figure 3.2 for an example of the reduction.

Suppose we have a backdoor to $\mathcal{H}$ of size at most k. Then, for each set S_j, at least one variable from $(y_j^1, \ldots, y_j^{r_m}, x_{\sigma_j(1)}, \ldots, x_{\sigma_j(p)}, z_j^1, \ldots, z_j^{r_m})$ must belong to the backdoor. Suppose this is not the case. Then, the language Γ of any

reduced instance would contain the relations of $\{\phi_j(R_1), (\psi_j \circ \phi_j)(\Gamma_m \backslash \{R_1\})\}$ plus the relations $R_{\psi_j^i}$. Now, observe that

$$R_{\psi_j}(w_1, w_2) = \exists w_j^1, \ldots, w_j^p \left(R_{\psi_j^0}(w_1, w_j^1) \wedge R_{\psi_j^1}(w_j^1, w_j^2) \wedge \ldots \wedge R_{\psi_j^p}(w_j^p, w_2) \right)$$

and therefore $R_{\psi_j} \in \langle \Gamma \rangle$. Furthermore,

$$(\psi_j \circ \phi_j)(R_1)(w_1, \ldots, w_{r_m}) = \exists w'_1, \ldots, w'_{r_m} \Big(\phi_j(R_1)(w'_1, \ldots, w'_{r_m}) \wedge R_{\psi_j}(w'_1, w_1) \wedge \ldots \wedge R_{\psi_j}(w'_{r_m}, w_{r_m}) \Big)$$

which implies that $(\psi_j \circ \phi_j)(R_1) \in \langle \Gamma \rangle$ and thus $(\psi_j \circ \phi_j)(\Gamma_m) \in \langle \Gamma \rangle$. By the same reasoning we have $\Gamma_m \in \langle (\psi_j \circ \phi_j)(\Gamma_m) \cup \{R_{(\psi_j \circ \phi_j)^{-1}}\} \rangle$. By value-renamability and algebraicity we have $(\psi_j \circ \phi_j)(\Gamma_m) \cup \{R_{(\psi_j \circ \phi_j)^{-1}}\} \in \mathcal{H}$ and finally $\Gamma_m \in \mathcal{H}$, a contradiction. Therefore, a hitting set of size at most k can be constructed by including every value u_i such that x_i is in the backdoor, and if any variable from $y_j^1, \ldots, y_j^{r_m}, z_j^1, \ldots, z_j^{r_m}$ belongs to the backdoor for some j, we also include $u_{\sigma_j(1)}$.

Conversely, a hitting set forms a backdoor. After every complete assignment to the variables from the hitting set, the set of constraints associated with any set S_j can be partitioned into sublanguages whose domains have an empty intersection (see Figure 3.2). The sublanguages are either:

- $\phi_j(R_1)$ together with some constraints $R_{\psi_j^i}$ and a residual unary constraint with a single tuple. This language is in $\mathcal{H}$ by value-renamability and idempotency.
- $(\psi_j \circ \phi_j)(\Gamma_m \backslash \{R_1\})$ together with some constraints $R_{\psi_j^i}$ and a residual unary constraint with a single tuple. This case is symmetric.
- A (possibly empty) chain of constraints $R_{\psi_j^i}$ plus unary constraints with a single tuple, which is again in $\mathcal{H}$ since $\mathcal{H}$ is idempotent, value-renamable and contains the language $\emptyset$.

Furthermore, the sublanguages associated with different sets S_j also have an empty domain intersection. Since $\mathcal{H}$ is domain-decomposable, the resulting language is in $\mathcal{H}$. □

This result does not contradict Theorem 12, since any class $\mathcal{H}$ that is domain-decomposable, value-renamable and not simple cannot have a finite Helly number (part of the proof of Theorem 13 amounts to showing that one can build in polynomial time arbitrarily large languages Γ such that every sublanguage is in $\mathcal{H}$ but

Γ is not). The strength of Theorem 13 lies in its generality, as it applies almost immediately to many known tractable classes without having to rely on class-specific constructions (see Figure 3.3). It is also quite easy to use as value-renamability, domain-decomposability and non-simplicity are straightforward properties of any large enough idempotent algebraic tractable class.

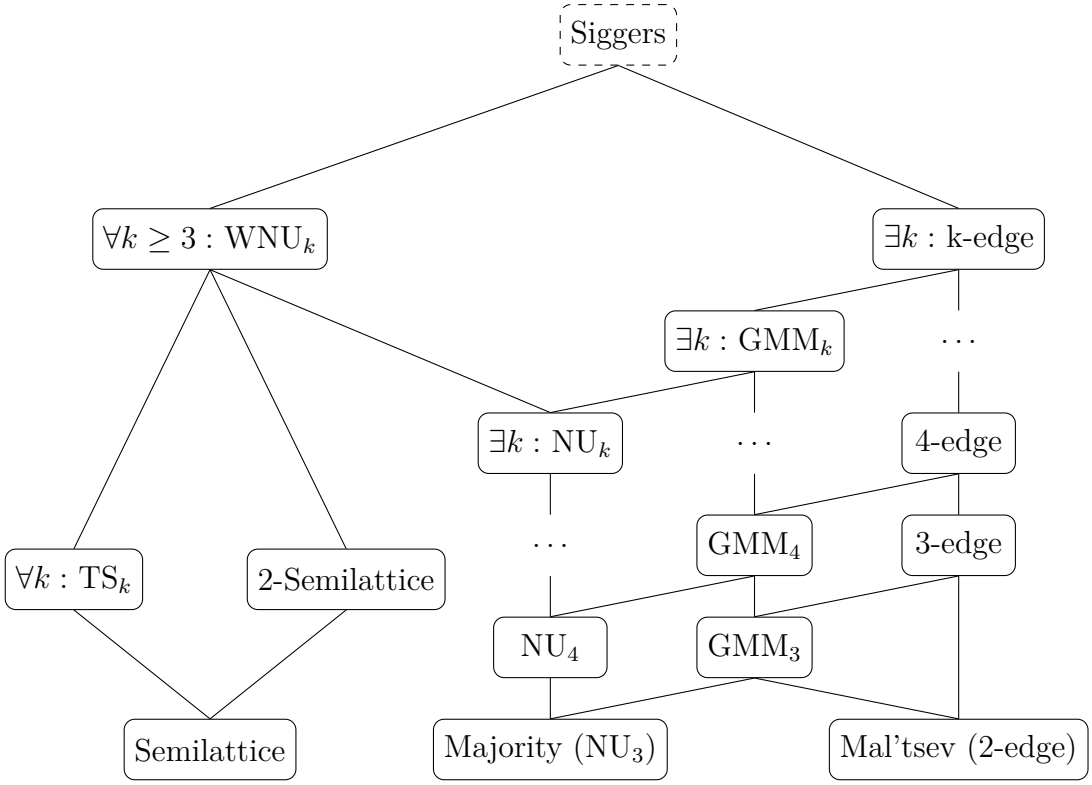

Figure 3.3: Hasse diagram of classical tractable classes in the idempotent case, ordered by inclusion. Theorem 13 applies to each of these classes (short proofs can be found in Appendix A). The subscript after the name of a polymorphism denotes its arity.

3.3.3 Constant-closed Classes

The results of Section 3.3.1 and 3.3.2 are sufficient to classify the complexity of Strong $\mathcal{H}$-Backdoor Detection for most language-based tractable classes $\mathcal{H}$ that can be found in the literature and could reasonably be used as targets for backdoor detection. However, every hardness result so far has been proved under the assumption that $\mathcal{H}$ is idempotent, which covers all natural polynomially

recognizable tractable classes except one: constant-closed languages.

A language Γ is constant-closed if and only if it admits a unary polymorphism that always map to the same value α (or, equivalently, if every nonempty relation $R \in \Gamma$ contains the tuple $(\alpha, \ldots, \alpha)$). Constant-closed languages are trivially tractable (every instance is satisfiable - just assign every variable to α) and are far too restricted to be interesting in practice, but form a well-known tractable class since two of the tractable cases of Schaefer's Dichotomy for generalised SAT correspond to constant-closed languages [128]. Note that on most instances minimum-size strong backdoor into α-closed languages for a fixed α is very likely to simply be the set of all variables, given the very small size of the tractable class and the non-monotonicity that is inherent to its associated strong backdoor detection problem (even if a relation is α-closed, branching on an additional variable in the scope is highly likely to create subrelations that are *not* α-closed). Still, we shall explore the complexity of backdoor detection for these classes for completeness. Theorem 12 tells us that detecting strong backdoors into constant-closed classes in FPT when the parameter is $k + r$, but we will see in this section that this problem is actually polynomially solvable.

In the context of SAT, finding a minimum-size strong backdoor into 0-closed languages (traditionally called *0-valid*) has been shown in [72] to be a polynomial-time problem. The case of 1-closed languages is symmetric. The three-line proof states that the minimum strong backdoor into 0-closed languages is exactly the set of variables that appear in clauses that contain only positive literals. Although the result itself is correct, the proof provided is not satisfying as it does not take into account the non-monotonicity discussed above. Let us consider an example consisting of two clauses $C_1 = (v_1 \vee v_4 \vee v_5)$ and $C_2 = (\overline{v_1} \vee v_2 \vee v_3)$. It is clear that $\{v_1, v_4, v_5\}$ must belong to every strong backdoor into 0-valid languages, since this clause is not 0-valid and branching on value 0 for any variable leaves a clause that also contains only positive literals. However, these variables do not form a strong backdoor, because assigning v_1 to the value 1 turns C_2 into a clause whose literals are all positive! A solution to this problem is simply to repeat the operation until a fixed point is reached: initialize the candidate backdoor set $\mathbf{B}$ to $\emptyset$ and add all variables appearing in any clause to $\mathbf{B}$. Then, *remove* these variables from the remaining clauses and repeat the operation (without reinitialising $\mathbf{B}$) until the rule does not find any variable to include in $\mathbf{B}$. The correctness is straightforward.

The contribution of this section is an extension of this algorithm to CSP over arbitrary relations. Let α be a fixed domain element, and let $\mathcal{H}_\alpha$ denote the class of all α-closed languages over a given domain $D(\mathcal{H})$. Given a tuple $\mathbf{t}$, we denote by $\#_\alpha \mathbf{t}$ the number of components in $\mathbf{t}$ that are equal to α. The following lemma describes a simple trick that will be exploited by our algorithm.

Lemma 3. *Let $I = (\mathcal{X}, \mathcal{D}, \mathcal{C})$ be a* CSP *instance, $C = (S, R) \in \mathcal{C}$, $\mathbf{t}_\alpha \in R$ be a tuple of R such that $\#_\alpha \mathbf{t}_\alpha$ is maximum and B be any strong backdoor into $\mathcal{H}_\alpha$. Then,*

$$\{x \in S \mid \mathbf{t}_\alpha[x] \neq \alpha\} \subseteq B$$

Proof. For the sake of contradiction, assume that there exists $x \in S$ such that $\mathbf{t}_\alpha[x] \neq \alpha$ and $x \notin B$. Let ϕ denote a partial assignment to B that maps every value x to $\mathbf{t}_\alpha[x]$. Let I' be the instance obtained from I by application of ϕ and let (S', R') be the constraint in I' that originates from C. Since B is a backdoor into $\mathcal{H}_\alpha$, R' is α-closed and contains the tuple $(\alpha, \ldots, \alpha)$. It follows that R contains a tuple $\mathbf{t}^*$ such that $\mathbf{t}^*[y] = \mathbf{t}_\alpha[y]$ whenever $y \in B$, and $\mathbf{t}^*[y] = \alpha$ otherwise. In particular, $\mathbf{t}^*[y] = \alpha$ whenever $\mathbf{t}_\alpha[y] = \alpha$ and $\mathbf{t}^*[x] = \alpha$. Since $\mathbf{t}_\alpha[x] \neq \alpha$, we have $\#_\alpha \mathbf{t}^* > \#_\alpha \mathbf{t}_\alpha$, which contradicts the definition of $\mathbf{t}_\alpha$. □

Given an instance $I = (\mathcal{X}, \mathcal{D}, \mathcal{C})$ and a set of variables $X' \subseteq X$, we call *(X',α)-obstruction* an assignment ϕ to X' such that the residual instance after application of ϕ is not over an α-closed language.

Theorem 14. *For a fixed domain element α,* Strong $\mathcal{H}_\alpha$-Backdoor Detection *can be solved in polynomial time.*

Proof. Let $I = (\mathcal{X}, \mathcal{D}, \mathcal{C})$ be a CSP instance over $D(\mathcal{H})$. We initialize a set $B = \emptyset$. We repeat the following operations until the algorithm terminates. Find a (B, α)-obstruction ϕ in polynomial time using Lemma 2 and the fact that $\mathcal{H}_\alpha$ is simple. If no obstruction can be found, B is a strong backdoor into $\mathcal{H}_\alpha$ and the algorithm returns `TRUE`. Otherwise, let I_ϕ denote the residual instance obtained after application of ϕ to I. Since ϕ is an obstruction, the language of I_ϕ is not α-closed and there exists a constraint $C = (S, R)$ in I_ϕ such that R does not contain the tuple $(\alpha, \ldots, \alpha)$. Find a tuple $\mathbf{t}_\alpha \in R$ such that $\#_\alpha \mathbf{t}_\alpha$ is maximum, and add to B the set $\{x \in S \mid \mathbf{t}_\alpha[x] \neq \alpha\}$. This set cannot be empty since R is not α-closed. If after the addition of new variables $|B| > k$, the algorithm returns `FALSE`; otherwise it starts looking for a new obstruction.

We now prove the correctness of this algorithm. Let $\mathbf{B}$ denote a minimum-size strong backdoor into $\mathcal{H}_\alpha$. We will prove by induction that at every iteration of the algorithm, $B \subseteq \mathbf{B}$. At the first iteration, $B = \emptyset \subseteq \mathbf{B}$. Now, suppose that at iteration i we also have $B \subseteq \mathbf{B}$. Then, if a (B, α)-obstruction ϕ exists $\mathbf{B} \backslash B$ is a strong backdoor of I_ϕ into $\mathcal{H}_\alpha$; therefore by Lemma 3 at iteration $i + 1$ we also have $B \subseteq \mathbf{B}$, which completes the induction. If the algorithm returns `TRUE` B is a strong backdoor of size at most k, and if it returns `FALSE` then $|\mathbf{B}| \geq |B| > k$ and I has no strong backdoor into $\mathcal{H}_\alpha$ of size at most k. □

In the case where $\mathcal{H}$ is a finite union of constant-closed classes, Strong $\mathcal{H}$-Backdoor Detection is still FPT when the parameter is $k+r$ by Theorem 12. In SAT it is known that finding a strong backdoor into the union of 0-valid and 1-valid formulas is W[2]-hard when the parameter is k [69], but this hardness result does not translate well to CSP because the extensional representation of a clause has exponentially many tuples. This leads to the following question, which we will leave open.

Question 1. *Let $\mathcal{U}$ be an infinite set, c be a fixed integer, $\alpha_1, \ldots, \alpha_c \in \mathcal{U}$ and $\mathcal{H} = \mathcal{H}_{\alpha_1} \cup \ldots \cup \mathcal{H}_{\alpha_c}$ be the set of all languages over $\mathcal{U}$ that are α_i-closed for at least one index i. What is the complexity of* Strong $\mathcal{H}$-Backdoor Detection *parameterized by k?*

3.4 Partition Backdoors

We have seen in Section 3.3.1 and Section 3.3.2 that strong backdoors are in general quite difficult to detect, even with the backdoor size as a parameter. The FPT cases identified in Theorem 12 do not include the most general tractable classes, and the complexity of the algorithm depends heavily on the Helly number of the class. In this section we introduce a relaxation of strong backdoors, *partition backdoors*, which are tailored to scale well with large target tractable classes.

The concept of partition backdoors is based on two main observations:

(i) As seen in Theorem 11, backdoor detection is significantly harder than a simple hitting set on the constraints with unwelcome properties because breaking optimally a given constraint into subrelations that belong to the target class $\mathcal{H}$ is a difficult task;

(ii) Even CSP instances that have numerous constraints are often over a language of limited size; i.e. a given relation can be used by many constraints (typically, this happens for the disequality relation). This suggests that the number of distinct relations used by the instance could be a reasonable parameter.

Partition backdoors solve (*i*) by simply ignoring what happens inside relations; either a relation is satifying as it is or it has to be (almost) entirely removed from the instance by branching on variables. By reasoning in terms of relations only, partition backdoors can then exploit the observation (*ii*). We only define partition backdoors for conservative tractable classes even though the idea can be extended to idempotent classes (simply replace "a vertex cover" by "the vertex set" in the definition below).

Definition 20. Let $I = (\mathcal{X}, \mathcal{D}, \mathcal{C})$ be a CSP instance and $\mathcal{H}$ be a fixed algebraic conservative tractable class. A **partition backdoor** of I into $\mathcal{H}$ is a subset of variables $B \subseteq \mathcal{X}$ such that there exists a partition $\mathcal{C}_1, \mathcal{C}_2$ of $\mathcal{C}$ satisfying

- $\mathcal{R}(\mathcal{C}_2) \in \mathcal{H}$
- B is a vertex cover of the primal graph of $\mathcal{C}_1$

Proposition 5. *Every partition backdoor is a strong backdoor.*

Proof. Let $I = (\mathcal{X}, \mathcal{D}, \mathcal{C})$ be a CSP instance and B be a partition backdoor into a conservative class $\mathcal{H}$. Let $\mathcal{C}_1, \mathcal{C}_2$ be a partition of the constraints satisfying the requirements of the theorem. Because B is a vertex cover of the primal graph of $\mathcal{C}_1$, the scope every $C \in \mathcal{C}_1$ contains only variables in B except at most one. Therefore, after any complete assignment to B the constraints of $\mathcal{C}_1$ are reduced to at most unary constraints. The residual instance can be expressed using only constraints in $\mathcal{C}_2$ plus unary relations and thus its language belongs to $\mathcal{H}$ by algebraicity. □

Intuitively, a minimum-size partition backdoor should be a decent approximation of the minimum-size strong backdoor if most constraints are of low arity. However, in the worst case it is clear that a minimum-size partition backdoor can be arbitrarily larger than a minimum-size strong backdoor (see e.g. [69] for an exemple of such a construction). As a minor remark, the requirement of algebraicity is purely technical and can be omitted with a slight change in the definition of a variable assignment. We can now define the associated detection problem.

Strong $\mathcal{H}$-Partition Backdoor Detection
Input: A CSP instance $I = (\mathcal{X}, \mathcal{D}, \mathcal{C})$, an integer k
Question: Does I have a partition backdoor into $\mathcal{H}$ of size at most k?

Proposition 6. *Let $\mathcal{H}$ be a conservative algebraic class recognizable in polynomial time.* Strong $\mathcal{H}$-Partition Backdoor Detection *is FPT for the parameter $k + l$, where l is the size of the language of the input* CSP *instance.*

Proof. Let $I = (\mathcal{X}, \mathcal{D}, \mathcal{C})$ be a CSP instance and $\mathcal{H}$ be a fixed algebraic conservative class. The main observation here is that we only have to examine partitions $\mathcal{C}_1, \mathcal{C}_2$ of $\mathcal{C}$ such that $\mathcal{R}(\mathcal{C}_1) \cap \mathcal{R}(\mathcal{C}_2) = \emptyset$ since otherwise we could set $\mathcal{C}'_1 = \mathcal{C}_1 \backslash \{C \in \mathcal{C}_1 \mid \mathcal{R}(C) \in \mathcal{R}(\mathcal{C}_2)\}$, $\mathcal{C}'_2 = \mathcal{C}_2 \cup \{C \in \mathcal{C}_1 \mid \mathcal{R}(C) \in \mathcal{R}(\mathcal{C}_2)\}$ and obtain a partition $\mathcal{C}'_1, \mathcal{C}'_2$ of $\mathcal{C}$ such that (i) the vertex cover of $\mathcal{C}'_1$ is no larger than that of C_1, (ii) $\mathcal{R}(\mathcal{C}'_2) \in \mathcal{H}$ if and only if $\mathcal{R}(\mathcal{C}_2) \in \mathcal{H}$ and (iii) $\mathcal{R}(\mathcal{C}'_1) \cap \mathcal{R}(\mathcal{C}'_2) = \emptyset$. There are at most 2^l such partitions, the membership of $\mathcal{R}(\mathcal{C}_2)$ in $\mathcal{H}$ can be checked in polynomial time using the recognition algorithm and computing a size-k vertex cover is FPT in the parameter k, so the claim follows. □

Proposition 6 is at the core of the partition backdoors approach. This proposition clearly does not hold for general strong backdoors because it would contradict Theorem 11. Note also that the algorithm scales much better with k than the algorithm of Theorem 12 as Vertex Cover algorithms are very efficient and the exponential factor is completely independent of the target tractable class. Finally, we can observe that the reduction used to prove Theorem 13 builds an instance in which the minimum-size strong backdoor is also a partition backdoor. Therefore, we can transpose this result to partition backdoors without any change (aside from dropping the requirement for idempotency since we assume that $\mathcal{H}$ is conservative).

Theorem 15. *For* CSP *with arity at most r, if $\mathcal{H}$ is a conservative algebraic class that is*

- *not* 1*-Helly (i.e. not simple) for constraints of arity at most r*
- *value-renamable*
- *domain-decomposable*

then Strong $\mathcal{H}$-Partition Backdoor Detection *is W[2]-hard for the parameter k.*

3.5 Preliminary Experiments

Our results are so far purely theoretical, and going from theory to practice is generally a giant step where clever heuristics, sophisticated data structures and preprocessing tricks can overshadow even a terrible worst-case complexity estimate. We see the FPT algorithms of previous sections as ethereal bricks that may be used to design efficient methods for solving CSP, but there is still a long way to go before backdoors (or related methods) can be explicitly used in a competitive CSP solver.

Nevertheless, we performed preliminary experiments with partition backdoors which already give some insight on the obstacles that will have to be overcome in the future. We implemented a simple partition backdoor detector for the class of all languages that have a conservative majority polymorphism. The only notable difference with the algorithm described in the proof of Proposition 6 is the replacement of the generate-and-test algorithm for enumerating the 2^l possible sublanguages with a branch-and-bound paired with a nogood structure that stores the minimal sublanguages that admit no conservative majority polymorphism. These simple improvements alone sped up the procedure by orders of magnitude. We performed the experiment on 191 families of instances of the XCSP repository,

with an added singleton arc-consistency preprocessing to reduce the size of the constraints. The maximum backdoor size was fixed at 40 variables.

Of these 191 families of instances, 135 were over languages including huge relations (global constraints, or simply extensional relations with thousands of tuples) that could not be handled by the detection algorithm for conservative majority polymorphisms, which is very memory-intensive. 16 families were generally solved by the SAC preprocessing (either a domain is emptied, or every variable is assigned). This left 40 families of instances on which we could perform the experiment properly. None had consistently small partition backdoors into conservative majority polymorphisms although we had positive results on several instances (summarized in Figure 3.4).

Instance	#Var	#Act	#Fix	#Rel	time (s)
`driverlogw-01c`	71	53	22	2/14	0.40
`primes-10-20-2-1`	100	79	3	2/20	439.92
`primes-10-40-2-1`	100	56	6	3/40	344.92
`primes-10-40-3-1`	100	43	3	3/40	7.80
`primes-10-60-2-1`	100	28	0	0/59	0.01
`primes-10-60-2-3`	100	14	0	0/59	0.09
`primes-10-60-3-1`	100	17	0	0/59	0.00
`primes-10-80-2-1`	100	11	0	0/79	0.00
`primes-10-80-2-3`	100	4	0	0/79	0.08
`primes-10-80-3-1`	100	8	0	0/79	0.01

Figure 3.4: Partition backdoors found. #Var is the number of variables, #Act is the number of non-assigned variables after SAC preprocessing, #Fix is the size of the backdoor and #Rel is the ratio of relations in $\mathcal{C}_1$.

Our conclusions for this preliminary experiment can be summarized as follows:

- The main limitation of the method seems to be its inability to handle global constraints, which are very common in repositories of CSP instances such as XCSP or CSPLib. A possible workaround would be to precompute databases (using possibly advanced knowledge representation systems) of polymorphisms for classical global constraints. We could also simply skip those instances and focus on the more favourable cases. This is less ambitious, but having performance gains only on some specific families of instances could be considered a success if the induced overhead is minimal.

- With our improvements (branch-and-bound and nogood recording) finding an optimal partition of the constraints never took more than a few dozen iterations, very far from the 2^l upper bound. This is very encouraging and

confirms the intuition that partition backdoors should be closer to practicality than general strong backdoors.

- Vertex covers are quite easy to compute even with an elementary implementation, and the high runtime observed on some instances was always a product of the inefficiency of the detection algorithm for conservative majority polymorphisms. This means that, surprisingly, the only polynomial-time step of the procedure is the costliest by far.

We did not perform experiments with other conservative tractable classes because of the general lack of efficient recognition algorithms, although for most tractable classes there is no known hardness result either.

3.6 Conclusion and Future Research

In this chapter we have proved quite general theorems that yield a near-complete classification of the complexity of strong backdoor detection when the target tractable class is language-based. If the parameter is backdoor size, strong backdoor detection is almost always W[2]-hard. When we augment the parameter to include the maximum arity as well, the complexity landscape becomes more diverse; in this setting we have introduced a mesure of heterogeneity of a class of constraint languages in the notion of *Helly number*, whose boundedness seems to correlate strongly with the existence of an FPT algorithm for algebraic idempotent classes. The obvious open question here is whether a bounded Helly number characterizes exactly the FPT cases. We did not investigate much further since we consider that our original goal of classifying the complexity of strong backdoors detection for relevant tractable classes is achieved, and pushing further the analysis is likely to raise technical problems which are of little interest to the CSP community. Finally, observing that strong backdoor detection becomes increasingly difficult as the heterogeneity of the tractable class increases (as measured by its Helly number) we have introduced *partition backdoors*, a type of strong backdoors that allow a fair approximation of the minimum-size strong backdoor for low-arity CSPs. Partition backdoors are easily computable in instances whose language has few relations, and work well even with very heterogeneous classes provided we have a fast enough recognition algorithm at our disposal. Finally, we have performed preliminary experiments on partition backdoors. The results were mixed but nonetheless provided valuable insight on various aspects of the method.

Overall, strong backdoor detection shows potential but still requires substantial improvements in order to be competitive. In-depth experiments with both strong backdoors and partition backdoors for various target tractable classes are needed to understand fully the strengths and weaknesses of the approach. In turn,

these experiments could be used to design solving hybrid solving methods, such as backdoor learning during search, backdoor-based variable-branching heuristics or an adaptive preprocessing technique that can analyze the instance to decide which target tractable class is the most relevant and choose the most reasonable type of backdoor it should attempt to compute.

Another important avenue of research raised by our experiments is designing efficient polynomial-time recognition algorithms for classical tractable classes. This topic will be investigated in depth in the next chapter.

Chapter 4

Meta-Problems in Conservative Constraint Satisfaction

The complexity of constraint satisfaction problems has enjoyed considerable scientific attention in the past decade, motivated by their practical usefulness, their generality, and the deep theoretical questions they relate to. A sizable part of this scientific effort has been directed towards the complexity of fixed-language restrictions and has produced a complex hierarchy of language-based tractable classes, generally defined by the existence of polymorphisms satisfying a certain set of identities. Many of these algebraic conditions characterize deep properties of constraint languages, such as definability in a certain logic [108][48] or solvability by a given (type of) algorithm [49][93][90].

A question that has been largely overlooked in the existing literature is the complexity of testing for these properties. There are many reasons for this lack of interest. The first that comes to mind is that these *meta-problems* do not fit well in the non-uniform CSP theory where languages are usually assumed to be fixed (thus only decidability matters). Naturally, the strong divide between the practical and theoretical CSP communities also plays an important role since meta-problems are inbetween, motivated by mostly practical considerations but requiring a secure grasp of the algebraic CSP theory in order to be properly understood. Another possible reason is that these meta-problems certainly seem difficult to solve, since polymorphism-based properties are by essence statements about the expressive power of constraint languages; for instance, a constraint language has a majority polymorphism if and only if every expressible relation is 2-decomposable [93] and a Mal'tsev polymorphism if and only if every expressible relation is rectangular [60].

The complexity of meta-problems, or almost equivalently the complexity of detecting polymorphisms with desirable properties in constraint languages, is a question that is completely central to the present thesis. Sophisticated algorithms for restrictions of CSP defined by elusive algebraic properties certainly define

facets of tractability that have yet to be harnessed, and having a polynomial-time meta-problem is a primordial quality of a tractable class that enables the whole backdoor approach that was the subject of the previous chapter. Beyond the obvious application as preprocessing, connections can be made between meta-problems and the classical complexity of CSP through the concept of *uniform algorithms*, which are polynomial-time algorithms associated with *sets of constraint languages* rather than just constraint languages and often allow more efficient solving (this aspect will be detailed in Section 4.3). The conception of efficient algorithms for meta-problems is also a first step towards automated complexity proofs, albeit this idea has hitherto only been used within the CSP world (to discover the smallest digraph whose associated CSP has unknown complexity [9][73] and to prove the NP-completeness of deciding if a CSP instance is max-closed modulo a permutation of the domains [78]). Last, and unfortunately least, meta-problems often have unusual properties that call for novel algorithmic techniques and hence are interesting in their own right, from a purely academic point of view.

Given a language Γ, the *conservative* CSP over Γ (c-CSP(Γ) for short) is a common variant of CSP where all unary relations are accepted in addition to those of Γ. These additional unary relations correspond to a widely used feature of practical constraint solving, arbitrary restrictions of variable domains. The complexity of c-CSP is very well understood, as a dichotomy was established and a nice polymorphism-based characterization of all conservatively tractable languages is known [21]. However, these results were obtained thirteen years ago and the complexity of testing for the tractability criterion has remained unknown since then. Moreover, conservatively tractable classes of languages are exactly those for which our results on partition backdoors apply (see Section 3.4, Definition 20), which gives us additional incentive to focus on meta-problems of this kind.

In this chapter we will develop a rich arsenal of algorithmic tools that allows us to draw a near-complete complexity classification of conservative meta-problems. Our contribution is threefold:

- We establish that determining if a constraint language is conservatively tractable is polynomial-time. In addition, if the language is proven conservatively tractable our algorithm also outputs the *coloured graph* of the language, which contains valuable information on its algebraic structure. This is undeniably the strongest result of the chapter, and possibly of the whole thesis.

- As a byproduct, we exhibit a general connection between the complexity of the meta-problem and the existence of a *semiuniform algorithm* on classes of conservatively tractable languages defined by certain algebraic identities known as *linear strong Mal'tsev conditions*. An immediate corollary is a

polynomial-time algorithm that detects conservative k-edge polymorphisms for a fixed k, which complements nicely the known results on conservative meta-problems.

- Observing that the polynomial-time algorithms of the first two contributions are impractical, we present highly efficient specialized algorithms for detecting conservative Mal'tsev and conservative majority polymorphisms. These new algorithms avoid the heavy artillery of the first two contributions by using an orthogonal, finer-grained approach to meta-problems that is interesting in its own right.

4.1 Meta-Problems

For a fixed class $\mathcal{H}$ of constraint languages, the *meta-problem* (or *metaquestion* [36]) for $\mathcal{H}$ is the problem of deciding if an input constraint language Γ belongs to $\mathcal{H}$. We shall assume that Γ is always given in extension, as we did in the previous chapter, but we impose no additional restriction and in particular we shall avoid the (quite common [20][21]) simplifying assumptions that the domain size or the maximum arity of Γ are bounded. Naturally, we will focus on meta-problems for classes that are tractable and definable by algebraic means. There exists a fragmented literature on such meta-problems, although a very recent paper has started to develop a more systematic approach [36].

The complexity of the meta-problem for a class defined by the existence of polymorphisms satisfying certain identities depends crucially on whether the said identities entail idempotency or not. Informally, the idea is that looking for non-idempotent polymorphisms is hard because it has connections with the coNP-complete problem of deciding if a constraint language Γ is a core [84]: if a polymorphism f of Γ is not idempotent, then $h(x) = f(x, \ldots, x)$ can possibly be a non-surjective endomorphism and one exists if and only if Γ is not a core. This reasoning has been (indirectly) used to show that deciding if a language Γ has bounded width is NP-complete [36], despite being polynomial-time if in addition the polymorphisms witnessing bounded width are required to be idempotent [6]. Similar results state that deciding if Γ has width 1 is NP-hard [108] (membership in NP is still open) and the meta-problem for the class of all languages with a Siggers polymorphism is NP-complete [25]. An example of idempotent family of polymorphism that is NP-complete to detect is that of semilattice polymorphisms [78][36]. In this case the computational hardness originates from the associativity of the polymorphisms instead of non-idempotency, and the result still holds in the conservative case and for semigroup polymorphisms [36]. Incidentally, because semilattice polymorphisms are idempotent witnesses to bounded width

this also highlights the fact that more general classes do not necessarily give rise to harder meta-problems.

The polynomial-time algorithm sketched in [6] to decide if a language has bounded width witnessed by idempotent polymorphisms can also be used to detect k-near-unanimity polymorphisms for a fixed k [36]. If k is not fixed the problem is decidable [5] but of unknown complexity. The applicability of this algorithmic technique has been characterized in [36], and we will give an in-depth treatment of the topic in Section 4.3. In contrast, deciding whether a language admits the median polymorphism (a particular case of majority polymorphism) with respect to an unknown ordering of the domain is NP-complete [78].

These results may seem numerous but are in fact quite scattered. Detecting a Siggers polymorphism is NP-complete, but what about an idempotent or conservative Siggers polymorphism? The same question can be asked for totally symmetric polymorphisms of all arities, which characterize width 1. For Mal'tsev polymorphisms, the meta-problem is polynomial-time on directed graphs [35] but for arbitrary languages nothing is known beyond membership in NP. More generally, the complexity of detecting k-edge polymorphisms is unknown even if k is fixed.

4.2 Conservative Constraint Satisfaction

In this section we will present the criterion for conservative tractability, as obtained in [21], along with a very useful theorem often referred to as the *Three Operations Theorem*. This criterion is naturally equivalent to the existence of a conservative Siggers polymorphism since it does not contradict the algebraic dichotomy conjecture, but its formulation will simplify greatly our analysis of the meta-problem.

In general, if Γ is a conservative language and there exists $\{a, b\} \subseteq \mathcal{D}$ such that every polymorphism of Γ is a projection when restricted to $\{a, b\}$ then $R \in \langle\Gamma\rangle$, where

$$R = \begin{pmatrix} a & b & b \\ b & a & b \\ b & b & a \end{pmatrix}$$

It follows that CSP(Γ) is NP-complete as CSP($\{R\}$) is equivalent to Positive 1-in-3-SAT. The Dichotomy Theorem for conservative CSP states that the converse is true: if for every $B = \{a, b\} \subseteq \mathcal{D}$ there exists a polymorphism f such that $f_{|B}$ is *not* a projection, then c-CSP(Γ) is polynomial-time. By Post's lattice [120], the polymorphism f can be chosen such that $f_{|B}$ is either a majority operation, a minority operation or a semilattice.

Theorem 16 ([21]). *Let Γ be a fixed constraint language over a domain $\mathcal{D}$. If for every $B = \{a, b\} \subseteq \mathcal{D}$ there exists a conservative polymorphism f such that $f_{|B}$ is either a majority operation, a minority operation or a semilattice then* c-CSP(Γ) *is in P. Otherwise,* c-CSP(Γ) *is NP-complete.*

This theorem provides a way to determine the complexity of c-CSP(Γ), since we can enumerate all ternary operations over $\mathcal{D}$ and list those that are polymorphisms of Γ. However, this naive procedure is super-exponential in time if the domain is part of the input and hence does not settle the complexity of the meta-problem.

Three different proofs of Theorem 16 have been published [21][4][22], and two of them rely heavily on a construction called the *coloured graph* of Γ and denoted by G_Γ. The definition of G_Γ is as follows. The vertex set of G_Γ is $\mathcal{D}$, and there is an edge between any two vertices. Each edge (a, b) is labelled with a colour following these rules:

- If there exists a polymorphism f such that $f_{|\{a,b\}}$ is a semilattice, then (a, b) is red;
- If there exists a polymorphism f such that $f_{|\{a,b\}}$ is a majority operation and (a, b) is not red, then (a, b) is yellow;
- If there exists a polymorphism f such that $f_{|\{a,b\}}$ is a minority operation and (a, b) is neither red nor yellow, then (a, b) is blue.

Additionally, red edges are directed: we have $(a \to b)$ if there exists f such that $f(a, b) = f(b, a) = b$. It is possible to have $(a \leftrightarrow b)$. By Theorem 16, G_Γ is entirely coloured if and only if c-CSP(Γ) is tractable. The next theorem, from ([21], Proposition 3.1), shows that the tractability of c-CSP(Γ) is always witnessed by three specific polymorphisms (instead of $O(d^2)$ in the original formulation).

Theorem 17 (The Three Operations Theorem [21]). *Let Γ be a language such that* c-CSP(Γ) *is tractable. There exist three conservative polymorphisms $f^*(x, y)$, $g^*(x, y, z)$ and $h^*(x, y, z)$ such that for every two-element set $B \subseteq \mathcal{D}$:*

- *$f^*_{|B}$ is a semilattice operation if B is red and $f^*(x, y) = x$ otherwise;*
- *$g^*_{|B}$ is a majority operation if B is yellow, $g^*_{|B}(x, y, z) = x$ if B is blue and $g^*_{|B}(x, y, z) = f^*(f^*(x, y), z)$ if B is red;*
- *$h^*_{|B}$ is a minority operation if B is blue, $h^*_{|B}(x, y, z) = x$ if B is yellow, and $h^*_{|B}(x, y, z) = f^*(f^*(x, y), z)$ if B is red.*

The original theorem also proves the existence of other polymorphisms, but we will only use f^*, g^* and h^* in our proofs.

4.3 Tools

A healthy place to start our quest for tractable conservative meta-problems is a review of the existing techniques that have been used to solve meta-problems in polynomial time for arbitrary constraint languages. This will not take long, for there is only one such technique.

The complexity of detecting polymorphisms can be expected to depend crucially on the nature of the desired identities. We have seen that looking for an associative polymorphism is often hard, so it is sensible to avoid identities involving compositions. In the same vein, a bound on the arity and number of polymorphisms is desirable as it ensures membership in NP. A set of identities with these favourable properties is usually called a *linear strong Mal'tsev condition*. Given that universal algebra is not the main topic of the present thesis, we will use a simplified exposition similar to that found in [36]. A *linear identity* is an expression of the form

$$f(x_1, \ldots, x_{a_f}) \approx g(y_1, \ldots, y_{a_g})$$

or

$$f(x_1, \ldots, x_{a_f}) \approx y$$

where f, g are operation symbols and $x_1, \ldots, x_{a_f}, y_1, \ldots, y_{a_g}, y$ are variables. It is *satisfied* by two interpretations for f and g on a domain $\mathcal{D}$ if the equality holds for any assignment to the variables. A *strong linear Mal'tsev condition* $\mathcal{M}$ is a finite set of linear identities. We say that a set of operations satisfy $\mathcal{M}$ if they satisfy every identity in $\mathcal{M}$. A strong linear Mal'tsev condition is said to be *idempotent* if it entails $f_i(x, \ldots, x) \approx x$ for all operation symbols f_i (that is, all sets of operations satisfying $\mathcal{M}$ are idempotent). Because linear strong Mal'tsev conditions are finite, the number of operation symbols and their maximum arity are constant.

Example 9. The set of identities

$$f(x, x, y) \approx x$$
$$f(x, y, x) \approx x$$
$$f(y, x, x) \approx x$$

is the idempotent linear strong Mal'tsev condition that defines majority operations. On the other hand, recall that semilattices are binary operations f satisfying

$$f(x, x) \approx x$$
$$f(x, y) \approx f(y, x)$$
$$f(x, f(y, z)) \approx f(f(x, y), z)$$

which does not form a linear strong Mal'tsev condition because the identity enforcing the associativity of f is not linear.

By extension, we say that a constraint language satisfies a linear strong Mal'tsev condition $\mathcal{M}$ if it has a collection of polymorphisms that satisfy $\mathcal{M}$. The definability of a class of constraint languages by a linear strong Mal'tsev condition $\mathcal{M}$ is strongly tied up with the meta-problem, because for such classes we can associate any constraint language Γ with a polynomial-sized CSP instance whose solutions, if any, are exactly the polymorphisms of Γ satisfying $\mathcal{M}$ [36]. We will describe the construction below.

Definition 21. Let $\mathcal{M}$ be a linear strong Mal'tsev condition over operation symbols $f_1, \ldots, f_m$ of respective arities $a_1, \ldots, a_m$ and Γ be a constraint language. The **indicator problem** of Γ for $\mathcal{M}$, denoted by $\mathcal{P}_{\mathcal{M}}(\Gamma)$, is a CSP instance $(\mathcal{X}, \mathcal{D}, \mathcal{C})$, where

(i) $\mathcal{X} = \{\, x_{f_i(v_1,\ldots,v_{a_i})} \mid i \in \{1,\ldots,m\}, (v_1,\ldots,v_{a_i}) \in \mathcal{D}^{a_i} \,\}$;

(ii) $\mathcal{D} = D(\Gamma)$;

(iii) For every $i \in \{1,\ldots,m\}$, $R^* \in \Gamma$ and $\mathbf{t_1}, \ldots, \mathbf{t_{a_i}} \in R^*$ there is a constraint $C^{R^*}_{\mathbf{f_i}(\mathbf{t_1},\ldots,\mathbf{t_{a_i}})}$ with relation R^* and scope S such that $\forall j \leq |S|$, $S[j] = x_{f_i(\mathbf{t_1}[j],\ldots,\mathbf{t_{a_i}}[j])}$;

(iv) For every identity $\mathcal{E} \in \mathcal{M}$ of the form $f_j(x_1,\ldots,x_{a_j}) \approx f_p(y_1,\ldots,y_{a_p})$ there is an equality constraint between $x_{f_j(\phi(x_1),\ldots,\phi(x_{a_j}))}$ and $x_{f_p(\phi(y_1),\ldots,\phi(y_{a_p}))}$ for every possible assignment ϕ to $\{x_1,\ldots,x_{a_j},y_1,\ldots,y_{a_p}\}$;

(v) For every identity $\mathcal{E} \in \mathcal{M}$ of the form $f_j(x_1,\ldots,x_{a_j}) \approx y$ there is a unary constraint with scope $x_{f_j(\phi(x_1),\ldots,\phi(x_{a_j}))}$ and relation $\{[(\phi(y))]\}$ for every possible assignment ϕ to $\{x_1,\ldots,x_{a_j},y\}$.

Because this elementary construction will be used extensively throughout the chapter, we shall complement the definition with an informal overview of its structure. For every operation symbol f_j and every a_j-tuple of values $(v_1,\ldots,v_{a_j})$, $\mathcal{P}_{\mathcal{M}}(\Gamma)$ contains one variable $x_{f_i(v_1,\ldots,v_{a_i})}$ that dictates where the operation f_j should map $(v_1,\ldots,v_{a_i})$. The constraints described in (iii) force each f_j to be a polymorphism of Γ, and the constraints (iv) and (v) ensure that $(f_1,\ldots,f_m)$ satisfy the identities in $\mathcal{M}$. The following is immediate.

Proposition 7. *Let $\mathcal{M}$ be a linear strong Mal'tsev condition and Γ be a constraint language. The solutions to $\mathcal{P}_{\mathcal{M}}(\Gamma)$ are exactly the polymorphisms of Γ satisfying $\mathcal{M}$.*

If $\mathcal{M}$ is an empty set of identities over a k-ary operation symbol f, $\mathcal{P}_{\mathcal{M}}(\Gamma)$ is called the indicator problem of order k of Γ [94] and will be denoted by $\mathcal{IP}^k(\Gamma)$. Because f is unconstrained, the solutions to $\mathcal{IP}^k(\Gamma)$ are exactly the k-ary polymorphisms of Γ.

Observe that each constraint in $\mathcal{P}_{\mathcal{M}}(\Gamma)$ is either a relation from Γ, a unary relation with a single tuple or an equality; hence every idempotent polymorphism of Γ is also a polymorphism of $\mathcal{L}(\mathcal{P}_{\mathcal{M}}(\Gamma))$. Thus, if the set of languages satisfying $\mathcal{M}$ is tractable these instances $\mathcal{P}_{\mathcal{M}}(.)$ have the unusual property to be over a tractable language *if they are satisfiable* - a property that will be used with great effect when combined with the final ingredient: uniformity. Let $\mathcal{M}$ denote a strong linear Mal'tsev condition, and let CSP($\mathcal{M}$) denote the CSP restricted to instances whose language satisfies $\mathcal{M}$.

Definition 22. A **uniform polynomial-time algorithm** for $\mathcal{M}$ is an algorithm that solves CSP($\mathcal{M}$) in polynomial time.

The term "uniform" here refers to the fact that the language is not fixed (as in the Feder-Vardi Dichotomy conjecture), but may range over all languages that satisfy $\mathcal{M}$. The existence of a uniform algorithm implies that CSP(Γ) is in P for every Γ that satisfies $\mathcal{M}$, but the converse is not guaranteed to be true. For instance, an algorithm for CSP($\mathcal{M}$) that is exponential only in the domain size is polynomial for every fixed Γ that satisfies $\mathcal{M}$, but is not uniform. The following proposition has been part of the folklore for some time (see e.g. [6]) and has been recently formalized in [36]. We give here the proof sketch because our contributions will reuse and refine the core idea.

Proposition 8 ([36])**.** *Let $\mathcal{M}$ be an idempotent strong linear Mal'tsev condition. If $\mathcal{M}$ has a uniform algorithm, then the meta-problem for $\mathcal{M}$ is polynomial time.*

Proof. The idempotency of $\mathcal{M}$ ensures that we have a uniform algorithm for the *search* problem (i.e. decide if the instance is satisfiable and produce a solution if one exists) because idempotent polymorphisms always preserve assignments to variables, which can be seen as unary relations with a single tuple. Given a constraint language Γ, to check if Γ satisfies $\mathcal{M}$ we build the instance $\mathcal{P}_{\mathcal{M}}(\Gamma)$ and invoke the uniform search algorithm. Since the language of $\mathcal{P}_{\mathcal{M}}(\Gamma)$ is Γ plus equalities and unary relations with a single tuple, $\mathcal{L}(\mathcal{P}_{\mathcal{M}}(\Gamma))$ satisfies $\mathcal{M}$ if and only if Γ does. If $\mathcal{P}_{\mathcal{M}}(\Gamma)$ is satisfiable then Γ satisfies $\mathcal{M}$ and the algorithm must produce a solution (which can be easily verified), and whenever the algorithm fails to do so we can safely conclude that Γ does not satisfy $\mathcal{M}$. □

This simple trick alone shows that detecting fixed-arity near-unanimity polymorphisms or deciding if a language has bounded width witnessed by idempotent polymorphisms can be done in polynomial time, since these classes correspond to idempotent strong linear Mal'tsev conditions and have uniform algorithms [6].

4.4 Semiuniform algorithms

The preliminaries over, we now narrow our attention to Proposition 8 and see in which ways it could be reinforced for conservative meta-problems. Let us examine in detail the hypotheses required by the method:

(i) There are finitely many identities in $\mathcal{M}$. Dropping this may cause $\mathcal{P}_{\mathcal{M}}(\Gamma)$ to have exponential size since the arity of the polymorphisms is no longer bounded. The number of extra constraints needed to enforce the identities may grow out of control as well.

(ii) The identities in $\mathcal{M}$ are linear. Identities that involve composition generally correspond to exponential-size extra constraints, and the language of $\mathcal{P}_{\mathcal{M}}(\Gamma)$ may no longer admit the polymorphisms of Γ (in fact it may even be NP-hard, even if Γ is tractable).

(iii) The identities in $\mathcal{M}$ entail idempotency. Always true in the conservative setting.

(iv) $\mathcal{M}$ has a uniform polynomial-time algorithm. Without this hypothesis the argument will not work, but there is no known example of tractable idempotent strong linear Mal'tsev condition with an NP-hard meta-problem either.

Considering all of the above, the requirement for a uniform algorithm seems to be the least impactful. We will confirm this reasoning by showing that in the case of conservative languages, the meta-problem for a linear strong Mal'tsev condition $\mathcal{M}$ can be solved in polynomial time assuming a weaker notion of uniformity that was introduced recently [36].

Definition 23. A **semiuniform polynomial-time algorithm** for $\mathcal{M}$ is an algorithm that solves $\mathrm{CSP}(\mathcal{M})$ in polynomial time provided each instance I is paired with polymorphisms $f_1, \ldots, f_m$ of $\mathcal{L}(I)$ that satisfy $\mathcal{M}$.

Semiuniform algorithms are tied to the identities in $\mathcal{M}$ rather than the class of languages it defines; even if $\mathrm{CSP}(\mathcal{M}_1)$ and $\mathrm{CSP}(\mathcal{M}_2)$ denote the exact same set of instances, the polymorphisms satisfying $\mathcal{M}_2$ can be more useful than those satisfying $\mathcal{M}_1$. The algorithm for Mal'tsev constraints is the archetypal example of a semiuniform algorithm since it is not exponential in $|\Gamma|$ but requires explicit access to a Mal'tsev polymorphism in order to solve the instance [24].

The relationship between uniformity and semiuniformity is quite subtle. There is no known example of idempotent linear strong Mal'tsev condition that has a semiuniform algorithm but provably no uniform algorithm (even modulo complexity theoretic assumptions), so the two notions might in fact coincide. The following

observation gives additional evidence that uniformity and semiuniformity can be difficult to distinguish.

Observation 1. *If $\mathcal{M}$ has a semiuniform algorithm, then* $\mathrm{CSP}(\mathcal{M}) \in NP \cap coNP$.

This follows from the fact that a set of polymorphisms satisfying $\mathcal{M}$ is a polynomially verifiable certificate for both satisfiable and unsatisfiable instances. Recall that $\mathcal{M}$ has a uniform algorithm if and only if $\mathrm{CSP}(\mathcal{M}) \in \mathrm{P}$; Observation 1 tells us that $\mathrm{CSP}(\mathcal{M})$ is not NP-hard unless NP = coNP [76]. The latter statement is weaker than the former, but not by a large margin. One could also be tempted to say that NP ∩ coNP problems not believed to be in P are few in number, but keep in mind that $\mathrm{CSP}(\mathcal{M})$ is a *promise* problem and constructing very hard NP ∩ coNP promise problems is easy by manipulating the promise. For example, given two SAT instances (I_1, I_2) deciding if I_1 is satisfiable is in NP ∩ coNP if we have the promise that exactly one instance among (I_1, I_2) is satisfiable, but this problem is unlikely to be in P. However, the certificates of those hard NP ∩ coNP promise problems often rely crucially on the promise to be useful, while the certificates of $\mathrm{CSP}(\mathcal{M})$ (polymorphisms) do not. Overall, Observation 1 should be taken as mild evidence that semiuniformity might imply uniformity, at least in some cases.

When the complexity of the meta-problem is taken into account as well, we obtain a picture where an idempotent strong linear Mal'tsev condition $\mathcal{M}$ with a semiuniform algorithm has a uniform algorithm if and only if the search version of the meta-problem is polynomial-time (a variant of the meta-problem where we have not only to decide if Γ satisfies $\mathcal{M}$, but also produce the polymorphisms that witness it). Because a polynomial-time algorithm for the meta-problem will almost always produce the polymorphisms, this means that replacing the requirement of uniformity with semiuniformity in Proposition 8 is unlikely to provide a generalization in the absolute sense but instead make its applicability easier.

4.5 Semiuniformity and Conservative Polymorphisms

We are now ready to present our main technical tool. Given a language Γ, we will use $\overline{\Gamma}$ to denote its *conservative completion*, that is, the language comprised of Γ plus all possible unary relations over $\mathrm{D}(\Gamma)$.

Theorem 18. *Let $\mathcal{M}$ be a linear strong Mal'tsev condition that admits a semiuniform algorithm. There exists a polynomial-time algorithm that, given as input a constraint language Γ, decides if $\overline{\Gamma}$ satisfies $\mathcal{M}$ and produces conservative polymorphisms of Γ satisfying $\mathcal{M}$ if any exist.*

The use of $\overline{\Gamma}$ in the above theorem is simply a trick to say that we are only interested in the *conservative* polymorphisms of Γ that satisfy $\mathcal{M}$ (conservativity is not a linear identity). The following is not quite a corollary, but rather an equivalent statement giving confirmation that semiuniformity and uniformity are strongly related.

Theorem 19. *Let $\mathcal{M}$ be a fixed strong Mal'tsev condition. In the case of conservative languages, $\mathcal{M}$ has a semiuniform algorithm if and only if $\mathcal{M}$ has a uniform algorithm.*

Our strategy to detect conservative polymorphisms satisfying $\mathcal{M}$ assuming a semiuniform algorithm for $\mathcal{M}$ is to cast the meta-problem as a CSP using the construction of Section 4.3 and then compute successively partial solutions $\phi_1, \dots, \phi_\alpha$ of slowly increasing size until a solution to the whole CSP is obtained. The originality of our approach is that ϕ_{i+1} is not computed directly from ϕ_i, but by solving a polynomial number of CSP instances whose languages admit ϕ_i as a polymorphism. One can see a semiuniform algorithm as a collection of polynomial-time algorithms, one per combination of polymorphisms satisfying $\mathcal{M}$; in our case, each partial solution is a *description* of a polynomial-time algorithm that can be used to compute a slightly larger one. This is strongly reminiscent of a treasure hunt, where each chest contains the key to unlock the next one.

Let $\mathcal{M}$ be a strong linear Mal'tsev condition with operation symbols $f_1, \dots, f_m$ of respective arities $a_1, \dots, a_m$. Let Γ be a constraint language over $\mathcal{D}$ and $\mathcal{P}_{\mathcal{M}}(\Gamma)$ be the CSP whose solutions are exactly the polymorphisms of Γ satisfying $\mathcal{M}$ (as described in Section 4.3). Recall that for every symbol f_i in $\mathcal{M}$ and $(d_1, \dots, d_{a_i}) \in \mathcal{D}^{a_i}$ we have a variable $x_{f_i(d_1,\dots,d_{a_i})}$ that dictates where f_i should map $d_1, \dots, d_{a_i}$, and for every $R^* \in \Gamma$ and a_i tuples $\mathbf{t_1}, \dots, \mathbf{t_{a_i}} \in R^*$ we have a constraint $C^{R^*}_{\mathbf{f_i}(\mathbf{t_1},\dots,\mathbf{t_{a_i}})}$ that forces the tuple $\mathbf{f_i}(\mathbf{t_1}, \dots, \mathbf{t_{a_i}})$ to belong to R^* (where $\mathbf{f_i}$ is the operation on tuples obtained by componentwise application of f_i). Our goal is to decide if $\overline{\Gamma}$ satisfies $\mathcal{M}$, which requires the polymorphisms of Γ satisfying $\mathcal{M}$ to be conservative. The solutions to $\mathcal{P}_{\mathcal{M}}(\Gamma)$ can easily be guaranteed to be conservative by adding the unary constraint $x_{f_i(d_1,\dots,d_{a_i})} \in \{d_1, \dots, d_{a_i}\}$ on every variable $x_{f_i(d_1,\dots,d_{a_i})} \in \mathcal{X}$. We will denote this new problem by $\mathcal{P}^c_{\mathcal{M}}(\Gamma)$, and each solution ϕ to $\mathcal{P}^c_{\mathcal{M}}(\Gamma)$ is a collection $(f_1, \dots, f_m)$ of conservative polymorphisms of Γ satisfying $\mathcal{M}$.

We need one more definition. Given a CSP instance $\mathcal{I}$, a *consistent restriction* of $\mathcal{I}$ is an instance obtained from $\mathcal{I}$ by adding new constraints that are either unary or equalities and then enforcing 1-minimality. We will be interested in the consistent restrictions of $\mathcal{P}^c_{\mathcal{M}}(\Gamma)$, and we will keep the same notations for constraints that already existed in $\mathcal{P}^c_{\mathcal{M}}(\Gamma)$.

Lemma 4. *Let $\mathcal{P} = (\mathcal{X}, \mathcal{D}, \mathcal{C})$ be a consistent restriction of $\mathcal{P}^c_{\mathcal{M}}(\Gamma)$. Let f_i and f_j be operation symbols in $\mathcal{M}$. If $C^{R^*}_{\mathbf{f_i}(\mathbf{t_1},\ldots,\mathbf{t_{a_i}})} \in \mathcal{C}$ and $\mathbf{t'_1},\ldots,\mathbf{t'_{a_j}} \in \mathcal{R}(C^{R^*}_{\mathbf{f_i}(\mathbf{t_1},\ldots,\mathbf{t_{a_i}})})$ then*

$$\mathcal{R}(C^{R^*}_{\mathbf{f_j}(\mathbf{t'_1},\ldots,\mathbf{t'_{a_j}})}) \subseteq \mathcal{R}(C^{R^*}_{\mathbf{f_i}(\mathbf{t_1},\ldots,\mathbf{t_{a_i}})})$$

Proof. Let $S = \mathcal{S}(C^{R^*}_{\mathbf{f_i}(\mathbf{t_1},\ldots,\mathbf{t_{a_i}})})$ and $S' = \mathcal{S}(C^{R^*}_{\mathbf{f_j}(\mathbf{t'_1},\ldots,\mathbf{t'_{a_j}})})$. Before 1-minimality was enforced, we had $\mathcal{R}(C^{R^*}_{\mathbf{f_i}(\mathbf{t_1},\ldots,\mathbf{t_{a_i}})}) = \mathcal{R}(C^{R^*}_{\mathbf{f_j}(\mathbf{t'_1},\ldots,\mathbf{t'_{a_j}})}) = R^*$. Thus, after enforcing 1-minimality we have $\mathcal{R}(C^{R^*}_{\mathbf{f_i}(\mathbf{t_1},\ldots,\mathbf{t_{a_i}})}) = R^* \cap (\pi_{x\in S} D(x))$ and $\mathcal{R}(C^{R^*}_{\mathbf{f_j}(\mathbf{t'_1},\ldots,\mathbf{t'_{a_j}})}) = R^* \cap (\pi_{x\in S'} D(x))$. However, since $\mathbf{t'_1},\ldots,\mathbf{t'_{a_j}} \in \mathcal{R}(C^{R^*}_{\mathbf{f_i}(\mathbf{t_1},\ldots,\mathbf{t_{a_i}})})$, the conservativity constraints ensure that for each k,

$$D(S'[k]) = D(x_{f_j(\mathbf{t'_1}[k],\ldots,\mathbf{t'_{a_j}}[k])}) \subseteq \{\mathbf{t'_1}[k],\ldots,\mathbf{t'_{a_j}}[k]\} \subseteq D(S[k])$$

Therefore, $\mathcal{R}(C^{R^*}_{\mathbf{f_j}(\mathbf{t'_1},\ldots,\mathbf{t'_{a_j}})}) \subseteq \mathcal{R}(C^{R^*}_{\mathbf{f_i}(\mathbf{t_1},\ldots,\mathbf{t_{a_i}})})$. □

Given two sets of variables $X_1, X_2 \subseteq \mathcal{X}$, we write $X_1 \lhd X_2$ if for each symbol f_i in $\mathcal{M}$, for each x in X_2 and for each $\mathbf{t}$ in $D(x)^{a_i}$ we have $x_{f_i(\mathbf{t})} \in X_1$. If $X_1 \lhd X_1$, we say that X_1 is *closed*.

Proposition 9. *Let $\mathcal{P} = (\mathcal{X}, \mathcal{D}, \mathcal{C})$ be a consistent restriction of $\mathcal{P}^c_{\mathcal{M}}(\Gamma)$. If X_1 and X_2 are subsets of variables such that $X_1 \lhd X_2$, then every solution to $\mathcal{P}_{|X_1}$ is a collection of polymorphisms of $\mathcal{L}(\mathcal{P}_{|X_2})$.*

Proof. Let $f_i, f_j \in \{f_1,\ldots,f_m\}$ be operation symbols in $\mathcal{M}$. Let $R^* \in \Gamma$, $\mathbf{t_1},\ldots,\mathbf{t_{a_i}} \in R^*$, $C_2 = (S_2, R_2) \in \mathcal{P}_{|X_2}$ be the projection of $C^{R^*}_{\mathbf{f_i}(\mathbf{t_1},\ldots,\mathbf{t_{a_i}})}$ onto X_2, and $\mathbf{t^2_1},\ldots,\mathbf{t^2_{a_j}} \in R_2$. By the nature of projections, there must exist $\mathbf{t'_1},\ldots,\mathbf{t'_{a_j}} \in \mathcal{R}(C^{R^*}_{\mathbf{f_i}(\mathbf{t_1},\ldots,\mathbf{t_{a_i}})})$ such that $\mathbf{t^2_1},\ldots,\mathbf{t^2_{a_j}}$ is the projection of $\mathbf{t'_1},\ldots,\mathbf{t'_{a_j}}$ onto X_2. Then, by Lemma 4 we have

$$\mathcal{R}(C^{R^*}_{\mathbf{f_j}(\mathbf{t'_1},\ldots,\mathbf{t'_{a_j}})}) \subseteq \mathcal{R}(C^{R^*}_{\mathbf{f_i}(\mathbf{t_1},\ldots,\mathbf{t_{a_i}})})$$

and in particular $\mathcal{R}(C^{R^*}_{\mathbf{f_j}(\mathbf{t'_1},\ldots,\mathbf{t'_{a_j}})}[X_2]) \subseteq \mathcal{R}(C^{R^*}_{\mathbf{f_i}(\mathbf{t_1},\ldots,\mathbf{t_{a_i}})}[X_2]) = R_2$. Now, note that because $X_1 \lhd X_2$ and $\mathcal{P}$ is 1-minimal, every variable $x_{f_j(\mathbf{t'_1}[k],\ldots,\mathbf{t'_{a_j}}[k])}$ in the scope of $C^{R^*}_{\mathbf{f_j}(\mathbf{t'_1},\ldots,\mathbf{t'_{a_j}})}[X_2]$ also belongs to X_1. We denote this constraint by C_1.

Let us summarize what we have: for every symbol f_j, every relation $R_2 \in \mathcal{L}(\mathcal{P}_{|X_2})$ other than equalities and unary relations (which are preserved by all conservative polymorphisms) and $\mathbf{t^2_1},\ldots,\mathbf{t^2_{a_j}} \in R_2$, there is a constraint $C_1 = (S_1, R_1) \in \mathcal{P}_{|X_1}$ such that $|S_1| = |S_2|$, $R_1 \subseteq R_2$ and for every k we have $S_1[k] = x_{f_j(\mathbf{t^2_1}[k],\ldots,\mathbf{t^2_{a_j}}[k])}$. It follows that for every solution $(f_1,\ldots,f_m)$ to $\mathcal{P}(\Gamma)_{|X_1}$, f_j is also a solution to the indicator problem of order a_j of $\mathcal{L}(\mathcal{P}(\Gamma)_{|X_2})$ and is therefore a polymorphism of $\mathcal{L}(\mathcal{P}(\Gamma)_{|X_2})$. □

Closed sets of variables allow us to turn partial solutions into true polymorphisms of a specific constraint language, hence enabling us to make (limited) use of semiuniform algorithms. A variable of $\mathcal{P}^c_{\mathcal{M}}(\Gamma)$ is a *singleton* if it is of the form $x_{f_i(v,\ldots,v)}$ for some $v \in \mathcal{D}$. The sets of variables corresponding to singletons and $\mathcal{X}$ constitute two closed sets; the next lemma shows that many intermediate, regurlarly-spaced closed sets exist in $\mathcal{P}^c_{\mathcal{M}}(\Gamma)$ between these two extremes.

Lemma 5. *Let $\mathcal{P}^c_{\mathcal{M}}(\Gamma) = (\mathcal{X}, \mathcal{D}, \mathcal{C})$ after applying 1-minimality. There exist $X_0 \subseteq \ldots \subseteq X_\alpha = \mathcal{X}$ such that X_0 is the set of all singleton variables, each X_i is closed and $|X_{i+1} - X_i| \leq ma^a$, where a and m denote respectively the maximum arity and number of operation symbols in $\mathcal{M}$.*

Proof. Let $(D_1, \ldots, D_\alpha)$ denote an arbitrary ordering of the subsets of $\mathcal{D}$ of size a. We define

$$X_0 = \{x_{f_j(v_i,\ldots,v_i)} \mid f_j \in \mathcal{M}, v_i \in \mathcal{D}\}$$

and for all $i \in [1..\alpha]$

$$X_i = X_{i-1} \cup \{x_{f_j(\mathbf{t})} \mid f_j \in \mathcal{M}, \mathbf{t} \in (D_i)^{a_j}\}$$

It is clear that X_0 is the set of all singleton variables and for all i, $|X_{i+1} - X_i| \leq m|(D_i)^a| = ma^a$. It remains to show that each set is closed. Let $k \geq 1$ and suppose that X_{k-1} is closed. By induction hypothesis, we only need to verify that $X_k \lhd X_k \backslash X_{k-1}$. Let $x_{f_j(v_1,\ldots,v_{a_j})}$ be a variable in $X_k \backslash X_{k-1}$. Because $\mathcal{P}^c_{\mathcal{M}}(\Gamma)$ is 1-minimal, we have $D(x_{f_j(v_1,\ldots,v_{a_j})}) \subseteq \{v_1, \ldots, v_{a_j}\} \subseteq D_k$. By construction X_k contains all variables of the form $x_{f_c(\mathbf{t})}$ where $\mathbf{t} \in (D_k)^{a_c}$ and because $D(x_{f_j(v_1,\ldots,v_{a_j})}) \subseteq \{v_1, \ldots, v_{a_j}\} \subseteq D_k$ it contains in particular all variables $x_{f_c(\mathbf{t})}$ such that $t \subseteq D(x_{f_j(v_1,\ldots,v_{a_j})})$. This implies that $X_k \lhd X_k \backslash X_{k-1}$ and concludes the proof. □

We now have every necessary tool at our disposal to start solving $\mathcal{P}^c_{\mathcal{M}}(\Gamma)$. It is straightforward to see that if a subset of variables X' is closed in $\mathcal{P}^c_{\mathcal{M}}(\Gamma)$, then it is closed in every consistent restriction as well.

Proposition 10. *If a solution to $\mathcal{P}^c_{\mathcal{M}}(\Gamma)_{|X_i}$ is known, then a solution to $\mathcal{P}^c_{\mathcal{M}}(\Gamma)_{|X_{i+1}}$ (if any exist) can be found in polynomial time.*

Proof. Let $(f^i_1, \ldots, f^i_m)$ be a solution to $\mathcal{P}^c_{\mathcal{M}}(\Gamma)_{|X_i}$. We assume that 1-minimality has been enforced on $\mathcal{P}^c_{\mathcal{M}}(\Gamma)$. This ensures, in particular, that the domain of each $x_{f_j(\mathbf{t})} \in X_{i+1} \backslash X_i$ contains at most a elements. It follows that $X_{i+1} \backslash X_i$ has at most $s = a^{ma^a}$ possible assignments $\phi_1, \ldots, \phi_s$. For every $j \in [1..s]$, we create a CSP instance $\mathcal{P}_j$ that is a copy of $\mathcal{P}^c_{\mathcal{M}}(\Gamma)$ but also includes the constraints corresponding to the assignment $X_{i+1} \backslash X_i \leftarrow \phi_j(X_{i+1} \backslash X_i)$. We enforce 1-minimality on every instance $\mathcal{P}_j$.

Now, observe that each $\mathcal{P}_j$ is a consistent restriction of $\mathcal{P}^c_{\mathcal{M}}(\Gamma)$, so X_i is still closed in $\mathcal{P}_j$. Moreover, every variable $x \in X_{i+1} \backslash X_i$ has domain size 1 in $\mathcal{P}_j$; since X_i contains all singleton variables, if follows that in $\mathcal{P}_j$ we have $X_i \lhd X_{i+1}$.

By Proposition 9, $(f^i_1, \ldots, f^i_m)$ is a collection of polymorphisms of $\mathcal{L}(\mathcal{P}_{j|X_{i+1}})$. We can then use the semiuniform algorithm to find in polynomial time a solution to $\mathcal{P}_{j|X_{i+1}}$ if one exists by backtracking search (every f^i_z is idempotent, so we can invoke the semiuniform algorithm at each node to ensure that the algorithm cannot backtrack more than one level). A solution to $\mathcal{P}^c_{\mathcal{M}}(\Gamma)_{|X_{i+1}}$ exists if and only if $\mathcal{P}_{j|X_{i+1}}$ has a solution for some $j \in \{1, \ldots, s\}$. □

The above proof depends critically on the fact that every complete instantiation of the variables in $X_{i+1} \backslash X_i$ (followed by 1-minimality) yields a residual instance over a language that admits $(f^i_1, \ldots, f^i_m)$ as polymorphisms. In other terms, $\mathcal{P}^c_{\mathcal{M}}(\Gamma)_{|X_{i+1}}$ has a *backdoor* of constant size into the class of languages that admit $(f^i_1, \ldots, f^i_m)$ as polymorphisms. We now have at our disposal every ingredient needed to prove Theorem 18.

Proof (of Theorem 18). The algorithm starts by building $\mathcal{P}^c_{\mathcal{M}}(\Gamma)$ and computes the sets $X_0, \ldots, X_\alpha$ as in Lemma 5. We have a solution to $\mathcal{P}^c_{\mathcal{M}}(\Gamma)_{|X_0}$ for free because of the conservativity constraints, and we can compute a solution to $\mathcal{P}^c_{\mathcal{M}}(\Gamma)$ by invoking repeatedly (at most $\alpha \leq |\mathcal{X}| \leq md^a$ times) Proposition 10. If the algorithm fails at any step, $\overline{\Gamma}$ does not satisfy $\mathcal{M}$. □

An immediate application of Theorem 18 concerns the detection of conservative k-edge polymorphisms for a fixed k. Recall that a k-edge operation on a set $\mathcal{D}$ is a $(k+1)$-ary operation e satisfying

$$
\begin{aligned}
e(x, x, y, y, y, \ldots, y, y) &\approx y \\
e(x, y, x, y, y, \ldots, y, y) &\approx y \\
e(x, y, y, x, y, \ldots, y, y) &\approx y \\
e(x, y, y, y, x, \ldots, y, y) &\approx y \\
&\ldots \\
e(x, y, y, y, y, \ldots, x, y) &\approx y \\
e(x, y, y, y, y, \ldots, y, x) &\approx y
\end{aligned}
$$

These identities form a linear strong Mal'tsev condition. The algorithm given in [90] is semiuniform, but in addition to e it must have access to three other

polymorphisms p, d, s derived from e and satisfying

$$\begin{aligned} p(x,y,y) &\approx x \\ p(x,x,y) &\approx d(x,y) \\ d(x,d(x,y)) &\approx d(x,y) \\ s(x,y,y,y,\ldots,y,y) &\approx d(y,x) \\ s(y,x,y,y,\ldots,y,y) &\approx y \\ s(y,y,x,y,\ldots,y,y) &\approx y \\ &\ldots \\ s(y,y,y,y,\ldots,y,x) &\approx y \end{aligned}$$

The authors provide a method to obtain these three polymorphisms from e that requires a possibly exponential number of compositions. However, conservative algebras are much simpler and we can observe that

$$\begin{aligned} s(x_1,x_2,\ldots,x_k) &= e(x_2,x_1,x_2,x_3,\ldots,x_k) \\ d(x,y) &= e(x,y,x,\ldots,x) \\ p(x,y,z) &= e(y,d(y,z),x,\ldots,x) \end{aligned}$$

satisfy the required identities and are easy to compute. It follows that in the conservative case their algorithm is semiuniform even if only a k-edge polymorphism e is given.

Corollary 3. *For every fixed k, the class of constraint languages admitting a conservative k-edge polymorphism is uniformly tractable and has a polynomially decidable meta-problem.*

4.6 Deciding the Conservative Dichotomy in Polynomial Time

While the criterion for the conservative dichotomy theorem can be stated as a linear strong Mal'tsev condition [129], none of the algorithms found in the literature are semiuniform. Still, Theorem 18 gives a uniform algorithm for constraint languages Γ whose coloured graph contains only yellow and blue edges: if $g^*(x,y,z)$ and $h^*(x,y,z)$ are the polymorphisms predicted by the Three Operations Theorem, then

$$m^*(x,y,z) = h^*(g^*(x,y,z), g^*(y,z,x), g^*(z,x,y))$$

is a generalized majority-minority polymorphism of Γ, which implies that Γ has a 3-edge polymorphism [10].

Our algorithm will reduce the meta-problem to a polynomial number of CSP instances over languages with conservative 3-edge polymorphisms using a refined version of the treasure hunt algorithm paired with a simple reduction rule. This reduction rule is specific to indicator problems and allows us to avoid the elaborate machinery presented in [22] to eliminate red edges in CSP instances over a tractable conservative language.

We start by the reduction rule. We say that a binary polymorphism is *maximally commutative* if it is commutative on as many 2-elements subsets as possible, and projects onto its first argument otherwise. By the Three Operations Theorem, a maximally commutative polymorphism of a conservatively tractable language Γ is commutative on a pair of values if and only if it is red.

Proposition 11. *Suppose that we know a polymorphism f^* of Γ that is maximally commutative if* c-CSP(Γ) *is tractable. There is a polynomial-time algorithm that determines if* c-CSP(Γ) *is tractable, in which case it also returns the colour of each edge in G_Γ.*

Proof. If c-CSP(Γ) is tractable, f^* is enough to identify every red pair of values. To determine the colour of non-red pairs we need to look at conservative ternary polymorphisms. Therefore, we start by building the instance $\mathcal{IP}^{3c}(\Gamma)$, which is the indicator problem of order 3 of Γ with conservativity constraints. For $i \in \{1,2,3\}$, let π_i be the solution to $\mathcal{IP}^{3c}(\Gamma)$ given by $\pi_i(x_{v_1,v_2,v_3}) = v_i$ for all $v_1, v_2, v_3 \in \mathcal{D}$. These solutions correspond to the three ternary polymorphisms of Γ that project onto their ith argument. We enforce 1-minimality and apply the algorithm `Reduce`.

Algorithm 1: `Reduce`

$s_1 \leftarrow \pi_1$;
$s_2 \leftarrow \pi_2$;
$s_3 \leftarrow \pi_3$;
while *There exist i, j and $x \in \mathcal{X}$ such that $\{s_i(x), s_j(x)\}$ is red and $f^*(s_i(x), s_j(x)) = s_j(x)$* **do**
  $s_1 \leftarrow f^*(s_1, s_j)$;
  $s_2 \leftarrow f^*(s_2, s_j)$;
  $s_3 \leftarrow f^*(s_3, s_j)$;
  for *all $x \in \mathcal{X}$ and $v \in D(x)$ s.t. $\forall k$, $s_k(x) \neq v$* **do**
    $D(x) \leftarrow D(x) \backslash v$;

We denote by $\mathcal{IP}^{3c}_R(\Gamma)$ the resulting CSP instance. An important invariant of this algorithm is that at the end of every iteration of the loop in `Reduce`, for

every $x \in \mathcal{X}$ and $v \in D(x)$ there exists $s \in \{s_1, s_2, s_3\}$ such that $s(x) = v$. This is straightforward, since we only remove v from $D(x)$ if none of $s_1(x), s_2(x), s_3(x)$ takes value v. It then follows from the loop condition that at the end of `Reduce`, no $x \in \mathcal{X}$ may have a domain that contains a red pair of elements.

We now show that if $\mathcal{IP}^{3c}(\Gamma)$ has a solution that is majority (resp. minority) on a non-red pair of values B, then so does $\mathcal{IP}^{3c}_R(\Gamma)$. We proceed by induction. Suppose that at iteration i of the loop of `Reduce`, a solution p_i that is majority (resp. minority) on B exists. Let $D_i(x)$ denote the domain of a variable x at step i. We set $p_{i+1} = f^*(p_i, s_j)$. Because f always projects onto its first argument on non-red pairs, a value v can only be removed from $D_i(x)$ at iteration $i+1$ if $\{v, s_j(x)\}$ is red and $f(v, s_j(x)) = s_j(x)$. Therefore, if $p_i(x)$ is removed at iteration i then $p_{i+1}(x) = f^*(p_i(x), s_j(x)) = s_j(x)$, and otherwise $p_{i+1}(x) \in \{p_i(x), s_j(x)\} \subseteq D_{i+1}(x)$; in any case $p_{i+1}(x) \in D_{i+1}(x)$. Furthermore, since B is not red, $p_{i+1}(x_{f(v_1,v_2,v_3)}) = p_i(x_{f(v_1,v_2,v_3)})$ for all $\{v_1, v_2, v_3\} \subseteq B$ and we can conclude that p_{i+1} is still majority (resp. minority) on B.

Now, we enforce 1-minimality again. We can ensure that every solution is a majority (resp. minority) polymorphism when restricted to B by assigning the 6 variables concerned by the majority (resp. minority) identity. Since the remaining instance I is red-free in G_Γ, either c-CSP(Γ) is intractable or $\mathcal{L}(I)$ admits a 3-edge polymorphism. We test for the existence of a 3-edge polymorphism using Theorem 18. If one exists we use the uniform algorithm given by Corollary 3 to decide if a solution exists and otherwise we can conclude that c-CSP(Γ) is intractable. □

With this result in mind, the last challenge is to design a polynomial-time algorithm that finds a binary polymorphism f^* that is commutative on as many 2-element subsets as possible, and projects onto its first argument otherwise. This can be achieved using a variant of the algorithm presented in Section 4.5 and the following lemma.

Lemma 6. *Let $\mathcal{P} = (\mathcal{X}, \mathcal{D}, \mathcal{C})$ denote an 1-minimal instance such that $\forall x \in \mathcal{X}$, $|D(x)| \leq 2$. Suppose that we have a conservative binary polymorphism f of $\mathcal{L}(\mathcal{P})$ and a partition (V_1, V_2) of the variables such that $f(a,b) = f(b,a) = f(D(x))$ whenever $x \in V_1$, and f projects onto its first argument otherwise. Then, every variable $x \in V_1$ can be assigned to $f(D(x))$ without altering the satisfiability of $\mathcal{P}$.*

Proof. Let $C = (S, R) \in \mathcal{C}$. Let $S_1 = S \cap V_1$, $S_2 = S \cap V_2$ and $\mathbf{t} \in R$. We assume without loss of generality that no variable in S is ground (i.e. has a singleton domain). If $x \in S$, let $\overline{\mathbf{t}[x]} = D(x) \backslash \mathbf{t}[x]$. Because $\mathcal{P}$ is 1-minimal, for every $x \in S_1$ there exists $\mathbf{t_x} \in R$ such that $\mathbf{t_x}[x] = \overline{\mathbf{t}[x]}$. Let $x_1, \ldots, x_s$ denote an arbitrary ordering of S_1. Then, let $\mathbf{t^{(0)}} = \mathbf{t}$ and for $i \in \{1, \ldots, s\}$,

$$\mathbf{t^{(i)}} = \mathbf{f}(\mathbf{t^{(i-1)}}, \mathbf{t_{x_i}})$$

It is immediate to see that if $x \in S_2$, then $\mathbf{t^{(s)}}[x] = \mathbf{t}[x]$ since f will project onto its first argument at each interation. On the other hand, if $x_k \in S_1$ and there exists j such that $\mathbf{t^{(j)}}[x_k] = f(D(x_k))$ then $\mathbf{t^{(i)}}[x_k] = f(D(x_k))$ for all $i \geq j$. This is guaranteed to happen for $j \leq k$, as either

- $\mathbf{t}[x_k] = f(D(x_k))$, in which case it is true for $j = 0$, or
- $\mathbf{t^{(k-1)}}[x_k] = f(D(x_k))$, in which case it is true for $j = k - 1$, or
- $\mathbf{t^{(k-1)}}[x_k] = \mathbf{t}[x_k] \neq f(D(x_k))$, in which case $\mathbf{t^{(k)}}[x_k] = f(\mathbf{t^{(k-1)}}[x_k], \mathbf{t_{x_k}}[x_k]) = f(\mathbf{t}[x_k], \mathbf{t}[x_k]) = f(D(x_k))$ and thus it is true for $j = k$.

It follows that $\mathbf{t^{(s)}}$ is a tuple or R that coincides with $\mathbf{t}$ on S_2, and $\mathbf{t^{(s)}}[x] = D(f(x))$ whenever $x \in S_1$. Therefore, assigning each $x \in S_1$ to $D(f(x))$ is always compatible with any assignment to S_2; since this is true for each constraint, it is true for $\mathcal{P}$ as well. □

We denote by $\mathcal{IP}^{2c}(\Gamma)$ the CSP instance obtained from $\mathcal{IP}^{2}(\Gamma)$ by adding the unary constraints enforcing conservativity. We can interpret $\mathcal{IP}^{2c}(\Gamma)$ as the meta-problem associated with an unconstrained conservative binary operation symbol f and reuse the definitions and lemmas about closed sets of variables seen in the last section. In the hierarchy of closed sets given by Lemma 5 applied to $\mathcal{IP}^{2c}(\Gamma)$, X_{i+1} contains the variables of X_i plus two variables $x_{f(a,b)}, x_{f(b,a)}$ for some $B_{i+1} = \{a, b\} \subseteq \mathcal{D}$.

Proposition 12. *Suppose that we know a solution f_i to $\mathcal{IP}^{2c}(\Gamma)_{|X_i}$ that is maximally commutative if* c-CSP(Γ) *is tractable. A solution f_{i+1} to $\mathcal{IP}^{2c}(\Gamma)_{|X_{i+1}}$ with the same properties can be found in polynomial time.*

Proof. The strategy is similar to the proof of Proposition 10. The two differences are that we do not have a semiuniform algorithm in general, which can be handled by Lemma 6, and the fact that we are not interested in *any* solution but in one that is maximally commutative.

Observe that if $\mathcal{IP}^{2c}(\Gamma)_{|X_{i+1}}$ is 1-minimal, then its language is conservatively tractable and the order-2 conservative indicator problem of $\mathcal{L}(\mathcal{IP}^{2c}(\Gamma)_{|X_{i+1}})$ is exactly $\mathcal{IP}^{2c}(\Gamma)_{|X_{i+1}}$ plus unconstrained variables (because X_{i+1} is closed). Therefore, by the Three Operations Theorem, a maximally commutative solution to $\mathcal{IP}^{2c}(\Gamma)_{|X_{i+1}}$ is commutative on some pair $\{u, v\}$ if and only if there is a solution to $\mathcal{IP}^{2c}(\Gamma)_{|X_{i+1}}$ that is also commutative on $\{u, v\}$. It follows from this same argument applied to X_i instead of X_{i+1} that if f_i is *not* commutative on some $(u, v) \in \mathcal{D}^2$ then either c-CSP(Γ) is NP-complete or Γ has a ternary conservative polymorphism $p_{u,v}$ that is either a majority or a minority operation on $\{u, v\}$.

Let $X_{i+1} = X_i \cup \{x_{f(a,b)}, x_{f(b,a)}\}$. We have only three assignments to examine for $(x_{f(a,b)}, x_{f(b,a)})$: (a, a), (b, b) and (a, b). The fourth assignment (b, a) is the

projection onto the second argument, which does not need to be tried since we are only interested in the maximally commutative solutions to $\mathcal{IP}^{2c}(\Gamma)_{|X_{i+1}}$. For each of these assignments, we build the CSP instances $\mathcal{P}^1, \mathcal{P}^2, \mathcal{P}^3$ by adding the constraints corresponding to the possible assignments to $(x_{f(a,b)}, x_{f(b,a)})$ to $\mathcal{IP}^{2c}(\Gamma)$ and enforcing 1-minimality.

For every $j \in \{1,2,3\}$ and every pair $\{u,v\}$ of elements in the domain of $\mathcal{P}^j_{|X_{i+1}}$ we create an instance $\mathcal{P}^j_{uv}$ by adding the constraint $x_{f(u,v)} = x_{f(v,u)}$ to $\mathcal{P}^j$ and enforcing 1-minimality. Since the variables in $X_{i+1} \backslash X_i$ are ground in $\mathcal{P}^j_{uv}$, X_i is closed and X_i contains all singleton variables, we have $X_{i+1} \lhd X_i$ in $\mathcal{P}^j_{uv}$. By Proposition 9, f_i is a polymorphism of $\mathcal{L}(\mathcal{P}^j_{uv|X_{i+1}})$. Now, if a variable x in $\mathcal{P}^j_{uv|X_{i+1}}$ has domain size 2 and f_i is commutative on $D(x)$, by Lemma 6 we can assign x to $f_i(D(x))$ without losing the satisfiability of the instance. Once this is done, we can enforce 1-minimality again; the polymorphisms $p_{u',v'}$ guarantee that if c-CSP(Γ) is tractable, the remaining instance has a conservative generalized majority-minority polymorphism and hence a conservative 3-edge polymorphism. Using Corollary 3, we can decide if the language of $\mathcal{P}^j_{uv|X_{i+1}}$ has a conservative 3-edge polymorphism. If it does not then c-CSP(Γ) is NP-complete, and otherwise we can decide if a solution exists in polynomial time.

At this point, for every pair (u,v) of elements in the domain of some variable in $\mathcal{IP}^{2c}(\Gamma)_{|X_{i+1}}$ we know if a solution to $\mathcal{IP}^{2c}(\Gamma)_{|X_{i+1}}$ that is commutative on (u,v) exists, except if $(u,v) = (a,b)$. This problem can be fixed by checking if any of $\mathcal{P}^{a,a}_{|X_{i+1}}$ or $\mathcal{P}^{b,b}_{|X_{i+1}}$ has a solution, where $\mathcal{P}^{a,a}$ and $\mathcal{P}^{b,b}$ are the subproblems corresponding respectively to the assignments $(x_{f(a,b)}, x_{f(b,a)}) \leftarrow (a,a)$ and $(x_{f(a,b)}, x_{f(b,a)}) \leftarrow (b,b)$.

We then add the equality constraint $x_{f(u,v)} = x_{f(v,u)}$ to $\mathcal{IP}^{2c}(\Gamma)_{|X_{i+1}}$ for *every* pair (u,v) (including (a,b) if applicable) such that a solution to $\mathcal{IP}^{2c}(\Gamma)_{|X_{i+1}}$ that is commutative on (u,v) exists. On all other pairs, we know that f_{i+1} must project on the first argument, so we can ground the corresponding variables. If c-CSP(Γ) is tractable, then this new CSP instance $\mathcal{P}$ has a solution and it must be maximally commutative. We can solve $\mathcal{P}$ by branching on the possible assignments to $(x_{f(a,b)}, x_{f(b,a)})$ and the usual arguments using f_i, Proposition 9 and Lemma 6. □

Theorem 20. *There exists a polynomial-time algorithm* `A` *that, given in input a constraint language* Γ*, decides if* c-CSP(Γ) *is in P or NP-complete. If* c-CSP(Γ) *is in P, then* `A` *also returns the coloured graph of* Γ*.*

Proof. We use Proposition 12 to find in polynomial time a conservative polymorphism f^* of Γ that is maximally commutative if c-CSP(Γ) is tractable. If the algorithm fails, then we know that c-CSP(Γ) is not tractable and the algorithm stops. Otherwise, we label every pair $\{a,b\}$ of domain elements with the colour red if f^* is commutative on $\{a,b\}$, and otherwise we use Proposition 11 to check if

there is a conservative ternary polymorphism that is either majority or minority on $\{a, b\}$. If a majority polymorphism is found then we label $\{a, b\}$ with yellow, else if a minority polymorphism is found then $\{a, b\}$ is blue, and otherwise we know that c-CSP(Γ) is NP-complete. The orientation of the red edges can be easily computed from $\mathcal{IP}^{2c}(\Gamma)$ using Lemma 6 and f^*. □

4.7 Specialized Algorithms

The worst-case complexity of the algorithm for solving CSP with 3-edge polymorphisms is $O(m(ndt)^{25})$ [90], were n is the number of variables, d is the domain size, t is the number of tuples and m is the number of constraints. Describing this algorithm as costly would be an understatement. To decide the conservative dichotomy in polynomial time, our "treasure hunt" method invokes repeatedly this algorithm on instances with $O(d^4)$ variables (indicator problems for 3-edge polymorphisms), so the dependency of our algorithm in the domain size is a polynomial whose degree is well over a hundred. This is likely to be an overestimation because we are using the "3-edge algorithm" on highly structured instances and with *conservative* 3-edge polymorphisms, but this is sufficient to claim that our method, despite solving an important theoretical problem, is utterly impractical.

The techniques we have employed so far are very coarse, and only use the most elementary properties of indicator problems. In particular, Theorem 18 is perhaps *too general* because we have made absolutely no assumption on the identities encapsulated by the strong linear Mal'tsev condition; the lack of information on the actual identities prevents us from characterizing the structure of the indicator problem. In this section we will provide specialized algorithms for detecting respectively conservative Mal'tsev and conservative majority polymorphisms that do not rely on the Mal'tsev semiuniform algorithm or the uniform algorithm for majority constraints, singleton arc-consistency. These ad hoc algorithms are very simple and efficient but the correctness is tricky to prove, especially for Mal'tsev polymorphisms. The only result we will reuse from the previous sections is Lemma 4.

4.7.1 Conservative Mal'tsev Polymorphisms

The outline of our approach to detect conservative Mal'tsev polymorphisms is as follows. We first reduce the problem to that of finding a conservative *minority* polymorphism using a simple algebraic trick. Then, we show that enforcing 1-minimality on the indicator problem associated with conservative minority polymorphisms leaves an extremely well-structured instance, and a simple reduction rule allows us to eliminate every variable whose domain contains more than two values. The residual instance is then shown to be equivalent to a system of linear

equations over GF(2), and can be solved by Gaussian elimination.

Lemma 7. *Let $\mathcal{F}$ be a clone. $\mathcal{F}$ contains a conservative Mal'tsev operation if and only if it contains a conservative minority operation.*

Proof. Every minority operation is a Mal'tsev operation, hence one implication is trivial. Suppose that $\mathcal{F}$ contains a conservative Mal'tsev operation m, and let

$$f(x,y,z) = m(z, m(y, m(x,z,y), x), m(x,z,y))$$

This operation belongs to $\mathcal{F}$ because $\mathcal{F}$ is a clone, and is conservative since m is conservative. Furthermore, for every a, b we have

$$f(a,b,a) = m(a, m(b, m(a,a,b), a), m(a,a,b)) = b$$
$$f(b,a,a) = m(a, m(a, m(b,a,a), b), m(b,a,a)) = b$$

and it is straightforward to see that $f(a,a,b) = m(b, m(a, m(a,b,a), a), m(a,b,a))$ is always equal to b, whether $m(a,b,a) = b$ or $m(a,b,a) = a$. Hence, f is a minority operation of $\mathcal{F}$. □

Although this lemma may be known to some, it appears to have never been pointed out in the literature. The closest results we could find were that digraphs with a conservative Mal'tsev polymorphisms also have a conservative minority polymorphism [35] and constraint languages with both a conservative majority and a conservative Mal'tsev polymorphism also have a conservative minority polymorphism [27]. In our case, this lemma is crucial, since the indicator problem corresponding to conservative minority polymorphisms has interesting (i.e., algorithmically useful) properties that its counterpart for Mal'tsev polymorphisms does not have.

Let $\mathcal{M}$ be the strong linear Mal'tsev condition defining minority polymorphisms, i.e.

$$\mathcal{M} \;:\; \begin{aligned} f(y,x,x) &\approx y \\ f(x,y,x) &\approx y \\ f(x,x,y) &\approx y \end{aligned}$$

Given a constraint language Γ, let $\mathcal{P}^c_{\mathcal{M}}(\Gamma)$ be the CSP instance whose solutions are exactly the conservative minority polymorphisms of Γ, as detailed in Section 4.5. For our structural analysis we will assume that for every $R^* \in \Gamma$, $\mathcal{P}^c_{\mathcal{M}}(\Gamma)$ also contains a constraint $C^{R'}_{\mathbf{f(t'_1,t'_2,t'_3)}}$ for every projection R' of R^* and $\mathbf{t'_1}, \mathbf{t'_2}, \mathbf{t'_3} \in R'$. These additional constraints are only needed to facilitate our analysis and will not be required by the algorithm.

A constraint $C = (S, R)$ is *functional* in $x \in S$ if for every valid assignment t of $S \backslash x$ there is at most one value $d \in \mathcal{D}$ such that $(S \backslash x \leftarrow t, x \leftarrow d)$ is an assignment

to S that satisfies C. If two relations R and R' differ only by a permutation of their columns, we write $R \approx R'$. We also remind the reader that if $C = C^{R^*}_{\mathbf{f}(\mathbf{t_1},\mathbf{t_2},\mathbf{t_3})}$ is a constraint of $\mathcal{P}^c_{\mathcal{M}}(\Gamma)$, the kth variable in its scope is $x_{f(\mathbf{t_1}[k],\mathbf{t_2}[k],\mathbf{t_3}[k])}$. Therefore, if $\mathbf{t_1}[k] = \mathbf{t_2}[k]$, the unary constraints will ensure that $x_{f(\mathbf{t_1}[k],\mathbf{t_2}[k],\mathbf{t_3}[k])}$ is ground (i.e. has a singleton domain) with value $\mathbf{t_3}[k]$.

Lemma 8. *Let $\mathcal{P}^c_{\mathcal{M}}(\Gamma) = (\mathcal{X}, \mathcal{D}, \mathcal{C})$ after applying 1-minimality. Let $C = (S, R) \in \mathcal{C}$ be a non-unary constraint and $x \in S$. Either C is functional in x, or $R \approx \mathcal{R}(C[S\backslash x]) \times D(x)$.*

Proof. Let $C = C^{R^*}_{\mathbf{f}(\mathbf{t_1},\mathbf{t_2},\mathbf{t_3})}$ and $x = x_{f(v_1,v_2,v_3)}$. Without loss of generality, we assume that x occurs last in S. First, suppose that there exists $\mathbf{t} \in \mathcal{R}(C[S\backslash x])$ such that $(\mathbf{t}, v_k) \in \mathcal{R}(C)$ for every $v_k \in D(x)$. We will show that every tuple must have the same property as $\mathbf{t}$. Let $\mathbf{t}' \in \mathcal{R}(C[S\backslash x])$ be such that $(\mathbf{t}', v_\alpha) \in \mathcal{R}(C)$ but $(\mathbf{t}', v_\beta) \notin \mathcal{R}(C)$ for some $\{v_\alpha, v_\beta\} \subseteq D(x)$. Then, because of the unary constraints, the constraint $C^{R^*}_{\mathbf{f}((\mathbf{t},v_\alpha),(\mathbf{t},v_\beta),(\mathbf{t}',v_\alpha))}$ has only ground variables in its scope, and its only possible support is $(\mathbf{t}', v_\beta)$. By Lemma 4, $\mathcal{R}(C^{R^*}_{\mathbf{f}((\mathbf{t},v_\alpha),(\mathbf{t},v_\beta),(\mathbf{t}',v_\alpha))}) \subseteq \mathcal{R}(C)$ and hence $(\mathbf{t}', v_\beta) \in \mathcal{R}(C)$, a contradiction. Therefore, such a partial tuple $\mathbf{t}'$ cannot exist and $R \approx \mathcal{R}(C[S\backslash x]) \times D(x)$.

Now, suppose that $D(x) = \{v_1, v_2, v_3\}$ and there exists $\mathbf{t} \in \mathcal{R}(C[S\backslash x])$ such that $(\mathbf{t}, v_k) \in \mathcal{R}(C)$ for exactly two indices k, say 1 and 2. Since C is 1-minimal, there exists $\mathbf{t}'$ such that $(\mathbf{t}', v_3) \in \mathcal{R}(C)$. However, the scope of $C^{R^*}_{\mathbf{f}((\mathbf{t},v_1),(\mathbf{t},v_2),(\mathbf{t}',v_3))}$ contains only ground variables and x; therefore $\mathcal{R}(C^{R^*}_{\mathbf{f}((\mathbf{t},v_1),(\mathbf{t},v_2),(\mathbf{t}',v_3))})$ contains the tuple $(\mathbf{t}', v_k)$ for all $k \in \{1, 2, 3\}$. By Lemma 4 we have $\mathcal{R}(C^{R^*}_{\mathbf{f}((\mathbf{t},v_1),(\mathbf{t},v_2),(\mathbf{t}',v_3))}) \subseteq \mathcal{R}(C)$, and the partial tuple $\mathbf{t}'$ brings us back to the first case.

If no tuples satisfy either of the above two conditions, C is functional in x. □

The key observation in our proof will be that variables with domain size 1 or 2 have very limited interactions with variables with domain size 3 once 1-minimality has been enforced. Given a constraint $C = (S, R)$ in $\mathcal{P}^c_{\mathcal{M}}(\Gamma)$, we denote by $\mathcal{S}_{|1,2}(C)$ the restriction of S to variables with domain size 1 or 2, and by $\mathcal{S}_{|3}(C)$ the restriction of S to variables with domain size 3.

Lemma 9. *Let $\mathcal{P}^c_{\mathcal{M}}(\Gamma) = (\mathcal{X}, \mathcal{D}, \mathcal{C})$ after applying 1-minimality. Let $C \in \mathcal{C}$ and $x \in \mathcal{S}_{|3}(C)$. It is true that $\mathcal{R}(C[\mathcal{S}_{|1,2}(C) \cup x]) \approx \mathcal{R}(C[\mathcal{S}_{|1,2}(C)]) \times D(x)$.*

Proof. Let $C_1 = C[\mathcal{S}_{|1,2}(C)] = (R_1, S_1)$, $C_2 = C[\mathcal{S}_{|1,2}(C) \cup x] = (R_2, S_2)$ and assume that $x = x_{f(v_1,v_2,v_3)}$ occurs last in the scope of C_2. By Lemma 8, either $R_2 = R_1 \times D(x)$ or C_2 is functional in x. If it is functional, then by 1-minimality there exist $\mathbf{t}, \mathbf{t}', \mathbf{t}'' \in R_1$ such that R_2 contains $(\mathbf{t}, v_1)$, $(\mathbf{t}', v_2)$ and $(\mathbf{t}'', v_3)$. Then, the scope of $C' = C^{R^*}_{\mathbf{f}((\mathbf{t},v_1),(\mathbf{t}',v_2),(\mathbf{t}'',v_3))}$ has only ground variables (those corresponding to $\mathcal{S}_{|1,2}(C)$) plus $x_{f(v_1,v_2,v_3)}$. Therefore, there exists $\mathbf{t}^*$ such that $\mathcal{R}(C')$ contains

$(\mathbf{t}^*, v_1)$, $(\mathbf{t}^*, v_2)$ and $(\mathbf{t}^*, v_3)$. By Lemma 4, $\mathcal{R}(C') \subseteq R_2$ and C_2 is not functional in x, a contradiction. □

Lemma 9 only deals with constraints whose scope contains exactly *one* variable with domain size 3. Unfortunately, for k variables it is not completely true that $\mathcal{R}(C[\mathcal{S}_{|1,2}(C) \cup \{x_1, \ldots, x_k\}]) \approx \mathcal{R}(C[\mathcal{S}_{|1,2}(C)]) \times D(x_1) \times \ldots \times D(x_k)$. Let $x_{f(v_1^1, v_2^1, v_3^1)}, \ldots, x_{f(v_1^k, v_2^k, v_3^k)}$ be k variables of the indicator problem. The *index-equality* constraint on these variables has three satisfying assignments: $(v_1^1, \ldots, v_1^k)$, $(v_2^1, \ldots, v_2^k)$ and $(v_3^1, \ldots, v_3^k)$. The next Proposition is the keystone of our proof, and gives the correct generalization of Lemma 9 to an arbitrary number of variables with domain size 3.

Proposition 13. *Let $\mathcal{P}^c_{\mathcal{M}}(\Gamma) = (\mathcal{X}, \mathcal{D}, \mathcal{C})$ after applying 1-minimality and $C \in \mathcal{C}$. There exists $n \geq 0$ and a set of constraints $C^*, C_1, \ldots, C_n$ such that*

$$C = C^* \wedge \left(\bigwedge_{i=1..n} C_i \right)$$

where the scope of C^* is $\mathcal{S}(C)$, the constraints C_i are (possibly unary) index-equalities whose scope are disjoint and cover $\mathcal{S}_{|3}(C)$, and

$$\mathcal{R}(C^*) \approx \mathcal{R}(C[\mathcal{S}_{|1,2}(C)]) \times \Pi_{x \in \mathcal{S}_{|3}(C)} D(x)$$

Proof. We proceed by induction on the size of $\mathcal{S}_{|3}(C)$. Let $k > 0$ and suppose that Proposition 13 is true for all constraints C' such that $|\mathcal{S}_{|3}(C')| \leq k$. Let $C = C^{R^*}_{\mathbf{f(t_1, t_2, t_3)}} = (S, R)$ be a constraint with $|\mathcal{S}_{|3}(C)| = k+1$, and $x \in \mathcal{S}_{|3}(C)$. By Lemma 8, either C is functional in x or $\mathcal{R}(C) = \mathcal{R}(C[S \backslash x]) \times D(x)$. In the latter case, C satisfies Proposition 13 by induction. Therefore, we shall assume that C is functional in x.

By induction, we know that $C[S \backslash x] = C^* \bigwedge_{i=1..n} C_i$. Let $y \in \{1..n\}$ and $Y = \mathcal{S}(C_y)$. Let $\mathbf{v_i}$, $i = 1, 2, 3$ be the three possible assignments to Y. We assume without loss of generality that $x = x_{f(u_1, u_2, u_3)}$ (hence, $D(x) = \{u_1, u_2, u_3\}$) and (Y, x) are the last variables in S. Let $\mathbf{t} \in \mathcal{R}(C[S \backslash \{Y, x\}])$, and define $\phi_{\mathbf{t}} : D(Y) \to D(x)$ such that $\phi_{\mathbf{t}}(\mathbf{v}) = \{u \in D(x) \mid (\mathbf{t}, \mathbf{v}, u) \in \mathcal{R}(C)\}$. We distinguish three cases.

1. $\phi_{\mathbf{t}}$ has range $\{u_i, u_j\}$ for some $i \neq j$. One of these two values, say u_i, has a preimage of size 2. Let $\{\mathbf{v_p}, \mathbf{v_s}\} = \phi^{-1}_{\mathbf{t}}(\{u_i\})$, $\mathbf{v_l} \notin \{\mathbf{v_p}, \mathbf{v_s}\}$, and $\mathbf{t^y_1}, \mathbf{t^y_2}, \mathbf{t^y_3}$ be the permutation of $(\mathbf{t}, \mathbf{v_p}, u_i), (\mathbf{t}, \mathbf{v_s}, u_i), (\mathbf{t}, \mathbf{v_l}, u_j)$ such that $\mathbf{t^y_h}[Y] = \mathbf{v_h}$. The constraint $C^{R^*}_{\mathbf{f(t^y_1, t^y_2, t^y_3)}}$ has only the variables in Y as active variables (i.e. variables with domain size ≥ 2), and by 1-minimality its relation must contain $(\mathbf{t}, \mathbf{v_p}, u_j), (\mathbf{t}, \mathbf{v_s}, u_j), (\mathbf{t}, \mathbf{v_l}, u_j)$. By Lemma 4, R must contain these tuples, a contradiction.

2. $\phi_{\mathbf{t}}$ is bijective. Suppose that there exist i, j such that $i \neq j$ and $\phi_{\mathbf{t}}(\mathbf{v_i}) = u_j$. Let $u_s \notin \{u_j, u_i\}$, $\mathbf{t'} = (\mathbf{t}, \mathbf{v_i}, u_j)$ and $\mathbf{t''} = (\mathbf{t}, \phi_{\mathbf{t}}^{-1}(u_s), u_s)$. Let $\mathbf{t_1^x}, \mathbf{t_2^x}, \mathbf{t_3^x}$ be the permutation of the tuples $\mathbf{t_i}, \mathbf{t'}, \mathbf{t''}$ such that $\mathbf{t_h^x}[x] = u_h$. Recall that $\mathbf{t_i}$ is one of the three tuples associated with the constraint $C = C^{R^*}_{\mathbf{f(t_1,t_2,t_3)}}$, and hence $\mathbf{t_i} \in R^*$, $\mathbf{t_i}[Y] = \mathbf{v_i}$ and $\mathbf{t_i}[x] = u_i$. Then, the constraint $C^{R^*}_{\mathbf{f(t_1^x,t_2^x,t_3^x)}}$ has x as the only active variable in its scope, and for every $u \in D(x)$ its relation must contain the tuple $\mathbf{t^u}$ such that $\mathbf{t^u}[l] = \mathbf{t_i}[l]$ if $l \notin Y \cup \{x\}$, $\mathbf{t^u}[Y] = \phi^{-1}(u_s)$, and $\mathbf{t^u}[x] = u$. Note that at this point, Lemma 4 cannot be applied because $\mathbf{t_i}$ may not belong to $\mathcal{R}(C)$. Let $\mathbf{t_1^a}, \mathbf{t_2^a}, \mathbf{t_3^a}$ be the permutation of $\mathbf{t_i}, \mathbf{t^s}, \mathbf{t'}$ such that $\mathbf{t_h^a}[x] = u_h$. The constraint $C^{R^*}_{\mathbf{f(t_1^a,t_2^a,t_3^a)}}$ has only ground variables in its scope except x, and its relation R' must contain the tuple $\mathbf{t^f}$ such that $\mathbf{t^f}[x] = u_j$ and $\mathbf{t^f}[l] = \mathbf{t''}[l]$ otherwise. However, since $R' \subseteq R$ we have $t_f \in R$, a contradiction. Therefore, if $\phi_{\mathbf{t}}$ is bijective then it must map every $\mathbf{v_i}$ to u_i.

   Now, suppose that there exists a partial tuple $\mathbf{t'}$ such that $\phi_{\mathbf{t'}}$ is not equal to $\phi_{\mathbf{t}}$. By Case 1 and the reasoning above, $\phi_{\mathbf{t'}}$ must map every $\mathbf{v_i}$ to the same value u_p. Let $\{\mathbf{v_i}, \mathbf{v_j}\} = D(Y) \backslash \mathbf{v_p}$. If we denote by $\mathbf{t_1^b}, \mathbf{t_2^b}, \mathbf{t_3^b}$ the permutation of $(\mathbf{t'}, \mathbf{v_j}, u_p)$, $(\mathbf{t'}, \mathbf{v_p}, u_p)$ and $(\mathbf{t}, \mathbf{v_i}, u_i)$ such that $\mathbf{t_h^b}[Y] = \mathbf{v_h}$, the constraint $C^{R^*}_{\mathbf{f(t_1^b,t_2^b,t_3^b)}}$ has only the variables in Y as active variables in its scope, and by 1-minimality its relation must contain the tuple $(\mathbf{t}, \mathbf{v_p}, u_i)$. By Lemma 4, this tuple must belong to R, a contradiction.

   Finally, in this case every tuple must induce an index-equality between Y and x. Therefore, we can add x to the scope of C_y and continue the induction.

3. $\phi_{\mathbf{t}}$ has range $\{u\}$. By Cases 1 and 2, we know that the only situation where the induction may not hold is when $\phi_{\mathbf{t'}}$ is in this case for every partial tuple $\mathbf{t'}$ and every choice of Y. For each $\mathbf{t'} \in \mathcal{R}(C[S \backslash x])$ and index-equality constrained set of variables Y', we define $\mathcal{J}_{Y'}(\mathbf{t'})$ to be $\mathbf{t'}$ plus the set of all tuples that differ from $\mathbf{t'}$ only on the assignment to Y'. By functionality, for each $\mathbf{t'} \in \mathcal{R}(C[S \backslash x])$ we can define $\psi(\mathbf{t'})$ to be the sole value $u \in D(x)$ such that $(\mathbf{t'}, u) \in R$. It is immediate that $\psi(\mathbf{t}^\alpha) = \psi(\mathbf{t}^\beta)$ for each $\mathbf{t}^\alpha, \mathbf{t}^\beta \in \mathcal{J}_{Y'}(\mathbf{t'})$, for any fixed $Y', \mathbf{t'}$. Furthermore, for any two tuples $\mathbf{t}^\alpha, \mathbf{t}^\beta \in \mathcal{R}(C[S \backslash x])$ such that $\mathbf{t}^\alpha[\mathcal{S}_{|1,2}(C)] = \mathbf{t}^\beta[\mathcal{S}_{|1,2}(C)]$, there exists $\mathbf{t_{Y_1}}, \ldots, \mathbf{t_{Y_n}}$ such that $\mathbf{t_{Y_1}} \in \mathcal{J}_{Y_1}(\mathbf{t}^\alpha)$, $\mathbf{t}^\beta \in \mathcal{J}_{Y_h}(\mathbf{t_{Y_n}})$ and for each i, $\mathbf{t_{Y_{i+1}}} \in \mathcal{J}_{Y_i}(\mathbf{t_{Y_i}})$. In other words, starting from $\mathbf{t}^\alpha$ one can obtain $\mathbf{t}^\beta$ by changing the assignments to each Y_i one by one. By transitivity of the equality, this means that $\psi(\mathbf{t}^\alpha) = \psi(\mathbf{t}^\beta)$. Since this is true for any pair $\mathbf{t}^\alpha, \mathbf{t}^\beta$ that share the same values for $\mathcal{S}_{|1,2}(C)$, it follows that $C[\mathcal{S}_{|1,2}(C) \cup x]$ is functional in x, a contradiction with Lemma 9.

□

Theorem 21. *There exists an algorithm that decides in time $O((rlt^3d^3 + rlt^4)$ if a constraint language Γ admits a conservative Mal'tsev polymorphism and outputs one if one exists, where r is the maximum arity, l is the number of relations in Γ, d is the domain size and t is the maximum number of tuples in a relation in Γ.*

Proof. By Lemma 7, we can look for a conservative minority polymorphism instead. The algorithm builds $\mathcal{P}^c_{\mathcal{M}}(\Gamma)$ in time $O(rlt^3)$, where l is the number of relations and t, r are respectively the maximum number of tuples and the maximum arity of a relation. $\mathcal{P}^c_{\mathcal{M}}(\Gamma)$ has $O(lt^3 + d^3)$ constraints and $O(d^3)$ variables. Then, we enforce 1-minimality in time $O(rlt^4)$. By Proposition 13, assigning every variable $x_{f(v_1,v_2,v_3)}$ with domain size 3 to v_1 does not violate any constraint (since it respects index-equalities) and is consistent with every satisfying assignment to the remaining variables. Therefore, we can eliminate every variable with domain size 3. We are left with an instance whose active variables have domain size 2, and if it has a solution its language must have a conservative minority polymorphism. Note that all minority operations coincide on 2-elements domains; therefore, we can rename each domain by $\{0, 1\}$ (arbitrarily) and obtain a CSP instance whose language has the unique Boolean minority polymorphism $m(x, y, z) = x - y + z \bmod 2$. This instance is equivalent to a system of linear equations over GF(2), and any such instance with n variables and m constraints of arity r can be solved in time $O(mnr)$ using iterative methods [126]. The complexity of the whole algorithm is $O(rlt^3d^3 + rlt^4)$. □

To put this result into perspective, the best known algorithm for solving Mal'tev instances has complexity $O(mn^4 + mn^2l^4t^4)$ [60]; therefore even in the case of bounded arity the complexity of the treasure hunt algorithm applied to conservative Mal'tsev polymorphisms is roughly $O(d^3(lt^3 + d^3)(d^{12} + d^6l^4))$, which is a polynomial whose dependency in d has degree 18. Finally, the algorithm of Theorem 21 is extremely simple: model the meta-problem as a CSP, apply 1-minimality and solve the remaining Boolean linear system using your favourite algorithm (although Gaussian elimination is not recommended here due to the sparsity of the system).

4.7.2 Conservative Majority Polymorphisms

As seen in Section 4.7.1, analyzing the structure of the indicator problem for languages of large arities can be tedious. Fortunately we need not do this twice, for languages with majority polymorphisms are *2-decomposable*: each constraint can be replaced by its binary projections without altering the solution set of the instance [93].

It is fairly straightforward to see that if a language Γ has a majority polymorphism, then the indicator problem of its 2-decomposition Γ_2 is equivalent to

the 2-decomposition of the indicator problem of Γ. Let $\mathcal{M}$ be the strong linear Mal'tsev condition that defines majority polymorphisms, i.e.

$$\mathcal{M} \; : \; \begin{array}{l} f(x,x,y) \approx x \\ f(y,x,x) \approx x \\ f(x,y,x) \approx x \end{array}$$

Given a constraint language Γ we denote by $\mathcal{P}^c_{\mathcal{M}}(\Gamma)$ the CSP instance whose solutions are exactly the conservative majority polymorphisms of Γ, as detailed in Section 4.5.

Lemma 10. *If $\mathcal{P}^c_{\mathcal{M}}(\Gamma_2)$ is 1-minimal, the assignment*

$$x_{f(u_1,u_2,u_3)} \leftarrow (u_i \in D(x_{u_1,u_2,u_3}) \mid i \textit{ is minimum})$$

is a solution.

Proof. We first consider $\mathcal{P}^c_{\mathcal{M}}(\Gamma_2)$ *before* 1-minimality is applied. Let $C^{R^*}_{\mathbf{f(t_1,t_2,t_3)}} = (S, R^*)$ be a constraint of $\mathcal{P}^c_{\mathcal{M}}(\Gamma_2)$ with scope $(x_{f(u_1,u_2,u_3)}, x_{f(v_1,v_2,v_3)})$ such that both variables are active (i.e. $|\{u_1,u_2,u_3\}| = |\{v_1,v_2,v_3\}| = 3$, as otherwise the unary majority constraints would force the variable to be ground). Suppose that there exists a pair $i \neq j$ such that $\mathbf{t} = (u_i, v_j) \in R$. Let k be the index such that $k \notin \{i,j\}$ and $(\mathbf{t'_1}, \mathbf{t'_2}, \mathbf{t'_3})$ be the permutation of the tuples $\mathbf{t}, \mathbf{t_i}, \mathbf{t_k}$ such that $\mathbf{t'_1}[2] = v_1$, $\mathbf{t'_2}[2] = v_2$ and $\mathbf{t'_3}[2] = v_3$. Consider the constraint $C^{R^*}_{\mathbf{f(t'_1,t'_2,t'_3)}} = (S', R^*)$. The second variable in S' is $x_{f(v_1,v_2,v_3)}$ and after 1-minimality the first variable will be fixed to the value u_i. Therefore, by Lemma 4, after 1-minimality the constraint $C^{R^*}_{\mathbf{f(t_1,t_2,t_3)}} = (S, R)$ will contain the tuple (u_i, v) for every $v \in D(x_{f(v_1,v_2,v_3)})$. From this we can deduce that, after 1-minimality, for every i we have either $(u_i, v_i) \in R$ or $(u_i, v) \in R$ for every v in the domain of $x_{f(v_1,v_2,v_3)}$. In particular, if i and j are the minimum indices such that both u_i and v_i are in the domains, (u_i, v_j) always belongs to R. □

Theorem 22. *Conservative majority polymorphisms can be detected in time $O(rlt^4)$ in constraint languages with l distinct relations of arity at most r and containing at most t tuples.*

Proof. The algorithm starts by *assuming* that a conservative majority polymorphism exists. We build $\mathcal{P}^c_{\mathcal{M}}(\Gamma)$ and enforce 1-minimality in time $O(rlt^4)$. Since $\mathcal{P}^c_{\mathcal{M}}(\Gamma)$ is equivalent to $\mathcal{P}^c_{\mathcal{M}}(\Gamma_2)$ if it has a solution, the assignment proposed by Lemma 10 is a solution if and only if Γ has a majority polymorphism. □

Again, this algorithm is much more efficient than using Proposition 8, although the improvement is not as overwhelming as for conservative Mal'tsev polymorphisms. Besides, the time complexity of our algorithm is roughly that of *checking*

if a given conservative majority operation is a polymorphism of Γ, so there is little room for improvement. If memory is a problem (as it was in the previous chapter, which implemented Proposition 8 to detect conservative majority polymorphisms), it may be better to build $\mathcal{P}^c_{\mathcal{M}}(\Gamma_2)$ instead of $\mathcal{P}^c_{\mathcal{M}}(\Gamma)$ and then verify that the obtained majority polymorphism of Γ_2 is also a polymorphism of Γ. Ideally, further experiments on partition backdoors should use Theorem 22 instead of Proposition 8 to detect conservative majority polymorphisms.

4.8 Conclusion and Future Research

We have shown that the dichotomy criterion for conservative CSP can be decided in true polynomial time, without any assumption on the arity or the domain size of the input constraint language. This solves an important question on the complexity of c-CSP among the few that remain. On the way, we have also proved that classes of conservative constraint languages defined by linear strong Mal'tsev conditions admitting a semiuniform algorithm always have a tractable meta-problem. This result is a major step towards a complete classification of meta-problems in conservative languages and complements nicely the results of [36]. Finally, observing the inefficiency of our most general algorithms we have developed simple specialized methods to detect conservative majority and conservative Mal'tsev polymorphisms.

A number of open questions remain on meta-problems and uniform algorithms. It is known that Proposition 8 does not hold in general if the linearity requirement on the Mal'tsev condition is dropped, as semilattices are NP-hard to detect even in conservative constraint languages despite having a uniform algorithm [78]. The same happens if the idempotency of the Mal'tsev condition is dropped instead [36]. However, the mystery remains if the requirement for a uniform algorithm is loosened since no tractable idempotent strong linear Mal'tsev condition is known to have a hard meta-problem. This prompts us to ask if our result on conservative constraint languages can extend to the general case. An affirmative answer would be very strong since it would imply that detecting (nonconservative) Mal'tsev polymorphisms is polynomial-time, which is unknown even in the elementary case where the Mal'tsev polymorphism is required to be affine.

Question 2. *Does there exist an idempotent strong linear Mal'tsev condition $\mathcal{M}$ that has a semiuniform polynomial-time algorithm but whose meta-problem is not in P, assuming some likely complexity theoretic conjecture?*

By producing the coloured graph in polynomial time, the algorithm of Theorem 16 would be very helpful in the design of a uniform algorithm that solves

every tractable conservative constraint language (should one exist). An interesting question is whether such an algorithm exists.

Question 3. *Does there exist a uniform polynomial-time algorithm for the class of all tractable conservative constraint languages?*

The successive simplifications [21][4][22] of the proof of the conservative dichotomy suggest that conservatively tractable languages may not be as complicated as they appear, and we believe that the answer to Question 3 is likely to be in the affirmative.

Finally, it is known that detecting a Siggers polymorphism is NP-hard but Theorem 16 states that it is polynomial-time if the polymorphism has to be conservative. It is natural to wonder what happens for *idempotent* Siggers polymorphisms, which is the middle ground between the general and conservative cases. This question might prove tricky because we do not know yet if languages with a Siggers polymorphism are tractable, which limits greatly the reach of methods based on ideas similar to Proposition 8.

Question 4. *What is the complexity of deciding if a constraint language has an idempotent Siggers polymorphism?*

Chapter 5

Kernel-based Propagators: the Vertex Cover Constraint

The combination of intuitive modelling and efficient solving methods has made constraint programming a popular paradigm for solving real-world computational problems. At the heart of its success lies the concept of *global constraints*, which are relations represented algorithmically (through a *propagator*) rather than extensionally. These constraints scale very well to large arities without being overly memory-intensive, and the extensive literature on propagators for common types of global constraints makes up for the loss of a direct access to the list of tuples.

Because a global constraint can be just any NP subproblem, finding a support for a given variable-value pair is potentially NP-hard. For such global constraints, which are quite numerous [13], there are two extreme competing approaches. The most common is to simply decompose the constraint further until GAC can be achieved in polynomial time, which is generally easy to do but may significantly hinder the reasoning made possible by the knowledge of the problem structure. At the extreme opposite, one could decide to enforce GAC anyway by probing every variable-value pair and hope that the increased quality of propagation will make up for its worst-case exponential computational cost. There are numerous intermediate methods, which include using consistency notions weaker than GAC (such as bound consistency) or simply settle for incomplete propagation rules.

Because propagating NP-hard global constraints is all about tradeoffs and good scalability, it is fertile ground for parameterized complexity. A study of the parameterized complexity of global constraints, and of their relevant parameters, has displayed very promising results [12]. For instance, propagating the NVALUE constraint is NP-hard in general [13] and polynomial-time if the domain of each variable is an interval, but a refined analysis using parameterized complexity has shown that propagation is FPT when the parameter is the number of *holes* in the variable domains [12] - a parameter that can be expected to be rather small when

using a tailored search heuristic. This FPT algorithm has been subsequently improved using kernelization [71], which is used as a preprocessing for probing: pick a variable-value pair, and kernelize the problem of deciding if a support exist. In this chapter we introduce a novel family of kernels, called *loss-less kernels*, which are tied to the whole propagation process rather than the subproblems created by probing. A powerful feature of these new kernels is the ability to propagate even in the absence of probing, which becomes optional.

In general, kernelization methods proceed to reduce the problem size by applying dominance rules. From the point of view of global constraints this means that most kernelization algorithms will remove a domain value because they have found another in the same domain that is at least as likely to belong to a tuple, but it does *not* mean that the removed value has no support. This type of inference can thus not be used directly for filtering. Unlike general kernelization, loss-less kernels maintain complete information on all supports and thus define sound propagation rules. Naturally, because we use more restricted dominance rules this comes at the price of a larger bound on the kernel size.

We will introduce three variants of loss-less kernels and investigate the applicability of the strongest of them to the VertexCover constraint, both in theory and practice. Vertex Cover is an ideal ground, because it is

- The flagship problem of the parameterized complexity community, and thus enjoys a great variety of pruning techniques;
- Useful for modelling a variety of other well-known problems, such as Maximum Independent Set, Maximum Clique, and Maximum Common Induced Subgraph with side constraints.
- Mostly unexplored as a global constraint, which makes our implementation the first of its kind.

In contrast with the rest of the thesis, the content of this chapter is largely prospective. We open up a number of research avenues in the design of propagators based on loss-less kernels, but only study the Vertex Cover case and even in this restricted context we leave important questions unanswered. Still, we believe that there is great value in the ideas presented as measured by their potential applications.

5.1 Loss-less Kernels

Let $\overline{\mathbb{Q}} = \mathbb{Q} \cup \{\infty\}$ denote the set of rational numbers with infinity. An n-ary *cost function* over a finite domain $\mathcal{D}$ is a function $\pi : \mathcal{D}^n \to \overline{\mathbb{Q}}$. Throughout the chapter

we will be interested in NP problems that correspond to the decision version of minimizing an input cost function π drawn from a fixed (but infinite) family Π of cost functions.

Π-Minimization-Decision

Input: A set of variables $X = \{x_1, \ldots, x_n\}$ with domains $D(x_1), \ldots, D(x_n) \subseteq \mathcal{D}$, $\pi \in \Pi$ over $\mathcal{D}$ of arity n, $k \in \mathbb{Q}$

Question: Does there exist an assignment $\phi : X \to \mathcal{D}$ such that $\forall x \in X, \phi(x) \in D(x)$ and $\pi(\phi(x_1), \ldots, \phi(x_n)) \leq k$?

Assignments to X that respect the domains and have finite cost are called *feasible*, and *solutions* are feasible assignments of cost less than k. For a given instance I with variable set X, $\sigma(I)$ denotes the minimum cost of a feasible assignment to X (which is less than k if and only if I is a yes-instance), $\mathcal{U}(I)$ denotes the set of all pairs (x, d) with $x \in X$ and $d \in D(x)$, and $\Sigma(I)$ denotes the set of all pairs $(x, d) \in \mathcal{U}(I)$ such that there exists a solution ϕ to I with $\phi(x) = d$ (in the CSP world, this is the set of all variable-value pairs with a support). Depending on the context, we may write k_I instead of k to stress that k belongs to the instance I. We shall assume that the choice of encoding for π is part of the definition of Π, but to preserve membership in NP we will assume that π can be evaluated in polynomial time. For example, Valued CSP is a problem of this kind where π is a sum of smaller cost functions (constraints), each of which is represented using tables of costs. In this case the size of π is linear in the total number of (valued) tuples in the constraints. As another example, the Vertex Cover problem (defined in Example 3) is a problem of this kind where each x_i is a Boolean variable that decides if the vertex i should be included in the cover and π is a cost function that returns ∞ if the assignment to X is not a vertex cover of G, and the number of variables set to `true` otherwise (the cover size). In this case, the size of π is roughly the size of the graph G. Given a fixed family Π of cost functions, we can turn an instance I of Π-Minimization-Decision into a global constraint C_π whose tuples are the solutions to I. The scope of that constraint is $(x_1, \ldots, x_n, K)$, where K is a *cost variable*, and the predicate of that constraint is $\pi(x_1, \ldots, x_n) \leq K$. The purpose of this section is to identify new notions of kernelization for those cost-minimization problems that are relevant to the propagation of their associated constraint.

Classical kernelization for propagating global constraints is typically used in tandem with probing: pick a variable-value pair (x, d), kernelize the problem of deciding if a support exists in C_π and solve it in FPT time using the kernel. This technique requires to solve a number of FPT problems (and compute a number of kernels) that is linear in the scope and domain size just to propagate one constraint, which may be excessively costly from the perspective of the constraint solver. Furthermore, while propagation is FPT it is difficult to associate the whole process

with a unique, clear-cut kernel.

As classical kernels typically use rules to identify which values are relevant to the decision problem (i.e. finding a tuple-solution), we want to define kernels which capture the values that are relevant to propagation. In other words, if the parameter is p we want a subproblem of size $f(p)$ such that enforcing domain consistency on this subproblem allows complete propagation of *all values* in polynomial time. This is much more consistent with the spirit of kernelization, extends smoothly constraints with polynomial-time propagators and yields a propagation algorithm with running time $O(g(p) + |I|^{O(1)}p^{O(1)})$, instead of $O(|I|^{O(1)}g(p))$ for probing plus classical kernelization.

Now that we have identified the foundations of the desired kernelization notion, we can refine it for cost minimization constraints using a recurrent phenomenon that is critical for CSP solving. Let us consider a cost minimization constraint C_π with cost variable K and underlying instance I. Let $\bar{K}$ denote the maximum value of K. If $\sigma(I)$ is much smaller than $\bar{K}$ then C_π is likely to contain a very large number of tuples, and thus propagating this constraint will probably not have a noticeable effect on the variable domains. On the other hand, during backtracking search the domain of K will be gradually reduced and it will eventually get to the point where $\bar{K} \leq \sigma(I) + z$ for a small constant z. This means that the constraint will get *tighter*, in the sense that it will contain less tuples and thus be more likely to perform meaningful propagations. Therefore, it makes sense to bolster the propagation methods when z is identified as small (for instance because we know a lower bound of $\sigma(I)$). From the kernelization perspective, this means that we should prioritize kernels whose size will provably decrease as z gets closer to 0, or even kernels that only exist for small values of z. Thus, instead of devising potentially large kernels that preserve information on all supports for non-kernel variables, we can focus on kernels that preserve information on all supports whose cost is at most z from the optimum cost - at the (very small, as argued above) price of being useful only when the constraint becomes tight.

We will define a hierarchy of three different types of kernels for propagation, which we call *z-loss-less* and will be respectively labelled as *direct*, *weak* and *strong*. Direct z-loss-less kernels, which are the smallest and most general, follow closely the thought process of the above paragraphs.

Definition 24. Let $z \in \overline{\mathbb{Q}}$ and Π be a fixed family of cost functions. A **direct z-loss-less kernelization** of Π-Minimization-Decision parameterized by p is a function that maps in polynomial time each parameterized instance (I, p) of Π-Minimization-Decision to a new parameterized instance (I', p'), such that

(i) There exist computable functions f, g such that $|I'| \leq f(p)$ and $p' \leq g(p)$;

(ii) I is satisfiable if and only if I' is;

(iii) $\mathcal{U}(I') \subseteq \mathcal{U}(I)$;

(iv) There exists a polynomial-time algorithm $\mathcal{A}$ which on input $(I, \sigma(I'))$ computes $\sigma(I)$;

(v) There exists a polynomial-time algorithm $\mathcal{B}$ which, on input $(I, \sigma(I), \Sigma(I'))$ and whenever $\sigma(I) \geq k_I - z$, outputs $\Sigma(I)$.

Weak z-loss-less kernels improve on their "direct" counterpart by allowing polynomial-time propagation of non-kernel values even if only the optimum cost $\sigma(I')$ of the kernel I' is known. This gives a nice flexibility: with n variables and d values in each domain, we can either be satisfied by propagating the $nd - f(p)$ non-kernel values at the cost of solving a single FPT optimization subproblem, or push further and decide to propagate the kernel values as well at the cost of solving a number of subproblems that depends only on the parameter.

Definition 25. Let $z \in \overline{\mathbb{Q}}$ and Π be a fixed family of cost functions. A **weak z-loss-less kernelization** of Π-Minimization-Decision parameterized by p is a function that maps in polynomial time each parameterized instance (I, p) of Π-Minimization-Decision to a new parameterized instance (I', p'), such that

(i) There exist computable functions f, g such that $|I'| \leq f(p)$ and $p' \leq g(p)$;

(ii) I is satisfiable if and only if I' is;

(iii) $\mathcal{U}(I') \subseteq \mathcal{U}(I)$;

(iv) There exists a polynomial-time algorithm $\mathcal{A}$ which on input $(I, \sigma(I'))$ computes $\sigma(I)$;

(v) There exists a polynomial-time algorithm $\mathcal{B}$ which, on input $(I, \sigma(I))$ and whenever $\sigma(I) \geq k_I - z$, outputs $\Sigma(I) \cap (\mathcal{U}(I) \backslash \mathcal{U}(I'))$.

Now, let us analyze these two first definitions. The items (i) and (ii) correspond to classical kernelization. It would be possible to do without (iii) but it is quite valuable in the CSP context: this constraint will be one among many, and this requirement ensures that the kernel makes sense from the perspective of the CSP instance as a whole. To see the necessity for (iv) and (v), recall that our purpose is to devise a type of kernelization that allows complete propagation on all values by solving a single problem on the kernel whenever the problem is tight; in this setting $\mathcal{A}$ will be used to determine if the constraint is tight enough, and if it is the case then $\mathcal{B}$ will perform the actual propagation. These two algorithms $\mathcal{A}$ and $\mathcal{B}$ are designed to work in tandem, with the output of $\mathcal{A}$ being the input of $\mathcal{B}$. So, why distinguish them?

In the upcoming sections we will present weak z-loss-less kernels for Vertex Cover with an additional property that turns out to be highly relevant to propagation: in these kernels, the algorithm $\mathcal{B}$ is capable of performing *partial* propagation if a *lower bound* of $\sigma(I)$ is given in input instead of $\sigma(I)$. This means that invoking $\mathcal{A}$ becomes optional, and hence the kernel becomes useful even if the FPT subproblem is not solved at all. Of course, for this to be meaningful from a theoretical point of view the quality of this partial propagation has to be quantified in precise terms. This is a bit tricky because the propagation is not guaranteed in terms of variables and values, as in weak kernels, but in terms of *convergence* towards complete propagation as the lower bound gets closer to the real value of $\sigma(I)$. Of course, if $\mathcal{B}$ does nothing unless $\sigma(I)$ is given as input it technically converges; but the key observation is that $\mathcal{B}$ cannot do that in general because it is a polynomial-time algorithm and checking if the input is indeed $\sigma(I)$ is likely to be difficult. Therefore, in most cases $\mathcal{B}$ will perform increasingly better propagation as the lower bound gets closer to $\sigma(I)$, even if it is unknown. If an algorithm $\mathcal{B}$ with this property exists, we will say that the kernel is *strong* rather than weak.

Definition 26. Let $z \in \overline{\mathbb{Q}}$ and Π be a fixed family of cost functions. A **strong z-loss-less kernelization** of Π-Minimization-Decision parameterized by p is a function that maps in polynomial time each parameterized instance (I, p) of Π-Minimization-Decision to a new parameterized instance (I', p'), such that

(i) There exist computable functions f, g such that $|I'| \leq f(p)$ and $p' \leq g(p)$;

(ii) I is satisfiable if and only if I' is;

(iii) $\mathcal{U}(I') \subseteq \mathcal{U}(I)$;

(iv) There exists a polynomial-time algorithm $\mathcal{A}$ which on input $(I, \sigma(I'))$ computes $\sigma(I)$;

(v) There exists a polynomial-time algorithm $\mathcal{B}$ which on input (I, l) where $l \in \mathbb{Q}, l \leq \sigma(I)$ outputs a set $\Sigma_l \subseteq \mathcal{U}(I) \backslash \mathcal{U}(I')$ such that

- For every l_1, l_2 with $l_1 \geq l_2$, $\Sigma_{l_1} \subseteq \Sigma_{l_2}$;
- If $\sigma(I) \geq k_I - z$ then $\Sigma_{\sigma(I)} = \Sigma(I) \cap (\mathcal{U}(I) \backslash \mathcal{U}(I'))$.

We will use *loss-less* as a shorthand for ∞-loss-less. There are numerous ways to generalize these definitions - for instance, making z a function of the parameter, omitting (iii), considering kernels for higher-order consistencies that preserve supports for simultaneous assignments to variables - but we feel that these definitions are a nice compromise, being at the same time strong theoretical foundations to build upon and still in touch with the practical side of constraint programming.

Example 10. *Subset minimization problems* ask for a subset S with property $\mathcal{P}$ of a given universe U such that $|S| \leq k$. Those problems are particular cases of our framework, with one Boolean variable for each $u \in U$ and a cost function that returns ∞ whenever the assignment corresponds to a set that does not have the property $\mathcal{P}$, and the number of variables set to `true` otherwise. In this context, Damaschke defined a *full kernel* as a set $W \subseteq U$ that contains the union of all inclusion-minimal solutions [50]. These kernels are useful for enumeration purposes when the property π is $\supset$-closed, i.e. $Y \supset X$ has the property π whenever X has. In this case, full kernels computable in polynomial time are always strong loss-less kernels where $\sigma(I) = \sigma(I')$ and $\mathcal{B}$ does not remove any values unless the input lower bound is exactly k, in which case it no longer finds supports for the value `true` in the domain of non-kernel variables. This is an example for which the convergence of $\mathcal{B}$ to complete propagation is the worst possible, since $\mathcal{B}$ does nothing until the lower bound is equal to k, despite z being infinite. Our own kernels for Vertex Cover will behave very differently, with propagation happening as soon as the lower bound exceeds $k - z$.

Generalizing polynomial-time computable full kernels, *enum-kernels* are special kernels which contain all necessary information to enumerate all solutions with FPT delay [44], and in that aspect they are incomparable with our loss-less kernels which are required to preserve one support (i.e. solution) for each variable-value pair, and the existence of these supports must be decidable in polynomial time (not FPT) from the knowledge of kernel supports. However, because the core idea is relatively similar we can expect many direct loss-less kernels to be enum-kernels as well (and vice versa). Finally, in the very specific context of strong backdoors Samer and Szeider have introduced *loss-free* kernels [127], which are essentially identical to full kernels and should not be confused with our own.

5.2 Vertex Cover

Let us start by introducing some elementary notions of graph theory and relevant notations. An *undirected graph* is an ordered pair $G = (V, E)$ where V is a set of vertices and E is a set of edges, that is, pairs in V. We denote the *neighbourhood* of a vertex v by $N(v) = \{u \mid \{v, u\} \in E\}$, its *closed neighbourhood* $N^+(v) = N(v) \cup \{v\}$ and $N(W) = \bigcup_{v \in W} N(v)$. The *subgraph* of $G = (V, E)$ induced by a subset of vertices W is denoted $G[W] = (W, 2^W \cap E)$. An *independent set* is a set $I \subseteq V$ such that no pair of elements in I is in E. A *clique* is a set $C \subseteq V$ such that every pair of elements in C is in E. A *clique cover* T of a graph $G = (V, E)$ is a collection of disjoint cliques such that $\bigcup_{C \in T} C = V$. A *matching* is a subset of pairwise disjoint edges. Recall that a *vertex cover* of G is a set $S \subseteq V$ such that every edge $e \in E$ is incident to at least one vertex in S, and observe that the

complement of a vertex cover is by definition an independent set. The size of a matching is a straightforward and widely used lower bound to that of any vertex cover because covering the matching itself requires at least one vertex per edge. A more refined lower bound can be derived from clique covers: if $(C_1, \ldots, C_h)$ is a clique cover, then every vertex cover has to contain at least $|C_i| - 1$ vertices of each clique C_i.

We will use the decision version of the Vertex Cover problem to showcase the interest of z-loss-less kernels.

Vertex Cover
Input: A graph $G = (V, E)$, an integer k
Question: Does G have a vertex cover of size at most k?

Vertex Cover is NP-complete [67] and commonly parameterized by the solution size k. The problem is easily seen as FPT via a bounded search tree (see Example 3), and more involved techniques yield an exact FPT algorithm with running time $O(1.2738^k + |V|k)$ [38]. The most recent research on Vertex Cover often focus on stronger parameters that measure the difference between k and easily computable lower bounds [68][102], but we shall keep things simple and use k as the parameter.

5.2.1 Preliminaries: Classical Kernelization

In this section we survey the main existing kernelization methods, and in particular crown-based kernelization which will serve as a baseline for our own contributions. Historically, the first kernelization method discovered for Vertex Cover is *Buss's rule* [28]. The idea is straightforward. If a vertex v has degree $k + 1$, it is clear that v has to belong to every size-k vertex cover since otherwise we would have to include its whole neighbourhood; we can therefore safely remove v from the graph and decrement k by one. If this rule is no longer applicable, then every vertex can cover at most k edges, and therefore no size-k vertex cover exists unless the graph has at most k^2 edges and $k^2 + k$ non-isolated vertices. A more refined kernelization algorithm works using decompositions called *crowns*.

Definition 27. Let $G = (V, E)$ be a graph. A **crown decomposition** of G is a partition (H, W, I) of V such that

(i) I is an independent set;

(ii) There is no edge between I and H;

(iii) There is a matching M between W and I of size $|W|$.

Given a crown decomposition of a graph G, every vertex cover of $G[W \cup I]$ has to be of size at least $|W|$ because of the matching M. Since I is an independent set, taking the vertices of W over those of I in the vertex cover is always a sound choice: they would cover all the edges between W and I at minimum cost, and as many edges in $G[W \cup H]$ as possible. This reduction rule is slightly more difficult to apply than Buss's rule because one has to actually compute a crown decomposition, which is not obviously easy. For this, two competing methods exist: one based on maximal matchings that leaves a residual instance $G[H]$ with $3k$ vertices [2], and one that uses the relaxation of the LP formulation instead and yields a kernel of size $2k$ [117]. The best known kernelization for Vertex Cover leaves a kernel of size $2k - c\log(k)$ by using a reduction to Almost-2-SAT on top of the LP method [107], but the algorithmic cost becomes quickly prohibitive as c increases.

There is a fundamental difference between Buss's rule and crown-based kernelization. Let us turn to the formulation as a cost function minimization problem, with one Boolean variable for each $v \in V$ that decides if v should be in the cover. When Buss's rule can be applied to a vertex v, it belongs to *all* vertex covers of size at most k. This means that the value 0 in the domain of v is inconsistent, and 1 is consistent if and only if a solution exists. After removal (i.e. assignment to 1) of those, every other non-kernel vertex is isolated, and for these the value 1 is consistent if and only if the minimum cost is strictly less than k. Overall, this is a strong ∞-loss-less kernelization. On the other hand, crown decompositions assert that taking the vertices in W in the cover is at least as good as taking those in I, but it may happen that size-k (and even minimum-size) vertex covers contain vertices in I. In other words, we preserve at least one minimum-size cover but information on supports may be lost in the process. This idea is formalized more rigorously in the next proposition, which is definitely not surprising but included for completeness. A full proof based on a reduction from Minimal CSP can be found in Appendix A.

Proposition 14. *Unless $P = NP$, there is no polynomial-time algorithm that takes as input:*

- *A graph G and a crown decomposition (H, W, I) of G,*
- *The minimum size of a vertex cover of $G[H]$,*

- *The list of all vertices of H that belong to all minimum-size vertex covers of $G[H]$, and*
- *The list of all vertices of H that belong to no minimum-size vertex covers of $G[H]$*

and decides if there exists a vertex in W that belongs to all minimum-size vertex covers of G.

From this proposition we can deduce that crown decompositions are unsafe to use for computing loss-less kernels, even for direct ones and with $z = 0$. To resolve this issue, Chlebík and Chlebíková defined special crown decompositions for which every *minimum-size* vertex cover of the bipartite graph (W, I) contains every vertex in W and none in I [39] (we use here (W, I) as a shorthand for $G[W \cup I]$ minus the edges between vertices in W). These crowns are said to be *strong*. An optimal strong crown decomposition can be found in polynomial time using a variant of the LP method, where "optimal" means that H does not have a strong crown decomposition and $|H| \leq 2k$ (thus matching the best known bound for standard crown kernelization). When the upper bound k is exactly the minimum size of a vertex cover, this kernelization method gives simple rules to completely propagate non-kernel vertices: this is a strong 0-loss-less kernelization.

5.2.2 z-loss-less Kernels for Vertex Cover

On the scale of z-loss-less kernelization, Buss's rule and strong crowns correspond to the two extreme cases $z = \infty$ and $z = 0$. The kernel obtained by Buss's rule is large (quadratic in k) but the propagation algorithm $\mathcal{B}$ can be used at any time. In contrast, the strong crown kernel is almost as small as a kernel can be but does not infer anything unless the cost k is exactly the minimum size of a vertex cover. In this section we show that a full hierarchy of intermediate z-loss-less kernelizations exist between these two, and the bound on the size of these kernels is linear in both z and k. To achieve this, we consider the following extension of strong crowns.

Definition 28. Let $G = (V, E)$ be a graph and z be a natural number. A **z-rigid crown decomposition** of G is a partition (H, W, I) of V such that

(i) I is an independent set;

(ii) There is no edge between I and H, and

(iii) Every vertex cover of the bipartite graph (W, I) of size at most $|W| + z$ contains W.

The concept of z-rigidity is illustrated in Figures 5.1 and 5.2. Using our terminology, strong crowns are 0-rigid. In order to find in polynomial time z-rigid crown decompositions, we will make a small detour and consider decompositions that are not quite crowns. A partition (H, W, I) of the vertex set of a graph G is said to be an *almost-crown* if I is an independent set and $N(I) = W$ (technically, an almost-crown is simply an alternative representation of independent sets). The *width* of an almost-crown $\Sigma = (H, W, I)$, denoted by $\delta(\Sigma)$, is the size of a maximum matching between I and W. A crown is then an almost-crown Σ with $\delta(\Sigma) = |W|$. A *subcrown* of an almost-crown (H, W, I) is a crown (H', W', I') such that $W' \subseteq W$ and $I' \subseteq I$.

Proposition 15. *Let $\Sigma = (H, W, I)$ be an almost-crown of a graph $G = (V, E)$ with no isolated vertex. If $|I| > (z+1)\delta(\Sigma)$, then Σ contains a z-rigid subcrown.*

Proof. We proceed by induction on the value of $\delta(\Sigma)$. If $\delta(\Sigma) = 1$, every vertex $i \in I$ has the same (unique) neighbour $v \in W$ because I does not contain isolated vertices. It follows that $W = \{v\}$ and Σ is itself a z-rigid crown since every vertex cover not containing v is of size at least $|I| > z + 1 = z + |W|$. Let $k \geq 1$ and suppose that the claim is true for all almost-crowns Σ' with $\delta(\Sigma') \leq k$. Let $\Sigma = (H, W, I)$ be an almost-crown with no isolated vertex and such that $|I| > (z+1)\delta(\Sigma) = (z+1)(k+1)$. If Σ is not z-rigid, then there exists a cover S of (W, I) of size at most $\delta(\Sigma) + z = k + 1 + z$ that does not contain W. Now, let $W_2 = W \cap S$, $I_2 = I \backslash S$ and observe that $\Sigma_2 = (I_2, W_2, V \backslash \{I_2, W_2\})$ is an almost-crown (the neighbours of I_2 must belong to W_2 because S is a cover).

Suppose that $\delta(\Sigma_2) = k + 1$. Then, there exists a matching M_2 of size $k + 1$ between W_2 and I_2. Because S does not contain W there exists a vertex $v \in W \backslash S$, and since $N(I) = W$ and S is a vertex cover there must exist $i \in I \cap S$ such that $(i, v) \in E$. Then, $M_2 \cup (i, v)$ is a matching of size $k + 2$ of Σ, which contradicts the hypothesis $\delta(\Sigma) = k + 1$. Therefore, $\delta(\Sigma_2) \leq k$.

It remains to prove that $|I_2| > (z+1)\delta(\Sigma_2)$. By construction, we have $|I_2| = |I| - |S \cap I| > (z+1)(k+1) - (k+1+z-|W_2|)$. Since $|W_2| \geq \delta(\Sigma_2)$, we have $|I_2| > (z+1)(k+1)+\delta(\Sigma_2)-(k+1+z) = (z+1)k-k+\delta(\Sigma_2)$. If we let $x = \delta(\Sigma_2)$, the difference $d(x) = |I_2| - (z+1)x$ satisfies

$$d(x) > (z+1)k - k + x - (z+1)x = zk - zx \geq 0$$

hence $|I_2| > (z+1)\delta(\Sigma_2)$ and we can use the induction hypothesis to obtain the claim. □

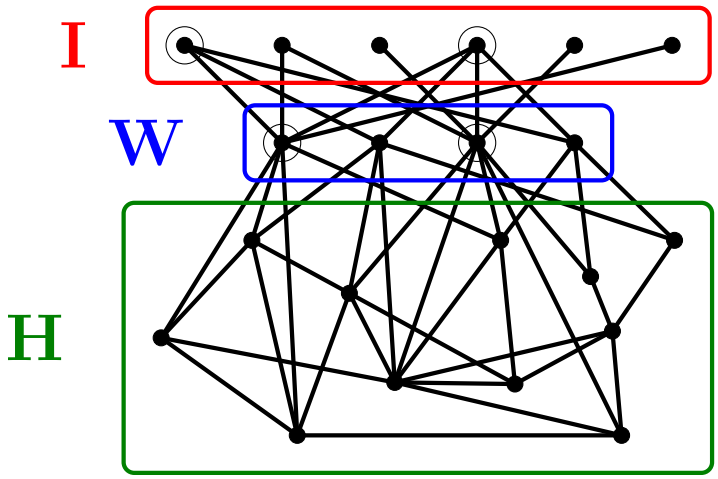

Figure 5.1: A non-rigid crown decomposition. The vertices of a minimum-size vertex cover of $G[W \cup I]$ that does not contain W are circled.

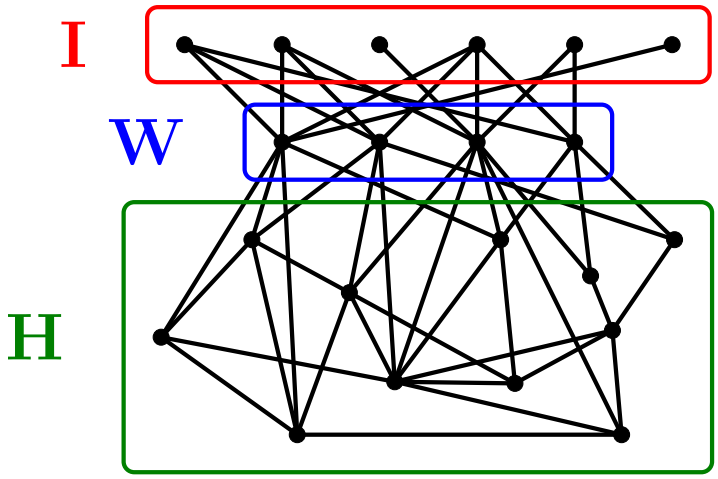

Figure 5.2: With two additional edges between W and I, the crown decomposition of Figure 5.1 becomes 1-rigid.

The above proposition actually gives a polynomial-time algorithm to compute the z-rigid subcrown. Starting from an almost-crown $\Sigma = (H, W, I)$ satisfying the criterion of Proposition 15, for each vertex v in W decide if (W, I) has a vertex cover of size $\leq k + z$ that does not include v (this is polynomial-time using the Hopcroft-Karp algorithm [88] and König's Theorem). If no such vertex cover is found then Σ is z-rigid, and otherwise follow the reasoning of the proof to reduce the problem to a strictly smaller almost-crown.

We now show how a suitable almost-crown can be found in polynomial time. We will use the following lemma, which is adapted from ([2], Theorem 7).

Lemma 11 ([2])**.** *Let (G, k) be an instance of* Vertex Cover*. There exists a polynomial-time algorithm that computes a crown decomposition $\Sigma_N = (H, W, I)$ of G such that $|H| \leq 2(k - |W|)$ and $G[H]$ has no 0-rigid crown decomposition.*

Theorem 23. *Given an instance (G, k) of* Vertex Cover *and an integer $z \leq k$, a strong z-loss-less kernel of (G, k) with at most $(z+2)k$ vertices can be computed in polynomial time.*

Proof. We remove all isolated vertices from G and compute the crown decomposition $\Sigma_N = (H, W, I)$ of G using Lemma 11. Because every z-rigid crown decomposition is also 0-rigid, every z-rigid crown in G must be a subcrown of Σ_N. If $|I| > (z + 1)|W|$, we can use Proposition 15 to find a z-rigid crown and reduce the problem further. Otherwise, we have $|G| = |H| + |W| + |I| \leq 2(k - |W|) + |W| + (z + 1)|W| = z|W| + 2k \leq (z + 2)k$ and the claim follows. □

For $z = 0$, we obtain a kernel of size $2k$ which matches the bound obtained by Chlebík and Chlebíková using more specialized methods. The key strength of their technique is a guarantee that the kernel does not have a strong crown decomposition, which we fail to achieve for z-rigid crowns in general. Such an improvement is left for future research, although we do not believe that very different methods are needed.

5.2.3 The VertexCover Constraint

We consider the Vertex Cover problem as a constraint over two variables: an *integer* variable K to represent the bound on the size of the vertex cover, and a *set variable* S to represent the cover itself. The former takes integer values in a *domain* $D(K)$ which minimum and maximum values are denoted $\underline{K}$ and $\overline{K}$, respectively. The latter takes its values in the sets that are supersets of a *lower bound* $\underline{S}$ and subsets of an *upper bound* $\overline{S}$. Moreover, the domain of a set variable is also often constrained by its cardinality given by an integer variable $|S|$. We consider a constraint on these two variables and whose predicate is the Vertex Cover problem on the graph $G = (V, E)$ given as a parameter:

Definition 29 (VertexCover constraint)**.**
VertexCover$[G](K, S) \iff |S| \leq K \ \& \ \forall \{v, u\} \in E, \ v \in S \vee u \in S$

The use of a set variable S is purely notational; in practice S will be implemented as an array of Boolean variables, so the predicate of the the constraint falls within the framework of cost function minimization. The propagation mechanism will make essential use of z-loss-less kernels, lower bound computations plus *witness pruning*, a novel pruning rule that yields no theoretical guarantees but appears to be useful in practice. This last rule is based on the following observation.

Observation 2. *If S is a minimum vertex cover of $G = (V, E)$ such that there exists $v \in S, J \subseteq N(v) \setminus S$ with $N(J) \subseteq N^+(v)$ then any vertex cover of G either contains v or at least $|S| + |J| - 1$ vertices.*

Proof. Let $v \in S$ be a vertex in an optimal vertex cover S. Consider $J \subseteq N(v) \setminus S$ such that $N(J) \subseteq N^+(v)$. Suppose there exists a vertex cover S' such that $|S'| < |S| + |J| - 1$ and $v \notin S'$. S' must contain every node in $N(v)$ and hence in J. However, we can build a vertex cover of size at most $|S| - 1$ by replacing J by v, since $V \setminus S$ and thus J are independent sets. □

In particular, if $|S|$ is at least $k - |J|$ then the node v belongs to every vertex cover. This rule is applicable whenever a minimum-size vertex cover is known, which will happen quite often as a byproduct of lower bound computation. With every necessary ingredient gathered, we can now describe the general skeleton of our propagation algorithm.

Algorithm 2 takes as input the set variable S standing for the vertex cover, an integer variable K standing for the cardinality of the vertex cover, and five parameters: the graph $G = (V, E)$, a "witness" vertex cover ω initialised to V, and three integers λ, z_m and δ.

The pruning in Line 1 is a straightforward application of the definition: $V \setminus \overline{S}$ is the set of vertices that are excluded from the cover, and the neighbourhood of these vertices must belong to any vertex cover that respects the current domains. Then, in Line 2, we apply the ∞-loss-less kernelization; in our case, H^b is the Buss kernel, and every non-kernel vertex is either forced to belong to the cover (F^b) or isolated (R^b).

The parameter w is a known vertex cover (the "witness") that is kept between calls to the propagation algorithm to avoid redundant computations, and z_m is the gap limit for computing z-loss-less kernels. If Condition 3 fails, there exists a vertex cover $\omega \cup \underline{S}$ of size strictly less than $\overline{K} - z_m$ and we cannot propagate using z-less-less kernels for $z \leq z_m$. Otherwise, we attempt on line Line 6 to compute a new witness using a classical kernel (Line 4) and a bruteforce algorithm `VertexCover`. To avoid unnecessary computations we stop the algorithm when we

Algorithm 2: `PropagateVertexCover`$(S, K, G = (V, E), \lambda, \omega, z_m, \delta)$

$\underline{S} \leftarrow \underline{S} \cup N(V \setminus \overline{S})$;
$(H^b, F^b, R^b) \leftarrow \infty$-`LossLessKernel`$(G[\overline{S} \setminus \underline{S}])$;
if $\omega \not\subseteq \overline{S} \ \vee \ |\omega \cup \underline{S}| \geq \overline{K}$ **then**
    $(H^k, F^k) \leftarrow$ `Kernel`(H^b);
    if $\lambda > 0$ **then**
        $\omega \leftarrow F^b \cup F^k \cup$ `VertexCover`(H^k, λ);
    if ω *is optimal* **then**
        $\underline{K} \leftarrow |\omega|$;
    else
        $\underline{K} \leftarrow \max(\underline{K}, |F^b| + |F^k| +$ `LowerBound`$(H^k))$;
if $\underline{K} \geq \overline{K} - z_m$ **then**
    $z \leftarrow \overline{K} - \underline{K}$;
    $(H^z, F^z, R^z) \leftarrow z$-`LossLessKernel`$(G[\overline{S} \setminus \underline{S}])$;
    $\underline{S} \leftarrow \underline{S} \cup F^z$;
    if $\underline{K} = \overline{K}$ **then**
        $\overline{S} \leftarrow \overline{S} \setminus (R^z \cup R^b)$;
if ω *is optimal* $\&$ $\overline{K} - \underline{K} \leq \delta$ **then**
    $\underline{S} \leftarrow$ `WitnessPruning`(G, ω, δ);
$\underline{S} \leftarrow \underline{S} \cup F^b$;

find a vertex cover whose size is stricly smaller than $\bar{K} - z_m$, or when the branch and bound has reached λ backtracks. In the first case, we know that z-loss-less kernels with $z \leq z_m$ cannot be used. The second stopping condition is simply used to control the amount of time spent within the bruteforce procedure. If the search tree is fully explored, we can conclude that the witness cover is minimum and therefore we can derive a valid lower bound (Line 8). Otherwise, we simply use the lower bound computed at the root node by `VertexCover`, denoted `LowerBound` in Line 10 - typically, a clique cover computed using a heuristic.

If the lower bound is tight, then we can apply the pruning from z-rigid crowns as described in Section 5.2.2. Algorithm z-`LossLessKernel` returns a triple H^z, F^z, R^z where H^z is the z-loss-less kernel, F^z are vertices that must belong to every vertex cover of size at most z from the minimum possible, and R^z is a set of vertices that are isolated after removal of F^z (i.e. that belonged to the independent set part of a z-rigid crown). The set F^z allows immediate pruning, but the vertices in R^z (and R^b, which are vertices isolated by Buss's rule in Line 2) can only be pruned when $\underline{K} = \bar{K}$ (Line 16). In Line 18 we apply the pruning corresponding to Observation 2. Because this operation is quite costly, here we have again an integer δ which limits the size of the subsets J to which Observation 2 is applied. Finally, we apply the pruning on the lower bound of S corresponding to the forced nodes computed by ∞-`LossLessKernel`.

5.3 Experiments

We experimentally evaluated Algorithm 2 on the following problem, in which we want to find a minimum vertex cover that is evenly distributed over a given partition of the vertices.

Balanced Vertex Cover

Input: A graph $G = (V, E)$, a partition $(V_1, \ldots, V_c)$ of V, two integers k and b

Question: Does there exist a vertex cover S of size at most k and such that $\forall (i, j), \left| |V_i \cap S| - |V_j \cap S| \right| \leq b$?

For instance, the vertex cover may represent a set of machines to shut down in a network so that all communications are interrupted. In this case, one might want to avoid shutting down too many machines of the same type, or same client, or in charge of the same service, etc. By varying the integer b, we can control the similarity of the problem to pure minimum vertex cover. This problem can be easily modelled as a CSP with a VertexCover constraint and a few side constraints ensuring that the balance criterion is met.

5.3.1 Implementation

We evaluated the performance of Algorithm 2 with λ set to 5000, z_m set to 0 and δ set to 2. The choice of 0 for z_m may seem odd because that means we restrict ourselves to strong crowns only. This decision is motivated by the belief that our kernelization algorithm is too crude to be useful in practice: the method will not do anything unless the number of non-isolated vertices is at least $(z + 2)$ times k, which is only true for very specific graphs (especially if z is large). For instance, the method is not even guaranteed to find the vertices with $z+1$ degree-1 neighbours. Improving the method so that it guarantees a kernel with no z-rigid crown decomposition would solve this problem, but for now our results are purely theoretical.

Our bruteforce algorithm for VERTEX COVER uses clique covers as a lower bound at each node, and a simple dominance rule (a vertex whose neighbourhood is a clique can be removed) that is applied until a fixed point is reached. This implementation is a compromise between powerful branch-and-bound algorithms, which spend considerable time at each node but excel at solving difficult problems, and streamlined algorithms which are very efficient for small problems but do not scale well. The classical kernelization is the straightforward Nemhauser-Trotter $2k$ kernel, which requires solving the relaxation of the VERTEX COVER basic ILP formulation. We solve it the usual way, by reducing the problem to a maximum matching on a certain bipartite graph. We refer the reader to [1] for more details. For 0-loss-less kernelization, we follow closely the algorithm sketched in ([39], Lemma 7). Our implementation is very incremental; if G has seen little change between two calls to Algorithm 2, the kernels (both classical and 0-loss-less) are not computed from scratch in the second call.

We compared 5 different implementations of the VERTEXCOVER constraint, which correspond to gradually activating the features of our propagation algorithm:

- **Decomposition** is a simple decomposition in 2-clauses and a cardinality constraint. This would be the most direct model of the problem as a CSP.

- **Clique Cover** uses only Buss kernelization, plus a clique cover to compute a lower bound and detect infeasibility. No 0-loss-less kernels, no witness cover, and no witness pruning.

- **Kernel Pruning** uses 0-loss-less kernels in addition to the **Clique Cover** approach. No witness is computed, and witness pruning is not used. This corresponds to Algorithm 2 with $z_m = 0$, $\lambda = 0$ and $\delta = -1$.

- **Kernel & Witness** is Algorithm 2 with $z_m = 0$, $\lambda = 5000$ and $\delta = -1$, i.e. a witness is maintained but witness pruning is not used.

- **VertexCover** is Algorithm 2 with $z_m = 0$, $\lambda = 5000$ and $\delta = 2$.

5.3.2 Methodology

We used a range of graphs from the `dimacs` and `snap` repositories. `dimacs` graphs meant for the Maximum Clique problem are complemented in order to effectively turn Balanced Vertex Cover into a clique problem with side constraints. The density of the `dimacs` graphs, once complemented, range from moderate to high. The `snap` graphs represent peer-to-peer and collaboration networks and tend to be relatively sparse but very large (between 5242 and 26518 vertices).

For each graph $G = (V, E)$, we post a VertexCover constraint on the set variable $\emptyset \subseteq S \subseteq V$. Then, we compute uniformly at random a balanced 4-partition $\{s_1, s_2, s_3, s_4\}$ of the vertices and we post the following constraint:

$$\max(\{|s_i \cap S| \mid 1 \leq i \leq 4\}) - \min(\{|s_i \cap S| \mid 1 \leq i \leq 4\}) \leq b$$

We then ask the solver to find a solution which minimizes $|S|$. For each graph, we generated 3 instances for $b \in \{0, 4, 8\}$ denoted "tight", "medium" and "loose" respectively. However, the classes `p2p` and `ca-` are much too large for these values to make sense. In this case we used three ratios $0.007, 0.008$ and 0.009 of the number of nodes instead. We compared all 5 methods described in Section 5.3.1, implemented in Mistral [83] and ran on CORE I7 processors with a time limit of 5 minutes for each instance. Because we are only interested in comparing propagation algorithms, we used a lexicographic branching heuristic.

5.3.3 Results

The results of these experiments are reported in Table 5.1 (for `dimacs`) and Table 5.2 (for `snap`). For `dimacs` instances, we have clustered the instances by classes whose cardinality is given in the first column to improve readability[1]. These classes are ordered from top to bottom by decreasing ratio of minimum vertex cover size over number of nodes.

We report four values for each class (or single instances for `snap`) and each method: '#s' is the number of instances of the class that were not solved to optimality, 'gap' is the average gap with respect to the smallest vertex cover found, 'cpu' and '#nd' are mean CPU time in seconds and number of nodes visited, respectively, until finding the best solution. Notice that CPU times and number of nodes are then only comparable when the objective values (gaps) are equal. We colour the tuples (#s, gap, cpu, #nd) that are lexicographically minimum for each class[2].

[1] Detailed results can be found in Appendix C

[2] With a "tolerance" of 1s and 1% nodes.

	Decomposition				Clique Cover				Kernel Pruning				Kernel & witness				VertexCover			
	#s	gap	cpu	#nd	#s	gap	cpu	#nd	#s	gap	cpu	#nd	#s	gap	cpu	#nd	#s	gap	cpu	#nd
balancing constraint: tight																				
3 kel	2	2.00	9.7	0.4M	2	2.00	10.6	0.2M	2	2.00	9.1	0.2M	2	2.00	26.6	0.1M	2	2.00	41.0	0.1M
15 p_h	12	5.73	8.6	0.5M	10	5.20	15.6	1.1M	11	5.20	11.2	0.6M	11	4.67	27.7	0.4M	11	4.67	28.8	0.4M
12 bro	9	3.67	0.1	11K	9	3.67	0.1	4K	9	3.67	0.1	3K	9	3.67	0.1	2K	9	3.67	0.2	2K
4 joh	1	0.00	0.1	10K	1	0.00	0.0	1K	1	0.00	0.0	1K	1	0.00	0.0	971	1	0.00	0.0	937
15 san	15	10.87	12.2	1.8M	11	9.80	13.3	1.9M	11	9.80	13.7	1.1M	11	9.80	10.8	0.6M	11	9.80	12.4	0.6M
7 c-f	3	10.29	0.2	9K	3	10.29	0.2	18K	3	10.29	0.1	7K	3	10.29	0.1	7K	3	10.29	0.1	7K
6 ham	4	9.00	26.3	2.2M	3	9.00	3.1	0.3M	3	9.00	3.4	0.2M	3	9.00	5.1	0.2M	3	9.00	5.1	0.2M
32 gra	29	40.47	24.6	2.5M	28	40.47	19.8	3.5M	28	39.22	17.6	2.0M	28	40.47	18.9	1.5M	28	41.22	9.8	0.5M
4 man	3	91.00	1.1	33K	3	91.00	1.1	51K	3	91.00	0.9	31K	3	91.00	1.4	31K	3	91.00	1.5	30K
5 mul	5	8.40	7.2	1.8M	4	7.60	41.4	6.3M	3	7.60	24.6	2.1M	3	7.60	25.1	2.1M	3	7.60	19.2	1.7M
3 fps	3	105.00	0.1	7K	3	103.67	40.5	4.2M	3	103.67	56.0	3.2M	3	103.67	61.0	3.2M	3	103.67	14.9	0.8M
3 zer	3	44.67	11.9	2.6M	3	44.67	11.4	1.6M	3	44.67	14.7	1.4M	3	44.67	13.9	1.4M	3	44.67	8.2	0.9M
3 ini	3	191.33	57.5	6.1M	3	191.33	72.7	6.1M	3	191.33	82.3	6.1M	3	191.33	82.6	6.1M	3	191.33	82.6	6.1M
balancing constraint: medium																				
3 kel	2	1.67	24.1	1.2M	2	0.67	35.9	1.0M	2	0.67	54.7	1.0M	2	0.00	32.8	2K	2	0.00	32.1	2K
15 p_h	12	3.07	21.5	1.2M	10	1.27	24.3	0.7M	11	1.27	34.4	0.6M	10	0.87	18.6	60K	10	0.87	18.8	59K
12 bro	9	0.83	15.6	1.9M	8	0.17	17.8	1.0M	8	0.17	25.4	1.0M	8	0.17	23.9	451	8	0.17	22.2	450
4 joh	1	0.00	0.0	11	1	0.00	0.0	11	1	0.00	0.0	11	1	0.00	0.0	11	1	0.00	0.0	11
15 san	15	8.33	35.6	5.0M	7	2.67	30.4	2.4M	7	2.73	33.5	1.6M	7	1.53	42.8	0.4M	7	1.53	43.1	0.4M
7 c-f	3	4.14	0.0	40	3	4.14	0.0	40	3	4.14	0.0	40	3	4.14	0.0	40	3	4.14	0.0	40
6 ham	4	4.67	0.2	53K	2	4.67	0.0	1K	2	4.67	0.0	360	2	4.67	0.0	359	2	4.67	0.0	359
32 gra	26	29.28	32.8	2.7M	22	26.50	21.9	2.2M	22	24.44	23.8	1.4M	22	24.25	21.5	0.9M	22	24.25	23.0	0.9M
4 man	3	89.00	29.3	1.3M	3	88.75	44.6	1.6M	3	88.75	21.1	0.6M	2	88.50	29.6	0.6M	2	88.50	33.4	0.6M
5 mul	5	1.20	0.3	61K	1	1.20	0.0	1K	1	1.20	0.0	682	1	1.20	0.0	682	1	1.20	0.0	560
3 fps	3	103.00	0.0	250	1	102.67	0.0	429	1	102.67	0.0	404	1	102.67	0.0	404	1	102.67	0.0	261
3 zer	3	3.33	37.4	8.2M	1	3.00	11.6	1.5M	1	3.00	25.4	1.5M	1	3.00	14.5	1.5M	1	3.00	14.5	1.5M
3 ini	3	189.00	0.6	65K	1	189.00	0.0	4K	1	189.00	0.0	3K	1	189.00	0.0	3K	1	189.00	0.0	3K
balancing constraint: loose																				
3 kel	2	1.67	43.3	1.8M	2	0.67	20.1	0.6M	2	0.67	30.4	0.6M	2	0.00	27.7	447	2	0.00	28.0	419
15 p_h	12	2.40	20.6	1.2M	10	0.73	32.1	1.0M	11	0.73	47.1	1.0M	9	0.27	18.0	3K	9	0.27	18.0	3K
12 bro	9	0.67	16.2	1.9M	8	0.00	10.6	0.7M	8	0.00	15.8	0.7M	8	0.00	13.6	264	8	0.00	13.6	264
4 joh	1	0.00	0.0	11	1	0.00	0.0	11	1	0.00	0.0	11	1	0.00	0.0	11	1	0.00	0.0	11
15 san	15	8.20	28.1	4.0M	7	2.13	40.9	2.3M	7	2.27	29.6	1.4M	5	0.27	36.8	3K	5	0.27	36.8	3K
7 c-f	2	0.71	0.0	1K	0	0.00	0.6	98K	0	0.00	0.1	7K	0	0.00	0.2	7K	0	0.00	0.2	7K
6 ham	4	2.00	0.0	118	2	2.00	0.0	118	2	2.00	0.0	118	2	2.00	0.0	118	2	2.00	0.0	118
32 gra	23	18.97	29.4	1.9M	17	12.84	12.9	0.8M	17	12.06	15.6	0.5M	17	11.56	14.5	66K	17	11.50	17.3	0.1M
4 man	3	88.50	38.1	0.8M	3	88.50	8.5	0.3M	3	87.75	30.6	0.8M	2	87.00	43.5	0.8M	2	86.50	52.4	0.8M
5 mul	5	0.00	0.0	93	0	0.00	0.0	92	0	0.00	0.0	91	0	0.00	0.0	91	0	0.00	0.0	91
3 fps	3	1.67	1.1	0.1M	1	1.00	12.6	1.0M	1	1.00	13.4	0.5M	1	1.00	13.9	0.5M	1	0.67	53.5	2.1M
3 zer	3	2.00	0.2	48K	0	1.00	2.6	0.4M	0	1.00	2.7	0.2M	0	1.00	2.6	0.2M	0	1.00	0.6	58K
3 ini	3	0.67	6.9	0.5M	0	0.00	20.9	1.3M	0	0.00	28.9	1.0M	0	0.00	31.9	1.0M	0	0.00	11.6	0.5M

Table 5.1: Comparison of approaches on Balanced Vertex Cover: `dimacs`

	Decomposition				Clique Cover				Kernel Pruning				Kernel & witness				VertexCover			
	#s	gap	cpu	#nd	#s	gap	cpu	#nd	#s	gap	cpu	#nd	#s	gap	cpu	#nd	#s	gap	cpu	#nd
balancing constraint: tight																				
ca-astroph	1	26	1.1	6K	1	23	0.8	7K	1	1	157.9	32K	1	3	178.9	27K	1	0	144.5	26K
ca-condmat	1	16	1.7	9K	1	13	3.5	11K	1	1	175.4	37K	1	3	152.7	30K	1	2	151.5	29K
ca-grqc	1	6	0.1	2K	0	0	1.5	4K	0	0	0.9	3K	0	0	0.9	3K	0	0	0.9	3K
ca-hepph	1	8	154.6	0.7M	1	5	170.8	1.2M	1	7	158.9	0.9M	1	7	173.6	0.9M	1	7	177.6	0.9M
ca-hepth	1	16	0.3	4K	1	4	1.4	9K	1	0	3.2	9K	0	0	5.1	9K	0	0	6.0	9K
p2p-gnutella04	1	66	0.5	6K	1	21	9.3	36K	0	0	4.8	13K	0	0	5.0	13K	0	0	5.9	13K
p2p-gnutella06	1	36	0.3	5K	1	24	1.6	12K	0	0	1.7	8K	0	0	1.8	8K	0	0	2.0	8K
p2p-gnutella08	1	34	0.2	4K	1	34	0.1	4K	1	34	0.1	4K	1	34	0.1	4K	1	34	0.1	4K
p2p-gnutella09	1	25	0.3	5K	1	25	0.1	5K	1	25	0.1	5K	1	25	0.1	5K	1	25	0.1	5K
p2p-gnutella24	1	32	3.8	21K	1	10	169.2	58K	0	0	7.1	25K	0	0	8.5	25K	0	0	9.1	25K
balancing constraint: medium																				
ca-astroph	1	26	1.1	6K	1	23	0.7	7K	1	1	155.2	32K	1	3	179.9	27K	1	0	138.5	26K
ca-condmat	1	16	1.7	9K	1	13	3.5	11K	1	1	178.6	37K	1	3	153.8	30K	1	2	152.5	29K
ca-grqc	1	6	0.1	2K	0	0	1.5	4K	0	0	0.9	3K	0	0	0.9	3K	0	0	0.9	3K
ca-hepph	1	8	0.5	4K	1	3	4.5	8K	0	0	25.4	11K	0	0	31.0	11K	0	0	22.6	9K
ca-hepth	1	16	0.3	4K	1	4	1.4	9K	1	0	3.0	9K	0	0	5.1	9K	0	0	5.6	9K
p2p-gnutella04	1	66	0.5	6K	1	21	9.4	36K	0	0	4.8	13K	0	0	5.0	13K	0	0	6.0	13K
p2p-gnutella06	1	36	0.3	5K	1	24	1.6	12K	0	0	1.7	8K	0	0	1.8	8K	0	0	2.0	8K
p2p-gnutella08	1	22	0.2	4K	1	22	0.1	4K	1	22	0.1	4K	1	22	0.1	4K	1	22	0.1	4K
p2p-gnutella09	1	21	0.3	5K	1	2	12.4	20K	0	0	3.0	8K	0	0	1.9	6K	0	0	2.0	6K
p2p-gnutella24	1	32	3.8	21K	1	10	168.1	58K	0	0	7.1	25K	0	0	8.5	25K	0	0	9.1	25K
balancing constraint: loose																				
ca-astroph	1	26	1.1	6K	1	23	0.7	7K	1	1	162.4	32K	1	3	177.8	27K	1	0	140.8	26K
ca-condmat	1	16	1.7	9K	1	13	3.4	11K	1	1	174.4	37K	1	3	155.0	30K	1	1	177.1	31K
ca-grqc	1	6	0.1	2K	0	0	1.5	4K	0	0	0.9	3K	0	0	0.9	3K	0	0	0.8	3K
ca-hepph	1	8	0.5	4K	1	3	5.3	8K	0	0	29.2	11K	0	0	30.8	11K	0	0	21.4	9K
ca-hepth	1	16	0.3	4K	1	4	1.4	9K	1	0	3.0	9K	0	0	5.2	9K	0	0	6.1	9K
p2p-gnutella04	1	66	0.5	6K	1	21	9.5	36K	0	0	4.8	13K	0	0	5.0	13K	0	0	6.1	13K
p2p-gnutella06	1	36	0.3	5K	1	24	1.6	12K	0	0	2.0	8K	0	0	1.8	8K	0	0	2.0	8K
p2p-gnutella08	1	10	0.2	4K	1	10	0.1	4K	1	10	0.1	4K	1	10	0.1	4K	1	10	0.1	4K
p2p-gnutella09	1	21	0.3	5K	1	2	12.3	20K	0	0	3.0	8K	0	0	1.9	6K	0	0	1.9	6K
p2p-gnutella24	1	32	3.8	21K	1	10	167.9	58K	0	0	7.1	25K	0	0	8.6	25K	0	0	9.2	25K

Table 5.2: Comparison of approaches on Balanced Vertex Cover: `snap`

Let us take a look at Table 5.1 first. The shift of coloured cells from left to right when going from top to bottom within each subtable means that our methods are more effective on instances with smaller vertex covers, which was to be expected. We can also observe another shift of coloured cells from left to right when moving from a subtable to the next. This was also an expected outcome since the pruning on this constraint becomes more prevalent when the problem is closer to pure Vertex Cover. On these families, every reasoning step (clique covers, 0-loss-less kernels, lower bound from the witness and pruning from the witness) improves the overall results but it is difficult to single out one step in particular as more impactful than the others. It should be noted that many instances from the `dimacs` repository are adverse to our methods as they tend to have very large vertex covers (up to 98% of the nodes).

The results on `snap` presented in Table 5.2 are more impressive. These graphs are quite sparse, so this was expected. We observe the same shifts in coloured cells but this time there is a sharp improvement in the objective function (gap) when a clique cover lower bound is used instead of a decomposition, and then another one when 0-loss-less kernels are computed for pruning. The pruning from 0-loss-less kernels has also a dramatic effect on the number of instances solved to optimality. Finally, the other features (witness computation and witness pruning) improve moderately the solving times and help proving optimality for a few more instances.

5.4 Conclusion and Future Research

Observing that standard kernelization does not quite meet the needs of constraint propagation problems, we have introduced *loss-less kernels* as a mean to reduce the whole propagation process to a subproblem whose size is only a function of the parameter. While the general idea can potentially be applied to any parameterized constraint, we focused on cost-minimization constraints and defined specialized variants which are guaranteed to perform increasingly better propagation as the constraint becomes tighter, i.e. the cost of every tuple is at most a small constant z from the optimum.

We investigated this new perspective of kernelization on the case study of Vertex Cover. Partially extending the results of Chlebík and Chlebíková [39], we showed that the standard crown kernelization techniques can be adapted to compute z-loss-less kernels with at most $(z + 2)k$ vertices for every z. We also note that our z-loss-less kernels preserve much more information than necessary as they actually allow FPT-delay enumeration of vertex covers with at most z vertices above the optimum. This suggests that either our techniques can be refined further, or the difference between loss-less kernels and enumeration kernels is thinner than expected for some reason. Overall, z-loss-less kernelization opens up a number of new research avenues, especially if other problems than Vertex Cover are considered.

We have designed a propagator for the VertexCover constraint based on z-loss-less kernels, and implemented a simplified version in the Mistral CSP solver. The experiments are quite promising as the propagator beats consistently a decomposed version of the constraint even on the most adverse instances. In particular, the pruning from loss-less kernels on the graphs with small vertex covers has had a quite spectacular impact. Finally, this propagator still has a lot of room for improvement and would especially benefit from advances in z-loss-less kernels for $z > 0$.

Chapter 6

Conclusion

The subject of this dissertation is easy to summarize: *find ways, if they exist, to bring theory closer to practice in constraint satisfaction problems.* Because the theoretical literature on CSP is so extensive and so disconnected with the reality of CSP solvers, this subject is immensely vast. There is a multitude of possible approaches, and we have no doubt that many other researchers would have opted for a completely different route to take on this challenge (see e.g. [116][96]).

Our contributions share a common philosophy: find new theoretical questions of interest from the practical limitations of tractability results. In Chapter 3, we observed that tractable classes would probably not appear often in practice and investigated the parameterized complexity of strong backdoor detection. In Chapter 4, we observed that the complexity of membership tests for many tractable classes is unknown and developed new polynomial-time algorithms for that purpose. In Chapter 5, we observed that standard kernelization poorly fits the context of constraint propagation and defined the loss-less variant.

We believe these contributions to be significant advances. The complexity classification of backdoor detection given in Chapter 3 covers all the natural tractable classes, the novel algorithmic techniques for meta-problems presented in Chapter 4 are vastly more sophisticated than existing techniques, and the loss-less kernelization method introduced in Chapter 3 has shown practical promise.

Still, there is a long way to go before we start to see competitive CSP solvers that make explicit use of tractable classes. Backdoor theory has not yet proven its worth, many important meta-problems still have either unknown complexity or only astronomically inefficient polynomial-time algorithms, and z-loss-less kernelization has so far only be applied to one global constraint, VertexCover. There are without a doubt countless other ways to harness tractability in CSP, and we have but prepared the grounds for future research.

Open problems

We recall here the open problems raised in this thesis. It is on purpose that we did not include Question 1, as it is of considerably narrower interest compared to the others.

(i) Does there exist an idempotent strong linear Mal'tsev condition $\mathcal{M}$ that has a semiuniform polynomial-time algorithm but whose meta-problem is not in P, assuming some likely complexity theoretic conjecture?

(ii) Does there exist a uniform polynomial-time algorithm for the class of all tractable conservative constraint languages?

(iii) What is the complexity of deciding if a constraint language has an idempotent Siggers polymorphism?

(iv) Is it possible to improve our algorithm for deciding the conservative dichotomy so that its complexity becomes a reasonable polynomial, as we did for conservative Mal'tsev and conservative majority polymorphisms?

(v) Is it true that for every z, a strong z-loss-less kernelization of Vertex Cover leaving a residual graph with at most $(z+2)k$ vertices and no z-rigid crown decomposition exists? We assume here that z is part of the input.

Appendices

Appendix A

Omitted Proofs

Proof (of the claim in Figure 3.3). The domain-decomposability of these classes is trivial, and for value-renamability we refer to Example 7 as the exact same reasoning applies to every class in the diagram. The only slightly nontrivial part is to prove that none of these classes is 1-Helly. Let Γ^1_{sml} be the set of all ternary Horn clauses and Γ^2_{sml} be the set of all ternary dual-Horn clauses. Both Γ^1_{sml} and Γ^2_{sml} have a semilattice polymorphism (respectively max and min on the Boolean domain), so they belong to the classes "Semilattice", "$\forall k : \mathrm{TS}_k$", "2-Semilattice", "$\forall k \geq 3 : \mathrm{WNU}_k$" and "Siggers". However, $\Gamma^1_{sml} \cup \Gamma^2_{sml}$ does not have a Siggers polymorphism and thus belongs to none of these classes. Therefore, they are not 1-Helly. Now, let R^1_{mal} be the equation $R^1_{mal}(x, y, z) \iff x+y = z$ over GF(3), and let $R^2_{mal} = [(1), (2)]$. Both of these relations have a Mal'tsev polymorphism (respectively, $f(x, y, z) = x-y+z \mod 3$ and $f(x, y, z) = x-y+z \mod 2+1$) so they belong to the classes "Mal'tsev", "k-edge" and "GMM_k" for all $k \geq 3$. However, the relation $R(y, z) : \exists x, R^2_{mal}(x) \wedge R^1_{mal}(x, y, z)$ is pp-definable in $\{R^1_{mal}, R^2_{mal}\}$ and coincides with $\neq_3$, so $\{R^1_{mal}, R^2_{mal}\}$ belongs to none of those classes, which then cannot be 1-Helly. At this point, only the classes of the near-unanimity hierarchy remain. Let $\Gamma = \{[(a_1, a_2), (a_1, b_2), (b_1, c_2)] \mid (a_1, a_2, b_1, b_2, c_2) \in \{0, 1, 2\}^5\}$. The simple gadget given in ([42], Figure 4, Case 1) shows that Γ is NP-complete, but every $R \in \Gamma$ has a majority polymorphism and thus belongs to "NU_k" for all $k \geq 3$. It follows that these classes are not 1-Helly. □

Proof (of Proposition 14). The proof is a Cook reduction from Minimal CSP, in which we are given a CSP instance I with the promise that every tuple in every constraint belongs to at least one solution and we are asked to produce a solution (this is a search problem, not a decision problem). Minimal CSP is known to be NP-hard even on binary instances and domain size 3 [77][61]. Let $I = (\mathcal{X}, \mathcal{D}, \mathcal{C})$ be a binary instance of Minimal CSP with domain size 3. Let $G_I = (V, E)$ be the graph with $V = \{ (x, d) \mid x \in \mathcal{X}, d \in \mathcal{D} \}$ and with an edge between (x, d) and

(y, d') if and only if there is a constraint C with scope (x, y) and such that (d, d') is not a tuple of C. Observe that the maximum-size independent sets of G_I are in one-to-one correspondance with the solutions to I, and therefore a solution to I can be computed in polynomial time from any vertex cover of G_I. Note also that every minimum vertex cover of G_I contains exactly $2n$ vertices, where $n = |\mathcal{X}|$.

Let B be a polynomial-time algorithm as in Proposition 14. For every subset $V' \subseteq V$, let $G^{V'}$ be the graph with vertex set $V \cup \{w, i\}$ and edge set $E \cup \{ \ (v, w) \mid v \in V' \ \} \cup \{ \ (w, i) \ \}$. By construction, $(V, \{w\}, \{i\})$ is always a crown decomposition of $G^{V'}$. Now, suppose that we invoke B with input $(G^{V'}, (V, \{w\}, \{i\}), 2n, \emptyset, \emptyset)$. The input is clearly correct, as every vertex of G_I belongs to at least one maximum-size independent set and at least one minimum-size vertex cover. If B says that there exists a minimum-size vertex cover S of $G^{V'}$ that does not contain w, then S must contain its neighbourhood $\{i\} \cup V'$. Therefore, there exists a minimum-size vertex cover of G_I that contains V'. We can use this property to compute greedily a minimum-size vertex cover of G_I: if we know a subset of vertices $V_S \subset V$ that is a strict subset of a minimum-size vertex cover of G_I, for all $v \in V \backslash V_S$ we invoke B with input $(G^{V_S \cup \{v\}}, (V, \{w\}, \{i\}), 2n, \emptyset, \emptyset)$. If B says that there exists a minimum-size vertex cover of $G^{V_S \cup \{v\}}$ that does not contain w, we set $V_S = V_S \cup \{v\}$ and repeat until a minimum-size vertex cover of G_I is known. By construction, the complement of this minimum-size vertex cover gives a solution to the instance I of Minimal CSP. □

Appendix B

Compendium of Problems

Almost-2-SAT

Input: A set $\mathcal{X}$ of Boolean variables, a CNF formula $\mathcal{F}$ over $\mathcal{X}$ with at most 2 literals per clause, an integer k

Question: Is there a subset $\mathcal{S}$ of at most k clauses and $\phi : \mathcal{X} \to \{0, 1\}$ such that $\mathcal{F} \backslash \mathcal{S}$ evaluates to `true`?

Complexity: NP-complete and FPT for the parameter k [121]

Betweenness

Input: A universe U, a set T of ordered triples of distinct elements of U

Question: Is it possible to find a total ordering $>_U$ of U such that for each triple $(a, b, c) \in T$, either $a >_U b >_U c$ or $c >_U b >_U a$?

Complexity: NP-complete [119]

Balanced Vertex Cover

Input: A graph $G = (V, E)$, a partition $(V_1, \ldots, V_c)$ of V, two integers k and b

Question: Does there exist a vertex cover S of size at most k and such that $\forall (i, j), \left| |V_i \cap S| - |V_j \cap S| \right| \leq b$?

Complexity: NP-complete by reduction from Vertex Cover

Clique

Input: A graph $G = (V, E)$, an integer k

Question: Does there exist $S \subseteq V$ such that $|S| \geq k$ and for all distinct u, v in S, $\{u, v\} \in E$?

Complexity: NP-complete [98] and W[1]-complete for the parameter k [58]

CSP

Input: A set $\mathcal{X}$ of variables, a set $\mathcal{D}$ of values, a set $\mathcal{C}$ of pairs (S, R) where S is a tuple of variables and $R \subseteq \mathcal{D}^{|S|}$

Question: Does there exist $\phi : \mathcal{X} \to \mathcal{D}$ such that $\forall (S, R) \in \mathcal{C}$, $\boldsymbol{\phi}(S) \in R$?

Complexity: NP-complete [98]

Dominating Set

Input: A graph $G = (V, E)$, an integer k

Question: Does there exist $S \subseteq V$ such that $|S| \geq k$ and for all $u \notin S$, $\exists v \in S$ such that $\{u, v\} \in E$?

Complexity: NP-complete [67] and W[2]-complete for the parameter k [57]

Vertex Cover

Input: A graph $G = (V, E)$, an integer k

Question: Does there exist $S \subseteq V$ such that $|S| \leq k$ and $\forall\, e \in E$, $S \cap e \neq \emptyset$?

Complexity: NP-complete [98] and FPT for the parameter k

Graph k-Coloring

Input: A graph $G = (V, E)$, an integer k

Question: Does there exist a partition $V_1, \ldots, V_k$ of v such that $\forall e \in E$ and $i \in \{1, \ldots, k\}$, $e \not\subset V_i$?

Complexity: NP-complete for $k \geq 3$ [130] and polynomial-time otherwise

d-Hitting Set

Input: A finite universe U, a collection $\mathcal{S}$ of sets containing each at most d elements from U, an integer k

Question: Does there exist $H \subseteq U$ such that $|H| \leq k$ and $\forall\, S \in \mathcal{S}$, $H \cap S \neq \emptyset$?

Complexity: NP-complete [98] and FPT for the parameter k [59]

Hitting Set

Input: A finite universe U, an integer p, a collection $\mathcal{S}$ of sets containing each at most p elements from U, an integer k

Question: Does there exist $H \subseteq U$ such that $|H| \leq k$ and $\forall\, S \in \mathcal{S}$, $H \cap S \neq \emptyset$?

Complexity: NP-complete [98] and W[2]-hard for the parameter k [59]

Integer Linear Programming

Input: A matrix A, a vector $\mathbf{b}$, a vector $\mathbf{c}$, a vector $\mathbf{x}$ of variables, an integer k

Question: Does there exist a integer solution ϕ to the system $A\mathbf{x} \leq \mathbf{b}$ such that $\mathbf{c}^T\boldsymbol{\phi}(\mathbf{x}) \geq k$?

Complexity: NP-complete [67]

Maximum Common Induced Subgraph

Input: Two graphs (G_1, G_2), an integer k

Question: Does there exist a graph H with at least k vertices that is isomorphic to induced subgraphs of both G_1 and G_2?

Complexity: NP-complete [67]

SAT

Input: A set $\mathcal{X}$ of Boolean variables, a CNF formula $\mathcal{F}$ over $\mathcal{X}$

Question: Does there exist $\phi : \mathcal{X} \to \{0, 1\}$ for which $\mathcal{F}$ evaluates to `true`?

Complexity: NP-complete [41]

Positive 1-in-3-SAT

Input: A set $\mathcal{X}$ of Boolean variables, a CNF formula $\mathcal{F}$ over $\mathcal{X}$ with only positive clauses and at most 3 literals per clause

Question: Does there exist $\phi : \mathcal{X} \to \{0, 1\}$ such that exactly one literal is set to `true` in each clause in $\mathcal{F}$?

Complexity: NP-complete [128]

3-SAT

Input: A set $\mathcal{X}$ of Boolean variables, a CNF formula $\mathcal{F}$ over $\mathcal{X}$ with at most 3 literals per clause

Question: Does there exist $\phi : \mathcal{X} \to \{0, 1\}$ for which $\mathcal{F}$ evaluates to `true`?

Complexity: NP-complete [98]

k-XORSAT

Input: A set S of linear equations over GF(2) with at most k variables each

Question: Does S have a solution?

Complexity: Polynomial-time

Strong $\mathcal{H}$-Backdoor Detection

Input: A CSP instance $I = (\mathcal{X}, \mathcal{D}, \mathcal{C})$, an integer k

Question: Does I have a strong backdoor into $\mathcal{H}$ of size at most k?

Complexity: Depends on $\mathcal{H}$ and the choice of parameter. See Theorem 10, Theorem 11, Theorem 12, Theorem 13, Theorem 14 and [69]

Strong $\mathcal{H}$-Partition Backdoor Detection

Input: A CSP instance $I = (\mathcal{X}, \mathcal{D}, \mathcal{C})$, an integer k

Question: Does I have a partition backdoor into $\mathcal{H}$ of size at most k?

Complexity: Depends on $\mathcal{H}$ and the choice of parameter. See Proposition 6 and Theorem 15

Π-Minimization-Decision

Input: A set of variables $X = \{x_1, \ldots, x_n\}$ with domains $D(x_1), \ldots, D(x_n) \subseteq \mathcal{D}$, $\pi \in \Pi$ over $\mathcal{D}$ of arity n, $k \in \mathbb{Q}$

Question: Does there exist an assignment $\phi : X \to \mathcal{D}$ such that $\forall x \in X, \phi(x) \in D(x)$ and $\pi(\phi(x_1), \ldots, \phi(x_n)) \leq k$?

Complexity: Depends on Π

Appendix C

Detailed Results for `dimacs` Instances

Table C.1: Comparison of approaches on Balanced Vertex Cover: `dimacs`

	Decomposition				Clique Cover				Kernel Pruning				Kernel & witness				VertexCover			
	#s	gap	cpu	#nd	#s	gap	cpu	#nd	#s	gap	cpu	#nd	#s	gap	cpu	#nd	#s	gap	cpu	#nd
balancing constraint: tight																				
mann_a27	1	4	0.2	22K	1	4	0.2	23K	1	4	0.2	21K	1	4	0.3	21K	1	4	0.3	21K
mann_a45	1	346	-	-	1	346	-	-	1	346	-	-	1	346	-	-	1	346	-	-
mann_a81	1	11	3.3	77K	1	11	3.0	0.1M	1	11	2.6	72K	1	11	3.9	72K	1	11	4.2	71K
mann_a9	0	3	0.0	23	0	3	0.0	23	0	3	0.0	23	0	3	0.0	23	0	3	0.0	23
brock200_1	1	5	0.0	14	1	5	0.0	14	1	5	0.0	14	1	5	0.0	14	1	5	0.0	14
brock200_2	0	0	0.1	3K	0	0	0.0	1K	0	0	0.0	1K	0	0	0.1	712	0	0	0.1	699
brock200_3	0	3	0.0	251	0	3	0.0	276	0	3	0.0	229	0	3	0.0	229	0	3	0.0	220
brock200_4	0	1	1.1	0.1M	0	1	0.4	40K	0	1	0.5	37K	0	1	1.4	18K	0	1	1.5	18K
brock400_1	1	4	0.0	143	1	4	0.0	150	1	4	0.0	140	1	4	0.0	140	1	4	0.0	140
brock400_2	1	4	0.0	1K	1	4	0.0	1K	1	4	0.0	1K	1	4	0.0	1K	1	4	0.1	1K
brock400_3	1	5	0.0	2K	1	5	0.0	2K	1	5	0.0	2K	1	5	0.1	2K	1	5	0.1	1K
brock400_4	1	5	0.0	2K	1	5	0.0	2K	1	5	0.0	1K	1	5	0.0	1K	1	5	0.1	1K
brock800_1	1	4	0.0	40	1	4	0.0	41	1	4	0.0	39	1	4	0.0	39	1	4	0.0	39
brock800_2	1	4	0.1	588	1	4	0.0	580	1	4	0.0	575	1	4	0.0	572	1	4	0.1	559
brock800_3	1	5	0.1	467	1	5	0.0	490	1	5	0.0	408	1	5	0.0	408	1	5	0.0	398
brock800_4	1	4	0.1	57	1	4	0.0	63	1	4	0.0	53	1	4	0.0	52	1	4	0.0	52
c-fat200-1	0	4	0.0	3	0	4	0.0	3	0	4	0.0	3	0	4	0.0	3	0	4	0.0	3
c-fat200-2	0	4	0.2	25K	0	4	0.3	41K	0	4	0.1	18K	0	4	0.2	18K	0	4	0.2	18K
c-fat200-5	1	26	0.0	25	1	26	0.0	25	1	26	0.0	25	1	26	0.0	25	1	26	0.0	25
c-fat500-1	0	2	0.0	10	0	2	0.0	10	0	2	0.0	10	0	2	0.0	10	0	2	0.0	10
c-fat500-10	1	14	0.0	101	1	14	0.0	101	1	14	0.0	101	1	14	0.0	101	1	14	0.0	101
c-fat500-2	0	2	0.9	42K	0	2	1.2	90K	0	2	0.5	34K	0	2	0.7	34K	0	2	0.7	34K
c-fat500-5	1	20	0.0	38	1	20	0.0	38	1	20	0.0	38	1	20	0.0	38	1	20	0.0	38
fpsol2.i.1	1	308	-	-	1	308	-	-	1	308	-	-	1	308	-	-	1	308	-	-
fpsol2.i.2	1	2	0.0	2K	1	2	0.0	2K	1	2	0.0	2K	1	2	0.0	2K	1	2	0.0	2K
fpsol2.i.3	1	5	0.1	11K	1	1	81.1	8.3M	1	1	112.0	6.5M	1	1	122.0	6.5M	1	1	29.8	1.5M
graph00	1	97	0.0	3K	1	97	0.0	3K	1	97	0.0	3K	1	97	0.0	3K	1	97	0.0	3K
graph01	1	143	28.3	3.8M	1	143	30.9	9.1M	1	143	19.7	3.8M	1	143	30.6	3.8M	1	143	36.0	3.8M
graph02	1	117	0.0	128	1	117	0.0	128	1	117	0.0	128	1	117	0.0	128	1	117	0.0	128
graph03	1	185	173.0	17.6M	1	209	159.4	29.3M	1	185	139.2	16.7M	1	209	159.4	14.3M	1	213	0.2	14K
graph04	1	37	0.0	200	1	37	0.0	200	1	37	0.0	200	1	37	0.0	200	1	37	0.0	200
graph05	1	37	0.0	201	1	37	0.0	201	1	37	0.0	201	1	37	0.0	201	1	37	0.0	201
graph06	1	37	0.0	196	1	37	0.0	196	1	37	0.0	196	1	37	0.0	196	1	37	0.0	196
graph07	1	29	0.0	212	1	29	0.0	212	1	29	0.0	212	1	29	0.0	212	1	29	0.0	212

graph09	1	2	0.0	457	0	2	0.0	457	0	2	0.0	457	0	2	0.0	457	0	2	0.0	457
graph10	1	19	3.9	0.4M	1	19	2.8	0.5M	1	19	2.2	0.3M	1	19	3.1	0.3M	1	19	3.4	0.3M
graph11	1	15	151.6	15.8M	1	15	2.7	0.3M	1	15	1.7	0.1M	1	15	1.4	43K	1	15	1.5	41K
graph12	1	89	39.0	3.8M	1	89	14.9	1.4M	1	89	14.9	1.0M	1	89	21.0	0.9M	1	89	20.9	0.9M
graph13	1	85	114.6	10.9M	1	85	55.0	12.5M	1	85	31.7	4.6M	1	85	47.9	4.6M	1	85	57.3	4.6M
graph14	1	73	157.4	17.3M	1	73	177.4	36.6M	1	61	136.1	19.3M	1	73	161.9	17.0M	1	93	0.0	2K
graph15	1	85	5.4	0.6M	1	85	4.1	1.0M	1	85	3.4	0.6M	1	85	5.8	0.6M	1	85	6.7	0.6M
graph16	1	21	10.0	0.9M	1	21	41.4	4.8M	1	21	8.7	0.9M	1	21	10.8	0.9M	1	21	12.1	0.9M
graph17	1	9	0.1	7K	1	9	0.7	55K	1	9	0.1	6K	1	9	0.1	6K	1	9	0.1	6K
graph18	1	21	1.3	0.1M	1	21	1.3	0.2M	1	21	0.9	0.1M	1	21	1.2	0.1M	1	21	1.4	0.1M
graph19	1	21	0.0	100	1	21	0.0	100	1	21	0.0	100	1	21	0.0	100	1	21	0.0	100
graph20	1	15	0.0	166	1	15	0.0	166	1	15	0.0	166	1	15	0.0	166	1	15	0.0	166
graph21	1	11	0.7	81K	1	7	29.8	3.2M	1	7	17.1	1.1M	1	7	18.8	1.1M	1	7	20.4	1.0M
graph22	1	8	0.3	32K	1	4	8.4	0.9M	1	4	5.2	0.3M	1	4	5.9	0.2M	1	4	6.4	0.2M
graph23	1	5	0.1	9K	1	5	0.1	6K	1	5	0.1	4K	1	5	0.1	4K	1	5	0.1	3K
graph24	1	26	0.1	13K	1	18	9.3	0.4M	1	18	7.5	0.4M	1	18	10.6	17K	1	18	10.6	16K
graph25	1	30	21.2	1.7M	1	22	44.1	2.2M	1	18	140.4	11.1M	1	22	75.1	0.8M	1	22	75.9	0.8M
graph26	1	10	78.4	6.6M	1	10	52.1	9.4M	1	10	34.9	4.2M	1	10	51.8	4.2M	1	10	59.0	4.2M
graph27	1	18	0.2	12K	1	18	0.2	23K	1	18	0.1	10K	1	18	0.1	10K	1	18	0.1	10K
graph28	0	6	0.2	12K	0	6	0.2	21K	0	6	0.1	10K	0	6	0.1	10K	0	6	0.2	10K
graph29	1	18	0.1	6K	1	18	0.1	9K	1	18	0.1	4K	1	18	0.1	4K	1	18	0.1	4K
graph30	0	6	0.1	1K	0	6	0.0	832	0	6	0.0	502	0	6	0.0	502	0	6	0.0	502
graph31	1	14	0.0	55	1	14	0.0	51	1	14	0.0	48	1	14	0.0	48	1	14	0.0	48
graph32	0	6	0.0	43	0	6	0.0	43	0	6	0.0	43	0	6	0.0	43	0	6	0.0	43
hamming10-2	1	34	0.0	467	1	34	0.0	467	1	34	0.0	467	1	34	0.0	467	1	34	0.0	467
hamming10-4	1	0	134.4	6.7M	1	0	11.2	0.5M	1	0	12.0	0.5M	1	0	20.9	0.1M	1	0	21.2	0.1M
hamming6-2	0	8	20.6	6.1M	0	8	6.8	1.4M	0	8	8.1	1.0M	0	8	8.8	1.0M	0	8	9.0	0.9M
hamming6-4	0	0	0.0	3	0	0	0.0	3	0	0	0.0	3	0	0	0.0	3	0	0	0.0	3
hamming8-2	1	12	0.0	105	1	12	0.0	105	1	12	0.0	105	1	12	0.0	105	1	12	0.0	105
hamming8-4	1	0	2.5	0.2M	0	0	0.4	15K	0	0	0.3	15K	0	0	0.7	4K	0	0	0.7	4K
inithx.i.1	1	567	-	-	1	567	-	-	1	567	-	-	1	567	-	-	1	567	-	-
inithx.i.2	1	4	115.0	12.2M	1	4	145.4	12.2M	1	4	164.5	12.2M	1	4	165.1	12.2M	1	4	165.2	12.2M
inithx.i.3	1	3	0.0	394	1	3	0.0	394	1	3	0.0	394	1	3	0.0	394	1	3	0.0	394
johnson16-2-4	0	0	0.0	12	0	0	0.0	12	0	0	0.0	12	0	0	0.0	12	0	0	0.0	12
johnson32-2-4	1	0	0.0	114	1	0	0.0	114	1	0	0.0	114	1	0	0.0	114	1	0	0.0	114
johnson8-2-4	0	0	0.0	10	0	0	0.0	10	0	0	0.0	10	0	0	0.0	10	0	0	0.0	10
johnson8-4-4	0	0	0.2	40K	0	0	0.0	5K	0	0	0.1	5K	0	0	0.1	3K	0	0	0.1	3K
keller4	0	0	0.1	5K	0	0	0.0	4K	0	0	0.1	4K	0	0	0.1	3K	0	0	0.1	3K
keller5	1	4	20.5	0.9M	1	4	22.1	0.6M	1	4	20.0	0.6M	1	4	72.2	0.3M	1	4	114.3	0.3M
keller6	1	2	8.4	0.1M	1	2	9.7	0.1M	1	2	7.3	0.1M	1	2	7.5	0.1M	1	2	8.5	0.1M
mulsol.i.1	1	3	0.0	108	0	3	0.0	108	0	3	0.0	108	0	3	0.0	108	0	3	0.0	108
mulsol.i.2	1	6	0.0	83	1	2	161.4	25.5M	0	2	79.9	5.9M	0	2	81.2	5.9M	0	2	59.0	4.6M
mulsol.i.3	1	18	0.0	65	1	18	0.0	65	1	18	0.0	65	1	18	0.0	65	1	18	0.0	65
mulsol.i.4	1	9	0.8	0.2M	1	9	1.3	0.2M	1	9	1.0	0.2M	1	9	1.0	0.2M	1	9	1.0	0.2M
mulsol.i.5	1	6	35.1	9.0M	1	6	44.2	5.8M	1	6	42.2	4.4M	1	6	43.0	4.4M	1	6	36.1	3.6M
p_hat1000-1	1	2	0.2	7	0	2	0.1	7	1	2	0.1	7	1	2	0.1	7	1	2	0.1	7
p_hat1000-2	1	9	6.9	0.3M	1	9	6.6	0.2M	1	9	3.8	0.2M	1	9	6.5	0.1M	1	9	6.8	0.1M
p_hat1000-3	1	12	85.4	5.0M	1	12	41.6	2.8M	1	12	41.6	2.2M	1	12	60.2	2.1M	1	12	63.7	2.1M
p_hat1500-1	1	4	0.2	76	1	4	0.1	76	1	4	0.2	76	1	4	0.2	76	1	4	0.2	76
p_hat1500-2	1	11	0.2	49	1	11	0.1	49	1	11	0.1	49	1	11	0.1	49	1	11	0.1	49
p_hat1500-3	1	13	14.6	0.5M	1	13	8.5	0.6M	1	13	7.4	0.4M	1	13	11.1	0.4M	1	13	12.4	0.4M
p_hat300-1	0	0	0.0	188	0	0	0.0	175	0	0	0.0	172	0	0	0.0	172	0	0	0.0	145
p_hat300-2	1	1	0.4	46K	0	1	0.1	7K	0	1	0.1	5K	0	1	0.1	3K	0	1	0.1	3K
p_hat300-3	1	4	0.0	135	1	4	0.0	149	1	4	0.0	98	1	4	0.0	93	1	4	0.0	93
p_hat500-1	0	1	0.0	153	0	1	0.0	153	0	1	0.0	153	0	1	0.0	153	0	1	0.0	153
p_hat500-2	1	4	15.2	1.4M	1	4	0.3	30K	1	4	0.3	19K	0	0	63.2	0.1M	0	0	64.3	0.1M
p_hat500-3	1	5	0.3	35K	1	5	0.2	24K	1	5	0.1	12K	1	1	157.1	0.5M	1	1	160.1	0.5M
p_hat700-1	0	3	0.1	37	0	3	0.0	37	0	3	0.0	37	1	3	0.0	37	1	3	0.0	37
p_hat700-2	1	8	0.1	1K	1	4	71.6	6.0M	1	4	44.5	2.3M	1	4	59.0	2.0M	1	4	62.0	1.9M
p_hat700-3	1	9	5.5	0.4M	1	5	105.2	7.1M	1	5	69.7	4.0M	1	5	58.3	1.4M	1	5	62.6	1.4M
san1000	1	16	-	-	1	16	-	-	1	16	-	-	1	16	-	-	1	16	-	-
san200_0.7_1	1	14	0.0	57	0	6	2.3	0.3M	0	6	3.2	0.2M	0	6	4.7	0.2M	0	6	4.9	0.2M

san200_0.7_2	1	6	0.0	95	1	6	0.0	95	1	6	0.0	95	1	6	0.0	95	1	6	0.0	95
san200_0.9_1	1	26	0.0	165	1	26	0.0	165	1	26	0.0	165	1	26	0.0	165	1	26	0.0	165
san200_0.9_2	1	24	0.0	86	1	16	113.4	18.9M	1	16	103.3	8.5M	1	16	44.3	2.6M	1	16	47.0	2.6M
san200_0.9_3	1	8	0.0	1K	1	8	0.0	1K	1	8	0.0	607	1	8	0.0	557	1	8	0.0	557
san400_0.5_1	1	5	0.0	1K	0	5	0.0	1K	0	5	0.0	1K	0	5	0.0	1K	0	5	0.0	1K
san400_0.7_1	1	20	0.0	29	1	20	0.0	29	1	20	0.0	29	1	20	0.0	29	1	20	0.0	29
san400_0.7_2	1	14	48.3	4.5M	1	14	46.7	4.5M	1	14	62.3	4.5M	1	14	73.3	4.5M	1	14	91.7	4.5M
san400_0.7_3	1	8	0.0	304	1	8	0.0	304	1	8	0.0	304	1	8	0.0	304	1	8	0.0	304
san400_0.9_1	1	8	13.2	1.8M	1	8	7.5	1.2M	1	8	7.6	0.5M	1	8	5.3	0.5M	1	8	5.8	0.4M
sanr200_0.7	1	2	2.8	0.4M	0	2	1.7	0.2M	0	2	2.2	0.2M	0	2	4.8	0.1M	0	2	5.0	0.1M
sanr200_0.9	1	6	106.0	18.5M	1	6	14.1	2.0M	1	6	13.6	1.1M	1	6	18.0	0.8M	1	6	19.3	0.7M
sanr400_0.5	1	1	0.2	10K	0	1	0.1	8K	0	1	0.1	7K	0	1	0.4	7K	0	1	0.4	6K
sanr400_0.7	1	5	0.0	17	1	5	0.0	17	1	5	0.0	17	1	5	0.0	17	1	5	0.0	17
zeroin.i.1	1	121	-	-	1	121	-	-	1	121	-	-	1	121	-	-	1	121	-	-
zeroin.i.2	1	4	0.0	177	1	4	0.0	177	1	4	0.0	177	1	4	0.0	177	1	4	0.0	177
zeroin.i.3	1	9	23.9	5.2M	1	9	22.8	3.1M	1	9	29.4	2.8M	1	9	27.7	2.8M	1	9	16.4	1.8M
balancing constraint: medium																				
mann_a27	1	1	17.7	2.5M	1	1	0.1	9K	1	1	0.0	853	0	0	1.1	48K	0	0	0.9	41K
mann_a45	1	346	-	-	1	346	-	-	1	346	-	-	1	346	-	-	1	346	-	-
mann_a81	1	9	70.0	1.2M	1	8	133.8	4.6M	1	8	63.2	1.7M	1	8	87.8	1.7M	1	8	99.4	1.7M
mann_a9	0	0	0.0	18	0	0	0.0	18	0	0	0.0	18	0	0	0.0	18	0	0	0.0	18
brock200_1	1	1	20.2	3.3M	0	1	0.2	18K	0	1	0.3	18K	0	1	0.2	51	0	1	0.2	51
brock200_2	0	0	0.1	8K	0	0	0.0	303	0	0	0.0	307	0	0	0.1	40	0	0	0.1	40
brock200_3	0	0	31.8	4.3M	0	0	1.0	48K	0	0	1.6	48K	0	0	2.9	90	0	0	3.0	90
brock200_4	0	0	64.8	9.1M	0	0	1.1	64K	0	0	1.8	64K	0	0	1.1	81	0	0	1.1	81
brock400_1	1	1	17.7	2.0M	1	0	90.2	6.0M	1	0	137.4	6.0M	1	0	109.0	2K	1	0	108.3	2K
brock400_2	1	1	7.1	0.8M	1	0	4.5	0.3M	1	0	6.8	0.3M	1	0	5.1	277	1	0	5.1	273
brock400_3	1	1	9.0	1.0M	1	0	22.9	1.4M	1	0	35.9	1.4M	1	0	50.0	647	1	0	29.7	643
brock400_4	1	2	0.1	9K	1	0	54.4	3.4M	1	0	85.0	3.4M	1	0	69.9	1K	1	0	70.2	1K
brock800_1	1	1	15.8	1.0M	1	0	29.6	0.7M	1	0	25.8	0.7M	1	0	36.1	595	1	0	35.5	594
brock800_2	1	1	17.3	1.1M	1	0	4.3	0.1M	1	0	3.7	0.1M	1	0	4.6	126	1	0	4.5	124
brock800_3	1	1	3.2	0.2M	1	1	0.1	2K	1	1	0.1	2K	1	1	0.2	163	1	1	0.2	162
brock800_4	1	1	0.1	1K	1	0	5.4	0.2M	1	0	6.8	0.2M	1	0	8.1	138	1	0	7.9	138
c-fat200-1	0	0	0.0	13	0	0	0.0	13	0	0	0.0	13	0	0	0.0	13	0	0	0.0	13
c-fat200-2	0	0	0.0	25	0	0	0.0	25	0	0	0.0	25	0	0	0.0	25	0	0	0.0	25
c-fat200-5	1	14	0.0	39	1	14	0.0	39	1	14	0.0	39	1	14	0.0	39	1	14	0.0	39
c-fat500-1	0	0	0.0	15	0	0	0.0	15	0	0	0.0	15	0	0	0.0	15	0	0	0.0	15
c-fat500-10	1	6	0.0	113	1	6	0.0	113	1	6	0.0	113	1	6	0.0	113	1	6	0.0	113
c-fat500-2	0	0	0.0	27	0	0	0.0	27	0	0	0.0	27	0	0	0.0	27	0	0	0.0	27
c-fat500-5	1	9	0.0	52	1	9	0.0	52	1	9	0.0	52	1	9	0.0	52	1	9	0.0	52
fpsol2.i.1	1	308	-	-	1	308	-	-	1	308	-	-	1	308	-	-	1	308	-	-
fpsol2.i.2	1	0	0.0	266	0	0	0.0	265	0	0	0.0	264	0	0	0.0	264	0	0	0.0	263
fpsol2.i.3	1	1	0.0	234	0	0	0.0	594	0	0	0.0	544	0	0	0.0	544	0	0	0.0	260
graph00	1	89	37.6	4.4M	1	57	30.0	4.1M	1	53	109.2	4.1M	1	53	106.9	4.1M	1	53	106.0	4.0M
graph01	1	135	0.0	95	1	135	0.0	95	1	135	0.0	95	1	135	0.0	95	1	135	0.0	95
graph02	1	93	2.7	0.2M	1	93	31.6	4.6M	1	93	3.4	0.2M	1	93	3.6	0.2M	1	93	4.0	0.2M
graph03	1	149	69.7	6.1M	1	173	1.7	0.3M	1	149	76.3	5.5M	1	149	84.5	5.5M	1	149	95.3	5.5M
graph04	1	25	0.0	217	1	25	0.0	217	1	25	0.0	217	1	25	0.0	217	1	25	0.0	217
graph05	1	25	0.0	214	1	25	0.0	214	1	25	0.0	214	1	25	0.0	214	1	25	0.0	214
graph06	1	25	0.0	208	1	25	0.0	208	1	25	0.0	208	1	25	0.0	208	1	25	0.0	208
graph07	1	17	0.0	226	1	17	0.0	226	1	17	0.0	226	1	17	0.0	226	1	17	0.0	226
graph09	1	0	0.0	463	0	0	0.0	463	0	0	0.0	463	0	0	0.0	463	0	0	0.0	463
graph10	1	13	38.7	4.6M	1	4	47.7	4.0M	1	4	30.2	1.2M	1	1	38.2	97K	1	1	38.8	95K
graph11	1	15	48.6	5.2M	1	11	103.9	7.2M	1	11	138.0	7.2M	1	10	32.1	881	1	10	31.8	879
graph12	1	81	44.9	4.6M	1	76	58.2	10.9M	1	76	35.4	3.3M	1	76	45.4	3.3M	1	76	50.8	3.3M
graph13	1	49	168.7	9.1M	1	27	76.0	13.5M	1	9	40.0	2.6M	1	9	44.3	2.6M	1	9	49.4	2.6M
graph14	1	53	6.3	0.2M	1	53	0.0	826	1	53	0.0	228	1	53	0.0	226	1	53	0.0	226
graph15	1	73	124.7	11.8M	1	93	0.0	29	1	73	110.6	11.8M	1	73	140.5	11.8M	1	73	162.8	11.8M
graph16	1	9	0.2	3K	1	9	0.0	539	1	9	0.0	288	1	9	0.0	288	1	9	0.0	288
graph17	0	1	1.5	4K	0	1	0.2	17K	0	1	0.1	523	0	1	0.2	523	0	1	0.2	523
graph18	1	9	0.0	208	1	9	0.0	540	1	9	0.0	145	1	9	0.0	145	1	9	0.0	145
graph19	1	9	0.0	114	1	9	0.0	114	1	9	0.0	114	1	9	0.0	114	1	9	0.0	114

graph20	1	10	3.0	0.3M	1	1	158.2	10.1M	1	1	23.2	0.7M	1	0	2.6	4K	1	0	2.6	4K
graph21	1	5	101.5	12.0M	1	2	1.2	0.1M	1	2	0.6	35K	1	2	0.3	17K	1	2	0.3	17K
graph22	1	4	71.4	7.7M	1	1	5.4	0.4M	1	1	7.8	0.4M	1	0	23.9	791	1	0	23.7	785
graph23	1	3	66.4	6.7M	1	0	81.3	3.7M	1	0	76.7	3.7M	1	0	24.6	520	1	0	24.6	521
graph24	1	21	52.5	5.0M	1	2	42.0	0.8M	1	2	79.5	0.8M	1	2	81.7	1K	1	2	82.1	1K
graph25	1	24	57.1	4.7M	0	0	18.0	2.6M	0	0	14.7	0.5M	0	0	34.8	0.4M	0	0	36.2	0.4M
graph26	1	0	130.8	3.6M	0	0	0.2	5K	0	0	0.4	1K	0	0	3.1	1K	0	0	3.1	1K
graph27	1	0	14.1	0.8M	0	0	21.0	4.4M	0	0	7.2	0.7M	0	0	10.2	0.7M	0	0	11.8	0.7M
graph28	0	0	0.1	684	0	0	0.0	145	0	0	0.0	143	0	0	0.0	143	0	0	0.0	143
graph29	0	0	8.2	0.7M	0	0	23.5	4.6M	0	0	7.7	0.7M	0	0	9.4	0.7M	0	0	11.1	0.7M
graph30	0	0	0.1	519	0	0	0.0	114	0	0	0.0	112	0	0	0.0	112	0	0	0.0	112
graph31	0	0	0.2	10K	0	0	0.9	98K	0	0	0.1	10K	0	0	0.2	10K	0	0	0.2	10K
graph32	0	0	0.0	51	0	0	0.0	51	0	0	0.0	51	0	0	0.0	51	0	0	0.0	51
hamming10-2	1	22	0.0	481	1	22	0.0	481	1	22	0.0	481	1	22	0.0	481	1	22	0.0	481
hamming10-4	1	0	0.1	5K	1	0	0.0	49	1	0	0.0	40	1	0	0.0	38	1	0	0.0	38
hamming6-2	0	3	1.0	0.3M	0	3	0.0	7K	0	3	0.0	1K	0	3	0.0	1K	0	3	0.0	1K
hamming6-4	0	0	0.0	5	0	0	0.0	5	0	0	0.0	5	0	0	0.0	5	0	0	0.0	5
hamming8-2	1	3	0.0	122	0	3	0.0	122	0	3	0.0	122	0	3	0.0	122	0	3	0.0	122
hamming8-4	1	0	0.0	17	0	0	0.0	17	0	0	0.0	17	0	0	0.0	17	0	0	0.0	17
inithx.i.1	1	567	-	-	1	567	-	-	1	567	-	-	1	567	-	-	1	567	-	-
inithx.i.2	1	0	1.2	0.1M	0	0	0.1	7K	0	0	0.1	6K	0	0	0.1	6K	0	0	0.1	6K
inithx.i.3	1	0	0.0	362	0	0	0.0	362	0	0	0.0	362	0	0	0.0	362	0	0	0.0	362
johnson16-2-4	0	0	0.0	9	0	0	0.0	9	0	0	0.0	9	0	0	0.0	9	0	0	0.0	9
johnson32-2-4	1	0	0.0	17	1	0	0.0	17	1	0	0.0	17	1	0	0.0	17	1	0	0.0	17
johnson8-2-4	0	0	0.0	5	0	0	0.0	5	0	0	0.0	5	0	0	0.0	5	0	0	0.0	5
johnson8-4-4	0	0	0.0	15	0	0	0.0	15	0	0	0.0	15	0	0	0.0	15	0	0	0.0	15
keller4	0	0	0.0	1K	0	0	0.0	78	0	0	0.0	70	0	0	0.0	29	0	0	0.0	28
keller5	1	4	57.0	3.4M	1	1	107.3	3.1M	1	1	163.3	3.1M	1	0	46.7	6K	1	0	46.2	6K
keller6	1	1	15.2	0.2M	1	1	0.6	5K	1	1	0.8	5K	1	0	51.7	448	1	0	50.0	440
mulsol.i.1	1	0	0.0	101	0	0	0.0	101	0	0	0.0	101	0	0	0.0	101	0	0	0.0	101
mulsol.i.2	1	0	0.0	8K	0	0	0.0	115	0	0	0.0	92	0	0	0.0	92	0	0	0.0	92
mulsol.i.3	1	6	0.0	78	1	6	0.0	78	1	6	0.0	78	1	6	0.0	78	1	6	0.0	78
mulsol.i.4	1	0	1.3	0.3M	0	0	0.0	4K	0	0	0.0	2K	0	0	0.0	2K	0	0	0.0	2K
mulsol.i.5	1	0	0.0	7K	0	0	0.0	282	0	0	0.0	259	0	0	0.0	259	0	0	0.0	259
p_hat1000-1	1	0	0.3	1K	0	0	0.1	91	1	0	0.1	78	1	0	0.1	24	1	0	0.1	23
p_hat1000-2	1	7	18.4	0.9M	1	5	14.9	0.9M	1	5	14.5	0.5M	1	4	131.6	0.7M	1	4	135.2	0.7M
p_hat1000-3	1	9	114.3	6.6M	1	4	63.7	2.7M	1	4	53.4	2.1M	1	2	33.6	0.2M	1	2	34.0	0.2M
p_hat1500-1	1	1	1.2	17K	1	0	98.1	0.3M	1	0	147.8	0.3M	1	1	1.7	42	1	1	1.7	41
p_hat1500-2	1	6	0.7	12K	1	2	0.7	27K	1	2	0.9	25K	1	1	1.0	277	1	1	1.0	277
p_hat1500-3	1	7	101.6	3.9M	1	5	21.1	1.1M	1	5	22.6	0.7M	1	3	76.8	21K	1	3	76.7	21K
p_hat300-1	0	0	0.0	464	0	0	0.0	47	0	0	0.0	43	0	0	0.0	14	0	0	0.0	14
p_hat300-2	1	0	0.2	29K	0	0	0.0	106	0	0	0.0	100	0	0	0.0	34	0	0	0.0	34
p_hat300-3	1	2	0.3	40K	1	2	0.0	162	1	2	0.0	89	1	1	13.3	231	1	1	12.8	231
p_hat500-1	0	0	0.1	1K	0	0	0.0	84	0	0	0.0	78	0	0	0.1	22	0	0	0.1	21
p_hat500-2	1	3	42.7	4.2M	1	0	99.9	2.2M	1	0	175.6	2.2M	0	0	7.6	225	0	0	7.3	224
p_hat500-3	1	4	6.3	0.7M	1	0	62.0	3.1M	1	0	91.1	3.1M	1	0	2.4	156	1	0	2.3	156
p_hat700-1	0	0	30.6	0.9M	0	0	4.1	18K	0	0	9.8	17K	0	0	11.0	71	0	0	10.7	70
p_hat700-2	1	2	0.7	43K	1	1	0.1	6K	1	1	0.1	3K	1	1	0.1	469	1	1	0.1	459
p_hat700-3	1	5	4.6	0.4M	1	0	0.5	49K	1	0	0.5	28K	1	0	0.1	1K	1	0	0.1	1K
san1000	1	6	103.8	3.0M	1	5	0.5	3K	1	5	0.5	3K	1	1	164.2	929	1	1	160.8	925
san200_0.7_1	1	13	0.1	21K	0	0	0.1	7K	0	0	0.2	7K	0	0	0.2	199	0	0	0.2	199
san200_0.7_2	1	3	63.5	12.6M	0	0	13.3	1.2M	0	0	22.4	1.2M	0	0	0.5	402	0	0	0.8	391
san200_0.9_1	1	23	2.1	0.4M	1	21	79.6	11.4M	1	21	114.6	9.1M	1	13	106.0	6.1M	1	13	112.8	6.1M
san200_0.9_2	1	19	113.7	23.2M	0	2	35.6	1.9M	0	2	34.4	1.9M	0	2	2.5	3K	0	2	4.1	3K
san200_0.9_3	1	5	50.4	9.6M	1	4	1.4	0.2M	1	4	2.0	0.2M	1	3	107.8	2K	1	3	109.6	2K
san400_0.5_1	1	5	0.0	2K	0	0	0.9	31K	0	0	1.8	31K	0	0	0.8	247	0	0	0.8	246
san400_0.7_1	1	19	0.0	22	0	2	35.3	1.1M	0	2	64.9	1.1M	0	2	20.1	472	0	2	20.3	470
san400_0.7_2	1	15	0.0	16	0	2	21.0	0.6M	0	2	40.2	0.6M	0	2	15.4	522	0	2	15.7	522
san400_0.7_3	1	5	9.1	1.0M	1	2	43.4	3.0M	1	2	74.2	3.0M	1	0	74.5	1K	1	0	73.9	1K
san400_0.9_1	1	7	7.2	1.0M	1	2	127.2	9.2M	1	3	3.6	0.2M	1	0	31.3	902	1	0	31.5	889
sanr200_0.7	1	0	115.7	17.4M	0	0	1.4	0.1M	0	0	2.2	0.1M	0	0	2.1	123	0	0	2.1	120
sanr200_0.9	1	4	5.5	1.1M	1	0	31.9	3.5M	1	0	46.8	3.5M	1	0	4.6	1K	1	0	4.8	1K

sanr400_0.5	1	0	53.0	4.5M	0	0	3.7	93K	0	0	5.5	93K	0	0	17.0	174	0	0	16.8	172
sanr400_0.7	1	1	10.3	1.1M	1	0	60.6	3.4M	1	0	89.5	3.4M	1	0	95.4	1K	1	0	92.5	1K
zeroin.i.1	1	9	19.3	4.3M	1	9	34.6	4.3M	1	9	76.1	4.3M	1	9	43.3	4.3M	1	9	43.3	4.3M
zeroin.i.2	1	0	0.0	128	0	0	0.0	128	0	0	0.0	128	0	0	0.0	128	0	0	0.0	128
zeroin.i.3	1	1	92.8	20.2M	0	0	0.2	24K	0	0	0.1	14K	0	0	0.1	14K	0	0	0.1	12K
balancing constraint: loose																				
mann_a27	1	1	0.0	964	1	1	0.0	189	1	1	0.0	127	0	0	0.0	252	0	0	0.0	252
mann_a45	1	346	-	-	1	346	-	-	1	346	-	-	1	346	-	-	1	346	-	-
mann_a81	1	7	114.4	2.3M	1	7	25.5	1.0M	1	4	91.9	2.3M	1	2	130.4	2.3M	1	0	157.3	2.3M
mann_a9	0	0	0.0	17	0	0	0.0	17	0	0	0.0	17	0	0	0.0	17	0	0	0.0	17
brock200_1	1	1	20.1	3.3M	0	0	2.3	0.2M	0	0	3.4	0.2M	0	0	2.2	94	0	0	2.2	94
brock200_2	0	0	0.1	8K	0	0	0.0	303	0	0	0.0	307	0	0	0.1	40	0	0	0.1	40
brock200_3	0	0	31.7	4.3M	0	0	1.0	48K	0	0	1.5	48K	0	0	3.0	90	0	0	2.9	90
brock200_4	0	0	64.6	9.2M	0	0	1.1	63K	0	0	1.7	63K	0	0	1.1	68	0	0	1.1	68
brock400_1	1	1	13.1	1.5M	1	0	29.9	2.0M	1	0	45.3	2.0M	1	0	38.2	687	1	0	38.5	690
brock400_2	1	1	0.4	43K	1	0	4.4	0.3M	1	0	6.7	0.3M	1	0	5.0	146	1	0	5.1	147
brock400_3	1	1	4.6	0.5M	1	0	22.8	1.4M	1	0	35.5	1.4M	1	0	29.7	509	1	0	29.6	509
brock400_4	1	2	0.1	9K	1	0	54.0	3.4M	1	0	84.5	3.4M	1	0	70.2	1K	1	0	69.7	1K
brock800_1	1	0	40.7	2.5M	1	0	1.0	25K	1	0	0.8	25K	1	0	0.8	84	1	0	0.8	84
brock800_2	1	1	17.3	1.1M	1	0	3.6	0.1M	1	0	3.6	0.1M	1	0	4.6	119	1	0	4.8	118
brock800_3	1	0	1.2	67K	1	0	0.0	677	1	0	0.0	637	1	0	0.1	68	1	0	0.1	68
brock800_4	1	1	0.1	1K	1	0	7.4	0.2M	1	0	6.8	0.2M	1	0	7.8	138	1	0	7.8	138
c-fat200-1	0	0	0.0	13	0	0	0.0	13	0	0	0.0	13	0	0	0.0	13	0	0	0.0	13
c-fat200-2	0	0	0.0	25	0	0	0.0	25	0	0	0.0	25	0	0	0.0	25	0	0	0.0	25
c-fat200-5	0	0	0.1	12K	0	0	1.1	0.2M	0	0	0.1	12K	0	0	0.1	12K	0	0	0.1	12K
c-fat500-1	0	0	0.0	15	0	0	0.0	15	0	0	0.0	15	0	0	0.0	15	0	0	0.0	15
c-fat500-10	1	1	0.0	122	0	0	0.0	339	0	0	0.0	124	0	0	0.2	124	0	0	0.2	124
c-fat500-2	0	0	0.0	27	0	0	0.0	27	0	0	0.0	27	0	0	0.0	27	0	0	0.0	27
c-fat500-5	1	4	0.0	59	0	0	3.0	0.5M	0	0	0.7	42K	0	0	0.8	42K	0	0	0.9	42K
fpsol2.i.1	1	5	3.2	0.4M	1	3	37.9	3.1M	1	3	40.3	1.6M	1	3	41.8	1.6M	1	2	160.6	6.4M
fpsol2.i.2	1	0	0.0	262	0	0	0.0	262	0	0	0.0	262	0	0	0.0	262	0	0	0.0	262
fpsol2.i.3	1	0	0.0	239	0	0	0.0	239	0	0	0.0	239	0	0	0.0	239	0	0	0.0	239
graph00	1	85	40.8	4.8M	1	42	26.2	2.1M	1	17	180.0	4.0M	1	19	81.2	1.9M	1	17	169.8	3.9M
graph01	1	125	0.9	99K	1	125	1.4	0.3M	1	125	1.1	99K	1	125	1.4	99K	1	125	1.5	99K
graph02	1	73	10.4	0.8M	1	73	0.1	9K	1	73	0.0	436	1	73	0.0	436	1	73	0.0	436
graph03	1	13	174.4	5.1M	1	13	0.9	0.1M	1	13	0.1	6K	1	13	0.2	6K	1	13	0.2	6K
graph04	1	13	0.0	235	1	13	0.0	235	1	13	0.0	235	1	13	0.0	235	1	13	0.0	235
graph05	1	13	0.0	228	1	13	0.0	228	1	13	0.0	228	1	13	0.0	228	1	13	0.0	228
graph06	1	13	0.0	228	1	13	0.0	228	1	13	0.0	228	1	13	0.0	228	1	13	0.0	228
graph07	1	8	0.0	238	1	8	0.0	238	1	8	0.0	238	1	8	0.0	238	1	8	0.0	238
graph09	1	0	0.0	463	0	0	0.0	463	0	0	0.0	463	0	0	0.0	463	0	0	0.0	463
graph10	1	11	1.8	0.2M	1	4	47.5	2.1M	1	4	26.8	1.1M	1	0	18.2	745	1	0	17.9	736
graph11	1	15	55.2	6.1M	1	10	92.7	6.6M	1	10	124.3	6.7M	1	0	157.8	2K	1	0	158.1	2K
graph12	1	71	141.5	12.4M	1	31	0.8	0.1M	1	31	0.3	15K	1	31	0.6	14K	1	31	0.6	14K
graph13	1	49	111.0	6.0M	0	0	0.1	5K	0	0	0.1	1K	0	0	0.6	1K	0	0	0.7	1K
graph14	1	0	152.0	2.4M	0	0	5.7	0.9M	0	0	1.0	53K	0	0	1.8	53K	0	0	1.9	53K
graph15	1	62	0.3	22K	1	62	1.4	0.1M	1	62	0.3	22K	1	62	0.3	22K	1	62	0.4	22K
graph16	0	0	1.2	7K	0	0	0.0	426	0	0	0.0	300	0	0	0.1	300	0	0	0.1	300
graph17	0	0	0.1	414	0	0	0.0	206	0	0	0.0	204	0	0	0.0	204	0	0	0.0	204
graph18	0	0	0.1	381	0	0	0.0	284	0	0	0.0	160	0	0	0.0	160	0	0	0.0	160
graph19	0	0	0.2	233	0	0	0.1	371	0	0	0.1	130	0	0	0.3	130	0	0	0.3	130
graph20	1	9	0.0	162	1	1	152.0	9.8M	1	1	22.6	0.7M	1	0	2.5	386	1	0	2.5	385
graph21	1	5	1.6	0.2M	1	2	0.0	4K	1	2	0.0	790	1	0	95.5	2K	1	0	94.8	2K
graph22	1	4	75.5	8.4M	1	1	4.9	0.4M	1	1	7.3	0.4M	1	0	21.4	495	1	0	21.4	489
graph23	1	3	0.0	1K	1	0	49.9	3.7M	1	0	74.4	3.7M	1	0	24.8	473	1	0	24.9	473
graph24	1	21	52.2	5.1M	1	0	28.3	0.5M	1	0	58.5	0.5M	1	0	53.9	652	1	0	54.2	652
graph25	1	14	111.7	8.0M	0	0	1.2	17K	0	0	2.3	17K	0	0	4.5	433	0	0	4.5	433
graph26	1	0	10.4	0.5M	0	0	0.0	646	0	0	0.1	584	0	0	0.1	334	0	0	0.1	334
graph27	1	0	0.1	926	0	0	0.0	199	0	0	0.0	162	0	0	0.0	162	0	0	0.0	161
graph28	0	0	0.1	679	0	0	0.0	147	0	0	0.0	140	0	0	0.0	140	0	0	0.0	140
graph29	0	0	0.1	772	0	0	0.0	124	0	0	0.0	124	0	0	0.0	124	0	0	0.0	124
graph30	0	0	0.1	472	0	0	0.0	118	0	0	0.0	100	0	0	0.0	100	0	0	0.0	100

graph31	0	0	0.1	80	0	0	0.0	66	0	0	0.0	66	0	0	0.0	66	0	0	0.0	66
graph32	0	0	0.0	51	0	0	0.0	51	0	0	0.0	51	0	0	0.0	51	0	0	0.0	51
hamming10-2	1	12	0.0	494	1	12	0.0	494	1	12	0.0	494	1	12	0.0	494	1	12	0.0	494
hamming10-4	1	0	0.0	33	1	0	0.0	33	1	0	0.0	33	1	0	0.0	33	1	0	0.0	33
hamming6-2	0	0	0.0	33	0	0	0.0	33	0	0	0.0	33	0	0	0.0	33	0	0	0.0	33
hamming6-4	0	0	0.0	5	0	0	0.0	5	0	0	0.0	5	0	0	0.0	5	0	0	0.0	5
hamming8-2	1	0	0.0	129	0	0	0.0	129	0	0	0.0	129	0	0	0.0	129	0	0	0.0	129
hamming8-4	1	0	0.0	17	0	0	0.0	17	0	0	0.0	17	0	0	0.0	17	0	0	0.0	17
inithx.i.1	1	2	20.6	1.4M	0	0	62.7	3.8M	0	0	86.6	3.1M	0	0	95.6	3.1M	0	0	34.7	1.5M
inithx.i.2	1	0	0.0	365	0	0	0.0	365	0	0	0.0	365	0	0	0.0	365	0	0	0.0	365
inithx.i.3	1	0	0.0	361	0	0	0.0	361	0	0	0.0	361	0	0	0.0	361	0	0	0.0	361
johnson16-2-4	0	0	0.0	9	0	0	0.0	9	0	0	0.0	9	0	0	0.0	9	0	0	0.0	9
johnson32-2-4	1	0	0.0	17	1	0	0.0	17	1	0	0.0	17	1	0	0.0	17	1	0	0.0	17
johnson8-2-4	0	0	0.0	5	0	0	0.0	5	0	0	0.0	5	0	0	0.0	5	0	0	0.0	5
johnson8-4-4	0	0	0.0	15	0	0	0.0	15	0	0	0.0	15	0	0	0.0	15	0	0	0.0	15
keller4	0	0	0.0	1K	0	0	0.0	79	0	0	0.0	70	0	0	0.0	29	0	0	0.0	28
keller5	1	4	69.6	4.4M	1	1	59.6	1.9M	1	1	90.5	1.9M	1	0	34.4	1K	1	0	34.5	957
keller6	1	1	60.2	0.9M	1	1	0.7	6K	1	1	0.8	6K	1	0	48.8	275	1	0	49.6	272
mulsol.i.1	1	0	0.0	101	0	0	0.0	101	0	0	0.0	101	0	0	0.0	101	0	0	0.0	101
mulsol.i.2	1	0	0.0	91	0	0	0.0	91	0	0	0.0	91	0	0	0.0	91	0	0	0.0	91
mulsol.i.3	1	0	0.0	87	0	0	0.0	87	0	0	0.0	87	0	0	0.0	87	0	0	0.0	87
mulsol.i.4	1	0	0.0	98	0	0	0.0	94	0	0	0.0	91	0	0	0.0	91	0	0	0.0	91
mulsol.i.5	1	0	0.0	89	0	0	0.0	89	0	0	0.0	89	0	0	0.0	89	0	0	0.0	89
p_hat1000-1	1	0	0.3	1K	0	0	0.0	91	1	0	0.1	78	1	0	0.1	24	1	0	0.1	23
p_hat1000-2	1	4	97.5	5.1M	1	2	8.8	0.6M	1	2	11.5	0.5M	1	1	52.0	44K	1	1	52.2	43K
p_hat1000-3	1	5	112.7	6.7M	1	2	99.7	4.7M	1	2	111.5	4.6M	1	0	10.0	3K	1	0	9.7	3K
p_hat1500-1	1	1	0.4	323	1	0	97.1	0.3M	1	0	149.1	0.3M	1	1	0.2	29	1	1	0.2	28
p_hat1500-2	1	5	19.6	0.6M	1	2	0.2	5K	1	2	0.2	4K	1	0	33.9	378	1	0	33.3	376
p_hat1500-3	1	7	1.4	48K	1	3	115.0	4.7M	1	3	156.7	4.6M	1	2	139.4	2K	1	2	140.4	2K
p_hat300-1	0	0	0.0	464	0	0	0.0	47	0	0	0.0	43	0	0	0.0	14	0	0	0.0	14
p_hat300-2	1	0	0.2	29K	0	0	0.0	106	0	0	0.0	100	0	0	0.0	34	0	0	0.0	34
p_hat300-3	1	2	0.2	33K	1	2	0.0	138	1	2	0.0	86	0	0	13.3	232	0	0	13.2	232
p_hat500-1	0	0	0.1	1K	0	0	0.0	84	0	0	0.0	78	0	0	0.1	22	0	0	0.1	21
p_hat500-2	1	3	32.4	3.2M	1	0	97.0	2.2M	1	0	176.3	2.2M	0	0	7.3	157	0	0	7.6	157
p_hat500-3	1	4	6.3	0.7M	1	0	59.5	3.1M	1	0	93.5	3.1M	1	0	2.4	156	1	0	2.3	156
p_hat700-1	0	0	30.7	0.9M	0	0	4.0	18K	0	0	6.7	17K	0	0	11.1	71	0	0	11.0	70
p_hat700-2	1	1	0.6	36K	1	0	0.0	1K	1	0	0.0	1K	1	0	0.0	97	1	0	0.0	96
p_hat700-3	1	4	6.3	0.5M	1	0	0.4	20K	1	0	0.3	14K	1	0	0.0	170	1	0	0.0	169
san1000	1	6	101.8	3.1M	1	4	165.7	2.4M	1	5	0.5	3K	0	0	163.1	936	0	0	161.6	932
san200_0.7_1	1	13	0.1	31K	0	0	0.1	7K	0	0	0.2	7K	0	0	0.2	199	0	0	0.2	199
san200_0.7_2	1	3	66.3	13.3M	0	0	7.7	0.6M	0	0	13.3	0.6M	0	0	0.4	257	0	0	0.4	255
san200_0.9_1	1	23	3.1	0.7M	1	21	77.7	11.4M	1	21	111.7	9.1M	0	3	2.0	45K	0	3	2.2	45K
san200_0.9_2	1	19	118.7	24.6M	0	0	20.5	1.3M	0	0	20.6	1.3M	0	0	0.7	588	0	0	0.8	585
san200_0.9_3	1	4	48.7	9.5M	1	3	0.6	67K	1	3	0.8	66K	1	1	136.1	2K	1	1	137.8	2K
san400_0.5_1	1	5	0.1	2K	0	0	0.9	31K	0	0	1.7	31K	0	0	0.8	250	0	0	0.8	249
san400_0.7_1	1	19	0.0	22	0	0	51.3	1.0M	0	0	59.8	1.0M	0	0	16.1	455	0	0	15.4	452
san400_0.7_2	1	15	0.0	16	0	0	21.1	0.6M	0	0	40.3	0.6M	0	0	15.4	485	0	0	15.7	485
san400_0.7_3	1	5	9.1	1.0M	1	2	42.9	3.0M	1	2	74.0	3.0M	1	0	74.5	1K	1	0	74.5	1K
san400_0.9_1	1	7	11.3	1.6M	1	2	132.3	9.2M	1	3	3.4	0.2M	1	0	30.9	639	1	0	30.8	642
sanr200_0.7	1	0	3.1	0.5M	0	0	0.1	3K	0	0	0.1	3K	0	0	0.1	58	0	0	0.1	57
sanr200_0.9	1	3	4.8	1.0M	1	0	27.7	1.9M	1	0	25.8	1.9M	1	0	2.6	178	1	0	2.6	178
sanr400_0.5	1	0	43.8	3.8M	0	0	5.2	78K	0	0	4.5	78K	0	0	13.8	136	0	0	13.9	136
sanr400_0.7	1	1	10.5	1.1M	1	0	59.5	3.4M	1	0	86.8	3.4M	1	0	95.3	1K	1	0	95.1	1K
zeroin.i.1	1	6	0.6	0.1M	0	3	7.7	1.1M	0	3	8.2	0.5M	0	3	7.9	0.5M	0	3	1.7	0.2M
zeroin.i.2	1	0	0.0	128	0	0	0.0	128	0	0	0.0	128	0	0	0.0	128	0	0	0.0	128
zeroin.i.3	1	0	0.0	124	0	0	0.0	124	0	0	0.0	124	0	0	0.0	124	0	0	0.0	124

Appendix D

Bibliography

[1] Faisal N Abu-Khzam, Rebecca L Collins, Michael R Fellows, Michael A Langston, W Henry Suters, and Christopher T Symons. Kernelization algorithms for the vertex cover problem: Theory and experiments. *ALENEX/ANALC*, 69, 2004.

[2] Faisal N Abu-Khzam, Michael R Fellows, Michael A Langston, and W Henry Suters. Crown structures for vertex cover kernelization. *Theory of Computing Systems*, 41(3):411–430, 2007.

[3] Noga Alon, Raphael Yuster, and Uri Zwick. Color-coding. *Journal of the ACM*, 42(4):844–856, 1995.

[4] Libor Barto. The dichotomy for conservative constraint satisfaction problems revisited. In *Proceedings of the Twenty-Sixth Annual IEEE Symposium on Logic in Computer Science*, pages 301–310, 2011.

[5] Libor Barto. Finitely related algebras in congruence distributive varieties have near unanimity terms. *Canadian Journal of Mathematics*, 65(1):3–21, 2013.

[6] Libor Barto. The collapse of the bounded width hierarchy. *Journal of Logic and Computation*, 26(3):923–943, 2016.

[7] Libor Barto and Marcin Kozik. Constraint satisfaction problems solvable by local consistency methods. *Journal of the ACM*, 61(1), 2014.

[8] Libor Barto, Marcin Kozik, and Todd Niven. The csp dichotomy holds for digraphs with no sources and no sinks (a positive answer to a conjecture of bang-jensen and hell). *SIAM Journal on Computing*, 38(5):1782–1802, 2009.

[9] Libor Barto and David Stanovský. Polymorphisms of small digraphs. *Novi Sad Journal of Mathematics*, 40(2):95–109, 2010.

[10] Joel Berman, Paweł Idziak, Petar Marković, Ralph McKenzie, Matthew Valeriote, and Ross Willard. Varieties with few subalgebras of powers. *Transactions of the American Mathematical Society*, 362(3):1445–1473, 2010.

[11] Christian Bessiere, Clément Carbonnel, Emmanuel Hebrard, George Katsirelos, and Toby Walsh. Detecting and exploiting subproblem tractability. In *Proceedings of the Twenty-Third International Joint Conference on Artificial Intelligence*, pages 468–474, 2013.

[12] Christian Bessiere, Emmanuel Hebrard, Brahim Hnich, Zeynep Kiziltan, Claude-Guy Quimper, and Toby Walsh. The Parameterized Complexity of Global Constraints. In *Proceedings of the Twenty-Third AAAI Conference on Artificial Intelligence*, pages 235–240, 2008.

[13] Christian Bessiere, Emmanuel Hebrard, Brahim Hnich, and Toby Walsh. The tractability of global constraints. In *Proceedings of the Tenth International Conference on Principles and Practice of Constraint Programming*, pages 716–720, 2004.

[14] Christian Bessiere, Emmanuel Hebrard, George Katsirelos, Zeynep Kiziltan, and Toby Walsh. Ranking constraints. In *Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence*, pages 705–711, 2016.

[15] Hans L. Bodlaender, Bart M.P. Jansen, and Stefan Kratsch. Kernelization lower bounds by cross-composition. *SIAM Journal on Discrete Mathematics*, 28(1):277–305, 2014.

[16] Lucas Bordeaux, Marco Cadoli, and Toni Mancini. A unifying framework for structural properties of csps: Definitions, complexity, tractability. *Journal of Artificial Intelligence Research*, 32:607–629, 2008.

[17] Andrei A. Bulatov. Mal'tsev constraints are tractable. Technical Report RR-02-05, Computing Laboratory, Oxford University, 2002.

[18] Andrei A. Bulatov. Combinatorial problems raised from 2-semilattices. *Journal of Algebra*, 298(2):321–339, 2006.

[19] Andrei A. Bulatov. A dichotomy theorem for constraint satisfaction problems on a 3-element set. *Journal of the ACM*, 53(1):66–120, 2006.

[20] Andrei A. Bulatov. Bounded relational width. Technical report, School of Computer Science, Simon Fraser University, 2010.

[21] Andrei A. Bulatov. The complexity of conservative constraint satisfaction problems. *ACM Transactions on Computational Logic*, 12(4):24, 2011.

[22] Andrei A. Bulatov. Conservative constraint satisfaction re-revisited. *Journal of Computer and System Sciences*, 82(2):347 – 356, 2016.

[23] Andrei A. Bulatov. Graphs of relational structures: Restricted types. In *Proceedings of the Thirty-first Annual ACM/IEEE Symposium on Logic in Computer Science*, pages 642–651, 2016.

[24] Andrei A. Bulatov and Víctor Dalmau. A simple algorithm for Mal'tsev constraints. *SIAM Journal on Computing*, 36(1):16–27, 2006.

[25] Andrei A. Bulatov and Peter G. Jeavons. Algebraic structures in combinatorial problems. Technical Report MATH-AL-4-2001, Technische Universität Dresden, 2001.

[26] Andrei A. Bulatov, Peter G. Jeavons, and Andrei A. Krokhin. Classifying the complexity of constraints using finite algebras. *SIAM Journal on Computing*, 34:720–742, 2005.

[27] Andrei A. Bulatov and Dániel Marx. The complexity of global cardinality constraints. *Logical Methods in Computer Science*, 6:1–27, 2010.

[28] Jonathan F. Buss and Judy Goldsmith. Nondeterminism within p. *SIAM Journal on Computing*, 22(3):560–572, 1993.

[29] Yixin Cao and Dániel Marx. Interval deletion is fixed-parameter tractable. *ACM Transactions on Algorithms*, 11(3):21, 2015.

[30] Clément Carbonnel. The dichotomy for conservative constraint satisfaction is polynomially decidable. In *Proceedings of the Twenty-Second International Conference on Principles and Practice of Constraint Programming*, pages 130–146, 2016.

[31] Clément Carbonnel. The Meta-Problem for Conservative Mal'tsev Constraints. In *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence*, 2016.

[32] Clément Carbonnel and Martin C. Cooper. Tractability in constraint satisfaction problems: a survey. *Constraints*, 21(2):115–144, 2016.

[33] Clément Carbonnel, Martin C. Cooper, and Emmanuel Hebrard. On backdoors to tractable constraint languages. In *Proceedings of the Twentieth International Conference on Principles and Practice of Constraint Programming*, pages 224–239, 2014.

[34] Clément Carbonnel and Emmanuel Hebrard. Propagation via kernelization: The vertex cover constraint. In *Proceedings of the Twenty-Second International Conference on Principles and Practice of Constraint Programming*, pages 147–156, 2016.

[35] Catarina Carvalho, László Egri, Marcel Jackson, and Todd Niven. On Maltsev digraphs. In *Computer Science–Theory and Applications*, pages 181–194. Springer, 2011.

[36] Hubie Chen and Benoit Larose. Asking the metaquestions in constraint tractability. *ACM Transactions on Computation Theory*, 2017. To appear.

[37] Jianer Chen, Xiuzhen Huang, Iyad A. Kanj, and Ge Xia. Strong computational lower bounds via parameterized complexity. *Journal of Computer and System Sciences*, 72(8):1346 – 1367, 2006.

[38] Jianer Chen, Iyad A. Kanj, and Ge Xia. Improved parameterized upper bounds for vertex cover. In *Proceedings of the International Symposium on Mathematical Foundations of Computer Science*, pages 238–249, 2006.

[39] Miroslav Chlebík and Janka Chlebíková. Crown reductions for the minimum weighted vertex cover problem. *Discrete Applied Mathematics*, 156(3):292–312, 2008.

[40] Alan Cobham. The intrinsic computational difficulty of functions. In *Proceedings of the International Congress for Logic, Methodology, and Philosophy of Science*, pages 24–30, 1964.

[41] Stephen A. Cook. The complexity of theorem-proving procedures. In *Proceedings of the Third annual ACM symposium on Theory of Computing*, pages 151–158, 1971.

[42] Martin C. Cooper, David A. Cohen, and Peter Jeavons. Characterising tractable constraints. *Artificial Intelligence*, 65(2):347–361, 1994.

[43] Martin C. Cooper and Stanislav Živný. Hybrid tractable classes of constraint problems. In *The Constraint Satisfaction Problem: Complexity and Approximability*, 2017. To appear.

[44] Nadia Creignou, Arne Meier, Julian-Steffen Müller, Johannes Schmidt, and Heribert Vollmer. Paradigms for parameterized enumeration. In *Proceedings of the International Symposium on Mathematical Foundations of Computer Science*, pages 290–301, 2013.

[45] Marek Cygan, Fedor V Fomin, Łukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michał Pilipczuk, and Saket Saurabh. *Parameterized algorithms*. Springer, 2015.

[46] Víctor Dalmau. Generalized majority-minority operations are tractable. *Logical Methods in Computer Science*, 2(4), 2006.

[47] Víctor Dalmau. There are no pure relational width 2 constraint satisfaction problems. *Information Processing Letters*, 109(4):213 – 218, 2009.

[48] Víctor Dalmau and Benoit Larose. Mal'tsev + datalog → symmetric datalog. In *Proceedings of the Twenty-Third Annual IEEE Symposium on Logic in Computer Science*, pages 297–306, 2008.

[49] Víctor Dalmau and Justin Pearson. Set functions and width 1 problems. In *Proceedings of the Fifth International Conference on Principles and Practice of Constraint Programming*, pages 159–173, 1999.

[50] Peter Damaschke. Parameterized enumeration, transversals, and imperfect phylogeny reconstruction. *Theoretical Computer Science*, 351(3):337–350, 2006.

[51] Philippe David. Using pivot consistency to decompose and solve functional CSPs. *Journal of Artificial Intelligence Research*, 2:447–474, 1995.

[52] Rina Dechter. Enhancement schemes for constraint processing: Backjumping, learning and cutset decomposition. *Artificial Intelligence*, 41(3):273–312, 1990.

[53] Rina Dechter. *Constraint Processing*. 2003.

[54] Frank K. H. A. Dehne, Michael R. Fellows, Michael A. Langston, Frances A. Rosamond, and Kim Stevens. An $O(2^{O(k)}n^3)$ FPT algorithm for the undirected feedback vertex set problem. In *Proceedings of the Eleventh Annual Conference on Computing and Combinatorics*, pages 859–869, 2005.

[55] Yves Deville, Olivier Barette, and Pascal Van Hentenryck. Constraint satisfaction over connected row convex constraints. *Artificial Intelligence*, 109(1-2):243–271, 1999.

[56] Bistra Dilkina, Carla P Gomes, and Ashish Sabharwal. Tradeoffs in the complexity of backdoor detection. In *Proceedings of the Thirteenth International Conference on Principles and Practice of Constraint Programming*, pages 256–270, 2007.

[57] Rod G Downey and Michael R Fellows. Fixed-parameter tractability and completeness I: Basic results. *SIAM Journal on Computing*, 24(4):873–921, 1995.

[58] Rod G. Downey and Michael R. Fellows. Fixed-parameter tractability and completeness II: On completeness for W[1]. *Theoretical Computer Science*, 141(1):109 – 131, 1995.

[59] Rodney G Downey and Michael Ralph Fellows. *Parameterized complexity*. Springer Science & Business Media, 2012.

[60] Martin Dyer and David Richerby. An effective dichotomy for the counting constraint satisfaction problem. *SIAM Journal on Computing*, 42(3):1245–1274, 2013.

[61] Guillaume Escamocher and Barry O'Sullivan. On the minimal constraint satisfaction problem: Complexity and generation. In *Proceedings of the Ninth International Conference on Combinatorial Optimization and Applications*, pages 731–745, 2015.

[62] Tomás Feder and Moshe Y. Vardi. The computational structure of monotone monadic SNP and constraint satisfaction: A study through Datalog and group theory. *SIAM Journal of Computing*, 28(1):57–104, 1998.

[63] Johannes K Fichte and Stefan Szeider. Backdoors to normality for disjunctive logic programs. *ACM Transactions on Computational Logic*, 17(1):7, 2015.

[64] Johannes Klaus Fichte and Stefan Szeider. Backdoors to tractable answer set programming. *Artificial Intelligence*, 220:64–103, 2015.

[65] Eugene C. Freuder. Complexity of k-tree structured constraint satisfaction problems. In *Proceedings of the 8th National Conference on Artificial Intelligence*, pages 4–9, 1990.

[66] Robert Ganian, Ramanujan M.S., and Stefan Szeider. Backdoors to tractable valued csp. In *Proceedings of the 22nd International Conference on Principles and Practice of Constraint Programming*, 2016.

[67] Michael Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W.H.Freeman and Company, San Francisco, CA., 1979.

[68] Shivam Garg and Geevarghese Philip. Raising the bar for vertex cover: fixed-parameter tractability above a higher guarantee. In *Proceedings of the Twenty-Seventh Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1152–1166. SIAM, 2016.

[69] Serge Gaspers, Neeldhara Misra, Sebastian Ordyniak, Stefan Szeider, and Stanislav Živný. Backdoors into heterogeneous classes of SAT and CSP. In *Proceedings of the Twenty-Eighth AAAI Conference on Artificial Intelligence*, pages 2652–2658, 2014.

[70] Serge Gaspers, Sebastian Ordyniak, MS Ramanujan, Saket Saurabh, and Stefan Szeider. Backdoors to q-horn. *Algorithmica*, 74(1):540–557, 2016.

[71] Serge Gaspers and Stefan Szeider. Kernels for global constraints. In *Proceedings of the Twenty-Second International Joint Conference on Artificial Intelligence*, IJCAI'11, pages 540–545. AAAI Press, 2011.

[72] Serge Gaspers and Stefan Szeider. Backdoors to satisfaction. In *The Multivariate Algorithmic Revolution and Beyond*, pages 287–317. Springer, 2012.

[73] Richard Gault and Peter G. Jeavons. Implementing a test for tractability. *Constraints*, 2004.

[74] David Geiger. Closed systems of functions and predicates. *Pacific journal of mathematics*, 27(1):95–100, 1968.

[75] Ian P. Gent and Barbara M. Smith. Symmetry breaking in constraint programming. In *Proceedings of the Fourteenth European Conference on Artificial Intelligence*, pages 599–603, 2000.

[76] Oded Goldreich. *P, NP, and NP-Completeness: The basics of computational complexity*. Cambridge University Press, 2010.

[77] Georg Gottlob. On minimal constraint networks. In *Proceedings of the Seventeenth International Conference on Principles and Practice of Constraint Programming*, pages 325–339, 2011.

[78] Martin J. Green and David A. Cohen. Domain permutation reduction for constraint satisfaction problems. *Artificial Intelligence*, 172(8-9):1094–1118, 2008.

[79] Martin Grohe. The complexity of homomorphism and constraint satisfaction problems seen from the other side. *Journal of the ACM*, 54(1):1–24, 2007.

[80] Martin Grohe and Dániel Marx. Constraint solving via fractional edge covers. *ACM Transactions on Algorithms*, 11(1):4, 2014.

[81] Jiong Guo, Jens Gramm, Falk Hüffner, Rolf Niedermeier, and Sebastian Wernicke. Compression-based fixed-parameter algorithms for feedback vertex set and edge bipartization. *Journal of Computer and System Sciences*, 72(8):1386–1396, 2006.

[82] William D Harvey and Matthew L Ginsberg. Limited discrepancy search. In *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence*, pages 607–615, 1995.

[83] Emmanuel Hebrard. Mistral, a Constraint Satisfaction Library. In *The Third International CSP Solver Competition*, pages 31–40, 2008.

[84] Pavol Hell and Jaroslav Nešetřil. The core of a graph. *Discrete Mathematics*, 109(1):117–126, 1992.

[85] Pavol Hell and Jaroslav Nešetřil. On the complexity of H-coloring. *Journal of Combinatorial Theory, Ser.B*, 48:92–110, 1990.

[86] Danny Hermelin and Xi Wu. Weak compositions and their applications to polynomial lower bounds for kernelization. In *Proceedings of the Twenty-Third Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 104–113, 2012.

[87] Marijn J. H. Heule, Oliver Kullmann, and Victor W. Marek. Solving and verifying the boolean pythagorean triples problem via cube-and-conquer. In *Proceedings of the Nineteenth International Conference Theory and Applications of Satisfiability Testing*, 2016.

[88] John E. Hopcroft and Richard M. Karp. An $n^{5/2}$ Algorithm for Maximum Matchings in Bipartite Graphs. *SIAM Journal on Computing*, 2(4):225–231, 1973.

[89] Falk Hüffner, Christian Komusiewicz, Hannes Moser, and Rolf Niedermeier. Fixed-parameter algorithms for cluster vertex deletion. *Theory of Computing Systems*, 47(1):196–217, 2010.

[90] Pawel M. Idziak, Petar Markovic, Ralph McKenzie, Matthew Valeriote, and Ross Willard. Tractability and learnability arising from algebras with few subpowers. *SIAM Journal on Computing*, 39(7):3023–3037, 2010.

[91] Yoichi Iwata, Magnus Wahlström, and Yuichi Yoshida. Half-integrality, lp-branching, and fpt algorithms. *SIAM Journal on Computing*, 45(4):1377–1411, 2016.

[92] Peter G. Jeavons. On the algebraic structure of combinatorial problems. *Theoretical Computer Science*, 200:185–204, 1998.

[93] Peter G. Jeavons, David A. Cohen, and Martin C. Cooper. Constraints, consistency and closure. *Artificial Intelligence*, 101(1–2):251–265, 1998.

[94] Peter G. Jeavons, David A. Cohen, and Marc Gyssens. Closure properties of constraints. *Journal of the ACM*, 44(4), 1997.

[95] Peter G. Jeavons and Martin C. Cooper. Tractable constraints on ordered domains. *Artificial Intelligence*, 79(2):327–339, 1995.

[96] Philippe Jégou, Samba Ndojh Ndiaye, and Cyril Terrioux. Dynamic heuristics for backtrack search on tree-decomposition of csps. In *Proceedings of the Twentieth International Joint Conference on Artifical Intelligence*, pages 112–117, 2007.

[97] Narendra Jussien and Olivier Lhomme. Local search with constraint propagation and conflict-based heuristics. *Artificial Intelligence*, 139(1):21–45, 2002.

[98] Richard M. Karp. Reducibility Among Combinatorial Problems. In *Complexity of Computer Computations*, pages 85–103. 1972.

[99] George Katsirelos. *Nogood processing in CSPs*. PhD thesis, University of Toronto, 2008.

[100] Marcin Kozik. Weak consistency notions for all the csps of bounded width. In *Proceedings of the Thirty-First Annual ACM/IEEE Symposium on Logic in Computer Science*, pages 633–641, 2016.

[101] Marcin Kozik, Andrei Krokhin, Matt Valeriote, and Ross Willard. Characterizations of several Maltsev conditions. *Algebra universalis*, 73(3-4):205–224, 2015.

[102] Stefan Kratsch. A randomized polynomial kernelization for vertex cover with a smaller parameter. In *Proceedings of the Twenty-Fourth Annual European Symposium on Algorithms*, pages 59:1–59:17, 2016.

[103] Stefan Kratsch and Magnus Wahlström. Representative sets and irrelevant vertices: New tools for kernelization. In *Proceedings of the Fifty-Third IEEE Annual Symposium on Foundations of Computer Science*, pages 450–459, 2012.

[104] Martin Kronegger, Sebastian Ordyniak, and Andreas Pfandler. Backdoors to planning. In *Proceedings of the Twenty-Eigth AAAI Conference on Artificial Intelligence*, pages 2300–2307, 2014.

[105] Martin Kronegger, Sebastian Ordyniak, and Andreas Pfandler. Variable-deletion backdoors to planning. In *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence*, pages 3305–3312, 2015.

[106] Richard E. Ladner. On the structure of polynomial time reducibility. *Journal of the ACM*, 22(1):155–171, 1975.

[107] Michael Lampis. A kernel of order 2k-clog(k) for vertex cover. *Information Processing Letters*, 111(23):1089–1091, 2011.

[108] Benoit Larose, Cynthia Loten, and Claude Tardif. A characterisation of first-order constraint satisfaction problems. *Logical Methods in Computer Science*, 3(4), 2007.

[109] John M. Lewis and Mihalis Yannakakis. The node-deletion problem for hereditary properties is NP-complete. *Journal of Computer and System Sciences*, 20(2):219–230, 1980.

[110] Tomasz Łuczak and Jaroslav Nešetřil. A probabilistic approach to the dichotomy problem. *SIAM Journal on Computing*, 36(3):835–843, 2006.

[111] Alan K. Mackworth. Consistency in networks of relations. *Artificial Intelligence*, 8:99–118, 1977.

[112] Alan K. Mackworth. On reading sketch maps. In *Proceedings of the Fifth International Joint Conference on Artificial Intelligence*, pages 598–606, 1977.

[113] Michael Maher, Nina Narodytska, Claude-Guy Quimper, and Toby Walsh. Flow-based propagators for the sequence and related global constraints. In *Proceedings of the Fourteenth International Conference on Principles and Practice of Constraint Programming*, pages 159–174, 2008.

[114] Ugo Montanari. Networks of constraints: Fundamental properties and applications to picture processing. *Information Sciences*, 7:95–132, 1974.

[115] Matthew W Moskewicz, Conor F Madigan, Ying Zhao, Lintao Zhang, and Sharad Malik. Chaff: Engineering an efficient sat solver. In *Proceedings of the 38th annual Design Automation Conference*, pages 530–535. ACM, 2001.

[116] Achref El Mouelhi. *Tractable Classes of Constraint Satisfaction Problems: From Theory to Practice*. PhD thesis, Aix-Marseille Université, 2014.

[117] George L Nemhauser and Leslie E Trotter Jr. Vertex packings: structural properties and algorithms. *Mathematical Programming*, 8(1):232–248, 1975.

[118] Naomi Nishimura, Prabhakar Ragde, and Stefan Szeider. Detecting backdoor sets with respect to horn and binary clauses. *Proceedings of the Seventh International Conference on Theory and Applications of Satisfiability Testing*, 4:96–103, 2004.

[119] Jaroslav Opatrny. Total ordering problem. *SIAM Journal on Computing*, 8(1):111–114, 1979.

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

[121] Igor Razgon and Barry O'Sullivan. Almost 2-sat is fixed-parameter tractable. *Journal of Computer and System Sciences*, 75(8):435–450, 2009.

[122] Bruce Reed, Kaleigh Smith, and Adrian Vetta. Finding odd cycle transversals. *Operations Research Letters*, 32(4):299–301, 2004.

[123] Jean-Charles Régin. Global constraints: A survey. In *Hybrid optimization*, pages 63–134. Springer, 2011.

[124] Neil Robertson and Paul D. Seymour. Graph minors. xiii. the disjoint paths problem. *Journal of Combinatorial Theory, Series B*, 63(1):65 – 110, 1995.

[125] Yongshao Ruan, Henry A Kautz, and Eric Horvitz. The backdoor key: A path to understanding problem hardness. In *Proceedings of the Nineteenth National Conference on Artificial Intelligence*, volume 4, pages 118–123, 2004.

[126] Yousef Saad. *Iterative methods for sparse linear systems*. Siam, 2003.

[127] Marko Samer and Stefan Szeider. Backdoor trees. In *Proceedings of the Twenty-Third AAAI Conference on Artificial Intelligence*, volume 8, pages 13–17, 2008.

[128] Thomas J. Schaefer. The complexity of satisfiability problems. In *Proceedings 10th ACM Symposium on Theory of Computing, STOC'78*, pages 216–226, 1978.

[129] Mark H Siggers. A strong mal'cev condition for locally finite varieties omitting the unary type. *Algebra universalis*, 64(1-2):15–20, 2010.

[130] Larry Stockmeyer. Planar 3-colorability is polynomial complete. *SIGACT News*, 5(3):19–25, July 1973.

[131] Stefan Szeider. Backdoor sets for dll subsolvers. *Journal of Automated Reasoning*, 35(1):73–88, 2005.

[132] Balder ten Cate and Víctor Dalmau. The product homomorphism problem and applications. In *Proceedings of the Eighteenth International Conference on Database Theory*, 2015.

[133] Peter van Beek and Rina Dechter. On the minimality and global consistency of row-convex constraint networks. *Journal of the ACM*, 42(3):543–561, 1995.

[134] Ross Willard. Testing expressibility is hard. In *Proceedings of the Sixteenth International Conference on Principles and Practice of Constraint Programming*, pages 9–23, 2010.

[135] Ryan Williams, Carla P. Gomes, and Bart Selman. Backdoors to typical case complexity. In *Proceedings of the Eighteenth International Joint Conference on Artificial Intelligence*, pages 1173–1178, 2003.

Résumé

Le problème de satisfaction de contraintes (CSP) est un problème NP-complet classique en intelligence artificielle qui a suscité un engouement important de la communauté scientifique grâce à la richesse de ses aspects pratiques et théoriques. Cependant, au fil des années un gouffre s'est creusé entre les praticiens, qui développent des méthodes exponentielles mais efficaces pour résoudre des instances industrielles, et les théoriciens qui conçoivent des algorithmes sophistiqués pour résoudre en temps polynomial certaines restrictions de CSP dont l'intérêt pratique n'est pas avéré. Dans cette thèse nous tentons de réconcilier les deux communautés en fournissant des méthodes polynomiales pour tester automatiquement l'appartenance d'une instance de CSP à une sélection de classes traitables majeures. Anticipant la possibilité que les instances réelles ne tombent que rarement dans ces classes traitables, nous analysons également de manière systématique la possibilité de décomposer efficacement une instance en sous-problèmes traitables en utilisant des méthodes de complexité paramétrée. Finalement, nous introduisons un cadre général pour exploiter dans les CSP les idées développées pour la *kernelization*, un concept fondamental de complexité paramétrée jusqu'ici peu utilisé en pratique. Ce dernier point est appuyé par des expérimentations prometteuses.

Mots-clefs : problème de satisfaction de contraintes, classe traitable, polymorphisme, backdoor, complexité paramétrée, kernelization.

Abstract

The Constraint Satisfaction Problem (CSP) is a fundamental NP-complete problem with many applications in artificial intelligence. This problem has enjoyed considerable scientific attention in the past decades due to its practical usefulness and the deep theoretical questions it relates to. However, there is a wide gap between practitioners, who develop solving techniques that are efficient for industrial instances but exponential in the worst case, and theorists who design sophisticated polynomial-time algorithms for restrictions of CSP defined by certain algebraic properties. In this thesis we attempt to bridge this gap by providing polynomial-time algorithms to test for membership in a selection of major tractable classes. Even if the instance does not belong to one of these classes, we investigate the possibility of decomposing efficiently a CSP instance into tractable subproblems through the lens of parameterized complexity. Finally, we propose a general framework to adapt the concept of kernelization, central to parameterized complexity but hitherto rarely used in practice, to the context of constraint reasoning. Preliminary experiments on this last contribution show promising results.

Keywords: Constraint satisfaction problem, tractable class, polymorphism, backdoor, parameterized complexity, kernelization.