Hybrid tractability of constraint satisfaction problems with global constraints

Evgenij Thorstensen
St. Anne's College, Oxford

Thesis submitted for the degree of
Doctor of Philosophy

Department of Computer Science
University of Oxford
Hilary term 2013

HYBRID TRACTABILITY
OF CONSTRAINT SATISFACTION PROBLEMS
WITH GLOBAL CONSTRAINTS

Evgenij Thorstensen
St. Anne's College, Oxford
Doctor of Philosophy, Hilary term 2013

Abstract

A wide range of problems can be modelled as constraint satisfaction problems (CSPs), that is, a set of constraints that must be satisfied simultaneously. Constraints can either be represented extensionally, by explicitly listing allowed combinations of values, or intensionally, whether by an equation, propositional logic formula, or other means. Intensionally represented constraints, known as global constraints, are a powerful modelling technique, and many modern CSP solvers provide them. We give examples to show how problems that deal with product configuration can be modelled with such constraints, and how this approach relates to other modelling formalisms.

The complexity of CSPs with extensionally represented constraints is well understood, and there are several known techniques that can be used to identify tractable classes of such problems. For CSPs with global constraints, however, many of these techniques fail, and far fewer tractable classes are known. In order to remedy this state of affairs, we undertake a systematic review of research into the tractability of CSPs. In particular, we look at CSPs with extensionally represented constraints in order to understand why many of the techniques that give tractable classes for this case fail for CSPs with global constraints. The above investigation leads to two discoveries.

First, many restrictions on how the constraints of a CSP interact implicitly rely on a property of extensionally represented constraints to guarantee tractability. We identify this property as being a bound on the number of solutions in key parts of the instance, and find classes of global constraints that also possess this property. For such classes, we show that many known tractability results apply. Furthermore, global constraints allow us to treat entire CSP instances as constraints. We combine this observation with the above result, and obtain new tractable classes of CSPs by dividing a CSP into smaller CSPs drawn from known tractable classes.

Second, for CSPs that simply do not possess the above property, we look at how the constraints of an instance overlap, and how assignments to the overlapping parts extend to the rest of the problem. We show that assignments that extend in the same way can be identified. Combined with a new structural restriction, this observation leads to a second set of tractable classes.

We conclude with a summary, as well as some observations about potential for future work in this area.

Declaration and acknowledgements

I have written this thesis entirely by myself, although the results in Chapter 5 were obtained in collaboration with David Cohen, Peter Jeavons, and Stanislav Živný. I thank them for their comments and collaboration. Furthermore, parts of this thesis have appeared in the following papers:

- D. A. Cohen, P. G. Jeavons, E. Thorstensen, and S. Živný. Tractable Combinations of Global Constraints. In *Proceedings of the 19th International Conference on Principles and Practice of Constraint Programming (CP'13)*, volume 8124 of *Lecture Notes in Computer Science*, pp. 230–246. Springer, 2013.
- E. Thorstensen. Lifting Structural Tractability to CSP with Global Constraints. In *Proceedings of the 19th International Conference on Principles and Practice of Constraint Programming (CP'13)*, volume 8124 of *Lecture Notes in Computer Science*, pp. 661–677. Springer, 2013.

However, a thesis such as this does not get written without input and support from many people. First and foremost, I would like to thank my primary supervisor, Peter Jeavons. His soft-spoken support and constructive criticism have made this thesis possible. When my research floundered during the end of my second year, his suggestions got me back in the game. In short, I could not have had a better supervisor.

Second, I would like to thank David Cohen, who taught me, among other things, how to do proofs well, and how to structure mathematical thought and writing so as to make the complicated understandable. From him, I learned to separate what's true from how it is to be computed.

I would also like to thank my second supervisor, Georg Gottlob, as well as my colleagues Markus Aschinger, Conrad Drescher, Justyna Petke, András Salamon, and Stanislav Živný, for interesting discussions and useful feedback.

Proofreading academic text is hard and requires a lot of effort. I am therefore grateful to Conrad Drescher, Jess Pumphrey, and András Salamon for proofreading.

Finally, a special thank you goes to Rick's Cafe, on Cowley road. Their excellent coffee and athmosphere helped me discover several key results in this thesis.

Table of contents

Abstract **3**

Declaration and acknowledgements **5**

Table of contents **7**

1 Introduction **9**
1.1 Basic definitions 11
1.2 Contributions and thesis structure 13

2 Global constraints and their use **17**
In which we formally define global constraints, demonstrate their use on a few example configuration problems, and discuss how they relate to other formal systems.
2.1 Motivating examples: Configuration problems 18
2.1.1 Connected graph partition 18
2.1.2 PartnerUnits 18
2.1.3 Car part configuration 20
2.2 Global constraints: Definitions and examples 20
2.2.1 Examples of global constraint types 22
2.2.2 Using global constraints for configuration 24
2.3 Existing formal systems for configuration 25
2.3.1 Conditional CSP 25
2.3.2 Composite CSP 27
2.3.3 The logic $\exists\mathsf{FO}_{\rightarrow,\wedge,+}$ 29
2.4 Summary 30

3 Tractable classes of CSP instances: An overview **31**
In which we give an overview of current research into tractable classes of CSP instances, and observe that of the results discussed, a significant number apply only to classic CSP instances.
3.1 Tractability of classic CSP instances 31
3.1.1 Restrictions on structure 32
3.1.2 Restrictions on language 41

3.1.3 Hybrid tractability . . . 41
3.2 Research specific to global constraints . . . 42
3.2.1 Local consistency . . . 43
3.2.2 Propagators . . . 45
3.3 Discussion . . . 47

4 Tractability due to few solutions in key places 49

In which we show that having few solutions in key places is what makes many structural restrictions work for classic CSP instances. Furthermore, we discover that this property applies in several contexts.

4.1 Useful width . . . 51
4.2 Subproblem decompositions . . . 57
4.2.1 An extended example . . . 62
4.2.2 Variations on a theme: WCSP . . . 64
4.3 Back doors . . . 67
4.4 Summary . . . 68

5 Tractability via equivalence classes of assignments 71

Where we find a way to split many solutions into few equivalence classes, and discuss when it is a good idea to do so.

5.1 Cooperating constraints . . . 72
5.1.1 Examples of cooperating catalogues . . . 75
5.2 From cooperation to structure . . . 78
5.3 Relational structures and cores . . . 84
5.4 Summary . . . 86

6 Summary and future work 87

6.1 Open questions . . . 88
6.1.1 Subproblem decompositions . . . 88
6.1.2 Equivalence classes . . . 89

References 91

Chapter 1

Introduction

This thesis is about the computational complexity of constraint satisfaction problems (CSPs) with global constraints. Computational complexity [Pap94] as a field studies the difficulty of algorithmically solving abstract problems, such as finding the shortest path between two vertices in a directed graph, or partitioning a set of numbers into two sets whose sums are equal. Such abstract problems frequently occur as part of concrete problems such as route planning or circuit design. Knowing which properties of such abstract problems make them difficult or easy thus helps us to better solve concrete problems.

Constraint satisfaction problems [RvBW06,Wal96] are a specific kind of such abstract problems. Informally, a constraint satisfaction problem is to assign values to a set of variables V, subject to a set of constraints that specify which combinations of values are allowed among various subsets of V. To illustrate the above with a concrete example, consider the well-known 3-colourability problem [GJ79]: Given an undirected graph, is it possible to colour the vertices using only three colours such that no edge is monochromatic? Represented as a CSP, this problem can have the vertices of the given graph as variables, to be assigned values from the set $\{r, g, b\}$. Furthermore, there will be a constraint on every pair of variables connected by an edge, specifying that only unequal pairs of values are allowed on each such pair of variables. As 3-colourability is an NP-complete problem, it follows that CSP is a powerful formal system — every problem in the class NP can be expressed as a CSP.

At this point, we could ask how the combinations of values allowed by a constraint are specified. For the example above, we could simply list them — there are only nine possible pairs of values, six of them unequal, for every pair of variables. Our informal description thus translates easily into a precise one. However, consider the above problem with an additional requirement: The number of red and green vertices should be equal. This is a constraint on all the variables of the problem. The number of combinations it allows is exponential in the input size, and listing them explicitly would therefore not be feasible. Such considerations lead to the idea of global constraints — constraints that capture relations between a growing set of variables, where listing the allowed combinations up

front may not be feasible [vHK06].

Let us call constraints where the allowed combinations are listed explicitly *classic* constraints, to distinguish them from *global* constraints. Global constraints can be represented in a variety of ways, depending on the solver and on the specific constraint [GJ08, Tac09]. Modern CSP solvers typically implement specific algorithms for families or types of global constraints, such as equality, above, or constraints that limit the number of times a domain element may occur [GJM06, WNS97]. Such constraints are a natural way to model certain kinds of problems (see below).

In this thesis, we are concerned with the complexity of CSP instances with global constraints. More specifically, we are concerned with tractability, that is, with identifying classes of instances that can be solved in polynomial time. In Chapter 3, we will demonstrate that a variety of restrictions that yield tractable classes of classic CSP instances fail to do so for CSP instances with global constraints. It comes as no surprise that this is due to the often succinct representation of global constraints. In this thesis, we are going investigate the representation of various global constraints in order to find out why restrictions on classic constraints fail to yield tractability for them. Armed with that knowledge, we will come up with better restrictions and new tractable classes of CSP instance with global constraints.

Configuration

While CSP is a powerful formal system with or without global constraints, one area where having global constraints is particularly useful is that of configuration. The problem of configuration [MF89] is to find, given a set of available components and a specification, a configuration that satisfies the specification — the configuration being a subset of the components and a description of how they are to be connected. This provides a conceptual model for practical problems encountered in industry, such as rack configuration in telephone switching systems, car configuration, etc.

Configuration problems have a number of distinguishing features. Firstly, the number of components required is not known up front, but can change dynamically as components are selected [MF90, SFH98, SH93, VJ05]. Second, configuration problems frequently deal with various kinds of resource requirements, such as power and fuel requirements of components, their heat tolerance, and so on. Such requirements usually involve arithmetic over properties of components [SCKN07, SF99a, SH93].

Since the number of components required is not known up front, configuration problems frequently feature constraints over growing sets of variables. Due to resource requirements, they also frequently feature numeric constraints, where the values of a set of variables must not exceed a bound, or be equal to certain other values, etc. As such, configuration problems are a type of problems where global constraints become very useful for modelling. Configuration prob-

lems are thus another motivating factor in our study of CSPs with global constraints. In Chapter 2, we will look at several example configuration problems and show how they can be modelled using global constraints. In Section 2.3, we will also look at other formal systems for configuration, and show how they can be encoded as global constraints in a natural way.

1.1 Basic definitions

As far as possible, we will introduce formal defintions of concepts as they are needed. Some basic concepts, however, come up all over this thesis, and so we define them below.

Notation 1.1 (Variables) Let V be a set of variables, each with an associated set of domain elements. We denote the set of domain elements (the domain) of a variable v by $D(v)$. We extend this notation to arbitrary subsets of variables, W, by setting $D(W) = \bigcup_{v \in W} D(v)$. □

In the literature [GLS00, GLS02, GJC94, Mar09b, PJ97], the usual way of defining and working with constraints comes from database theory, and represents combinations of values allowed by a classic constraint by a relation, itself represented by tuples of values.

Notation 1.2 (Tuples) Let t be a tuple. We denote the number of elements in t by $|t|$, and the i-th coordinate of t by t_i, for $1 \leq i \leq |t|$. □

Using tuples, a classic constraint is a pair $\langle \sigma, \rho \rangle$, where σ is a tuple of variables, and $\rho \subseteq D(\sigma_1) \times \cdots \times D(\sigma_{|\sigma|})$ a set of tuples of domain values. In other words, ρ is a $|\sigma|$-ary relation over the domain elements.

We can then proceed to define a classic CSP instance as follows. A classic CSP instance $P = \langle V, C \rangle$ is a set of variables V together with a set C of classic constraints over V. A solution to P is a mapping s of variables to values such that for every constraint $\langle \sigma, \rho \rangle$ we have $\langle s(\sigma_1), \ldots, s(\sigma_{|\sigma|}) \rangle \in \rho$, that is, we have an allowed tuple of values for every constraint.

In this thesis, while we are going to use the above view of constraints occasionally, mainly we are going to adopt a different notation. We will represent the combinations a constraint allows by listing assignments, i.e. mappings from variables to values. This notation has been used previosuly [CG06], and will greatly simplify matters for us in the rest of this thesis.

Definition 1.3 (Variables and assignments) An *assignment* of a set of variables V is a function $\theta : V \to D(V)$ that maps every $v \in V$ to an element $\theta(v) \in D(v)$. We denote the restriction of θ to a set of variables $W \subseteq V$ by $\theta|_W$.

We also allow the special assignment $\perp$ of the empty set of variables. In particular, for every assignment θ, we have $\theta|_\emptyset = \perp$.

To write assignments explicitly, we will use substitution notation. An assignment to a set of variables $\{v, w\}$ that assigns 1 to both of them will be written as $\{v/1, w/1\}$. □

Definition 1.4 (Classic constraint) A *classic constraint* is a pair $\langle \sigma, \Theta \rangle$, where σ is a set of variables, and Θ is a set of assignments to σ. □

Observe that we can convert such a constraint to be represented by tuples simply by imposing an arbitrary linear order on the variables in σ to obtain a tuple of them, before applying each assignment in Θ to each variable in the tuple to obtain a set of tuples of domain values. Vice versa, every tuple of values in a classic constraint corresponds directly to an assignment.

Notation 1.5 (Variables of objects) Given a previously defined object Q involving variables, we will write $\mathcal{V}(Q)$ for the variables of Q when this is unambigous. Thus, the set of variables of a classic constraint $c = \langle \sigma, \Theta \rangle$ is $\mathcal{V}(c) = \sigma$, the set of variables of an assignment θ is $\mathcal{V}(\theta)$, etc. □

Definition 1.6 (Classic CSP instance) A *classic CSP instance* is a pair $\langle V, C \rangle$ where V is a set of variables, and C is a set of classic constraints with $\mathcal{V}(c) \subseteq V$ for every $c \in C$.

A *solution* to a classic CSP instance $\langle V, C \rangle$ is an assignment θ to V such that for every $\langle \sigma, \Theta \rangle \in C$ we have $\theta|_\sigma \in \Theta$. □

Example 1.7 To illustrate the two types of notation, we will use the problem of 3-colourability from the beginning of this chapter. Let G be a graph to be coloured with at most three colours. In either notation, the CSP instance $\langle V, C \rangle$ would have $V = \mathcal{V}(G)$ as variables, each with domain $\{r, g, b\}$, as well as a classic constraint $\langle \sigma, \rho \rangle \in C$ for every pair of variables connected by an edge in G.

Represented using tuples, a classic constraint $\langle \sigma, \rho \rangle$ on a pair of variables v, w would have $\sigma = \langle v, w \rangle$, and

$$\rho = \left\{ \begin{array}{c} \langle r, g \rangle, \langle g, r \rangle \\ \langle r, b \rangle, \langle b, r \rangle \\ \langle g, b \rangle, \langle b, g \rangle \end{array} \right\}.$$

Using assignments, on the other hand, we have $\sigma = \{v, w\}$, and

$$\rho = \left\{ \begin{array}{c} \{v/r, w/g\}, \{v/g, w/r\} \\ \{v/r, w/b\}, \{v/b, w/r\} \\ \{v/g, w/b\}, \{v/b, w/g\} \end{array} \right\}.$$

□

The usual operation of relational algebra, such as projection, work just fine using assignments instead of tuples.

Definition 1.8 (Projection) Let Θ be a set of assignments of a set of variables V. The *projection* of Θ onto a set of variables $X \subseteq V$ is the set of assignments $\pi_X(\Theta) = \{\theta|_X \mid \theta \in \Theta\}$. □

Note that when $\Theta = \emptyset$ we have $\pi_X(\Theta) = \emptyset$, but when $X = \emptyset$ and $\Theta \neq \emptyset$, we have $\pi_X(\Theta) = \{\perp\}$.

Definition 1.9 (Disjoint union of assignments) Let θ_1 and θ_2 be two assignments of disjoint sets of variables V_1 and V_2, respectively. The *disjoint union* of θ_1 and θ_2, denoted $\theta_1 \oplus \theta_2$, is the assignment of $V_1 \cup V_2$ such that $(\theta_1 \oplus \theta_2)(v) = \theta_1(v)$ for all $v \in V_1$, and $(\theta_1 \oplus \theta_2)(v) = \theta_2(v)$ for all $v \in V_2$. □

To discuss the complexity of CSP instances, the following general definitions will be useful.

Definition 1.10 (Tractable) Let $\mathcal{A}$ be a class of problem instances, for example CSP instances. We say that $\mathcal{A}$ is *tractable* if there exists a constant $c \in \mathbb{N}$ and an algorithm that can solve any instance $P \in \mathcal{A}$ in time $O(|P|^c)$, where $|P|$ denotes the size of the instance. □

The size of the instance is, roughly speaking, the number of bits needed to encode it. We will define the size of a CSP instance precisely in Section 2.2.

1.2 Contributions and thesis structure

The main contribution of this thesis is a systematic study of the complexity of CSP instances with global constraints. In particular, this study results in a set of techniques that extend known tractability results for classic constraint satisfaction problems to problems with global constraints.

To do this, we first give a new definition of global constraints and problems with such constraints in Chapter 2. This definition allows us to look at a global constraint's representation, thus providing a measure of input size, as well as the assignments allowed by the constraint. In Section 2.2, we show that the definition offered is flexible enough to capture global constraints discussed in previous research, and demonstrate how various global constraints can be used to model problems from the literature. In particular, in Section 2.3 we show that global constraints can be used to model configuration problems in a natural way, and also that they can be used to encode existing formal systems for configuration in a way that preserves their structure.

Second, in Chapter 3, we look at existing research into the complexity of constraint satisfaction problems, and provide an overview of the current state of the art. Here, we make the observation that many structural restrictions, such as query and hypertree width, that yield tractable classes of classic CSP instances fail to do so for instances with global constraints. As it turns out, this discrepancy comes from the fact that these restrictions rely on properties specific

to the extensional representation of classic constraints. This point motivates the research presented in this thesis. Since our definition of global constraints allows us to discuss representation in a uniform manner, we can try to identify these properties and look for global constraints that have them.

We begin this work in Chapter 4, where we collect a few insights from the literature to show that the structural restrictions mentioned produce tractability when a CSP instance has polynomially many solutions in its size, even if restricted to a subset of variables (Lemma 4.5). This property holds for classic instances, as they have “big” representations, and the size of an instance depends to a large extent on the size of the constraints’ representation.

To apply this insight, we adapt the notion of a useful structural restriction for a set of constraints, found in the literature. Using this notion, we show that query and hypertree width, among others, are useful for sparse sets of global constraints, where each member has polynomially many solutions in its size (Theorem 4.17). This gives us tractable classes for instances satisfying these structural restrictions if their constraints are sparse, provided we can for each constraint check in polynomial time if a partial assignment extends to a satisfying one. We also show that treewidth is the only useful structural restriction for arbitrary sets of global constraints (Theorem 4.9).

To extend this result, in Section 4.2 we consider treating CSP instances as global constraints. Our definitions allow this, and in a sense, doing this gives us the most general kind of global constraints. We define the notion of subproblem decompositions for a CSP instance, and show that the notions of useful structural restriction, sparse constraints, etc. all carry over to this case (Theorem 4.28). This means that the results obtained above hold also for this case. Furthermore, the notion of checking if a partial assignment extends to a satisfying one also carries over, and corresponds to having CSP instances drawn from known tractable classes. In other words, tractability results in this framework correspond to tractable combinations of existing tractable classes.

We next show that the condition of every constraint having few solutions in its size turns out to be too strong, and that it is in fact sufficient to consider the number of solutions of an instance when restricted to certain key sets of variables (Theorem 4.34). Doing so gives us a more general version of the tractable classes that we found for sparse constraints (Corollary 4.35).

As a bonus, we show how these results can be applied to weighted constraint satisfaction problems, where every satisfying assignment of every constraint has a cost, and we would like to find an optimal solution (Theorem 4.44 and Corollary 4.45).

Separately, in Chapter 5 we also consider the case of global constraints that simply do not possess the properties identified above — that is, those that simply have too many satisfying assignments. For such constraints, structural restrictions such as query and hypertree width will not give us tractable classes. However, by looking at how many different assignments overlapping sets of constraints can distinguish, we can define an equivalence relation on assignments.

For sets of constraints that end up with few equivalence classes of assignments, we define a new structural restriction that gives us yet another set of tractable classes (Theorem 5.33). Furthermore, using relational structures rather than hypergraphs to represent the structure of an instance, we obtain a slightly stronger version of this result (Theorem 5.41).

Finally, we finish this thesis by making concluding remarks in Chapter 6, where we also discuss the potential for future work in this area and list a few open questions.

Chapter 2

Global constraints and their use

In which we formally define global constraints, demonstrate their use on a few example configuration problems, and discuss how they relate to other formal systems.

Historically, global constraints grew out of the need to represent relations between unbounded numbers of variables. Such relations frequently cannot be represented extensionally, i.e. by listing the allowed assignments, in reasonable time [vHK06]. As specific kinds of relations, such as not-all-equal or all-different, occur in a variety of problems, having solvers provide these as special constraints became the norm [BC94]. Currently, the various solvers in use [GJM06, WNS97] provide many global constraints to the constraint programmer, as evidenced by the Global Constraint Catalogue[1]. Global constraints have been cited as a major reason for the practical success of constraint solvers [BHHW07].

Global constraints are thus well-researched [BHHW04, BKN+10, FFH+11, GS11, KEKM08]. In particular, a lot is known about achieving various kinds of consistency on the one hand, and about decomposing global constraints into fixed-arity table constraints on the other. The former allows constraint solvers to provide efficient propagators with known running times for specific global constraints, while the latter, when feasible, allows us to eliminate specific global constraints and use a solver that does not support them. We give an overview of some of these results in Section 3.2.

The goal of this chapter is to formally define global constraints in a way that will allow us to treat the many different types of global constraints (cf. Section 2.2.1) in a uniform manner, which in turn will help us investigate their complexity. To illustrate our definitions, we will use the configuration problems below. Observe that in Problems 2.1 and 2.2, there are constraints involving numbers, and these numbers are part of the input. In Section 2.2.2, we will show that global constraints are particularly well suited for such problems.

Furthermore, in Section 2.3 we will discuss previous approaches to configuration in order to show how they, too, can be encoded using global constraints. To illustrate these approaches, we will use Problem 2.6, as it exhibits conditional

[1] `http://www.emn.fr/z-info/sdemasse/gccat/`

requirements, that is, components that are only required if other components are present.

2.1 Motivating examples: Configuration problems

2.1.1 Connected graph partition

The connected graph partition problem [GJ79, p. 209], formally defined below, is the problem of partitioning the vertices of a graph into disjoint bags of a given size so as to minimize the number of edges that span the bags. The vertices of the graph can, for example, represent components to be placed on circuit boards of a fixed size while minimizing the number of inter-board connections.

Problem 2.1 (Connected graph partition (CGP)) We are given an undirected and connected graph $\langle V, E\rangle$, as well as $\alpha, \beta \in \mathbb{N}$. Can V be partitioned into disjoint sets $V_1, \ldots, V_m$ with $|V_i| \leq \alpha$ such that the set of broken edges $E' = \{\{u, v\} \in E \mid u \in V_i, v \in V_j, i \neq j\}$ has cardinality β or less? □

This problem is NP-complete [GJ79, p. 209], even for fixed $\alpha \geq 3$. However, for fixed β the problem can be solved in polynomial time, by successively guessing sets E', with $|E'| \leq \beta$, of broken edges, and checking whether the connected components of the graph $\langle V, E - E'\rangle$ all have α or fewer vertices. The number of such sets E' is bounded by $\sum_{i=1}^{\beta} \binom{|E|}{i} \leq (|E| + 1)^{\beta}$, which is polynomial if β is fixed. This property of having *few solutions* to a part of the problem is something that we will exploit and turn into a general result in Chapter 4.

2.1.2 PartnerUnits

Problem 2.2 below is a formalized version of the following problem, encountered initially by the configuration group at Siemens Austria [FHS10], and investigated in more detail by Aschinger et al. [ADF+11, ADG+11b] as well as Teppan et al. [TFF12]. In this problem, a company needs to track people coming in and going out of various rooms in a building. The rooms are organised into zones of interest, and have sensors on the doors leading in and out of the zones. As zones may overlap, a sensor can be connected to several zones, and a zone can of course have several sensors. We need to assign zones and sensors to control units that monitor them. The units are identical, and each can take a certain maximum number of sensors and zones. Furthermore, if a zone and a sensor are connected but assigned to different units, then we need to also connect the units. Each unit can in turn only be connected to a limited number of other units. Finally, the units are expensive, so we'd like to minimize the number used.

Problem 2.2 (PartnerUnits) We are given a bipartite undirected graph $G = \langle V_1, V_2, E\rangle$, a value $uc \in \mathbb{N}$ called the unit cap, a value $iuc \in \mathbb{N}$ called the interunit cap, and $k \in \mathbb{N}$. Can $V_1 \cup V_2$ be partitioned into k or fewer disjoint sets S (the units) such that for every S,

1. $|S \cap V_1| \leq uc$ and $|S \cap V_2| \leq uc$, and
2. $|\{S' \mid v \in S, w \in S' \text{ and } (v,w) \in E\}| \leq iuc$? □

In other words, every unit S must contain no more than uc vertices from either side of the graph, and the number of units connected to S by an edge in G must not exceed iuc. While similar to Problem 2.1, here we are not concerned with the number of edges spanning units, but rather the per-unit number of adjacent units. We also have a bound on the number of units, which was absent in the CGP.

Example 2.3 ([ADG+11b]) Consider the instance of PartnerUnits depicted in Figure 2.4. The graph G is $K_{6,6}$, that is, the complete bipartite graph with six vertices on each side, and $uc = iuc = 2$. Figure 2.4 also shows a solution to this instance, with the units depicted as squares. □

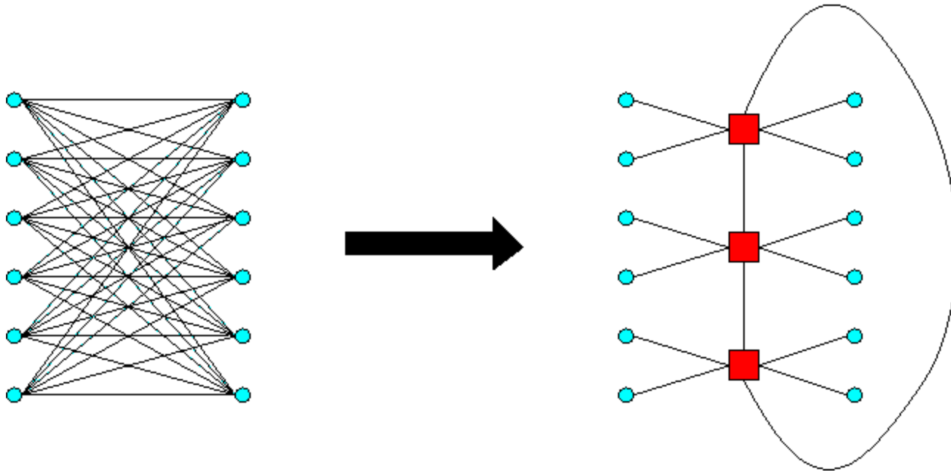

Figure 2.4: A $K_{6,6}$ PartnerUnits instance, with solution.

Problem 2.2 is a relaxed version of the original problem Siemens Austria encountered, where the size uc of the units is fixed and equal to 2 [FHS10].

Theorem 2.5 *The PartnerUnits problem is* NP*-complete.* □

Proof Reduction from bin packing [GJ79, p. 226]. In instances of bin packing, we are given a set of items I. Each item $i \in I$ has a size $s(i) \in \mathbb{N}$. We are also given a bin size b and a cap $k \in \mathbb{N}$. The problem is to partition I into k or fewer

sets B such that $\sum_{i \in B} s(i) \leq b$. Note that this problem is strongly NP-complete — it remains NP-complete if all numbers are encoded in unary.

We create a corresponding instance of the PartnerUnits problem by creating for every item i a biclique of size $s(i)$, that is, a complete bipartite graph with $\lceil \frac{s(i)}{2} \rceil$ vertices on one side and $\lfloor \frac{s(i)}{2} \rfloor$ vertices on the other. The bin size b becomes the unit cap uc, the interunit cap iuc is zero, and k is the number of units we may use. Since the interunit cap is zero, every vertex in a biclique must be assigned to the same unit, and thus the solutions to this instance of PartnerUnits correspond to the solutions for the given instance of bin packing. ■

It is tempting to attempt to use the reduction given above to also obtain NP-completeness for the special case where uc is fixed and equal to some constant. However, as bin packing with a fixed bin size is solvable in polynomial time [GJ79, p. 226], this approach will not work. Likewise, the polynomial-time algorithm for bin packing with fixed bin size does not give us a polynomial algorithm for this case, as bin packing has no notion of connections between bins.

As for tractability, Aschinger et al. [ADG+11b] have recently shown that for every fixed $a \in \mathbb{N}$, the special case of Problem 2.2 where $uc = a$ and $iuc = 2$ is in NL, and hence tractable.

2.1.3 Car part configuration

This problem, while quite simple, exhibits components that are only required if other components are present in other parts of the problem. As we mentioned in Chapter 1, this is a feature of configuration problems, and we will use it to discuss configuration specific systems in Section 2.3.

Problem 2.6 (Car part configuration) A car needs, among other things, an engine, an electrical system, and an exhaust system. The engine can either be a petrol or diesel engine. Both types of engines need pumps and cylinders; the petrol engine also needs spark plugs, while the diesel one requires them to be absent. The spark plugs, if present, must be compatible with the electrical system, while the pumps must match the cylinders.

To make this slightly more formal, assume we have a variable En with domain $\{p, d\}$, a variable Sp with domain $\{s\}$, as well as variables El, Ex, Pu, Cy with constraints $C(El, Sp)$ and $D(Pu, Cy)$. However, note that a solution need not necessarily assign all the variables, nor satisfy both constraints. □

2.2 Global constraints: Definitions and examples

Global constraints are traditionally defined, somewhat vaguely, as constraints without a fixed arity, possibly also with a compact representation of the con-

straint relation. For example, in [vHK06] a global constraint is defined as "a constraint that captures a relation between a non-fixed number of variables". Below, we offer a precise definition similar to the one used by Bessiere et al. [BHHW07], where the authors define global constraints for a domain D over a list of variables σ as being given intensionally by a function $D^{|\sigma|} \rightarrow \{0, 1\}$ computable in polynomial time. Our definition differs from this one in that we separate the general *algorithm* of a global constraint (which we call its *type*) from the specific description. This separation allows us a better way of measuring the size of a global constraint, which in turn helps us to establish new complexity results.

Definition 2.7 (Global constraints) A *global constraint type* is a parametrised polynomial-time algorithm that determines the acceptability of an assignment of a given set of variables.

Each global constraint type, e, has an associated set of *descriptions*, $\Delta(e)$. Each description $\delta \in \Delta(e)$ specifies appropriate parameter values for the algorithm e. In particular, each $\delta \in \Delta(e)$ specifies a set of variables, denoted by $\mathcal{V}(\delta)$.

A *global constraint* $e[\delta]$, where $\delta \in \Delta(e)$, is a function that maps assignments of $\mathcal{V}(\delta)$ to the set $\{0, 1\}$. Each assignment that is allowed by $e[\delta]$ is mapped to 1, and each disallowed assignment is mapped to 0. The *extension* or *constraint relation* of $e[\delta]$ is the set of assignments, θ, of $\mathcal{V}(\delta)$ such that $e[\delta](\theta) = 1$. We also say that such assignments *satisfy* the constraint, while all other assignments *falsify* it. □

When we are only interested in describing the set of assignments that satisfy a constraint, and not in the complexity of determining membership in this set, we will sometimes abuse the notation by writing $\theta \in e[\delta]$ to mean $e[\delta](\theta) = 1$.

As can be seen from the definition above, a global constraint is not usually explicitly represented by listing all the assignments that satisfy it. Instead, it is represented by some description δ and some algorithm e that allows us to check whether the constraint relation of $e[\delta]$ includes a given assignment. To stay within the complexity class NP, this algorithm is required to run in polynomial time. As the algorithms for many common global constraints are built into modern constraint solvers, we measure the *size* of a global constraint's representation by the size of its description.

It is worth noting that this definition allows us to encode instances of NP-complete problems as global constraints. For example, given an instance of the satisfiability problem, SAT [GJ79], we can check in polynomial time whether an assignment is a satisfying one. Therefore, an algorithm that takes a SAT instance and does so satisfies the definition of a global constraint type. The reason for not excluding such problems is twofold. On the one hand, there are global constraints currently used and studied, such as EGC (see below), that encode NP-complete problems. On the other hand, as we shall see in later chapters, excluding NP-complete problems from the definition does not by itself gain us anything.

Definition 2.8 (CSP instance) An instance of the constraint satisfaction problem (CSP) is a pair $\langle V, C\rangle$ where V is a finite set of *variables*, and C is a set of *global constraints* such that for every $e[\delta] \in C$, $\mathcal{V}(\delta) \subseteq V$. In a CSP instance, we call $\mathcal{V}(\delta)$ the *scope* of the constraint $e[\delta]$.

The *size* of a CSP instance $P = \langle V, C\rangle$ is $|P| = |V| + \sum_{v\in V} |D(v)| + \sum_{e[\delta]\in C} |\delta|$.

A *solution* to a CSP instance $\langle V, C\rangle$ is an assignment θ of V which satisfies every global constraint, i.e., for every $e[\delta] \in C$ we have $\theta|_{\mathcal{V}(\delta)} \in e[\delta]$. □

2.2.1 Examples of global constraint types

Below, we give examples of various well-known global constraints using our definition. We also show how they can be used to encode the problems we gave earlier.

Example 2.9 (EGC) A very general global constraint type is the *extended global cardinality* constraint type [QLOvBG04, SS11]. This form of global constraint is defined by specifying for every domain element a a finite set of natural numbers $K(a)$, called the cardinality set of a. The constraint requires that the number of variables which are assigned the value a is in the set $K(a)$, for each possible domain element a.

Using our notation, the description δ of an EGC global constraint specifies a function $K_\delta : D(\mathcal{V}(\delta)) \to \mathcal{P}(\mathbb{N})$ that maps each domain element to a set of natural numbers. The algorithm for the EGC constraint then maps an assignment θ to 1 if and only if, for every domain element $a \in D(\mathcal{V}(\delta))$, we have that $|\{v \in \mathcal{V}(\delta) \mid \theta(v) = a\}| \in K_\delta(a)$.

EGC constraints in particular have been used to model a number of different problems (cf. [SS11] for some examples). While deciding whether an arbitrary EGC constraint has a satisfying assignment is NP-complete [QLOvBG04], quite a bit is known about when single instances of such constraints are tractable. As we shall see in Chapter 3, however, this type of tractability by itself will not prove very valuable.

A global constraint does not always need a description, as the example below demonstrates.

Example 2.10 (Difference constraints) The *Alldifferent* constraint type [vHK06] takes an empty string as its description δ. The algorithm for this constraint then maps an assignment θ to 1 if and only if $|\{\theta(v) \mid v \in \mathcal{V}(\delta)\}| = |\mathcal{V}(\delta)|$, i.e. every variable is assigned a distinct value. □

Example 2.11 (Table constraints) A rather degenerate example of a a global constraint type is the *table* constraint.

In this case the description δ is simply a list of assignments of some fixed set of variables, $\mathcal{V}(\delta)$. The algorithm for a table constraint then decides, for any

assignment of $\mathcal{V}(\delta)$, whether it is included in δ. This can be done in a time which is linear in the size of δ and so meets the polynomial time requirement of Definition 2.7.

We observe that any global constraint can be rewritten as a table constraint. However, this rewriting will, in general, incur an exponential increase in the size of the description. □

While the table constraint type is not interesting as a global constraint, table constraints are precisely those global constraints that have their extension as their description, that is, classic constraints (cf. Chapter 1). Every classic CSP instance can therefore be viewed as a CSP instance where every constraint is a table constraint. In other words, classic CSP instances fit nicely into Definition 2.8.

Example 2.12 (Clauses) We can view the disjunctive clauses used to define propositional satisfiability problems as a global constraint type in the following way.

The description δ of a clause is simply a list of the literals that it contains, and $\mathcal{V}(\delta)$ is the corresponding set of variables. The algorithm for the clause then maps any Boolean assignment θ of $\mathcal{V}(\delta)$ that satisfies the disjunction of the literals in δ to 1, and all other assignments to 0.

Note that a clause forbids precisely one assignment to $\mathcal{V}(\delta)$ (the one that falsifies all of the literals in the clause). Hence the extension of a clause contains $2^{|\mathcal{V}(\delta)|} - 1$ assignments, so the size of the constraint relation grows exponentially with the number of variables, but the size of the constraint description grows only linearly. □

Example 2.12 means that the general satisfiability problem, SAT, which is given as a set of clauses to be satisfied [GJ79], can be viewed as a CSP instance with global constraints quite directly — each clause c becomes a clause global constraint with c as the description.

Example 2.13 (Negative Constraints) Another example of a global constraint type is provided by *negative* constraints. These are complementary to table constraints, in that they are described by listing *forbidden* assignments.

In this case the description δ is again a list of assignments of some fixed set of variables, $\mathcal{V}(\delta)$, but the algorithm for a negative constraint then decides, for any assignment of $\mathcal{V}(\delta)$, whether it is *not* included in δ. This can be done in a time which is linear in the size of δ and so meets the polynomial time requirement of Definition 2.7.

We observe that any global constraint can be rewritten as a negative constraint. However, this rewriting will, in general, incur an exponential increase in the size of the description.

The clauses described in Example 2.12 are a special case of the negative constraint type, where exactly one assignment is forbidden. □

2.2.2 Using global constraints for configuration

Below, we are going to demonstrate the power of global constraints by showing how EGC and table constraints can be used to model the configuration problems from Section 2.1 in a very natural way as CSP instances.

Example 2.14 (The CGP as global constraints) Given a connected graph $G = \langle V, E\rangle$, α, and β, we build a CSP instance $\langle A \cup B, C\rangle$ as follows. The set A will have a variable v for every $v \in V$ with domain $D(v) = \{1, \dots, |V|\}$, while the set B will have a boolean variable e for every edge in E.

The set of constraints C will have an EGC constraint C^α on A with $K(i) = \{0, \dots, \alpha\}$ for every $1 \leq i \leq |V|$. Likewise, C will have an EGC constraint C^β on B with $K(0) = \{0, \dots, |E|\}$ and $K(1) = \{1, \dots, \beta\}$.

Finally, to connect A and B, the set C will have for every edge $\{u, v\} \in E$, with corresponding variable $e \in B$, a table constraint on $\{u, v, e\}$ requiring $u \neq v \rightarrow e = 1$. □

This encoding follows the definition of Problem 2.1 quite closely, and can be done in polynomial time. We will use this encoding to illustrate the main result of Chapter 4 in detail.

Modelling the PartnerUnits problem

Given an instance of Problem 2.2, we can encode it as a CSP by letting the vertices of the graph be variables $V_1 = \{v_1^1, \dots, v_n^1\}$ and $V_2 = \{v_1^2, \dots, v_m^2\}$. For the domains we use the k or fewer sets that the vertices are to be partitioned into. If we denote the sets $S_1, \dots, S_k$, the domain of every variable v_j^i is $D(v_j^i) = \{S_1, \dots, S_k\}$.

We can now post an EGC constraint on V_1 and also an identical one on V_2 to ensure that every set S_i contains (is assigned to) uc or fewer vertices from each side of the graph, i.e. the description will have $K(S_i) = \{0, \dots, uc\}$.

We also need auxiliary boolean variables $a_{1,1}, \dots, a_{k,k}$ to track the connections between sets. For every three variables $v_i^1, v_j^2, a_{x,y}$, we post a table constraint requiring $v_i^1 = s_x \wedge v_j^2 = s_y \rightarrow a_{x,y} = 1$ to ensure that the boolean variables do indeed track connections.

Finally, there are several ways to ensure that the number of connections does not exceed iuc. One way is to post knapsack constraints on $a_{x,1}, \dots, a_{x,k}$ for every $1 \leq x \leq k$, to ensure that their sum is uic or less. These constraints would all have as their description the pair $\langle 0, p\rangle$. Alternatively, we could post EGC constraints on these variables with $K(0) = \{0, \dots, k\}$ and $K(1) = \{0, \dots, iuc\}$. This serves to illustrate the fact that global constraint types overlap in their functionality.

As it happens, the two sets of cardinality constraints mentioned in this encoding both have descriptions where every cardinality set is an interval. It is known [SS11] that such constraints can be individually solved in polynomial

time. However, in our encoding of Problem 2.2 they interact via the table constraints, and so we cannot guarantee that the assignment we find to one extends to an assignment that also satisfies the other. As such, this does not prove that the CSP *as a whole* is tractable. We pick up on this point in Chapter 3.

2.3 Existing formal systems for configuration

Several formal systems [GGM07, MF90, SF96] that extend or adapt the classic CSP framework have been proposed, and solvers based on them have been implemented [Mai98, SFH98, SF99a]. These formal systems highlight several interesting aspects of the configuration task and serve to clarify the structure of practical configuration problems. As we use configuration as an application area for global constraints, we ought to discuss how CSP with global constraints compares to such systems. Below, we shall show that these formal systems can all be reasonably expressed using global constraints. In particular, the translation preserves the structure of the given instance, i.e. the hypergraph formed by the constraint scopes, for two of them. We discuss the relevance of a problem's structure in determining the problem's complexity in Chapter 3.

Definition 2.15 (Hypergraph of a CSP instance) A hypergraph $G = \langle V, H \rangle$ is a set of vertices V together with a set of hyperedges $H \subseteq \mathcal{P}(V)$. The *rank* of G is $\max(\{|h| \mid h \in H\})$.

Given a CSP instance $P = \langle V, C \rangle$, the hypergraph of P, denoted $\mathsf{hyp}(P)$, has vertex set V together with a hyperedge $\mathcal{V}(\delta)$ for every $e[\delta] \in C$. □

In this section we describe *conditional CSPs* [MF90], *composite CSPs* [SF96], and the fragment of first-order logic $\exists\mathsf{FO}_{\rightarrow,\wedge,+}$ [GGM07]. All three frameworks were designed to address the fact that configuration problems feature explicit requirements that are conditional on choices made elsewhere in the problem. To illustrate the frameworks under discussion, we will use Problem 2.6, which features such requirements.

2.3.1 Conditional CSP

This framework was developed by Mittal and Falkenhainer [MF90] as an extension of the CSP framework. It deals with conditional requirements by featuring variables and constraints that need only be considered when certain other variables have been assigned.

Definition 2.16 (Conditional CSP syntax) A conditional CSP instance is a tuple $\Delta = \langle V, V_I, C_C, C_A \rangle$, where V is a set of variables, $V_I \subseteq V$ a set of *initial variables*, C_C a set of constraints over V, called *compatibility constraints*, and C_A a set of *activation constraints*. Activation constraints have two forms, $A \overset{RV}{\Longrightarrow} v$ and $A \overset{RN}{\Longrightarrow} v$, with A a compatibility constraint and v a variable.

The hypergraph of Δ has vertex set V and a hyperedge for every constraint in $C_C \cup C_A$. □

Definition 2.17 (Conditional CSP semantics) Let $\Delta = \langle V, V_I, C_C, C_A \rangle$ be a conditional CSP instance, and θ an assignment to a subset of V. The assignment θ satisfies Δ iff

1. $V_I \subseteq \mathcal{V}(\theta)$,
2. for every constraint $C \in C_C$ with $\mathcal{V}(C) \subseteq \mathcal{V}(\theta)$, we have that $\theta \models C$,
3. for every constraint $(A \overset{RV}{\Longrightarrow} v) \in C_A$ with $\mathcal{V}(A) \subseteq \mathcal{V}(\theta)$ and $\theta \models A$, we have that $v \in \mathcal{V}(\theta)$, and
4. for every constraint $(A \overset{RN}{\Longrightarrow} v) \in C_A$ with $\mathcal{V}(A) \subseteq \mathcal{V}(\theta)$ and $\theta \models A$, we have that $v \notin \mathcal{V}(\theta)$. □

In [MF90], this formalism is called dynamic CSP, but it has since been renamed to conditional CSP [SF99b] due to a name collision.

Example 2.18 We can represent the problem in Problem 2.6 as a conditional CSP instance $\langle V, V_I, C_C, C_A \rangle$ by letting

- $V = \{En, El, Ex, Pu, Cy, Sp\}$,
- $V_I = \{En, El, Ex\}$,
- $C_C = \{C(El, Sp), D(Pu, Cy)\}$, and
- $C_A = \left\{ \begin{array}{l} (En = p) \overset{RV}{\Longrightarrow} Sp, (En = p) \overset{RV}{\Longrightarrow} Pu, (En = p) \overset{RV}{\Longrightarrow} Cy, \\ (En = d) \overset{RN}{\Longrightarrow} Sp, (En = d) \overset{RV}{\Longrightarrow} Pu, (En = d) \overset{RV}{\Longrightarrow} Cy \end{array} \right\}.$ □

Below, we show a simple reduction from conditional CSP instances to CSP instances with global constraints. The reduction simulates the activation of variables by allowing inactive variables to be assigned a special value. Using this idea, the original constraints of the conditional CSP instance need only be satisfied if none of their variables are assigned this special value.

Theorem 2.19 *A conditional CSP instance can be reduced to a CSP instance with the same hypergraph in polynomial time.* □

Proof Let $\langle V, V_I, C_C, C_A \rangle$ be a conditional CSP. For every variable $v \in V - V_I$, add a special domain value $\perp$ to $D(v)$. Then, create for every $C \in C_C$ a constraint

$$\left(\bigwedge_{v \in \mathcal{V}(C)} v \neq \perp \right) \to C$$

specifying that the condition of the constraint C needs to be considered only if the variables in it are not assigned $\perp$.

For any activation constraint $A \overset{RV}{\Longrightarrow} v$, create the constraint $A \rightarrow v \neq \bot$, and for $A \overset{RN}{\Longrightarrow} v$ create the constraint $A \rightarrow v = \bot$. This reduction takes polynomial time, and the hypergraph formed by the constraint scopes of either instance is the same.

Let $\langle V', C' \rangle$ be the CSP instance constructed, and θ a solution for $\langle V', C' \rangle$. Letting $W = \{v' \in V' \mid \theta(v') = \bot\}$, the assignment $\theta|_{V'-W}$ is a solution for the original conditional CSP instance. In the other direction, a solution θ to the conditional CSP instance becomes a solution to $\langle V', C' \rangle$ by letting $\theta(v') = \bot$ for every $v' \in V' - \mathcal{V}(\theta)$. ■

In the above reduction, given a constraint C, we generate a constraint that, in addition to allowing the assignments that satisfy C, allows all assignments that assign at least one variable to $\bot$. Representing this as a classic constraint would result in an exponential blowup, since there are $2^{|\mathcal{V}(C)|} - 1$ assignments that assign at least one variable to $\bot$: one for every subset of $\mathcal{V}(C)$ except $\emptyset$. Using global constraints, this consideration disappears.

2.3.2 Composite CSP

This framework was introduced informally by Sabin and Freuder [SF96]. A composite CSP instance is a CSP instance where variables can have subproblems (other composite CSP instances) as values. If such a variable v is assigned a subproblem T as the value, any constraint containing v is removed and the constraints and variables in T are added to the composite CSP instance. This mechanism allows conditional requirements with a great deal of flexibility. However, to explore the properties of this framework we need to precisely define it. Below is our formalization of the ideas found in [SF96].

Definition 2.20 (Composite CSP syntax) A composite CSP instance is a strict partially ordered set $\langle S, < \rangle$ with $S = \{T_1, \ldots, T_n\}$ a set of CSP instances and $<$ a reachability relation with a single minimal element T_r. If $T_i < T_j$, then any variable $v \in \mathcal{V}(T_i)$ may contain T_j as a domain element.

The hypergraph of a composite CSP instance $\langle S, < \rangle$ is $\langle V, H \rangle$, where V and H are the unions of the vertex and hyperedge sets of $\mathsf{hyp}(T)$ for each $T \in S$. □

Definition 2.21 (Variables and constraints) Let $\langle S, < \rangle$ be a composite CSP instance. The set of variables in S is $\mathcal{V}(S) = \bigcup\{\mathcal{V}(T) \mid T \in S\}$. Likewise, the set of constraints in S is $\mathcal{C}(S) = \bigcup\{C \mid \langle V, C \rangle \in S\}$. □

Definition 2.22 (Subproblem variables) Let $\langle S, < \rangle$ be a composite CSP instance and θ an assignment to a subset of $\mathcal{V}(S)$. The set of *subproblem variables* in θ, denoted $\mathcal{SV}(\theta)$, is the set of variables $v \in \mathcal{V}(\theta)$ such that $\theta(v) = T$ for some $T \in S$. □

In other words, $\mathcal{SV}(\theta)$ is the set of variables assigned subproblems as values, and we have to treat them in a special way.

Definition 2.23 (Composite CSP semantics) Let $\langle S, < \rangle$ be a composite CSP instance with minimal element T_r, and θ an assignment to a subset of $\mathcal{V}(S)$. For every $T \in S$ we define $\theta \models T$ recursively as follows:

1. $\mathcal{V}(T) \subseteq \mathcal{V}(\theta)$,
2. for every constraint C in T with $\mathcal{V}(C) \cap \mathcal{SV}(\theta) = \emptyset$, we have that $\theta \models C$, and
3. for every $v \in \mathcal{V}(T)$ such that $\theta(v) = U$ for some $U \in S$, we have that $\theta \models U$.

The assignment θ satisfies $\langle S, < \rangle$ iff $\theta \models T_r$. □

In other words, we have to deal with any subproblem that we have assigned to a variable in our initial problem (recursively), but we do not need to check constraints containing such variables, as they are replaced by constraints from the subproblem.

Example 2.24 We can represent the problem in Problem 2.6 as a composite CSP $\langle S, < \rangle$ as follows: $S = \{S_1 = \langle V_1, C_1 \rangle, S_2 = \langle V_2, C_2 \rangle, S_3 = \langle V_3, C_3 \rangle\}$, where

- $V_1 = \{En, El, Ex\}$, $C_1 = \emptyset$, and the domain of En is $\{S_2, S_3\}$,
- $V_2 = \{El, Cy, Pu, Sp\}$ and $C_2 = \{C(El, Sp), D(Pu, Cy)\}$, and
- $V_3 = \{Cy, Pu\}$ and $C_3 = \{D(Pu, Cy)\}$.

The minimal element is S_1, with $S_1 < S_2$ and $S_1 < S_3$. □

Below, we show that composite CSP instances can be reduced to CSP instances. Here, the structure does not stay the same. While there are several ways to do this reduction, leading to different kinds of hypergraphs, below we show a reduction that uses the relationships between subproblems. As before, we use a special value to keep track of which variables are or are not required. However, here we also need to check for subproblems.

Theorem 2.25 *A composite CSP instance can be reduced to a CSP instance in polynomial time.* □

Proof Given a composite CSP instance $\langle S, < \rangle$ with minimal element T_r, we add a special value $\perp$ to the domain of every variable in $\mathcal{V}(S) - \mathcal{V}(T_r)$. The CSP instance we construct will have $\mathcal{V}(S)$ as its set of variables. For each constraint C of every $T \in S$, we add a constraint that checks if C is satisfied only if no variable is assigned $\perp$ or any subproblem in S. We also add for every $T \in S$ and variable $v \in \mathcal{V}(S)$ with $T \in D(v)$ a constraint on $\{v\} \cup \mathcal{V}(T)$ requiring that the variables in $\mathcal{V}(T)$ are not assigned $\perp$ if v is assigned T.

Given a solution θ to the CSP instance we constructed, let $W = \{v \in \mathcal{V}(\theta) \mid \theta(v) = \perp\}$. It is easy to verify that assignment $\theta|_{\mathcal{V}(S)-W}$ is a solution to $\langle S, < \rangle$.

Likewise, given a solution to $\langle S, < \rangle$ we obtain a solution to the CSP instance constructed by letting $\theta(v) = \bot$ for every $v \in \mathcal{V}(S) - \mathcal{V}(\theta)$. ■

2.3.3 The logic $\exists\mathsf{FO}_{\rightarrow,\wedge,+}$

This fragment of first-order logic defined by Gottlob et al. [GGM07] gives a logical characterization of configuration problems by extending a common logical characterization of classic CSPs as conjunctions of database predicates [GLS01, GJC94]. The logic $\exists\mathsf{FO}_{\rightarrow,\wedge,+}$ adds implication, which allows us to model conditional requirements. The close tie-in with database theory allows very precise discussions of expressive power. In particular, Gottlob et al. [GGM07] show that given a fixed set of database predicates, that is, sets of tuples of values (that is, classic constraints, cf. Chapter 1), any $\exists\mathsf{FO}_{\rightarrow,\wedge,+}$ sentence can be encoded as a conditional CSP using the same set of constraints, and vice versa.

Definition 2.26 ($\exists\mathsf{FO}_{\rightarrow,\wedge,+}$ syntax) An $\exists\mathsf{FO}_{\rightarrow,\wedge,+}$ sentence is a formula of the form

$$\exists\vec{x}. \bigwedge_{1\leq i\leq n} \left(\phi_i(\vec{x}) \rightarrow \exists\vec{y}.\psi_i(\vec{x},\vec{y})\right)$$

where $\vec{x}$, $\vec{y}$ are lists of variables, $n \geq 0$, and ϕ_i, ψ_i are conjunctions of atoms, possibly empty.

The hypergraph of an $\exists\mathsf{FO}_{\rightarrow,\wedge,+}$ sentence ϕ has the variables of ϕ as vertices. Furthermore, the hypergraph has for every conjunct $\phi_i(\vec{x}) \rightarrow \exists\vec{y}.\psi_i(\vec{x},\vec{y})$ a hyperedge containing the variables of this conjunct, that is, $\vec{x}$ and $\vec{y}$. □

The semantics of $\exists\mathsf{FO}_{\rightarrow,\wedge,+}$assume a constraint database that specifies for every atom the tuples allowed by it. We may therefore view $\exists\mathsf{FO}_{\rightarrow,\wedge,+}$as an extension of classic CSPs if we prefer simply by ignoring the quantifiers. The semantics for problems then follow those of first-order logic except when dealing with partial assignments.

Definition 2.27 ($\exists\mathsf{FO}_{\rightarrow,\wedge,+}$ semantics) Let ϕ be an $\exists\mathsf{FO}_{\rightarrow,\wedge,+}$ sentence, and θ an assignment to a subset of $\mathcal{V}(\phi)$. Denote by ϕ' the formula obtained from ϕ by replacing any atom R such that $\mathcal{V}(R) \not\subseteq \mathcal{V}(\theta)$ by $\bot$ (meaning false). The formula ϕ is satisfied by θ iff $\theta \models \phi'$. □

Example 2.28 We can represent the problem in Problem 2.6 as the following sentence of $\exists\mathsf{FO}_{\rightarrow,\wedge,+}$ (outermost conjunction omitted):

$$\exists En, El, Ex. \left(\begin{array}{c} T(El) \wedge T(Ex) \wedge T(En) \\ (En = d) \rightarrow \exists Pu, Cy. D(Pu, Cy) \\ (En = d) \wedge (Sp = s) \rightarrow \bot \\ (En = p) \rightarrow \exists Pu, Cy, Sp. \left(C(El, Sp) \wedge D(Pu, Cy)\right) \end{array} \right)$$

The predicate T is a new predicate that allows all values in the domain of a variable. Since there are no constraints on some of the required variables, we need it to make sure that they are present in any satisfying assignment. □

Next, we show that $\exists\mathsf{FO}_{\rightarrow,\wedge,+}$ sentences can be reduced to CSP instances with the same structure. The reduction follows the semantics of $\exists\mathsf{FO}_{\rightarrow,\wedge,+}$ quite directly, and once again uses a special value to keep track of variables that can be ignored.

Theorem 2.29 *An $\exists\mathsf{FO}_{\rightarrow,\wedge,+}$ sentence can be reduced to a CSP instance with the same hypergraph in polynomial time.* □

Proof Given an $\exists\mathsf{FO}_{\rightarrow,\wedge,+}$ sentence $\exists\vec{x}. \bigwedge_{1\leq i\leq n} \left(\phi_i(\vec{x}) \rightarrow \exists\vec{y}.\psi_i(\vec{x},\vec{y})\right)$, we first add the special value $\perp$ to the domain of every variable. Then, we create for every formula $\phi_i(\vec{x}) \rightarrow \exists\vec{y}.\psi_i(\vec{x},\vec{y})$ a constraint that, given an assignment to the variables in this formula, first replaces every atom with at least one variable assigned $\perp$ by false, and then checks if the resulting formula is true. It is clear that this transformation preserves the solutions of the $\exists\mathsf{FO}_{\rightarrow,\wedge,+}$ sentence given, as well as being feasible in polynomial time. Furthermore, the hypergraph of the CSP instance constructed is the same as that of the $\exists\mathsf{FO}_{\rightarrow,\wedge,+}$ sentence. ■

2.4 Summary

In this chapter, we have formally defined global constraints. Our definition of global constraints separates them into an algorithm or type, such as EGC, which is usually built into a solver, and a description, which specifies the allowed assignments. As we have seen from the examples, global constraints allow us to model a wide range of well-known problems, such as SAT, in a natural way. Furthermore, we have seen how global constraints can be used to model a few configuration problems, a type of problem that is difficult to model with classic constraints.

We've also discussed several existing formal systems for configuration, and shown how they admit natural encodings using global constraints. For two of the systems discussed, the encoding preserves the hypergraph of the original problem. As we shall see in Chapter 3, the hypergraph of a problem plays a major role in determining its complexity.

As we are concerned with the complexity of CSP instances with global constraints, we next undertake a review of research into the complexity of CSP instances, in particular at what is known about their tractable classes. As we shall show, while a lot is known about the complexity of classic CSP instances, many of those results do not apply to instances with global constraints.

Chapter 3

Tractable classes of CSP instances: An overview

In which we give an overview of current research into tractable classes of CSP instances, and observe that of the results discussed, a significant number apply only to classic CSP instances.

This chapter is divided into three parts. In Section 3.1, we look at research into classic CSP instances, and discuss which of these results also apply to instances with global constraints. In Section 3.2, on the other hand, we look at research aimed specifically at CSP instances with global constraints. Finally, in Section 3.3 we conclude this chapter by a discussion of how the results presented lead us to consider the research questions that we investigate in Chapters 4 and 5.

3.1 Tractability of classic CSP instances

Research into tractable classes of classic CSP instances is usually divided up into two families, tractability due to *structural restrictions* [ADG+11a, BFMY83, DKV02, GLS02, GJC94, RS86], and tractability due to restrictions on the *constraint language* [BD06, CJ06, PJ97, Sch78]. Structural restrictions have to a large extend been inspired by database theory [ADG+11a], while language restrictions commonly draw on the field of universal algebra [CJ06]. There are also *hybrid* tractable classes that use restrictions of both kinds [CJS10, SJ08]. While we did briefly discuss the structure of a CSP instance in Section 2.3, to aid the reader we repeat the definition here.

Definition 3.1 (Hypergraph of a CSP instance) A hypergraph $G = \langle V, H \rangle$ is a set of vertices V together with a set of hyperedges $H \subseteq \mathcal{P}(V)$. The *rank* of G is $\max(\{|h| \mid h \in H\})$.

Given a CSP instance $P = \langle V, C \rangle$, the hypergraph of P, denoted $\mathsf{hyp}(P)$, has vertex set V together with a hyperedge $\mathcal{V}(\delta)$ for every $e[\delta] \in C$. □

To avoid separate definitions for graphs and hypergraphs, we will adopt the common approach [Ber84] of considering undirected graphs to be hypergraphs

of rank 2, that is, those having hyperedges with at most two elements.

Definition 3.2 (Constraint relation) Given a constraint $e[\delta]$ and a linear ordering of $\mathcal{V}(\delta)$ into a tuple $\langle v_1, \ldots, v_n \rangle$, the *constraint relation* given by $e[\delta]$ is $\{\langle \theta(v_1), \ldots, \theta(v_n) \rangle \mid \theta \in e[\delta]\}$. □

Definition 3.3 (Constraint language) A *constraint language* L is a set of relations. A CSP instance $\langle V, C \rangle$ is said to be *over* the language L if for every $e[\delta] \in C$ there is some linear ordering of $\mathcal{V}(\delta)$ such that the constraint relation given by $e[\delta]$ is in L. □

A structural restriction is a restriction on the hypergraphs of CSP instances. Given a class of hypergraphs $\mathcal{H}$, we write CSP($\mathcal{H}$) for the class containing every CSP instance whose hypergraph is in $\mathcal{H}$. Likewise, a restriction on language is a restriction on the relations given by the constraints in a CSP instance. Given a language L, we write CSP(L) for the class containing every CSP instance over L. Finally, a class of CSP instances defined by restrictions on both structure and language is a *hybrid class*.

In other words, then, structural restrictions deal with how the constraints of a CSP instance interact, while language restrictions deal with the relations of the constraints. Below, we present examples of all three kinds of restrictions, and discuss how they can be used to obtain tractable classes of CSP instances. Since our results will use structural restrictions and properties in far greater detail when compared to language and hybrid restrictions, however, the focus will be on structure.

All of these definitions can be applied to CSP instances with global constraints just as well as for classic CSP instances. Therefore, whenever we will state a result that is only known to be true for a class of classic CSP instances, we will explicitly say so.

3.1.1 Restrictions on structure

Treewidth

The notion of treewidth is, perhaps, the central notion when it comes to tractability due to structural restrictions [CJG08,GPW06,Gro07]. Tractable classes based on treewidth are known for many NP-complete problems [ADG$^+$11a], and so we start here.

Notation 3.4 (Tree) A tree $T = (V, E)$ is an acyclic graph with a single vertex denoted r called the root. For any vertex $v \in V$, we write T_v for the subtree of T rooted at v. □

Definition 3.5 (Tree decomposition) Given a hypergraph $G = \langle V, H \rangle$, a *tree decomposition* of G is a pair $\langle T, \lambda \rangle$ where T is a tree and λ is a labelling function from nodes of T to sets of vertices, such that

1. for every $v \in V$, there exists a node t of T such that $v \in \lambda(t)$,
2. for every hyperedge $h \in H$, there exists a node t of T such that $h \subseteq \lambda(t)$, and
3. for every $v \in V$, the set of nodes $\{t \mid v \in \lambda(t)\}$ induces a connected subtree of T.

The width of a tree decomposition is $\max(\{|\lambda(t)| - 1 \mid t \text{ node of } T\})$. □

In other words, a tree decomposition groups the vertices of a hypergraph into sets to make the hypergraph a tree while preserving the hypergraph's connectivity. Originally defined on graphs [RS86], a tree decomposition of a hypergraph G has sometimes been defined on the primal graph of G [GLS02, ADG+11a], where every hyperedge is represented by a clique. The definition given above is equivalent [GM06] and, perhaps, more natural.

Quite a long time ago, it was observed that many NP-complete problems on arbitrary graphs could be efficiently solved given a tree decomposition [AP89, Bod88]. The general idea is to walk the tree decomposition bottom-up. At each node, we exhaustively search through the possible combinations (cartesian product) of values and propagate the results to the parent node. The fact that for every vertex v the nodes containing v form a connected subtree ensures that no information is lost [AP89].

The idea of the above algorithm can be applied to CSP instances with no major modifications. The key property is that the variables of every constraint are entirely contained in some node of a tree decomposition.

Theorem 3.6 ([DKV02]) *Given a CSP instance $P = \langle V, C \rangle$, as well as a tree decomposition of* $\mathrm{hyp}(P)$ *with width k, deciding whether P has a satisfying assignment can be done in time $O(|V| \times \max(\{|D(v)| \mid v \in V\})^{2k})$.* □

As the running time in Theorem 3.6 is exponential in the width of the decomposition, we would like to use a decomposition with low width. The treewidth $\mathrm{tw}(G)$ of a hypergraph G is the minimum width over all tree decompositions of G. Intuitively, then, treewidth measures how close a hypergraph is to a tree in terms of connectivity, and how hard a CSP instance with this hypergraph is to solve relative to those with other hypergraphs.

Much is known about the properties of tree decompositions [Die10]. In particular, acyclic graphs (i.e. trees) have treewidth 1, complete graphs on n vertices have treewidth $n - 1$, and hypergraphs of rank n have treewidth at least $n - 1$. Unfortunately there is also some bad news, originally proven by Arnborg et al. using a different terminology.

Theorem 3.7 ([ACP87]) *Given a hypergraph G and $k \in \mathbb{N}$, the problem of deciding whether* $\mathrm{tw}(G) \leq k$ *is* NP*-complete.* □

However, as with many other problems with a parameter, this problem becomes tractable if k is *fixed* rather than given [ACP87]. In other words, if we

have a bound on the width of decompositions beyond which we simply do not bother, a suitably small decomposition can be found in polynomial time if it exists.

By Theorem 3.6, given a class of hypergraphs $\mathcal{H}$, if there is a constant k such that $\mathsf{tw}(G) \leq k$ for every $G \in \mathcal{H}$, then CSP($\mathcal{H}$) is a tractable class. However, for hypergraphs of bounded rank, Grohe [Gro07] has shown that this can be strengthened to a dichotomy. To state this result, the following notation will be helpful. Given a class of hypergraphs $\mathcal{H}$, let $\mathsf{tw}(\mathcal{H})$ be the maximum treewidth over the hypergraphs in $\mathcal{H}$. If $\mathsf{tw}(\mathcal{H})$ is unbounded, we write $\mathsf{tw}(\mathcal{H}) = \infty$; otherwise $\mathsf{tw}(\mathcal{H}) < \infty$.

Theorem 3.8 ([Gro07]) *Let $\mathcal{H}$ be a class of hypergraphs with bounded rank. Assuming $\mathsf{FPT} \neq \mathsf{W}[1]$, CSP($\mathcal{H}$) is tractable if and only if $\mathsf{tw}(\mathcal{H}) < \infty$.* □

In other words, for structural restrictions on classes of CSP instances where the scope of every constraint has bounded size, bounded treewidth is the only tractable case. In Section 2.2, we noted that in this case we can convert each instance to a classic CSP instance, so the two coincide.

Incidence width

Using a notion such as treewidth to deal with problems, like CSPs, whose structure is more naturally described by a hypergraph can be misleading. In the trivial case, consider a CSP instance with a few large, disjoint constraints. The time required to solve this CSP instance is certainly not exponential in the size of the constraints, as e.g. a tree decomposition algorithm would tell us. Furthermore, for global constraints we are interested in classes of instances with constraints of unbounded arity, in which case the associated class of hypergraphs will have unbounded rank and hence unbounded treewidth.

One way to remedy these issues is to consider the treewidth of the incidence graph of a hypergraph.

Definition 3.9 (Incidence width) The *incidence graph* $\mathsf{inc}(G)$ of a hypergraph $G = \langle V, H \rangle$ is the bipartite graph with set of vertices $V \cup H$ that contains for every $v \in V$ and $h \in H$ with $v \in h$ the edge $\{v, h\}$.

The *incidence width* $\mathsf{iw}(G)$ of a hypergraph G is the treewidth of the incidence graph of G. For a class of hypergraphs $\mathcal{H}$, we write $\mathsf{iw}(\mathcal{H})$ for the maximum incidence width over the hypergraphs in $\mathcal{H}$. If $\mathsf{iw}(\mathcal{H})$ is unbounded we write $\mathsf{iw}(\mathcal{H}) = \infty$; otherwise $\mathsf{iw}(\mathcal{H}) < \infty$. □

Unlike treewidth, the incidence width of a hypergraph can be much smaller than its rank. As a simple example, the incidence with of a hypergraph with a single hyperedge is one. More generally, the following holds.

Theorem 3.10 ([KV00]) *For every hypergraph G, $\mathsf{iw}(G) \leq \mathsf{tw}(G) + 1$.* □

Given a classic CSP instance P, we can in polynomial time obtain a classic CSP instance P' with $\mathsf{inc}(\mathsf{hyp}(P)) = \mathsf{hyp}(P')$ [RPD90], which means that we can use Theorem 3.6 to solve P given a tree decomposition of $\mathsf{inc}(\mathsf{hyp}(P))$. Unfortunately, as we show in Section 3.2.1 (Theorem 3.45), this is not true for CSP in general. However, incidence width remains useful for instances with certain types of global constraints; in particular, Szeider [Sze03] has shown that it remains useful for SAT instances.

Theorem 3.11 ([Sze03]) *Let $\mathcal{H}$ be a class of hypergraphs, and let* SAT$(\mathcal{H})$ *be a class of CSP instances that only contain clause constraints (cf. Example 2.12) and whose hypergraphs are in $\mathcal{H}$. If* $\mathsf{iw}(\mathcal{H}) < \infty$*, then the class* SAT$(\mathcal{H})$ *is tractable.* □

The above result has since been improved upon by Chen and Grohe [CG10], who have shown that Theorem 3.11 holds for the more general class of CSP instances with *generalized disjunctive normal form* (GDNF) constraints.

Query and hypertree width

While incidence width is already a better notion for CSP instances with constraints of unbounded arity, it is still a concept that works on a graph representation of a hypergraph. To go beyond this, notions of width that deal directly with the hypergraph have been defined. Perhaps the most famous of these is the notion of a hypertree decomposition [GLS02], which improves upon the earlier notion of a query decomposition [CR00].

Definition 3.12 (Query decomposition) A *query decomposition* of a hypergraph $G = \langle V, H\rangle$ is a pair $\langle T, \lambda\rangle$, where T is a tree and λ is a labelling function from nodes of T to subsets of H, such that

1. for every $h \in H$, there exists a node $t \in T$ such that $h \in \lambda(t)$,
2. for every $h \in H$, the set of nodes $\{t \mid h \in \lambda(t)\}$ induces a connected subtree of T.
3. for every $v \in V$, the set of nodes $\{t \mid v$ occurs in some $h \in \lambda(t)\}$ induces a connected subtree of T.

The width of a query decomposition is $\max(\{|\lambda(t)| \mid t \in T\})$. □

The original definition in [CR00] allows decomposition nodes to contain variables as well as hyperedges, which serves no purpose in the CSP framework. As such, we give the definition of *pure* query decompositions [GLS01] instead, as has become customary.

Query decompositions preserve connectivity of vertices, and can be used to efficiently solve classic CSP instances using an algorithm very similar to the one used with tree decompositions. However, they cannot be used for CSP instances in general. To see why, observe that because the satisfying assignments of each constraint are listed explicitly, taking the join of a set C of k table constraints can

be done in time $\max(\{|\delta| \mid e[\delta] \in C\})^k$. For global constraints this is not true in general, and so we obtain a theorem that is essentially identical to Theorem 3.6, but which is only true for classic CSP instances.

Theorem 3.13 ([CR00]) *Given a classic CSP instance $P = \langle V, C\rangle$, as well as a query decomposition of* $\mathsf{hyp}(P)$ *having width k, deciding whether P has a solution can be done in time* $O(|V| \times \max(\{|\delta| \mid e[\delta] \in C\})^{2k})$. □

As with treewidth, the *query width* $\mathsf{qw}(G)$ of a hypergraph G is the minimum width over its query decompositions. To be of any use, we would like the query width of a hypergraph to be less than the treewidth. Here we obtain some good news.

Theorem 3.14 ([CR00]) *For every hypergraph G,* $\mathsf{qw}(G) \leq \mathsf{iw}(G) + 1$. □

In other words, we are always better off using query width to solve classic CSP instances. To do so, however, we need to be able to efficiently find query decompositions with low width. Unfortunately, here we hit some very bad news.

Theorem 3.15 ([GLS02]) *Given a hypergraph G, the problem of deciding whether* $\mathsf{qw}(G) \leq 4$ *is* NP*-complete.* □

One way to interpret the above result is that the definition of a query decomposition is too strict. In [GLS02], Gottlob et. al note that query decompositions indirectly require that certain vertices of the hypergraph be exactly covered by the hyperedges at every node, and introduce the notion of a hypertree decomposition to avoid this problem. A hypertree decomposition is a decomposition of a hypergraph into a hypertree, which in turn is a tree structure with two labelling functions. Having two labelling functions will allow us to define a weaker notion of connectivity to preserve.

Definition 3.16 (Hypertree) Let G be a hypergraph. A *hypertree* for G is a triple $\langle T, \chi, \lambda\rangle$ where T is a tree and χ, λ are labelling functions from nodes of T to sets of vertices and sets of hyperedges of G respectively.

For every node t of T, we define $\mathcal{V}(\lambda(t)) = \bigcup \lambda(t)$, and extend χ to subtrees. Thus,

$$\chi(T_t) = \bigcup \{\chi(p) \mid p \text{ node of } T_t\},$$

where T_t is the subtree of T rooted at t. □

Definition 3.17 (Hypertree decomposition) Given a hypergraph $G = \langle V, H\rangle$, a *hypertree decomposition* of G is a hypertree $\langle T, \chi, \lambda\rangle$ such that

1. for every $h \in H$, there exists a node $t \in T$ such that $h \subseteq \chi(t)$,
2. for every $v \in V$, the set of nodes $\{t \mid v \in \chi(t)\}$ induces a connected subtree of T,
3. for every $t \in T$, $\chi(t) \subseteq \mathcal{V}(\lambda(p))$, and

4. for every $t \in T$, $\mathcal{V}(\lambda(t)) \cap \chi(T_t) \subseteq \chi(t)$.

The width of a hypertree decomposition is $\max(\{|\lambda(t)| \mid t \text{ node of } T\})$. □

If we were to restate Definition 3.12 of a query decomposition as a hypertree satisfying certain conditions, we would at every tree node t have $\chi(t) = \mathcal{V}(\lambda(t))$. In [GLS02], Gottlob et al. point this out as the requirement of an exact covering that causes NP-completeness in Theorem 3.15. Having the two functions separately allows us to relax this requirement.

Given a hypertree decomposition of the hypergraph of a classic CSP instance, we can still solve the instance in time polynomial in the width of the decomposition, even though the connectivity requirement is weaker.

Theorem 3.18 ([GLS02]) *Given a classic CSP instance $P = \langle V, C \rangle$, as well as a hypertree decomposition of* $\mathsf{hyp}(P)$ *having width k, deciding whether P has a solution can be done in time $O(|V| \times \max(|\delta| \mid e[\delta] \in C\})^{2k})$.* □

As for other notions of width, the hypertree width $\mathsf{hw}(G)$ of a hypergraph G is the minimum width over its hypertree decompositions. Unlike query width, however, the problem of deciding whether $\mathsf{hw}(G) \leq k$ for fixed k is tractable. Finally, as an extra bonus we have the following result.

Theorem 3.19 ([GLS02]) *For every hypergraph G,* $\mathsf{hw}(G) \leq \mathsf{qw}(G)$. □

As we already know that query width is always smaller than treewidth of primal graphs, it follows that hypertree width is an even better measure of how hard a classic CSP instance with a specific hypergraph is to solve relative to instances on other hypergraphs.

As it turns out, for theoretical analysis the last requirement in Definition 3.17 can be dropped to obtain what has become known as a *generalized* hypertree decomposition. While deciding whether the generalized hypertree width of a given hypergraph is less than or equal to k for fixed k is no longer tractable, the two width measures are closely related. Denoting the generalized hypertree width of a hypergraph G by $\mathsf{ghw}(G)$, the following holds.

Theorem 3.20 ([AGG07]) *For every hypergraph G,* $\mathsf{ghw}(G) \leq \mathsf{hw}(G) \leq 3 \cdot \mathsf{ghw}(G) + 1$. □

In other words, if one is bounded for a class of hypergraphs, then so is the other. Since we are concerned with characterizing tractable classes, the two are (for us) equivalent. It also turns out that generalized hypertree width, as well as the structural notions that we will present below, admit a restatement using the framework of width functions, introduced by Adler [Adl06]. Below, we will therefore restrict ourselves to generalized hypertree width, and drop the word generalized.

Definition 3.21 (Width function) Let $G = \langle V, H\rangle$ be a hypergraph. A *width function* on G is a function $f : \mathcal{P}(\mathcal{V}(G)) \to \mathbb{R}^+$ that assigns a positive real number to every nonempty subset of vertices of G. A width function f is monotone if $f(X) \leq f(Y)$ whenever $X \subseteq Y$.

Let $\langle T, \lambda\rangle$ be a tree decomposition of G, and f a width function on G. The *f-width* of $\langle T, \lambda\rangle$ is $\max(\{f(\lambda(t)) \mid t \text{ node of } T\})$. The *$f$-width* of G is the minimal f-width over all its tree decompositions. □

In other words, a width function on a hypergraph G tells us how to assign weights to nodes of tree decompositions of G. It is easy top verify that the treewidth of a hypergraph G is the f-width of G where f is the width function $f(X) = |X| - 1$. We can also express hypertree width using this framework. Below, we show how to do so in considerable detail, since we will use width functions a lot in subsequent parts of this thesis.

Let $G = \langle V, H\rangle$ be a hypergraph, and $X \subseteq V$. An edge cover for X is any set of hyperedges $H' \subseteq H$ that satisfies $X \subseteq \bigcup H'$. The edge cover number $\rho(X)$ of X is the size of the smallest edge cover for X. It is clear that ρ is a width function. Below, we prove that ρ captures hypertree width.

Theorem 3.22 ([Adl06, Chapter 2]) *For every hypergraph G, the ρ-width of G is the hypertree width of G.* □

Proof Given G, consider an arbitrary tree decomposition $\langle T, \psi\rangle$ of G, and let $\lambda(t)$ be the smallest edge cover of $\psi(t)$ in G for each node t of T. We claim that $\langle T, \psi, \lambda\rangle$ is a hypertree decomposition of G, since the first two conditions of Definition 3.17 follow from the fact that $\langle T, \psi\rangle$ is a tree decomposition, while $\psi(t) \subseteq \mathcal{V}(\lambda(t))$ for every node t follows from the definition of an edge cover. Furthermore, the width of $\langle T, \psi, \lambda\rangle$ is the ρ-width of $\langle T, \psi\rangle$. Since $\langle T, \lambda\rangle$ was arbitrary this holds for every tree decomposition of G, so $\mathsf{hw}(G)$ is less than or equal to the ρ-width of G.

In the other direction, given an arbitrary hypertree decomposition $\langle T, \chi, \lambda\rangle$ of G, it is clear that $\langle T, \chi\rangle$ is a tree decomposition of G. Since $\chi(t) \subseteq \mathcal{V}(\lambda(t))$ for every node t of T by Definition 3.17, it follows that $\lambda(t)$ is an edge cover for $\chi(t)$, and hence that the ρ-width of $\langle T, \chi\rangle$ is less than or equal to the width of $\langle T, \chi, \lambda\rangle$. Since $\langle T, \chi, \lambda\rangle$ was arbitrary, this holds for every hypertree decomposition of G, and we are done. ■

Fractional hypertree width

Using the framework of width functions, we can define a relaxation of hypertree width known as fractional hypertree width, introduced by Grohe and Marx [GM06].

Definition 3.23 (Fractional edge cover) Let $G = \langle V, H\rangle$ be a hypergraph, and $X \subseteq V$. A *fractional edge cover* for X is a function $\gamma : H \to [0, 1]$ such that

$\sum_{v \in h \in H} \gamma(h) \geq 1$ for every $v \in X$. We call $\sum_{h \in H} \gamma(h)$ the weight of γ. The *fractional edge cover number* $\rho^*(X)$ of X is the minimum weight over all fractional edge covers for X. It is known that this minimum is always rational [GM06]. □

For a hypergraph G, it is clear that ρ^* is a width function on G. The *fractional hypertree width* $\mathsf{fhw}(G)$ of G is the ρ^*-width of G.

Theorem 3.24 ([GM06]) *Given a classic CSP instance P, deciding whether P has a solution can be done in time* $|P|^{\mathsf{fhw}(\mathsf{hyp}(P))+O(1)}$. □

Recall from Chapter 2 that for a CSP instance $P = \langle V, C \rangle$, we have

$$|P| = |V| + \sum_{v \in V} |D(v)| + \sum_{e[\delta] \in C} |\delta|.$$

To prove Theorem 3.24, Grohe and Marx show that the number of solutions to a classic CSP instance P is bounded by $|P|^{\mathsf{fhw}(\mathsf{hyp}(P))}$, and that this property is preserved when considering parts of an instance. From there, they obtain a polynomial-time algorithm. We will return to this proof in Chapter 4.

If we write $\mathsf{fhw}(\mathcal{H})$ for the maximum fractional hypertree width over the hypergraphs in $\mathcal{H}$, with $\mathsf{fhw}(\mathcal{H}) = \infty$ if this is unbounded, and $\mathsf{fhw}(\mathcal{H}) < \infty$ otherwise, we can state the following.

Corollary 3.25 *Let $\mathcal{H}$ be a class of hypergraphs. If* $\mathsf{fhw}(\mathcal{H}) < \infty$*, then the class of classic CSP instance* CSP($\mathcal{H}$) *is tractable.* □

Furthermore, using the characterization of hypertree width as a width function it becomes clear that $\mathsf{fhw}(G) \leq \mathsf{hw}(G)$ for every hypergraph G. Unlike hypertree width, however, the problem of deciding whether $\mathsf{fhw}(G) \leq k$ for a fixed k is not known to be tractable. The best we can do is to approximate $\mathsf{fhw}(G)$ in this case: A recent result by Marx [Mar10a] shows that for fixed k, the problem of either deciding that $\mathsf{fhw}(G) > k$ or else finding a tree decomposition of G with ρ^*-width at most $7k^3 + 31k + 7$ is tractable.

Submodular width

To go beyond fractional hypertree width, Marx [Mar10b, Mar09b] recently introduced the concept of submodular width. This concept uses a set of width functions satisfying a condition (submodularity), and considers the f-width of a hypergraph for every such function f.

Definition 3.26 (Submodular width function) Let $G = \langle V, H \rangle$ be a hypergraph. A width function f on G is *submodular* if for every set $X, Y \subseteq V$, we have $f(X) + f(Y) \geq f(X \cap Y) + f(X \cup Y)$. □

Definition 3.27 (Submodular width) Let G be a hypergraph. The *submodular width* $\mathsf{subw}(G)$ of G is the maximum f-width of G taken over all monotone submodular width functions f on G.

For a class of hypergraphs $\mathcal{H}$, we write $\mathsf{subw}(\mathcal{H})$ for the maximal submodular width over the hypergraphs in $\mathcal{H}$. If this is unbounded we write $\mathsf{subw}(\mathcal{H}) = \infty$; otherwise $\mathsf{subw}(\mathcal{H}) < \infty$. □

Unlike for fractional hypertree width and every other structural restriction discussed so far, bounded submodular width is not sufficient to obtain a tractable class. Instead, Marx shows the following.

Theorem 3.28 ([Mar09b]) *Let P be a classic CSP instance with $\mathsf{subw}(\mathsf{hyp}(P)) = k$. Deciding whether P has a solution can be done in time $O(2^{k \times 2^{O(n)}} \times |P|^{O(k)})$, where n is the number of vertices of $\mathsf{hyp}(P)$.* □

The rather complex proof uses properties of submodular functions fractional independent sets to split an instance into smaller instances that can be solved and their solutions combined.

As we can see, the running time has an exponential dependence on the number of vertices in the hypergraph of the instance. The class of classic CSP instances with bounded submodular width is therefore not tractable. However, this class is what is called fixed-parameter tractable [DF99, FG06].

Definition 3.29 (Fixed-parameter tractable) A *parameterized problem instance* is a pair $\langle k, P \rangle$, where P is a problem instance, such as a CSP instance, and $k \in \mathbb{N}$ a parameter.

Let S be a class of parameterized problem instances. We say that S is *fixed-parameter tractable* (in FPT) if there is a function f of one argument, as well as a constant c, such that every problem $\langle k, P \rangle \in S$ can be solved in time $O(f(k) \times |P|^c)$. □

The function f can be arbitrary, but must only depend on the parameter k. For CSP instances, a natural parameterization is by the size of the hypergraph of an instance, measured by the number of vertices in it. Since the hypergraph of an instance has a vertex for every variable, for every CSP instance P we consider the parameterized instance $\langle |\mathcal{V}(P)|, P \rangle$.

Corollary 3.30 *Let $\mathcal{H}$ be a class of hypergraphs. If $\mathsf{subw}(\mathcal{H}) < \infty$, then the class of classic CSP instances $\mathrm{CSP}(\mathcal{H})$ is fixed-parameter tractable.* □

Marx also shows that submodular width is always better than fractional hypertree width.

Theorem 3.31 ([Mar09b]) *For every hypergraph G, $\mathsf{subw}(G) \leq \mathsf{fhw}(G)$.* □

As for fractional hypertree width, there is currently no known polynomial-time algorithm for computing submodular width [Mar09b].

3.1.2 Restrictions on language

Language restrictions are usually defined in terms of closure properties of a constraint language L [CJ06,JCP98]. Below, we discuss one such property, referring the reader to [CJ06] for a much more comprehensive survey.

Definition 3.32 (Relational closure) Let R be a relation of arity n over a set D, and $f : D^k \to D$ a function. We say that R *is closed under* f if for every set of k tuples

$$\left\{ \begin{array}{c} \langle t_1^1, \ldots, t_n^1 \rangle \\ \vdots \\ \langle t_1^k, \ldots, t_n^k \rangle \end{array} \right\} \subseteq R$$

the tuple obtained by applying f componentwise, that is,

$$\left\langle f(t_1^1, \ldots, t_1^k), \ldots, f(t_n^1, \ldots, t_n^k) \right\rangle$$

also belongs to R. □

Language restrictions usually require the relations in a constraint language L to be closed under a specific function, or, more generally, some function with certain properties. As an example of the former, consider the next definition, which assumes that the relations are over a totally ordered set.

Definition 3.33 (Max-closed language) A constraint language L is *max-closed* if every relation in L is closed under the binary function max that returns the greater of two elements. □

Jeavons and Cooper have shown the following.

Theorem 3.34 ([JC95]) *Let L be a constraint language. A class of classic CSP instances* CSP(L) *is tractable if L is max-closed.* □

Cohen and Jeavons [CJ06] observe that solving a CSP instance over a max-closed language can be done by enforcing a certain notion of local consistency known as generalized arc consistency on the constraints. We discuss this notion in Section 3.2.1, and show examples of global constraints for which enforcing it can be done in polynomial time. This result therefore also applies to many classes of instances with global constraints. In particular, it applies to instances where all constraints are max-closed SAT clauses (cf. Example 2.12).

3.1.3 Hybrid tractability

Recent research into hybrid tractable classes of CSP instances has looked at the following way of representing instances [CJS10, Jég93, SJ08]. This strain of research has mainly focused on binary CSP instances, that is, those where all constraints are over two variables.

Definition 3.35 (Microstructure) Let $P = \langle V, C \rangle$ be a binary CSP instance. The *microstructure* of P is a graph $\mathsf{ms}(P)$ containing for each $v \in V$ and $a \in D(v)$ a vertex x^v_a. Furthermore, two vertices x^v_a and x^w_b are connected by an edge if for every constraint $e[\delta] \in C$ with $v, w \in \mathcal{V}(\delta)$, the assignment $\{v/a, w/b\}$ is in $e[\delta]$. □

In other words, the microstructure of a binary instance has all possible variable-value pairs as vertices, and edges connecting variable-value pairs that are allowed by all the constraints. Observe that by the definition above, if there are no constraints on two variables v, w, then the microstructure will have edges between every pair of vertices of the form x^v_a, x^w_b. As the microstructure of an instance depends on both the hypergraph of the instance and the relations of the constraints, it is indeed a hybrid property.

To define the restriction on microstructures that we need, the following concepts will be useful.

Definition 3.36 (Induced subgraph) Let $G = \langle V, E \rangle$ be a graph. A *subgraph* of G is any graph $G' = \langle V', E' \rangle$ such that $V' \subseteq V$ and $E' \subseteq E$. If E' contains every edge $\{x, y\} \in E$ such that $x, y \in V'$, then G' is an *induced subgraph* of G. □

Definition 3.37 (Perfect graph) Let G be a graph. A *clique* in G is a complete subgraph induced by G. The *chromatic number* of G is the smallest number of colours required to colour the vertices of G such that no edge is monochromatic.

A graph G is *perfect* if for every induced subgraph G' of G, the chromatic number of G' is equal to the size of largest clique in G'. □

Using these definitions, we can now state the following result.

Theorem 3.38 ([SJ08]) *Given a CSP instance P such that* $\mathsf{ms}(P)$ *is perfect, deciding whether P has a solution can be done in time polynomial in the size of* $\mathsf{ms}(P)$*.* □

The proof uses the observation that every solution to a binary CSP instance P corresponds to a maximal clique in $\mathsf{ms}(P)$. A result of Grötschel et al. [GLS81] states that finding a maximal clique in a perfect graph can be done in polynomial time in the size of the graph, and the theorem follows.

Among other examples, Salamon and Jeavons [SJ08] also show, using a recent characterization of perfect graphs, that CSP instances on trees, i.e. those whose hypergraphs have treewidth one, have perfect microstructures.

3.2 Research specific to global constraints

In the previous section, most of the results we discuss were developed with classic CSP instances in mind. In this section, we look at research aimed explicitly at CSP instances with global constraints. Here, quite a lot is known about global

constraint *propagation*, i.e. the enforcement of various kinds of *local consistency* on specific global constraints [Bes06, Tac09].

3.2.1 Local consistency

Notions of local consistency require that the domain values of the variables of a constraint or instance satisfy certain conditions, either for each variable or for sets of them. To quote Bessiere [Bes06], “Constraint propagation embeds any reasoning which consists in explicitly forbidding values or combinations of values for some variables of a problem because a given subset of its constraints cannot be satisfied otherwise.”

If the variables of a constraint do not satisfy a given notion of local consistency, we can make it locally consistent (enforce local consistency on it) by removing values from the domain of each variable [BHHW04]. If we have a notion of consistency that considers sets of variables, such as k-consistency [ABD07, Coo89], we can enforce it by adding (learning) new constraints that forbid unsuitable combinations.

Below, we discuss a few examples of local consistency conditions on the variables of individual constraints, since this is the type commonly found in solvers [Bes06]. For more details, we refer the reader to [Tac09] for an overview of propagation and local consistency techniques and how they are used by constraint solvers.

As a starting point, consider the widely used notion of generalized arc consistency [Bes06, BHHW04, KNW12].

Definition 3.39 (Generalized arc consistency) A global constraint $e[\delta]$ is *generalized arc consistent* (GAC) if for every variable $v \in \mathcal{V}(\delta)$ and domain element $a \in D(v)$ there exists an assignment $\theta \in e[\delta]$ such that $\theta(v) = a$ and $\theta(w) \in D(w)$ for every $w \in \mathcal{V}(\delta)$. □

In other words, if a constraint is GAC, then every domain value of every variable is part of some satisfying assignment. A constraint that is not GAC can be made GAC by removing domain elements that do not occur in any satisfying assignment. Having efficient algorithms for enforcing GAC on the constraints in a CSP instance allow a constraint solver to significantly prune the search space when solving an instance.

In general, however, the problem of deciding, for a given constraint, whether there exists a satisfying assignment that assigns a specific value to a specific variable may not be feasible in polynomial time. Examples of constraints for which this decision problem is known to be hard include constraints that encode known NP-complete problems, such as SAT, as well as unrestricted EGC constraints [QLOvBG04] (cf. Chapter 2).

An example of a global constraint type that can be made GAC in polynomial time is the Alldifferent constraint (cf. Example 2.10).

Theorem 3.40 ([Rég94]) *An Alldifferent constraint $e[\delta]$ can be made GAC in time $O(|\mathcal{V}(\delta)|^2 \times d^2)$, where $d = \max(\{|D(v)| \mid v \in \mathcal{V}(\delta)\})$.* □

The algorithm used represents the given constraint $e[\delta]$ as a bipartite graph having $\mathcal{V}(\delta)$ on one side and the domain elements of all the variables on the other (the value graph), with edges connecting each variable v to its set of domain elements $D(v)$. In this graph, an allowed assignment for $e[\delta]$ corresponds to a matching, i.e. a set of edges with no vertices in common, that covers every $v \in \mathcal{V}(\delta)$ by an edge. The constraint is GAC if and only if every edge belongs to at least one such matching, and finding such matchings in bipartite graphs can be done in polynomial time [HK73].

Another example of a constraint that can be made GAC in polynomial time is the EGC constraint (cf. Example 2.9) where all cardinality sets are intervals.

Definition 3.41 (Gaps and intervals) Let S be a finite set of natural numbers. We say that S has a *gap of length m* if there exists $i \in \mathbb{N}$ such that $\min(S) < i < \max(S)$ and $\{i, \dots, i + m - 1\} \cap S = \emptyset$. If S has no gap of length m, then S is m-gap free. If S is 1-gap free, we say that S is an *interval*.

Let $e[\delta]$ be an EGC constraint. We say that the cardinality sets of $e[\delta]$ have gaps of length at most m if they are all $m + 1$-gap free. □

Theorem 3.42 ([Rég96]) *An EGC constraint $e[\delta]$ whose cardinality sets are intervals can be made GAC in time $O(|\mathcal{V}(\delta)|^2 \times |D(V)|^2)$.* □

The proof also makes use of the value graph, this time to transform an EGC constraint into a network flow problem. GAC is then achieved by using network flow algorithms. Samer and Szeider [SS11] have since shown that EGC constraints where the cardinality sets have gaps of length at most one, i.e. are 2-gap free, can also be made GAC in polynomial time. To show this result, they reduce the problem of enforcing GAC on an EGC constraint to the problem of finding general factors in graphs, then apply a theorem due to Cornuéjols [Cor88] about the latter.

As GAC can be hard to enforce efficiently, weaker notions of local consistency have been studied, particularly for numeric constraints such as EGC. Below, we give one example of such a notion of consistency, which assumes that the domain values are ordered.

Definition 3.43 (Range consistency) A global constraint $e[\delta]$ is *range consistent* if for every variable $v \in \mathcal{V}(\delta)$ and domain element $a \in D(v)$ there exists an assignment $\theta \in e[\delta]$ such that $\theta(v) = a$, and for every $w \in \mathcal{V}(\delta)$, $\min(D(w)) \leq \theta(w) \leq \max(D(w))$. □

Observe that since $\min(D(w)) \leq \theta(w) \leq \max(D(w))$ does not imply that $\theta(w)$ is actually in $D(w)$, range consistency is a weaker condition than GAC. Furthermore, while checking whether there exists a satisfying assignment is trivial on a constraint that is GAC, it is not necessarily so on a constraint that is

range consistent. Enforcing range consistency therefore tends to be easier than enforcing GAC. As an example, Quimper et al. have shown the following.

Theorem 3.44 ([QLOvBG04]) *An EGC constraint $e[\delta]$ can be made range consistent in time $O(|\mathcal{V}(\delta)|^2 + |D(V)| \times |\mathcal{V}(\delta)|)$.* □

Unfortunately, a result due to Kutz et al. demonstrates that local consistency techniques are not well suited for finding tractable classes of CSP *instances* with global constraints.

Theorem 3.45 ([KEKM08]) *Let e be the Alldifferent constraint type. Deciding whether a given CSP instance $\langle V, \{e[\delta_1], e[\delta_2]\}\rangle$ has a solution is NP-complete.* □

The proof uses a somewhat involved reduction from the NP-hard problem of set packing [GJ79], and implies that having two Alldifferent constraints is enough to encode an NP-complete problem. As Alldifferent can be expressed both as EGC and NValue constraints, this also applies to them. Local consistency approaches are therefore of limited use by themselves. However, Green and Jefferson [GJ08] have recently investigated structural restrictions that yield tractability for classes of CSP instances where every constraint can be made e.g. GAC in polynomial time.

3.2.2 Propagators

Green and Jefferson's [GJ08] approach is to represent all constraints as propagators, that is, functions that enforce local consistency on the variables of a constraint. Below, we restate their definition of a propagator using our notation.

Definition 3.46 (Propagator) Given a set of variables V, a *set of subdomains* for V is any set $L = \{\langle v, A\rangle \mid v \in V \text{ and } A \subseteq D(v)\}$. The set of assignments induced by L, denoted $\Theta(L)$, is the greatest set of assignments θ to V such that for every $\langle v, A\rangle \in L$ we have $\theta(v) \in A$.

Let $e[\delta]$ be a constraint. A *propagator* for $e[\delta]$ is a function f that maps a list of subdomains for $\mathcal{V}(\delta)$ to another list of subdomains for $\mathcal{V}(\delta)$ subject to the following conditions:

1. $\Theta(f(L)) \subseteq \Theta(L)$,
2. if $\Theta(L) \subseteq \Theta(L')$, then $\Theta(f(L)) \subseteq \Theta(f(L'))$,
3. if $\theta \in \Theta(L) \cap e[\delta]$, then $\theta \in \Theta(f(L))$, and
4. if $\Theta(L) = \{\theta\}$ and $\theta \notin e[\delta]$ then $\Theta(f(L)) = \emptyset$. □

The definition looks technical but is not complicated. Condition one states that applying a propagator can only remove domain elements. Conditions two and three state, respectively, that propagators are monotone and preserve satisfying assignments. Finally, condition four requires that a propagator checks

whether a single assignment is a satisfying one. In our terminology, representing a constraint by a propagator corresponds to a constraint $e[\delta]$ being a propagator, rather than simply a parameterized algorithm that maps satisfying assignments to 1 (cf. Definition 2.7). However, condition four above means that a propagator satisfies Definition 2.7, and can be considered as a special case of our approach to global constraints.

Looking at the values that have to be removed rather than those that are preserved makes it easier to define a certain notion of local consistency that is used in some solvers.

Definition 3.47 (PAM) A *partial assignment membership* (PAM) propagator f for a constraint $e[\delta]$ is any propagator that satisfies the following condition: If L is a list of subdomains for $\mathcal{V}(\delta)$ such that for every $\langle v, A\rangle \in L$, either $|A| = 1$ or $A = D(v)$, then $\Theta(f(L)) = \emptyset$ if $\Theta(L) \cap e[\delta] = \emptyset$. □

In other words, a PAM propagator allows us to check if a partial assignment extends to a full one using the complete domains of $\mathcal{V}(\delta)$. Observe that a propagator that implements GAC (i.e. always removes all domain elements not part of any satisfying assignment) is also a PAM propagator. In fact, Green and Jefferson [GJ08] show that a propagator that implements GAC is the strongest possible kind of propagator for a constraint, in the sense that is removes every domain value that can possibly be removed while respecting the conditions of Definition 3.46.

We say that a CSP instance is represented by PAM propagators if every constraint in it is represented by a PAM propagator, and that an isolated vertex in a hypergraph is a vertex that occurs in at most one hyperedge.

Theorem 3.48 ([GJ08]) *Let $\mathcal{H}$ be a class of hypergraphs, and $\mathcal{H}'$ the class of hypergraphs obtained from $\mathcal{H}$ by removing all isolated vertices. Assuming* $\mathsf{FPT} \neq \mathsf{W}[1]$*, the class of CSP instances* CSP($\mathcal{H}$)*, represented by PAM propagators, is tractable if and only if* $\mathrm{tw}(\mathcal{H}') < \infty$. □

The proof is by a reduction from a CSP instance represented by PAM propagators to a classic CSP instance, and the only if part uses Theorem 3.8. Using a similar approach, Green and Jefferson also exhibit a tractable class for instances represented by GAC propagators.

To restate this result without reference to a specific representation, let Γ be a set of constraints such that we can compute a PAM propagator for any constraint in Γ in polynomial time. Given a class of hypergraphs $\mathcal{H}$, the class CSP($\mathcal{H}$) *of instances that only contain constraints from* Γ is tractable if $\mathrm{tw}(\mathcal{H}') < \infty$. Likewise, for the tractable class of instances represented by GAC propagators, consider a set of constraints Γ such that we can compute a GAC propagator for any of them in polynomial time.

3.3 Discussion

Considering the research we have seen in this chapter, on the structural side we observe a pattern: While bounded treewidth gives us tractability for arbitrary CSP instances, query width and beyond do so only for classic CSP instances. As the following example shows, this cannot be helped.

Example 3.49 The NP-complete problem of 3-colourability [GJ79] is to decide, given a graph $\langle V, E \rangle$, whether the vertices V can be coloured with three colours such that no two adjacent vertices have the same colour (cf. Chapter 1).

We may reduce this problem to a CSP with EGC constraints (cf. Example 2.9) as follows: Let V be the set of variables for our CSP instance, each with domain $\{r, g, b\}$. For every edge $\langle v, w \rangle \in E$, we post an EGC constraint with scope $\{v, w\}$, parametrised by the function K such that $K(r) = K(g) = K(b) = \{0, 1\}$. Finally, we make the hypergraph of this CSP instance acyclic by adding an EGC constraint with scope V parametrised by the function K' such that $K'(r) = K'(g) = K'(b) = \{0, \ldots, |V|\}$. This reduction clearly takes polynomial time in the size of the input graph $\langle V, E \rangle$. Furthermore, since the hypergraph of the resulting instance is acyclic, it has query, hypertree, and fractional hypertree width at most one.

As the constraint with scope V allows all possible assignments, any solution to this CSP is also a solution to the 3-colourability problem, and vice versa. □

Constructing arbitrary propagators for binary constraints can be done in polynomial time, and the constraint with scope V, which allows all assignments, is likewise trivial to propagate [GJ08]. Therefore, the above example works even when we consider the propagator representation of global constraints. Hence, using global constraints an NP-complete problem can be reduced in polynomial time to a class of CSP instances with acyclic hypergraphs, and so treewidth is the only structural restriction that guarantees tractability. Furthermore, Theorem 3.45 shows that even restricting ourselves to just the Alldifferent constraint type is not sufficient to go beyond treewidth.

Another observation to make is that if we attempt the reduction in Example 3.49 using classic constraints, it will no longer take polynomial time due to the constraint with scope V having exponentially many satisfying assignments to list. However, we can still do the reduction and obtain classic constraints with the same relations. One way to interpret the above example is that structural restrictions beyond treewidth are, in fact, not structural but "hybrid" — they rely on properties specific to classic constraints in addition to those of an instance's hypergraph.

However, unlike the hybrid results in Section 3.1.3, the properties of classic constraints that structural restrictions beyond treewidth rely on are not just properties of the constraint relations, but also of their representations. The ubiquity of extensionally represented constraints in the study of CSP complexity obscures this fact. As soon as we move to global constraints, however, these prop-

erties fail, as we have seen. In the light of this chapter, we therefore consider the following two questions.

First, what properties of classic constraints make classic CSP instances tractable on hypergraphs of e.g. bounded hypertree width? In particular, is there a single property that works for all the restrictions we discussed in Section 3.1.1? We investigate this question in Chapter 4, and identify such a property. We then identify classes of global constraints that possess this property, and that, in turn, allows us to use structural restrictions beyond treewidth to obtain tractable and fixed-parameter tractable classes of CSP instances with global constraints.

Second, what classes of global constraints, if any, allow structural restrictions *between* treewidth and query width that yield tractable classes? We investigate this question in Chapter 5, and identify such a class and restriction.

In a sense, this chapter, and more specifically the summary above, explains the title of this thesis. Both questions that we have asked here will be answered by exploiting both the structures of CSP instances as well as the constraints these instances contain. As such, we are firmly in the realm of hybrid tractability.

Chapter 4

Tractability due to few solutions in key places

In which we show that having few solutions in key places is what makes many structural restrictions work for classic CSP instances. Furthermore, we discover that this property applies in several contexts.

The goal of this chapter is to understand why the structural restrictions beyond treewidth, such as hypertree and fractional hypertree width, define tractable classes for classic CSP instances but not for instances with global constraints. Once we know the relevant properties of classic constraints, we can proceed to look for global constraints that also possess such properties. For instances restricted to such global constraints, we should be able to apply the results from Section 3.1.1 to achieve tractability.

We are going to start our investigation by considering fractional hypertree width in more detail. In Section 3.1.1, we mention that Grohe and Marx [GM06] use a bound on the number of solutions to a classic CSP instance to obtain tractability for classic CSP instances of bounded fractional hypertree width. We also said that this property is preserved when we consider parts of a CSP instance. The following definition formalizes what we mean by "parts", and is required to state the algorithm that Grohe and Marx use in their paper.

Definition 4.1 (Constraint projection) Let $e[\delta]$ be a constraint. The *projection of* $e[\delta]$ onto a set of variables $X \subseteq \mathcal{V}(\delta)$ is the constraint $\mathrm{pj}_X(e[\delta])$ such that $\mu \in \mathrm{pj}_X(e[\delta])$ if and only if there exists $\theta \in e[\delta]$ with $\theta|_X = \mu$.

For a CSP instance $P = \langle V, C\rangle$ and $X \subseteq V$ we define $\mathrm{pj}_X(P) = \langle X, C'\rangle$, where C' is the least set containing for every $e[\delta] \in C$ such that $X \cap \mathcal{V}(\delta) \neq \emptyset$ the constraint $\mathrm{pj}_{X\cap\mathcal{V}(\delta)}(e[\delta])$. □

Their algorithm is given as Algorithm 1, and is essentially the usual recursive search algorithm for finding all solutions to a CSP instance by considering smaller and smaller sub-instances using constraint projections. To show that Algorithm 1 does indeed find all solutions, we will use the following property of constraint projections.

Algorithm 1 Enumerate all solutions of a CSP instance

```
procedure ENUMERATESOLUTIONS(CSP instance P)            ▷ Returns sol(P)
    Solutions ← ∅
    if 𝒱(P) = ∅ then
        return {⊥}                                      ▷ The empty assignment
    else
        w ← chooseVar(𝒱(P))                             ▷ Pick a variable from 𝒱(P)
        Θ = EnumerateSolutions(pj_{𝒱(P)−{w}}(P))
        for θ ∈ Θ do
            for a ∈ D(w) do
                if θ ∪ ⟨w, a⟩ is a solution to P then
                    Solutions.add(θ ∪ ⟨w, a⟩)
                end if
            end for
        end for
    end if
    return Solutions
end procedure
```

Lemma 4.2 *Let P be a CSP instance. For every $X \subseteq \mathcal{V}(P)$, we have* $\mathsf{sol}(\mathsf{pj}_X(P)) \supseteq \pi_X(\mathsf{sol}(P))$. □

Proof Given $P = \langle V, C\rangle$, let $X \subseteq \mathcal{V}(P)$ be arbitrary, and let $C' = \{e[\delta] \in C \mid X \cap \mathcal{V}(\delta) \neq \emptyset\}$. For every $\theta \in \mathsf{sol}(P)$ and constraint $e[\delta] \in C'$ we have that $\theta|_{\mathcal{V}(\delta)} \in e[\delta]$ since θ is a solution to P. By Definition 4.1, it follows that for every $e[\delta] \in C'$, $\theta|_{X\cap\mathcal{V}(\delta)} \in \mathsf{pj}_{X\cap\mathcal{V}(\delta)}(e[\delta])$. Since the set of constraints of $\mathsf{pj}_X(P)$ is the least set containing for each $e[\delta] \in C'$ the constraint $\mathsf{pj}_{X\cap\mathcal{V}(\delta)}(e[\delta])$, we have $\theta|_X \in \mathsf{sol}(\mathsf{pj}_X(P))$, and hence $\mathsf{sol}(\mathsf{pj}_X(P)) \supseteq \pi_X(\mathsf{sol}(P))$. Since X was arbitrary, the claim follows. ■

Using Lemma 4.2, we can now show that Algorithm 1 is correct, that is, finds all solutions to a CSP instance.

Theorem 4.3 (Correctness of Algorithm 1) *Let P be a CSP instance. We have that* EnumerateSolutions$(P) = \mathsf{sol}(P)$. □

Proof The proof is by induction on the set of variables in P. For the base case, if $\mathcal{V}(P) = \emptyset$, the empty assignment is the only solution.

Otherwise, choose a variable $w \in \mathcal{V}(P)$, and let $X = \mathcal{V}(P) - \{w\}$. By induction, we can assume that EnumerateSolutions$(\mathsf{pj}_X(P)) = \mathsf{sol}(\mathsf{pj}_X(P))$. Since for every $\theta \in \mathsf{sol}(P)$ there exists $a \in D(w)$ such that $\theta = \theta|_X \cup \langle w, a\rangle$, and furthermore $\theta|_X \in \pi_X(\mathsf{sol}(P))$, it follows by Lemma 4.2 that $\theta|_X \in \mathsf{sol}(\mathsf{pj}_X(P))$. Since Algorithm 1 checks every assignment of the form $\mu \cup \langle w, a\rangle$ for every $\mu \in \mathsf{sol}(\mathsf{pj}_X(P))$ and $a \in D(w)$, it follows that EnumerateSolutions$(P) = \mathsf{sol}(P)$. ■

The time required for this algorithm depends on three key factors, which we are going to enumerate and discuss below. Let

1. $s(P)$ be the maximum of the number of solutions to each of the instances $\mathsf{pj}_{\mathcal{V}(P)-\{w\}}(P)$,
2. $c(P)$ be the maximum time required to check whether an assignment is a solution to P, and
3. $b(P)$ be the maximum time required to construct any instance $\mathsf{pj}_{\mathcal{V}(P)-\{w\}}(P)$.

There are $|\mathcal{V}(P)|$ calls to EnumerateSolutions. For each call, we need $b(P)$ time to construct the projection, while the double loop takes at most $s(P) \times |D(w)| \times c(P)$ time. Therefore, letting $d = \max(\{|D(w)| \mid w \in \mathcal{V}(P)\})$, the running time of Algorithm 1 is bounded by $O\big(|\mathcal{V}(P)| \times (s(P) \times d \times c(P) + b(P))\big)$.

Since constructing the projection of a classic CSP instance can be done in polynomial time, and likewise checking that an assignment is a solution, the whole algorithm runs in polynomial time if $s(P)$ is a polynomial in the size of P. As we mentioned in Section 3.1.1, for fractional hypertree width Grohe and Marx show the following.

Lemma 4.4 ([GM06]) *A classic CSP instance P has at most $|P|^{\mathsf{fhw}(\mathsf{hyp}(P))}$ solutions.* □

The proof uses a combinatorial result about hypergraphs, known as Shearer's Lemma [CGFS86], to bound the number of solutions. Since fractional hypertree width is a monotone width function (cf. Section 3.1.1), it follows that for any instance P and $X \subseteq \mathcal{V}(P)$, $\mathsf{fhw}(\mathsf{hyp}(\mathsf{pj}_X(P))) \leq \mathsf{fhw}(\mathsf{hyp}(P))$. Therefore, for classic CSP instances of bounded fractional hypertree width $s(P)$ is indeed polynomial in $|P|$.

4.1 Useful width

As it turns out, analogues of Lemma 4.4 also hold for treewidth, as well as for incidence, query, and hypertree width.

Lemma 4.5 *Let $\alpha \in \{\mathsf{tw}, \mathsf{iw}, \mathsf{qw}, \mathsf{hw}\}$. The number of solutions to a classic CSP instance P is at most $|P|^{\alpha(\mathsf{hyp}(P))+2}$.* □

Proof For hw, we can use the characterization of hw as the edge cover number of a hypergraph from Section 3.1.1. Since an edge cover is also a fractional edge cover, the claim follows immediately from Lemma 4.4.

Next, since we have that $\mathsf{hw}(\mathsf{hyp}(P)) \leq \mathsf{qw}(\mathsf{hyp}(P)) \leq \mathsf{iw}(\mathsf{hyp}(P)) + 1 \leq \mathsf{tw}(\mathsf{hyp}(P)) + 2$ by Theorems 3.10, 3.14 and 3.19, it follows that the claim also holds for $\alpha \in \{\mathsf{tw}, \mathsf{iw}, \mathsf{qw}\}$, and we are done. ■

Since all of these width measures are also monotone [Mar09a], it follows that $s(P)$ is polynomial for classic CSP instances with any of them bounded.

On the other hand, for instances with global constraints Lemma 4.5 is only true for treewidth, as demonstrated by Example 3.49. Therefore, the property of having polynomially many solutions in the size of the instance *for every projection* can be said to be the reason why structural restrictions beyond treewidth yield tractability for classic CSP instances. We formalize this idea below, using an informal description that can be found in a talk by Marx [Mar09a]. Since our formalization will make heavy use of width functions, the results in this section will not include incidence width. While it is possible to define incidence width in terms of width functions, it would be somewhat cumbersome to do so. Instead, we will include incidence width in Section 4.2, where we prove a more general version of the results from this section.

As we discussed in Section 3.3, the property that we are trying to pin down depends, on the constraint side, not only on the constraints' relations but also on the way these relations are represented, since representation plays a big part in determining the size of an instance. Therefore, while this is a hybrid property, the notion of a constraint language from Section 3.1.2 is not the right way of viewing the constraints — it ignores the way constraints are represented. Instead, we will use the notion below to describe classes of global constraints that possess this property of having few solutions for every projection.

Definition 4.6 (Constraint catalogue) A *constraint catalogue* Γ is a set of global constraints. A CSP instance $\langle V, C\rangle$ is said to be *over* Γ if for every $e[\delta] \in C$ we have $e[\delta] \in \Gamma$.

Let $\mathcal{H}$ be a class of hypergraphs, and Γ a constraint catalogue. We write $\text{CSP}(\mathcal{H}, \Gamma)$ for the class of CSP instances over Γ whose hypergraphs are in $\mathcal{H}$. □

Example 4.7 An example of a constraint catalogue would be the set of all possible table constraints, which we will denote by **Ext**. Using this notation, we can restate Theorem 3.18 as follows: For a class of hypergraphs $\mathcal{H}$ such that $\mathsf{hw}(\mathcal{H}) < \infty$, $\text{CSP}(\mathcal{H}, \mathbf{Ext})$ is tractable.

Another example of a constraint catalogue would be the set of all possible EGC constraints whose cardinality sets are intervals (cf. Definition 3.41 and Theorem 3.42). Denote this catalogue by Γ_{IntEGC}, and let $\mathcal{H}$ be the maximal class of hypergraphs such that $\mathsf{hw}(\mathcal{H}) = 1$. Using Example 3.49, we have shown that $\text{CSP}(\mathcal{H}, \Gamma_{IntEGC})$ is intractable. □

Observe that it is not at all clear how to restate Example 4.7 using constraint languages rather than catalogues, since the former ignore representation.

Definition 4.8 (Useful width function for a catalogue) Let Γ be a constraint catalogue, and G a hypergraph. A monotone width function f on G is a *useful width function* for Γ up to a constant factor c if, for every CSP instance $P \in \text{CSP}(\{G\}, \Gamma)$ and subset of variables $W \subseteq \mathcal{V}(P)$, we have $|\mathsf{sol}(\mathsf{pj}_W(P))| \leq |P|^{cf(W)}$.

A hypergraph G has *useful width at most* $k \in \mathbb{N}$ for Γ if there exists a useful width function f for Γ on G such that the f-width of G is at most k. A class of

hypergraphs has useful width at most k for Γ if every member of it does, using the same constant factor. We say that such a class has bounded useful width for Γ. □

If we are given a classic CSP instance P with $\mathsf{hyp}(P)$ having useful width at most k, then there is a tree decomposition $\langle T, \lambda \rangle$ of $\mathsf{hyp}(P)$ such that for every node t of T, $|\mathsf{sol}(\mathsf{pj}_{\lambda(t)}(P))| \leq |P|^{ck}$ for some c, and this holds for every *subset* of $\lambda(t)$ since bounded useful width requires a monotone width function. Therefore, invoking Algorithm 1 on $\mathsf{pj}_{\lambda(t)}(P)$ will only take polynomial time. After doing so for every node of T, we can solve P by standard consistency techniques.

As we have observed above, treewidth, query width, hypertree width, and fractional hypertree width are useful width functions for any classic catalogue, that is, one that only contains table constraints. However, if we consider useful width functions without any restrictions on the constraint catalogue (i.e. allow arbitrary global constraints), then there is some bad news.

Theorem 4.9 *If a class of hypergraphs $\mathcal{H}$ has bounded useful width for every constraint catalogue, then it has bounded treewidth.* □

Proof Let $\mathcal{H}$ be a class of hypergraphs with bounded useful width for every constraint catalogue. For every $G \in \mathcal{H}$, consider the CSP instance P with $\mathsf{hyp}(P) = G$ where every constraint allows all possible assignments, and every variable has domain size $n = |\mathcal{V}(P)|$. By assumption, $\mathcal{H}$ has bounded useful width for some catalogue that contains all these constraints, so let k be the bound. Since the descriptions of these constraints all just specify the constraint scope, they have the same fixed size. Therefore, the size of P is at most n^2. Furthermore, by construction we have for every $W \subseteq \mathcal{V}(P)$ that $|\mathsf{sol}(\mathsf{pj}_W(P))| = n^{|W|}$.

Let f be a useful width function for G such that the f-width of G is at most k, and let $\langle T, \lambda \rangle$ be a tree decomposition of G with f-width at most k. Since f is useful, we have for every node t of T that $|\mathsf{sol}(\mathsf{pj}_{\lambda(t)}(P))| = n^{|\lambda(t)|} \leq n^{2ck}$, and hence that $tw(G) \leq 2ck$. Since this holds for every $G \in \mathcal{H}$ using the same constant c, it follows that $\mathrm{tw}(\mathcal{H}) \leq 2ck$, and we are done. ■

Since treewidth as a width function is clearly useful, the above means that for arbitrary global constraints, any useful width function is essentially treewidth. In other words, for every class of hypergraphs $\mathcal{H}$ with bounded useful width for arbitrary global constraints, the class of CSP instances whose hypergraphs are in $\mathcal{H}$ behaves just like a class of classic instances — precisely because such a class *is* a class of classic instances in disguise. Furthermore, since bounded treewidth for a class of hypergraphs implies bounded rank, this is the only tractable case per Theorem 3.8.

We therefore need to look for restrictions on constraint catalogues that would make width functions encoding e.g. hypertree width useful. However, we would first need to show that if a class of hypergraphs $\mathcal{H}$ has bounded useful width

for an arbitrary catalogue Γ, then CSP($\mathcal{H}, \Gamma$) is tractable. This, however, is not true for arbitrary constraint catalogues, since we allow constraints such as EGC for which checking whether there exists a satisfying assignment is NP-hard. As constructing the projection of a constraint requires checking whether there is a satisfying assignment, the running time of Algorithm 1 will not stay polynomial. On the other hand, the following property suffices.

Definition 4.10 (Partial assignment checking) A global constraint catalogue Γ allows *partial assignment checking* if for any constraint $e[\delta] \in \Gamma$ we can decide in polynomial time whether a given assignment θ to a set of variables $W \subseteq \mathcal{V}(\delta)$ is contained in an assignment that satisfies $e[\delta]$, i.e. whether there exists $\mu \in e[\delta]$ such that $\theta = \mu|_W$. □

As it happens, this property is equivalent to the notion of having a polynomial-time partial assignment membership propagator (cf. Section 3.2.2) in the following sense: If a constraint catalogue Γ satisfies Definition 4.10, then we can in polynomial time build a PAM propagator for any constraint from Γ. Likewise, if we have a set of constraints Γ such that we can build a PAM propagator for any of them in polynomial time, then Γ satisfies Definition 4.10. However, since we are not interested in propagation per se, the above definition will be easier to work with. The connection to propagation also implies that there are many well-known global constraints that satisfy Definition 4.10, such as clauses, Alldifferent, and global cardinality constraints whose cardinality sets have gaps of length at most one (cf. Section 3.2.1).

If a catalogue Γ satisfies Definition 4.10, we can for any constraint $e[\delta] \in \Gamma$ build arbitrary projections of it, that is, construct the global constraint $\mathsf{pj}_X(e[\delta])$ for any $X \subseteq \mathcal{V}(\delta)$, in polynomial time. That allows us to prove the following.

Theorem 4.11 *Let $\mathcal{H}$ be a class of hypergraphs with bounded useful width for a constraint catalogue Γ that allows partial assignment checking.* CSP($\mathcal{H}, \Gamma$) *is tractable.* □

Proof We need only to verify that Algorithm 1 runs in polynomial time. Since Γ allows partial assignment checking, constructing any projection of a CSP instance P from CSP($\mathcal{H}, \Gamma$) can be done in polynomial time. Checking that an assignment is a solution runs in polynomial time by definition for any global constraint. Finally, bounded useful width means, as before, that there exists a tree decomposition $\langle T, \lambda \rangle$ of $\mathsf{hyp}(P)$ such that for every node t of T, $|\mathsf{sol}(\mathsf{pj}_{\lambda(t)}(P))|$ is polynomial in the size of P, and furthermore that this holds for subsets of $\lambda(t)$. ■

It is tempting to think that partial assignment checking gives us an interesting tractable case by itself. However, it follows from Theorem 3.48 that if Γ is the maximal catalogue that allows partial assignment checking for a fixed polynomial (i.e. contains all such constraints), then for a class of hypergraph $\mathcal{H}$, CSP($\mathcal{H}, \Gamma$) is tractable if and only if $\mathcal{H}$ has bounded treewidth. Taken together

with Theorem 4.11, this means that here, once again, bounded useful width implies bounded treewidth. However, consider the property defined below.

Definition 4.12 (Sparse catalogue) A constraint catalogue Γ is *sparse* if there exists $c \in \mathbb{N}$ such that for every $e[\delta] \in \Gamma$, $|\{\theta \mid \theta \in e[\delta]\}| \leq |\delta|^c$. □

In other words, the constraints in a sparse catalogue all have few satisfying assignments. Note, however, that this does not imply that such constraints allow partial assignment checking — there are classes of problems in NP with few solutions that, nevertheless, are not known to be solvable in polynomial time, and indeed are conjectured not to be. An example is unique-SAT, the class of SAT instances that have one or no solutions [VV86]. As we have discussed in Chapter 2, arbitrary SAT instances can be seen as single global constraints. Therefore, the class of unique-SAT instances forms a sparse catalogue, but is not known to allow partial assignment checking.

Furthermore, the problem of checking whether a constraint $e[\delta]$ from a given constraint catalogue Γ satisfies Definition 4.12 with respect to a given polynomial is hard even when the constraints in Γ can all be solved in polynomial time. To see this, observe that if we can decide this problem in polynomial time, then we can find the exact number of satisfying assignments for $e[\delta]$ by binary search. However, this latter problem is in the complexity class #P [Val79a], which is believed to be hard [Tod91]. Also, the problem of finding the number of solutions to a given instance of 2SAT, that is, SAT where every clause contains at most two literals, is complete for #P [Val79b], even though solving such an instance can be done in polynomial time.

Despite such bad news, however, it is not always difficult to recognise constraints with polynomially many satisfying assignments. A trivial example would be table constraints. For a less trivial example, consider the constraint C^β from Example 2.14 (cf. Section 4.2.1 for the analysis). Consider also negative constraints (cf. Example 2.13) that disallow a lot of assignments.

Having stated Definitions 4.10 and 4.12, we can now proceed to establish some new tractable classes of CSP instances.

Lemma 4.13 *If a sparse catalogue Γ allows partial assignment checking, then any $e[\delta] \in \Gamma$ can be converted to an equivalent table constraint in polynomial time.* □

Proof Let Γ be a sparse catalogue that allows partial assignment checking. For every $e[\delta] \in \Gamma$, the CSP instance $P = \langle \mathcal{V}(\delta), \{e[\delta]\}\rangle$ has the property that for every $X \subseteq \mathcal{V}(\delta)$, $\mathsf{sol}(\mathsf{pj}_X(P)) = \pi_X(\mathsf{sol}(P))$. Since Γ allows partial assignment checking, constructing projections of P can be done in polynomial time, and since Γ is sparse, the number of solutions to every projection is polynomial. Therefore, Algorithm 1 can find all solutions to P, which are exactly the satisfying assignments for $e[\delta]$, in polynomial time, giving us the table constraint required. ■

Lemma 4.14 *Let ϕ be a polynomial. If a constraint $e[\delta]$ can be converted to a constraint $e'[\delta']$ in time $\phi(|\delta|)$, then $|\delta'| = O(\phi(|\delta|))$.* □

Proof If $|\delta'|$ is not $O(\phi(|\delta|))$, then outputting δ' it cannot be done in time $O(\phi(|\delta|))$, contradicting the time required for the conversion. ■

Lemma 4.15 *If every constraint in a catalogue Γ can be converted to an equivalent constraint in a catalogue Γ' in polynomial time, then a width function f on a hypergraph G which is useful for Γ' is also useful for Γ.* □

Proof Let $P \in \mathrm{CSP}(\{G\}, \Gamma)$, and convert it to $P' \in \mathrm{CSP}(\{G\}, \Gamma')$. Since f is useful for Γ', for every $W \subseteq \mathcal{V}(P')$ we have $|\mathsf{sol}(\mathsf{pj}_W(P'))| \leq |P'|^{cf(W)}$ for a constant c. By Lemma 4.14, the size of P' is at most $O(\phi(|P|))$, where ϕ is the polynomial denoting the time to convert each constraint. Taken together, the two imply that $|P'|^{cf(W)} \leq |P|^{cf(W)ab}$, where a is the degree of ϕ, and b a possible constant factor from $O(\phi(|P|))$. Furthermore, since $\mathsf{sol}(\mathsf{pj}_W(P)) = \mathsf{sol}(\mathsf{pj}_W(P'))$, we have that f is useful for Γ up to the factor cab. ■

Corollary 4.16 *If every constraint in a catalogue Γ can be converted to an equivalent constraint in a catalogue Γ' in polynomial time, then a class of hypergraphs $\mathcal{H}$ that has bounded useful width for Γ' also has bounded useful width for Γ.* □

We can now use the lemmas above to combine partial assignment checking and sparse catalogues into the following.

Theorem 4.17 *If a class of hypergraphs $\mathcal{H}$ has bounded useful width for the catalogue* **Ext** *containing every table constraint, then $\mathcal{H}$ has bounded useful width for any sparse catalogue that allows partial assignment checking.* □

Proof Let $\mathcal{H}$ be given, and let Γ be an arbitrary sparse catalogue that allows partial assignment checking. By Lemma 4.13, every constraint in Γ can be converted into an equivalent table constraint in polynomial time. Since $\mathcal{H}$ has bounded useful width for **Ext**, by Corollary 4.16 $\mathcal{H}$ therefore has bounded useful width for Γ. ■

Corollary 4.18 *If $\mathcal{H}$ is a class of hypergraphs that has bounded useful width for the catalogue* **Ext***, and Γ a sparse catalogue that allows partial assignment checking, then $\mathrm{CSP}(\mathcal{H}, \Gamma)$ is tractable.* □

Proof Let $\mathcal{H}$ and Γ be given. By Theorem 4.17, $\mathcal{H}$ has bounded useful width for Γ, and therefore $\mathrm{CSP}(\mathcal{H}, \Gamma)$ is tractable by Theorem 4.11. ■

In other words, sparse catalogues that allow partial assignment checking allow the same useful width functions as table constraints. It is now tempting to conjecture a dichotomy — that if a class of hypergraphs $\mathcal{H}$ has bounded useful width for a constraint catalogue Γ that is *not* sparse, then $\mathcal{H}$ has bounded

treewidth. Unfortunately, as stated the conjecture is false for trivial reasons: Let every $G \in \mathcal{H}$ have hyperedges of odd arity, while every constraint in Γ has an even number of variables. The class $\mathrm{CSP}(\mathcal{H},\Gamma)$ is then empty, and any $\mathcal{H}$ has bounded useful width in this case.

We could try to patch this up. Observe that we are not interested in classes of hypergraphs and constraint catalogues themselves, but rather in classes of CSP instances with certain hypergraphs and constraints. The hypergraphs and constraints should therefore match: For each hypergraph there should be an instance with that hypergraph, and for every constraint there should be an instance with that constraint. In other words, the class of CSP instances above is really $\mathrm{CSP}(\emptyset,\emptyset)$, and the rest is padding. The definition below captures this idea.

Definition 4.19 A class of hypergraphs $\mathcal{H}$ and a constraint catalogue Γ form a *matching pair* if $\mathcal{H} = \{\mathsf{hyp}(P) \mid P \in \mathrm{CSP}(\mathcal{H},\Gamma)\}$ and $\Gamma = \bigcup\{C \mid \langle V, C\rangle \in \mathrm{CSP}(\mathcal{H},\Gamma)\}$. □

Unfortunately, even with this definition, the conjecture does not hold. The example below makes it clear that having bounded useful width is not a property of individual constraints, but can arise due to their interaction.

Example 4.20 Consider a class of hypergraphs $\mathcal{H}$ containing for every complete graph that graph with a single hyperedge on all vertices, and let Γ have binary equality constraints, as well as constraints that allow all assignments, of every arity. $\mathcal{H}$ and Γ form a matching pair.

However, for every instance $P \in \mathrm{CSP}(\mathcal{H},\Gamma)$ we have $|\mathsf{sol}(\mathsf{pj}_W(P))| \leq D(V) \leq |P|$, so $f(X) = 1$ is a useful width function. Therefore, $\mathcal{H}$ has bounded useful width for Γ even thought Γ is not sparse and $\mathcal{H}$ has unbounded treewidth. □

Furthermore, even though we did not include incidence width as a useful width function, it does possess the key properties that we make use of. As such, the class of SAT instances with bounded incidence width (cf. Theorem 3.11) provides further evidence against our conjecture, as the catalogue of SAT clauses is not sparse.

4.2 Subproblem decompositions

To generalize Theorem 4.17, consider the fact that our definition of a global constraint allows us to view a CSP instance $\langle V, C\rangle$ as a single constraint $e[\delta]$, by letting $\mathcal{V}(\delta) = V$ and $\delta = C$. The algorithm e then checks if an assignment satisfies all constraints. Of course, such a constraint encodes an NP-complete problem, but this is no different from e.g. the EGC constraint (cf. Chapter 2). With this in mind, in this section we are going to investigate what happens if a CSP instance is split up into a set of smaller instances.

Splitting up a CSP instance into smaller instances has previously been considered by Cohen and Green [CG06]. They use a very general framework of

guarded decompositions [CJG08] to define what they call "typed guarded decompositions". This notion allows them to obtain a tractability result for a CSP instance that can be split into smaller instances drawn from known tractable classes. In this section, we are going to prove more general results, and derive their tractable class as a special case.

Definition 4.21 (CSP subproblem) Given two CSP instances $P = \langle V, C\rangle$ and $P' = \langle V', C'\rangle$, we say that P' is a *subproblem* of P if $C' \subseteq C$ and $V' = \mathcal{V}(C)$. □

In other words, a subproblem of a CSP instance is given by a subset of the constraints in that instance. In [CG06], Cohen and Green call a subproblem a *component* of P.

Definition 4.22 (CSP join) Let $Q_1 = \langle V_1, D_1, C_1\rangle$ and $Q_2 = \langle V_2, D_2, C_2\rangle$ be two CSP instances. The *join* of Q_1 and Q_2 is the instance $Q_1 \sqcup Q_2 = \langle V_1 \cup V_2, C_1 \cup C_2\rangle$.

Definition 4.23 (Subproblem decomposition) Let P be a CSP instance. A set S of subproblems of P is a *subproblem decomposition* of P if $\bigsqcup S = P$.

A subproblem decomposition of a CSP instance is *proper* if no element of the decomposition is a subproblem of any other. □

A subproblem decomposition of an instance P, then, is a set of subproblems that together contain all the constraints and variables of P. Note that a constraint may occur in more than one subproblem in a decomposition.

Below, we shall assume that all subproblem decompositions are proper. Since the solutions to a CSP instance can be projected down to solutions for any subproblem, there is no benefit in allowing one subproblem to contain another.

Example 4.24 Let $P = \langle V, C\rangle$ be a CSP instance. A very simple subproblem decomposition of P would be $\{\langle \mathcal{V}(\delta), e[\delta]\rangle \mid e[\delta] \in C\}$, that is, every constraint of P is a separate subproblem. This subproblem decomposition is clearly proper.

For a more complicated example, consider a family of CSP instances on the set of boolean variables $\{x_i, y_i, z_i \mid 1 \leq i \leq n \in \{4, 6, 8, \ldots\}\}$, with the following constraints: An EGC constraint A on $\{x_1, \ldots, x_n\}$ with $K(1) = 4$ and $K(0) = \{0, \ldots, n\}$. A second EGC constraint B, on $\{y_1, \ldots, y_n, z_1, \ldots, z_n\}$ with $K(1) = K(0) = \{n\}$, and binary constraints on each pair $\{x_i, y_i\}$ enforcing equality. A possible subproblem decomposition for an instance from this family would be $\{P, Q\}$, where P contains A as well as the binary constraints, and Q contains the constraint B. This family is depicted in Figure 4.25, with Q marked by a dashed line. □

Viewing subproblems as constraints and a subproblem decomposition S as a CSP instance $\langle \mathcal{V}(\bigsqcup S), S\rangle$, we have $\mathsf{sol}(\langle \mathcal{V}(\bigsqcup S), S\rangle) = \mathsf{sol}(\bigsqcup S)$, since we have lost no information. As such, we will treat S as a CSP instance when it is convenient to simplify notation.

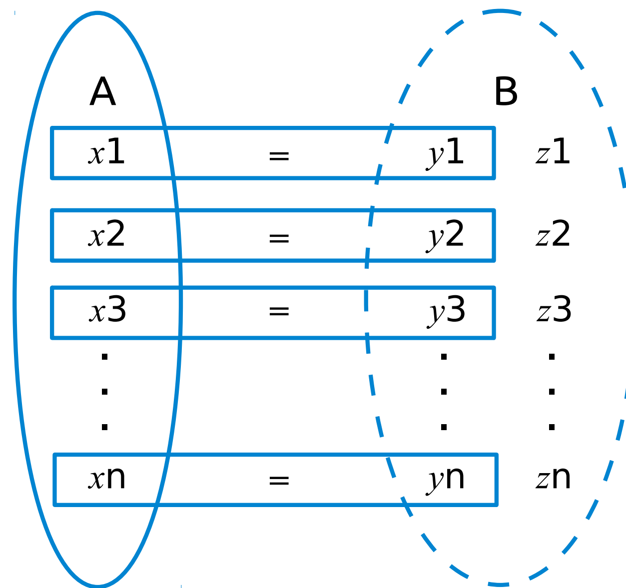

Figure 4.25: Family of instances from Example 4.24, with decomposition.

Using Definition 4.23, we can treat any set of CSP instances S as a subproblem decomposition of the instance $\bigsqcup S$. With that in mind, whenever we say that S is a subproblem decomposition without specifying what it is a decomposition of, we mean that S is a decomposition of the CSP instance $\bigsqcup S$.

Definition 4.26 (CSP instances given by subproblem decompositions) Let $\mathcal{F}$ be a family of subproblem decompositions. We define $\mathrm{CSP}(\mathcal{F})$ to be the class of CSP instances $\{\bigsqcup S \mid S \in \mathcal{F}\}$. □

Definition 4.27 (Hypergraph of a subproblem decomposition) Let S be a subproblem decomposition. The hypergraph of S, denoted $\mathrm{hyp}(S)$, has vertex set $\mathcal{V}(\bigsqcup S)$ and set of hyperedges $\{\mathcal{V}(P) \mid P \in S\}$.

For a family $\mathcal{F}$ of subproblem decompositions, let $\mathrm{hyp}(\mathcal{F}) = \{\mathrm{hyp}(S) \mid S \in \mathcal{F}\}$. □

Since a CSP instance can be seen as a global constraint, Definition 4.10 (partial assignment checking) and Definition 4.12 (sparse catalogue) carry over unchanged. To apply them to a family of subproblem decompositions $\mathcal{F}$, we need only consider the catalogue $\bigcup \mathcal{F}$ in both cases.

One way of interpreting Definition 4.10 for a catalogue of CSP instances is that every instance has been drawn from a tractable class — not necessarily the same one, as long as these classes all allow us to check in polynomial time whether a partial assignment extends to a solution. Most known tractable classes of CSP instances have this property; in particular, all the classes discussed in Section 3.1 have it.

In other words, viewing a CSP instance as a set of smaller instances allows us to combine known tractable classes. As an example, we immediately obtain the theorem below.

Theorem 4.28 *Let $\mathcal{F}$ be a sparse family of subproblem decompositions that allows partial assignment checking. If* $\mathsf{hyp}(\mathcal{F})$ *has bounded useful width for the catalogue* **Ext***, then* $\mathrm{CSP}(\mathcal{F})$ *is tractable.* □

Proof Let Γ be the catalogue containing every subproblem of every decomposition in $\mathcal{F}$, that is, $\Gamma = \bigcup \mathcal{F}$. By Definitions 4.26 and 4.26, we have that $\mathrm{CSP}(\mathcal{F}) = \mathrm{CSP}(\mathsf{hyp}(\mathcal{F}), \Gamma)$. By assumption, $\mathsf{hyp}(\mathcal{F})$ and Γ satisfy the conditions of Corollary 4.18, and the conclusion follows. ■

In order to go beyond this result, observe firstly that by Lemma 4.13, we can convert a CSP instance over a sparse catalogue that allows partial assignment checking into a classic CSP instance in polynomial time, while preserving the hypergraph. It follows that any property on a class of hypergraphs that ensures tractability for classic CSP will do here — we are not, in fact, restricted to just bounded useful width, even in the theorem above. This means that we can incorporate incidence width into our results, even though it is not defined as a width function. Furthermore, in [Mar09a] Marx makes the observation that fractional hypertree width is the best possible notion of useful width for table constraints.

Theorem 4.29 ([Mar09a], adapted) *Let G be a hypergraph. If a monotone width function f is a useful width function on G for* **Ext***, then the f-width of G is greater than or equal to the fractional hypertree width of G.* □

As table constraints have the largest representation size in terms of their satisfying assignments, Theorem 4.29 also applies to every global constraint type we have discussed so far.

A second way that we will improve our results is by examining the requirement of Definition 4.12, that every constraint (or subproblem) has few solutions, in more detail. Below, we are going to show that this requirement is too strict and relax it, then amplify the results to go beyond useful width.

Definition 4.30 (Intersection vertices) Let S be a subproblem decomposition. The set of *intersection vertices* of any subproblem $P \in S$ is $\mathsf{iv}(P) = \bigcup\{\mathcal{V}(P) \cap \mathcal{V}(Q) \mid Q \in S - \{P\}\}$. □

Definition 4.31 (Table constraint induced by a subproblem) Let S be a subproblem decomposition. For every $T \in S$, let μ^* be the assignment to $\mathcal{V}(T) - \mathsf{iv}(T)$ that assigns a special value $*$ to every variable. The *table constraint induced by* T is $\mathsf{ic}(T) = e[\delta]$, where $\mathcal{V}(\delta) = \mathcal{V}(T)$, and δ contains for every assignment $\theta \in \mathsf{sol}(\mathsf{pj}_{\mathsf{iv}(T)}(S))$ the assignment $\theta \oplus \mu^*$. □

If every subproblem in a decomposition S allows partial assignment checking, then building $\mathsf{ic}(T)$ for any $T \in S$ can be done in polynomial time when $|\mathsf{sol}(\mathsf{pj}_{\mathsf{iv}(T)}(S))|$ is itself polynomial in the size of $\bigsqcup S$ for every subset of $\mathsf{iv}(T)$. To do so, we can invoke Algorithm 1 on $\mathsf{pj}_{\mathsf{iv}(T)}(S)$.

Definition 4.32 (Sparse intersections) A family of subproblem decompositions $\mathcal{F}$ *has sparse intersections* if there exists a constant c such that for every $T \in S \in \mathcal{F}$ and $X \subseteq \mathsf{iv}(T)$, $|\mathsf{sol}(\mathsf{pj}_X(S))| \leq |\bigsqcup S|^c$. □

Observe that this definition relaxes Definition 4.12 — a family $\mathcal{F}$ where every $T \in S \in \mathcal{F}$ has few solutions satisfies Definition 4.32, but the reverse is not true. Checking if an intersection is sparse, however, remains hard in general, since the projection of a CSP instance is itself a CSP instance (cf. the discussion after Definition 4.12).

Example 4.33 Recall the second family of subproblem decompositions in Example 4.24. For a decomposition $S = \{P, Q\}$ from this family, the set of intersection vertices for both subproblems is $\{y_1, \ldots, y_n\}$. Furthermore, the EGC constraint A requires that there are exactly 4 variables assigned 1 among $\{x_1, \ldots, x_n\}$, so there are $\binom{n}{4}$ satisfying assignments for this constraint. The equality constraints ensure that this is the number of solutions to the whole subproblem P, so for every $X \subseteq \{y_1, \ldots, y_n\}$ we have that $|\mathsf{sol}(\mathsf{pj}_X(S))| \leq \binom{n}{4}$. Therefore, this family of subproblem decompositions has sparse intersections. □

Another example where Definition 4.32 would be satisfied is when every set of intersection vertices is covered by a fixed number of table constraints. In this case, the number of possible solutions is bounded by the size of the join of all these constraints (cf. Section 3.1.1). This is the condition used by Cohen and Green [CG06]. In other words, we can derive the main theorem in as a special case of ours. We, however, do not need to cover the intersection vertices of *every* subproblem by a fixed number of table constraints.

Theorem 4.34 *Let $\mathcal{F}$ be a family of subproblem decompositions that allows partial assignment checking. If $\mathcal{F}$ has sparse intersections, then we can in polynomial time reduce any subproblem decomposition $S \in \mathcal{F}$ to a classic CSP instance P with* $\mathsf{hyp}(P) = \mathsf{hyp}(S)$*, such that P has a solution if and only if S does.* □

Proof Let S be a subproblem decomposition from a family $\mathcal{F}$. For each $T \in S$, P will contain the table constraint $\mathsf{ic}(T)$ from Definition 4.31. Since $\mathcal{F}$ allows

partial assignment checking and has sparse intersections, computing $\mathsf{ic}(T)$ can be done in polynomial time by invoking Algorithm 1 on $\mathsf{pj}_{\mathsf{iv}(T)}(S)$.

It is clear that $\mathsf{hyp}(P) = \mathsf{hyp}(S)$. All that is left to show is that P has a solution if and only if S does. Let θ be a solution to S. For every $T \in S$, $\theta|_{\mathsf{iv}(T)} \in \mathsf{pj}_{\mathsf{iv}(T)}(S)$ by Definitions 4.1 and 4.30, so the assignment μ that assigns the value $\theta(v)$ to each $v \in \bigcup_{T \in S} \mathsf{iv}(T)$, and $*$ to every other variable is a solution to P.

In the other direction, if θ is a solution to P, then θ satisfies $\mathsf{ic}(T)$ for every $T \in S$. By Definition 4.31, this means that $\theta|_{\mathsf{iv}(T)} \in \mathsf{sol}(\mathsf{pj}_{\mathsf{iv}(T)}(S))$, and by Definition 4.1, there exists an assignment μ^T with $\mu^T|_{\mathsf{iv}(T)} = \theta|_{\mathsf{iv}(T)}$ that satisfies T. By Definition 4.30, the variables not in $\mathsf{iv}(T)$ do not occur in any other subproblem from S, so we can combine all the assignments μ^T to form a solution μ to S such that for $T \in S$ and $v \in \mathcal{V}(T)$ we have $\mu(v) = \mu^T(v)$. ∎

From Theorem 4.34, we get tractable and fixed-parameter tractable classes of CSP instances with global constraints.

Corollary 4.35 *Let $\mathcal{F}$ be a family of subproblem decompositions that allows partial assignment checking and has sparse intersections. If* CSP($\mathsf{hyp}(\mathcal{F})$, **Ext**) *is tractable or in* FPT*, then so is* CSP($\mathcal{F}$). □

Proof Let $\mathcal{F}$ be given. By Theorem 4.34, we can reduce any $S \in \mathcal{F}$ to an instance $P \in$ CSP($\mathsf{hyp}(\mathcal{F})$, **Ext**) in polynomial time. Since P has a solution if and only if S does, tractability of CSP($\mathsf{hyp}(\mathcal{F})$, **Ext**) implies the same for CSP($\mathcal{F}$). ∎

Example 4.36 The family of subproblem decompositions in Example 4.24 has sparse intersections (cf. Example 4.33). For any subproblem decomposition $S = \{P, Q\}$ from this family, it is easy to see that both subproblems allow partial assignment checking; P due to the argument in Example 4.33, and Q because the cardinality sets of EGC constraint B are intervals (cf. Section 3.2.1). Furthermore, the hypergraph of S has hypertree width 2, and therefore the family of CSP instances in Example 4.24 is tractable by Corollary 4.35. □

An example of a class of instances CSP($\mathcal{H}$, **Ext**) that is fixed-parameter tractable is given by $\mathcal{H}$ having bounded submodular width (cf. Theorem 3.28).

4.2.1 An extended example

Recall the connected graph partition problem (CGP, Problem 2.1): Given a connected graph G, as well as natural numbers α and β, can the vertices of G be partitioned into bags of size at most α, such that no more than β edges are broken. We looked at an encoding of the CGP using global constraints in Chapter 2 (cf. Example 2.14), where we also gave a proof sketch that this problem is tractable for fixed β. Below, we will illustrate Corollary 4.35 by using it to formally derive this result. To simplify the analysis, we assume without loss of

generality that $\alpha < |V|$, which means that any solution has at least one broken edge.

Given an instance of the CGP encoded as in Example 2.14, let S be the subproblem decomposition $\{P, Q\}$ with P containing the single constraint C^α and Q every other constraint. Such a decomposition for the CGP on the graph C_5, that is, a simple cycle on five vertices, is shown in Figure 4.37. The dashed line represents the constraint C^α in subproblem P, while solid lines mark the constraints in subproblem Q.

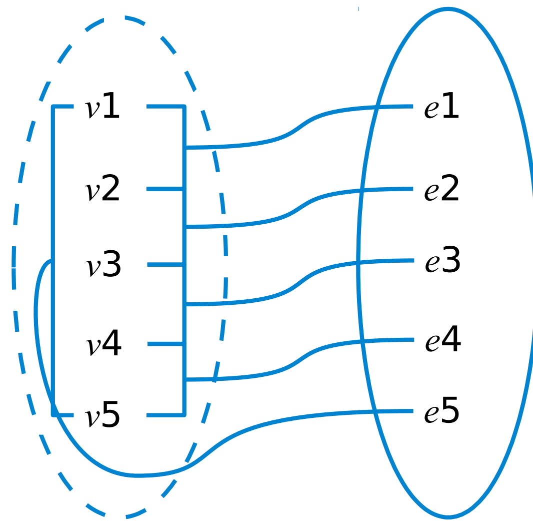

Figure 4.37: Subproblem decomposition of the CGP on the graph C_5.

We will show that if β is fixed, then Q allows partial assignment checking, and has only a polynomial number of solutions. The latter implies that $|\mathsf{sol}(\mathsf{pj}_{\mathsf{iv}(P)}(S))| = |\mathsf{sol}(\mathsf{pj}_{\mathsf{iv}(Q)}(S))|$ is polynomial in the size of $\bigsqcup S$ for every subset of $\mathsf{iv}(P) = \mathsf{iv}(Q)$. That P allows partial assignment checking follows from Theorem 3.42, since the cardinality sets of C^α are intervals.

First, we show that the number of solutions to Q is limited. Since C^β limits the number of ones in any solution to β or fewer, the number of satisfying assignments to this constraint is the number of ways to choose up to β variables to be assigned one. This is bounded by $\sum_{i=1}^{\beta} \binom{|E|}{i} \leq (|E|+1)^\beta$, and so we can

generate them all in polynomial time. Now, let θ be such a solution. How many solutions to Q contain θ? Well, every constraint on $\{u, v, e\}$ with $e = 1$ allows at most $|V|^2$ assignments, and there are at most β such constraints. So far we therefore have at most $(|E| + 1)^\beta \times |V|^{2\beta}$ assignments.

On the other hand, a constraint on $\{u, v, e\}$ with $e = 0$ requires $u = v$. Consider the graph G_0 containing for every constraint on $\{u, v, e\}$ with $e = 0$ the vertices u and v as well as the edge $\{u, v\}$. Since the original graph was connected, every connected component of G_0 contains at least one vertex which is in the scope of some constraint with $e = 1$. Therefore, since equality is transitive, each connected component of G_0 allows at most one assignment for each of the $(|E| + 1)^\beta \times |V|^{2\beta}$ assignments to the other variables of Q. We therefore get a total bound of $(|E| + 1)^\beta \times |V|^{2\beta}$ on the total number of solutions to the subproblem Q.

The above also gives us an algorithm that performs partial assignment checking on Q in polynomial time, simply by checking whether a given assignment is contained in any of the solutions.

The hypergraph of our subproblem decomposition has two hyperedges, so the hypertree width of this hypergraph is clearly one. Therefore, we may apply Corollary 4.35 to obtain tractability when β is fixed. As this problem is NP-complete for fixed $\alpha \geq 3$ [GJ79, p. 209], β is a natural parameter to try and use.

This tractability result relies on several properties of the problem, which the framework of subproblem decompositions helps unify. First, we exploited the fact that fixed β guarantees few solutions to one of the EGC constraints, and hence that the CSP instance formed by that constraint as well as the ternary constraints has few solutions. Second, we used the fact that the other EGC constraint allows partial assignment checking, and combining the two into a subproblem decomposition then gave us tractability.

As it happens, in this problem we can drop the requirement of partial assignment checking for subproblem P, since its variables are entirely covered by subproblem Q, which has polynomially many solutions for fixed β. We elucidate this point in Section 4.3, where we turn this observation into a general result.

4.2.2 Variations on a theme: WCSP

Having few solutions in key parts of a CSP instance has turned out to be a property we can exploit to obtain tractability. In this section, we are going to apply this property to an extension of the CSP framework called weighted CSP instances [GGS09, dGSV06], where every constraint assigns a cost to every satisfying assignment, and we would like to find a solution with smallest cost. This type of CSP is itself a special case of the more general valued CSP framework [SFV95, Živ09], where every constraint has a function that assigns a cost to every possible assignment for the variables of that constraint.

Definition 4.38 (Weighted constraint) A *weighted global constraint* $e[\delta]$ is a global constraint that assigns to each $\theta \in e[\delta]$ a value $\mathsf{cost}(e[\delta], \theta)$ from $\mathbb{Q}$. □

Definition 4.39 (WCSP instance) A *WCSP instance* is a pair $P = \langle V, C \rangle$, where V is a set of variables and C a set of weighted constraints. An assignment is a solution to P if it satisfies every constraint in C, and we denote the set of all solutions to P by $\mathsf{sol}(P)$.

For every solution θ to P we define $\mathsf{cost}(P, \theta) = \sum_{e[\delta] \in C} \mathsf{cost}(e[\delta], \theta|_{\mathcal{V}(\delta)})$. An assignment θ is an *optimal* solution to P if and only if it is a solution to P *with the smallest cost*, i.e. $\mathsf{cost}(P, \theta) = \min(\{\mathsf{cost}(P, \theta') \mid \theta' \in \mathsf{sol}(P)\})$. □

Definition 4.40 (WCSP decision problem) Given a WCSP instance P and $k \in \mathbb{Q}$, the *WCSP decision problem* is to decide whether P has a solution θ with $\mathsf{cost}(P, \theta) \leq k$. □

As for CSP instances, a classic WCSP instance is one where all constraints are table global constraints. Since we are free to ignore the costs a weighted constraint puts on assignments and treat it as an "ordinary" constraint, definitions of subproblems and subproblem decompositions carry over unchanged. Note that since the WCSP decision problem in clearly in NP, we can view a WCSP instance as a *weighted* global constraint. Therefore, Definition 4.10 will now be subtly different.

Definition 4.41 (Weighted part. assignment checking) A weighted constraint catalogue Γ allows *partial assignment checking* if for any weighted constraint $e[\delta] \in \Gamma$ we can decide in polynomial time, given an assignment θ to a set of variables $W \subseteq \mathcal{V}(\delta)$ and $k \in \mathbb{Q}$, whether θ is contained in an assignment that satisfies $e[\delta]$ and has cost at most k, i.e. whether there exists $\mu \in e[\delta]$ such that $\theta = \mu|_W$ and $\mathsf{cost}(e[\delta], \mu) \leq k$. □

In other words, given a partial assignment we need to be able to solve the WCSP decision problem for our constraint in polynomial time. As for the projection of a constraint, we need to alter Definition 4.1 to take costs into account.

Definition 4.42 (Weighted constraint projection) Let $e[\delta]$ be a weighted constraint. The *projection* of $e[\delta]$ onto a set of variables $X \subseteq \mathcal{V}(\delta)$ is the constraint $\mathsf{pj}_X(e[\delta])$ such that $\mu \in \mathsf{pj}_X(e[\delta])$ if and only if there exists $\theta \in e[\delta]$ with $\theta|_X = \mu$. The cost of an assignment $\theta \in \mathsf{pj}_X(e[\delta])$ is $\mathsf{cost}(\mathsf{pj}_X(e[\delta]), \theta) = \min(\{\mathsf{cost}(e[\delta], \mu) \mid \mu \in e[\delta] \text{ and } \mu|_X = \theta\})$.

For a CSP instance $P = \langle V, C \rangle$ and $X \subseteq V$ we define $\mathsf{pj}_X(P) = \langle X, C' \rangle$, where C' is the least set containing for every $e[\delta] \in C$ such that $X \cap \mathcal{V}(\delta) \neq \emptyset$ the constraint $\mathsf{pj}_{X \cap \mathcal{V}(\delta)}(e[\delta])$. □

Definition 4.43 (Weighted table constraint induced by a subproblem) Let S be a subproblem decomposition. For every $T \in S$, let μ^* be the assignment

to $\mathcal{V}(T) - \mathsf{iv}(T)$ that assigns a special value $*$ to every variable. The *weighted table constraint induced by T* is $\mathsf{ic}(T) = e[\delta]$, where $\mathcal{V}(\delta) = \mathcal{V}(T)$, and δ contains for every assignment $\theta \in \mathsf{sol}(\mathsf{pj}_{\mathsf{iv}(T)}(S))$ the assignment $\theta \oplus \mu^*$ with $\mathsf{cost}(\mathsf{ic}(T), \theta \oplus \mu^*) = \mathsf{cost}(\mathsf{pj}_{\mathsf{iv}(T)}(T), \theta)$. □

Since the variables of a subproblem $T \in S$ not in $\mathsf{iv}(T)$ occur only in T itself, if we have a solution to $\mathsf{pj}_{\mathsf{iv}(T)}(S)$, it doesn't matter what solution to T we extend it to. We should therefore pick the one that has the smallest cost, and that cost is precisely $\mathsf{cost}(\mathsf{pj}_{\mathsf{iv}(T)}(T), \theta)$ by Definition 4.42. The same as for CSP instances, if every subproblem in a weighted decomposition S allows weighted partial assignment checking, building $\mathsf{ic}(T)$ for any $T \in S$ can be done in polynomial time when $|\mathsf{sol}(\mathsf{pj}_{\mathsf{iv}(T)}(S))|$ is polynomial in the size of $\bigsqcup S$ for every subset of $\mathsf{iv}(T)$, again by using Algorithm 1. Since the definition of sparse intersections (Definition 4.32) carries over unchanged, we are ready to prove the following analogue of Theorem 4.34 for weighted subproblem decompositions.

Theorem 4.44 *Let $\mathcal{F}$ be a family of weighted subproblem decompositions that allows partial assignment checking. If $\mathcal{F}$ has sparse intersections, then we can in polynomial time reduce any weighted subproblem decomposition $S \in \mathcal{F}$ to a classic weighted CSP instance P with $\mathsf{hyp}(P) = \mathsf{hyp}(S)$, such that P has a solution with cost at most $k \in \mathbb{N}$ if and only if S does.* □

Proof Let S be a subproblem decomposition from $\mathcal{F}$. For each $T \in S$, P will contain the table constraint $\mathsf{ic}(T)$ from Definition 4.31. Since $\mathcal{F}$ allows partial assignment checking and has sparse intersections, computing $\mathsf{ic}(T)$ can be done in polynomial time by invoking Algorithm 1 on $\mathsf{pj}_{\mathsf{iv}(T)}(S)$.

It is clear that $\mathsf{hyp}(P) = \mathsf{hyp}(S)$. All that is left to show is that P has a solution with cost at most $k \in \mathbb{N}$ if and only if S does. Let θ be a solution to S. For every $T \in S$, $\theta|_{\mathsf{iv}(T)} \in \mathsf{pj}_{\mathsf{iv}(T)}(S)$ by Definitions 4.30 and 4.42, so the assignment μ that assigns the value $\theta(v)$ to each $v \in \bigcup_{T \in S} \mathsf{iv}(T)$, and $*$ to every other variable is a solution to P. Furthermore, for every $T \in S$ we have by Definition 4.43 that $\mathsf{cost}(\mathsf{ic}(T), \mu|_{\mathcal{V}(T)}) = \mathsf{cost}(\mathsf{pj}_{\mathsf{iv}(T)}(T), \mu|_{\mathsf{iv}(T)})$, so by Definition 4.42 $\mathsf{cost}(\mathsf{ic}(T), \mu|_{\mathcal{V}(T)}) \leq \mathsf{cost}(T, \theta|_{\mathcal{V}(T)})$ and therefore $\mathsf{cost}(P, \mu) \leq \mathsf{cost}(S, \theta)$.

In the other direction, if θ is a solution to P, then θ satisfies $\mathsf{ic}(T)$ for every $T \in S$. By Definition 4.43, this means that $\theta|_{\mathsf{iv}(T)} \in \mathsf{sol}(\mathsf{pj}_{\mathsf{iv}(T)}(S))$, and by Definition 4.42, there exists an assignment μ^T with $\mu^T|_{\mathsf{iv}(T)} = \theta|_{\mathsf{iv}(T)}$ that satisfies T, such that $\mathsf{cost}(\mathsf{ic}(T), \theta|_{\mathcal{V}(T)}) = \mathsf{cost}(T, \mu^T)$. By Definition 4.30, the variables not in $\mathsf{iv}(T)$ do not occur in any other subproblem from S, so we can combine all the assignments μ^T to form a solution μ to S such that for $T \in S$ and $v \in \mathcal{V}(T)$ we have $\mu(v) = \mu^T(v)$, with $\mathsf{cost}(P, \theta) = \mathsf{cost}(S, \mu)$. ■

As before, for a family of weighted subproblem decompositions $\mathcal{F}$ we define $\mathrm{WCSP}(\mathcal{F}) = \{\bigsqcup S \mid S \in \mathcal{F}\}$, and for a class of hypergraphs $\mathcal{H}$ we let $\mathrm{WCSP}(\mathcal{H}, \mathbf{Ext})$ be the class of classic WCSP instances whose hypergraphs are

in $\mathcal{H}$. With that in mind, we can use Theorem 4.44 to obtain new tractable and fixed-parameter tractable classes of weighted CSP instances with global constraints.

Corollary 4.45 *Let $\mathcal{F}$ be a family of weighted subproblem decompositions that allows partial assignment checking and has sparse intersections. If* $\mathsf{WCSP}(\mathsf{hyp}(\mathcal{F}), \mathbf{Ext})$ *is tractable or in* FPT*, then so is* $\mathsf{WCSP}(\mathcal{F})$. □

Proof Let $\mathcal{F}$ be given. By Theorem 4.44, we can reduce any $S \in \mathcal{F}$ to an instance $P \in \mathsf{WCSP}(\mathsf{hyp}(\mathcal{F}), \mathbf{Ext})$ in polynomial time. Since P has a solution with cost k if and only if S does, tractability of $\mathsf{WCSP}(\mathsf{hyp}(\mathcal{F}), \mathbf{Ext})$ implies the same for $\mathsf{WCSP}(\mathcal{F})$. ■

4.3 Back doors

If a family of weighted subproblem decompositions is not known to allow partial assignment checking, we may still be able to obtain tractability in some cases.

A (strong) back door set [GS12, WGS03] is a set of variables in an instance that, when assigned, make the instance easy to solve.

Definition 4.46 (Back door) Let Γ be a global constraint catalogue. A *back door* for a constraint $e[\delta] \in \Gamma$ is any set of variables $W \subseteq \mathcal{V}(\delta)$ (called a back door set) such that we can decide in polynomial time whether a given assignment θ with $\mathcal{V}(\theta) \supseteq W$ is contained in an assignment that satisfies $e[\delta]$, i.e. whether there exists $\mu \in e[\delta]$ such that $\mu|_{\mathcal{V}(\theta)} = \theta$. □

Trivially, for every constraint $e[\delta]$ the set of variables $\mathcal{V}(\delta)$ is a back door set, since we can always check in polynomial time if an assignment to $\mathcal{V}(\delta)$ satisfies the constraint. While finding back doors is difficult in general, for some problems, such as SAT, there are known classes of instances that allow efficient detection of back doors [GS12].

The key point about back doors is that given a catalogue Γ, adding to each $e[\delta] \in \Gamma$ with back door set W an arbitrary table constraint with scope W produces a catalogue Γ' that allows partial assignment checking. Adding a table constraint here means to add a set of assignments Θ to the description, and modify the algorithm e to only accept an assignment if it contains a member of Θ in addition to previous requirements. Furthermore, as long as $\Theta \supseteq \pi_W(\mathsf{sol}(e[\delta]))$, the new constraint has exactly the same satisfying assignments. This point leads to the following definition.

Definition 4.47 (Sparse back door cover) Let Γ_{PAC} be a catalogue that allows partial assignment checking and Γ_{BD} a catalogue. A family of subproblem decompositions $\mathcal{F}$ over $\Gamma_{PAC} \cup \Gamma_{BD}$ has *sparse back door cover* if there exists a constant c such that for every $T \in S \cap \Gamma_{BD}$ there exists a back door set W with $|\mathsf{sol}(\mathsf{pj}_X(S \cap \Gamma_{PAC}))| \leq |\bigsqcup S|^c$ for every $X \subseteq W$. □

Sparse back door cover means that for each subproblem that isn't from a catalogue that allows partial assignment checking, we can get a set of assignments for its back door set using Algorithm 1, and so turn this subproblem into one that does allow partial assignment checking.

Theorem 4.48 *If a family of subproblem decompositions $\mathcal{F}$ has sparse back door cover, then we can in polynomial time reduce any subproblem decomposition $S \in \mathcal{F}$ to a subproblem decomposition S' such that* $\mathsf{hyp}(S) = \mathsf{hyp}(S')$ *and* $\mathsf{sol}(S) = \mathsf{sol}(S')$. *Furthermore, the family $\{S' \mid S \in \mathcal{F}\}$ allows partial assignment checking.* $\square$

Proof Let $S \in \mathcal{F}$. We construct S' by adding to every $T \in S \cap \Gamma_{BD}$ with back door set W a table constraint $e[\delta]$ with $\mathcal{V}(\delta) = W$ containing $\mathsf{sol}(\mathsf{pj}_W(S \cap \Gamma_{PAC}))$, which we can obtain using Algorithm 1. By Definition 4.47, we have for every $X \subseteq W$ that $|\mathsf{sol}(\mathsf{pj}_W(S \cap \Gamma_{PAC}))| \leq |\bigsqcup S|^c$, so Algorithm 1 takes polynomial time since Γ_{PAC} allows partial assignment checking.

It is clear that $\mathsf{hyp}(S') = \mathsf{hyp}(S)$, and since $\mathsf{sol}(\mathsf{pj}_W(S \cap \Gamma_{PAC})) \supseteq \pi_W(\mathsf{sol}(S))$, the set of solutions stays the same, i.e. $\mathsf{sol}(S') = \mathsf{sol}(S)$. Finally, since the catalogue $\Gamma' = \bigcup\{S' - (S \cap \Gamma_{PAC}) \mid S \in \mathcal{F}\}$ allows partial assignment checking, so does $\Gamma' \cup \Gamma_{PAC}$, and hence $\mathcal{F}$ is a family that allows partial assignment checking. $\blacksquare$

One consequence of Theorem 4.48 is that we can apply Theorem 4.34 to a subproblem decomposition that contains an NP-hard subproblem when the variables of that subproblem are covered by the variables of another subproblem that has few solutions.

As a concrete example of this, consider again the subproblem decomposition for the CGP that we gave in Section 4.2.1. The variables of subproblem A are entirely covered by the variables of subproblems B, that is, $\mathcal{V}(A) \subset \mathcal{V}(B)$. As the entire set of variables of a subproblem is a back door set for it, and subproblem B has few solutions, these subproblem decompositions have sparse back door cover. As such, the constraints in subproblem A can, in fact, be arbitrary without affecting the tractability of that problem. In particular, the requirement that A allows partial assignment checking can be dropped.

As the reduction in Theorem 4.48 preserves all solutions, it also applies equally to WCSP instances and can be used together with Theorem 4.44.

4.4 Summary

In this chapter we have seen how structural restrictions such as hypertree width lead to tractability of classic CSP instances by bounding the number of solutions a CSP instance has when projected down to a subset of its variables. However, as it turns out, a CSP instance that has this property does not necessarily become tractable under structural restrictions such as bounded hypertree width, since global constraints with few solutions may encode difficult problems. To achieve tractability, we need to combine the property of having few solutions with the

additional requirement that the global constraints of our instance allow partial assignment checking.

Furthermore, treating instances as single constraints allows us view an instance as made up of smaller instances. If all the smaller instances are drawn from known tractable classes, we can once again use the property of having few solutions to obtain new tractable classes, strictly generalizing the first set of results at the cost of heavier machinery.

Finally, we applied the techniques discussed above to weighted CSP instances, and showed that there is also a set of tractable classes to be had there.

To end this chapter, consider the following family of CSP instances. The instances in it do not have subproblem decompositions that satisfy Corollary 4.35, even thought they are all easy to solve.

Example 4.49 Consider a family of CSP instances on three sets of boolean variables $\{x_1, \ldots, x_n\}$, $\{y_1, \ldots, y_n\}$, and $\{z_1, \ldots, z_n\}$ where for each n, the instance has two negative constraints (cf. Example 2.13), one on $\{x_1, \ldots, x_n, y_1, \ldots, y_n\}$ and the other on $\{y_1, \ldots, y_n, z_1, \ldots, z_n\}$. □

To solve such an instance, we need only project the forbidden assignments of each constraint down to $\{y_1, \ldots, y_n\}$ and check if any assignment is missing. However, since negative constraints can have exponentially many allowed assignments (in the size of their description), the conditions we used in this chapter fail. In the next chapter, we look at what can be done in such cases.

Chapter 5

Tractability via equivalence classes of assignments

Where we find a way to split many solutions into few equivalence classes, and discuss when it is a good idea to do so.

In Chapter 4, we considered constraints with few satisfying assignments, and extended this property to CSP instances with few solutions in key places. In this chapter, we explore the second question posed at the end of Chapter 3 — what structural restrictions between treewidth and query width yield tractability for nontrivial classes of global constraints?

To answer this question, we explore conditions that allow us to treat sets of assignments as equivalent with respect to a set of constraints. Doing so transforms a set of assignments into a smaller set of equivalence classes of assignments, and allows us to define a structural restriction between treewidth and query width that yields tractability for classes of constraints with sufficiently small sets of equivalence classes.

As a running example, consider the following family of constraint problems involving clauses and cardinality constraints of unbounded arity.

Example 5.1 Consider a family of constraint problems on a set of Boolean variables $\{x_1, x_2, \ldots, x_{3n}\}$ (where $n = 2, 3, 4, \ldots$), with the following five constraints:

- C_1 is the binary clause $x_1 \vee x_{2n+1}$;
- C_2 is an EGC constraint on $\{x_1, x_2, \ldots, x_n\}$ specifying that exactly one of these variables takes the value 1;
- C_3 is an EGC constraint on $\{x_{2n+1}, x_{2n+2}, \ldots, x_{3n}\}$ specifying that exactly one of these variables takes the value 1;
- C_4 is an EGC constraint on $\{x_2, x_3, \ldots, x_{3n}\} - \{x_{2n+1}\}$ specifying that exactly $n+1$ of these variables take the value 1;
- C_5 is the clause $\bigvee_{i=1}^{n-1} \neg x_{n+i} \vee x_{n+i+1}$.

This problem is illustrated in Figure 5.2. □

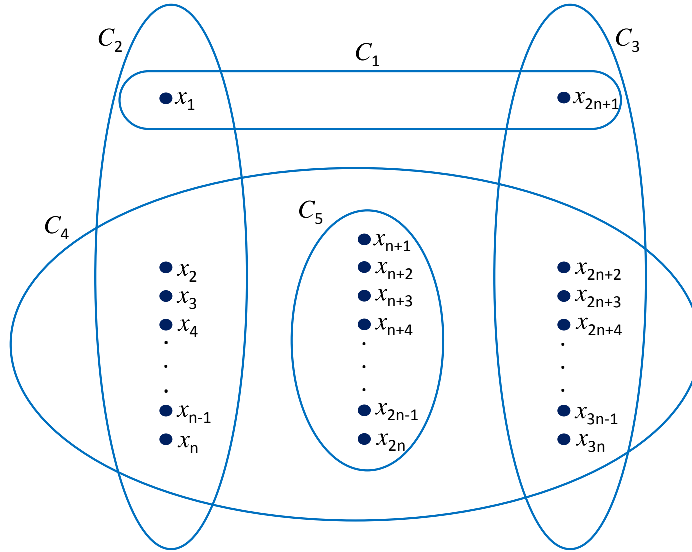

Figure 5.2: The structure of the constraint problems in Example 5.1

This family of problems is not included in any previously known tractable class. In particular, it does not admit a subproblem decomposition that satisfies Theorem 4.34. To see this, observe that since the constraints C_4 and C_5 both have exponentially many satisfying assignments on the variables they have in common, treating them as two subproblems means that we have a non-sparse intersection. On the other hand, the subproblem containing both constraints does not allow partial assignment checking — the flow-based algorithm for cardinality constraints with interval cardinality sets (cf. Section 3.2.1) fails because of the clause in C_5.

5.1 Cooperating constraints

Whenever constraint scopes overlap, we may ask whether the possible assignments to the variables in the overlap are essentially different. It may be that some assignments extend to precisely the same satisfying assignments in each of the overlapping constraints. If so, we may as well identify such assignments. This observation leads to the following definitions.

Definition 5.3 (Assignment extension) Let $e[\delta]$ be a global constraint, and $X \subseteq \mathcal{V}(\delta)$. For every assignment μ of X, let $\text{ext}(\mu, e[\delta]) = \pi_{\mathcal{V}(\delta)-X}(\{\theta \in e[\delta] \mid \theta|_X = \mu\})$. □

In other words, for any assignment μ of X, the set $\text{ext}(\mu, e[\delta])$ is the set of assignments of $\mathcal{V}(\delta) - X$ that extend μ to a satisfying assignment for $e[\delta]$; i.e., those assignments θ for which $\mu \oplus \theta \in e[\delta]$.

Definition 5.4 (Extension equivalence) Let $e[\delta]$ be a global constraint, and $X \subseteq \mathcal{V}(\delta)$. We say that two assignments θ_1, θ_2 to X are *extension equivalent* on X with respect to $e[\delta]$ if $\mathsf{ext}(\theta_1, e[\delta]) = \mathsf{ext}(\theta_2, e[\delta])$. We denote this equivalence relation by $\mathsf{equiv}[e[\delta], X]$; that is, $\mathsf{equiv}[e[\delta], X](\theta_1, \theta_2)$ holds if and only if θ_1 and θ_2 are extension equivalent on X with respect to $e[\delta]$. □

In other words, two assignments to some subset of the variables of a constraint $e[\delta]$ are extension equivalent if every assignment to the rest of the variables combines with both of them to give either two assignments that satisfy $e[\delta]$, or two that falsify it. Below, we give some examples to illustrate this rather technical notion.

Example 5.5 Consider the special case of extension equivalence with respect to a clause (cf. Example 2.12).

Given any clause $e[\delta]$, and any non-empty set of variables $X \subseteq \mathcal{V}(\delta)$, any assignment to X will either satisfy one of the corresponding literals specified by δ, or else falsify all of them. If it satisfies at least one of them, then any extension will satisfy the clause, so all such assignments are extension equivalent. If it falsifies all of them, then an extension will satisfy the clauses if and only if it satifies one of the other literals. Hence the equivalence relation $\mathsf{equiv}[e[\delta], X]$ has precisely 2 equivalence classes, one containing the single assignment that falsifies all the literals corresponding to X, and one containing all other assignments. □

Example 5.6 Consider the special case of extension equivalence on the whole scope of a constraint. We will show that in this case, regardless of the constraint type, we always have at most two equivalence classes.

Given any global constraint $e[\delta]$, and assignments θ_1, θ_2 to $\mathcal{V}(\delta)$, we have that $\mathsf{equiv}[e[\delta], \mathcal{V}(\delta)](\theta_1, \theta_2)$ if and only if either $\theta_1, \theta_2 \in e[\delta]$ or $\theta_1, \theta_2 \notin e[\delta]$. Consequently, $\mathsf{equiv}[e[\delta], \mathcal{V}(\delta)]$ partitions the assignments to $\mathcal{V}(\delta)$ into at most two equivalence classes, one containing all the assignments that satisfy $e[\delta]$, and the other one all those that do not. □

For a set S of global constraints, we will write $\mathsf{iv}(S)$ for the set of variables common to all of their scopes, that is, $\mathsf{iv}(S) = \bigcap_{e[\delta] \in S} \mathcal{V}(\delta)$.

Definition 5.7 (Join) For any set S of global constraints, we define the *join* of S, denoted $\mathsf{join}(S)$, to be a global constraint $e'[\delta']$ with $\mathcal{V}(\delta') = \bigcup_{e[\delta] \in S} \mathcal{V}(\delta)$ such that for any assignment θ to $\mathcal{V}(\delta')$, we have $\theta \in e'[\delta']$ if and only if for every $e[\delta] \in S$ we have $\theta|_{\mathcal{V}(\delta)} \in e[\delta]$. □

The join of a set of global constraints may have no simple compact description, and computing its extension may be computationally expensive. However,

we introduce this construct simply in order to describe the combined effect of a set of global constraints in terms of a single constraint.

Example 5.8 Let $V = \{v_1, \ldots, v_n\}$, for some $n \geq 3$, be a set of variables with $D(v_i) = \{a, b, c\}$, and let $S = \{e_1[\delta_1], e_2[\delta_2]\}$ be a set of two global constraints as defined below:

- $e_1[\delta_1]$ is a table constraint with $\mathcal{V}(\delta_1) = \{v_1, \ldots, v_{n-1}\}$ which enforces *equality*, i.e., $\delta_1 = \{\theta_a, \theta_b, \theta_c\}$, where for each $x \in D(V)$ and $v \in \mathcal{V}(\delta_1)$, $\theta_x(v) = x$.
- $e_2[\delta_2]$ is a negative constraint with $\mathcal{V}(\delta_2) = \{v_2, \ldots, v_n\}$ which enforces a *not-all-equal* condition, i.e., $\delta_2 = \{\theta_a, \theta_b, \theta_c\}$, where for each $x \in D(V)$ and $v \in \mathcal{V}(\delta_2)$, $\theta_x(v) = x$.

We will use substitution notation to write assignments explicitly; thus, an assignment of $\{v, w\}$ that assigns a to both variables is written $\{v/a, w/a\}$.

We have that $\mathsf{iv}(S) = \{v_2, \ldots, v_{n-1}\}$. The equivalence classes of assignments to $\mathsf{iv}(S)$ under $\mathsf{equiv}[\mathsf{join}(S), \mathsf{iv}(S)]$ are $\{\{v_2/a, \ldots, v_{n-1}/a\}\}$, $\{\{v_2/b, \ldots, v_{n-1}/b\}\}$, and $\{\{v_2/c, \ldots, v_{n-1}/c\}\}$, each containing the single assignment shown, as well as (for $n > 3$) a final class containing all other assignments, for which we can choose an arbitrary representative assignment, such as $\{v_2/a, v_3/b, \ldots, v_{n-1}/b\}$. Denote this representative assignment by θ_0.

Each assignment in the first 3 classes has just 2 possible extensions that satisfy $\mathsf{join}(S)$, since the value assigned to v_1 must equal the value assigned to $v_2, \ldots, v_{n-1}$, and the value assigned to v_n must be different. The assignment θ_0 has no extensions, since $\mathsf{ext}(\theta_0, e_1[\delta_1]) = \emptyset$.

Hence the number of equivalence classes in $\mathsf{equiv}[\mathsf{join}(S), \mathsf{iv}(S)]$ is at most 4, even though the total number of possible assignments of $\mathsf{iv}(S)$ is 3^{n-2} □

Definition 5.9 (Cooperating constraint catalogue) We say that a constraint catalogue Γ is a *cooperating* catalogue if for any finite set of global constraints $S \subseteq \Gamma$, we can compute a set of assignments of the variables $\mathsf{iv}(S)$ containing at least one representative of each equivalence class of $\mathsf{equiv}[\mathsf{join}(S), \mathsf{iv}(S)]$ in polynomial time in the size of $\mathsf{iv}(S)$ and the total size of the constraints in S. □

Note that this definition requires two things. First, that the number of equivalence classes in the equivalence relation $\mathsf{equiv}[\mathsf{join}(S), \mathsf{iv}(S)]$ is bounded by some fixed polynomial in the size of $\mathsf{iv}(S)$ and the size of the constraints in S. Secondly, that a suitable set of representatives for these equivalence classes can be computed efficiently from the constraints.

These requirements imply that each of the individual constraints in a cooperating catalogue has the “tractable nonemptiness” property which is suggested in [CG10] as a minimal requirement for any reasonable constraint representation. Hence any CSP instance over a cooperating catalogue which has only a single constraint is tractable.

In general, the problem of checking whether a constraint catalogue is cooperating should be hard. The reason for this is similar to the reason that checking whether a catalogue is sparse is hard, namely that finding the number of solutions to a given constraint is a hard problem (cf. the discussion after Definition 4.12). While we do not know any sufficiently general conditions that make it possible to determine in polynomial time whether a catalogue Γ is cooperating, below we exhibit a rather large catalogue that does.

5.1.1 Examples of cooperating catalogues

Definition 5.9 applies to a wide range of problems, as we will show below. We first remark that Definition 5.9 is only of interest when we have constraints of unbounded arity: any constraint catalogue where the constraints all have some fixed bound on their arity is automatically a cooperating catalogue.

Example 5.10 Consider a constraint catalogue consisting entirely of clauses (of arbitrary arity). It was shown in Example 5.5 that for any clause $e[\delta]$ and any non-empty $X \subseteq \mathcal{V}(\delta)$ the equivalence relation $\mathsf{equiv}[e[\delta], X]$ has precisely 2 equivalence classes.

If we consider some finite set, S, of clauses, then a similar argument shows that the equivalence relation $\mathsf{equiv}[\mathsf{join}(S), \mathsf{iv}(S)]$ has at most $|S| + 1$ classes. These are given by the single assignments of the variables in $\mathsf{iv}(S)$ that falsify the literals corresponding to the variables of $\mathsf{iv}(S)$ in each clause (there are at most $|S|$ of these — they may not all be distinct) together with at most one further equivalence class containing all other assignments (which must satisfy at least one literal in each clause of S).

Hence the total number of equivalence classes in the equivalence relation $\mathsf{equiv}[\mathsf{join}(S), \mathsf{iv}(S)]$ increases at most linearly with the number of clauses in S, and a representative for each class can be easily obtained from the descriptions of these clauses, by projecting the falsifying assignments down to the set of common variables, $\mathsf{iv}(S)$, and adding at most one more, arbitrary, assignment. □

Example 5.11 If we consider some finite set, S, of table constraints, then the equivalence relation $\mathsf{equiv}[\mathsf{join}(S), \mathsf{iv}(S)]$ has at most one class for each assignment allowed by each table constraint in S, together with at most one further class containing all other assignments. A representative assignment for each class can therefore be obtained by taking the projections onto $\mathsf{iv}(S)$ of the assignments allowed by each table constraint in S (these projections may not all be distinct, and they may be extension equivalent), together with at most one further, arbitrary, assignment.

Hence the total number of equivalence classes in the equivalence relation $\mathsf{equiv}[\mathsf{join}(S), \mathsf{iv}(S)]$ increases at most linearly with the total size of the descriptions of the table constraints in S, and a representative for each class can be easily obtained from the descriptions of these table constraints. □

However, it is also relatively easy to find constraint types that do not form a cooperating catalogue.

Example 5.12 The *duplicate* constraint type is parametrised by a description $\delta = \langle X_1, X_2, f \rangle$ containing two sets of variables X_1, X_2 of equal size that partition $\mathcal{V}(\delta)$, as well as a bijective function $f : X_1 \to X_2$. The constraint allows an assignment θ of $\mathcal{V}(\delta)$ if and only if for every $v \in X_1$, $\theta(v) = \theta(f(v))$.

Given a duplicate constraint $e[\delta]$, where $\delta = \langle X_1, X_2, f \rangle$, we have that, for any two assignments θ_1, θ_2 of X_1, $\text{ext}(\theta_1, e[\delta]) = \text{ext}(\theta_2, e[\delta])$ if and only if $\theta_1 = \theta_2$. Therefore, by Definition 5.4, there is a separate equivalence class in the equivalence relation $\text{equiv}[e[\delta], X_1]$ for each individual assignment of X_1. (In other words, this equivalence relation is the equality relation.) This implies that the number of equivalence classes is equal to the total number of distinct assignments of X_1, and hence is exponential in the size of the constraint description. □

In general, arbitrary extended global cardinality constraints (cf. Example 2.9) do not form a cooperating catalogue. However, we will show that if we bound the size of the variable domains, then the resulting extended global cardinality constraints do form a cooperating catalogue.

Definition 5.13 (Counting function) Let X be a set of variables and set $D = \bigcup_{x \in X} D(x)$. A *counting function* for X is any function $K : D \to \mathbb{N}$ such that $\sum_{a \in D} K(a) = |X|$.

Every assignment θ to X defines a corresponding counting function K_θ given by $K_\theta(a) = |\{x \in X \mid \theta(x) = a\}|$ for every $a \in D$. □

It is easy to verify that no EGC constraint can distinguish two assignments with the same counting function; for any EGC constraint, either both assignments satisfy it, or they both falsify it. It follows that two assignments with the same counting function are extension equivalent with respect to EGC constraints. We therefore state this property explicitly.

Definition 5.14 (Counting constraints) A global constraint $e[\delta]$ is called a *counting constraint* if, for any two assignments θ_1, θ_2 of $\mathcal{V}(\delta)$ which have the same counting function, either $\theta_1, \theta_2 \in e[\delta]$ or $\theta_1, \theta_2 \notin e[\delta]$. □

EGC constraints are not the only constraint type with this property. Constraints that require the sum (or the product) of the values of all variables in their scope to take a particular value, and constraints that require the minimum (or maximum) value of the variables in their scope to take a certain value, are also counting constraints.

Another example is given by the NValue constraint type, which requires that the number of distinct domain values taken by an assignment is a member of a specified set of acceptable numbers.

Example 5.15 (NValue constraints [BKN+10]) In an NValue constraint $e[\delta]$, the description δ specifies a finite set of natural numbers $L_\delta \subset \mathbb{N}$.

The algorithm e maps an assignment θ to 1 if $|\{\theta(v) \mid v \in \mathcal{V}(\delta)\}| \in L_\delta$. □

The reason for introducing counting functions is the following key property, previously used by Bulatov and Marx [BM10], although they do not state it explicitly.

Property 5.16 *The number of possible counting functions for a set of variables X is at most $\binom{|X|+|D|-1}{|D|-1} = O(|X|^{|D|})$, where $D = \bigcup_{x \in X} D(x)$.* □

Proof If every variable $x \in X$ has D as its set of domain elements, that is, $D(x) = D$, then every counting function corresponds to a distinct way of partitioning $|X|$ variables into at most $|D|$ boxes. There are $\binom{|X|+|D|-1}{|D|-1}$ ways of doing so [RMG+00, Section 2.3.3]. On the other hand, if there are variables $x \in X$ such that $D(x) \subset D$, then that disallows some counting functions, so the stated bound holds. ■

The main result of this section is that counting constraints on bounded domains, table constraints, and negative constraints (including clauses) all cooperate.

Theorem 5.17 *Any constraint catalogue that contains only counting constraints with bounded domain size, table constraints, and negative constraints, is a cooperating catalogue.* □

Proof Let Γ be a constraint catalogue containing only global constraints of the specified types, and let $S \subseteq \Gamma$ be a finite subset of Γ. Partition S into two subsets: S^C, containing only counting constraints and $S^\pm$ containing only table and negative constraints.

Let $\mathcal{K}$ be a set containing assignments of $\mathsf{iv}(S)$, such that for every counting function K for $\mathsf{iv}(S)$, there is some assignment $\theta_K \in \mathcal{K}$ with $K_\theta = K$. By Property 5.16, the number of counting functions for $\mathsf{iv}(S)$ is bounded by $|\mathsf{iv}(S)|^d$, where d is the bound on the domain size for the counting constraints in Γ. Hence such a set $\mathcal{K}$ can be computed in polynomial time in the size of $\mathsf{iv}(S)$.

For each constraint in $S^\pm$ we have that the description is a list of assignments (these are the allowed assignments for the table constraints and the forbidden assignments for the negative constraints, see Examples 2.11 and 2.13).

As we described in Example 5.11, for each table constraint $e[\delta] \in S$, we can obtain a representative for each equivalence class of $\mathsf{equiv}[e[\delta], \mathsf{iv}(S)]$ by taking the projection onto $\mathsf{iv}(S)$ of each allowed assignment, which we can denote by $\pi_{\mathsf{iv}(S)}(\delta)$, together with at most one further, arbitrary, assignment, θ_0, that is not in this set. This set of assignments contains at least one representative for each equivalence class of $\mathsf{equiv}[e[\delta], \mathsf{iv}(S)]$ (and possibly more than one representative for some of these classes).

Similarly, for each negative constraint $e[\delta] \in S$, we can obtain a representative for each equivalence class of $\mathsf{equiv}[e[\delta], \mathsf{iv}(S)]$, by taking the projection onto $\mathsf{iv}(S)$ of each forbidden assignment, which we can again denote by $\pi_{\mathsf{iv}(S)}(\delta)$, together with at most one further, arbitrary, assignment, θ_0, that is not in this set. This set of assignments contains at least one representative for each equivalence class of $\mathsf{equiv}[e[\delta], \mathsf{iv}(S)]$ (and possibly more than one representative for some of these classes).

Now consider the set of assignments $\mathcal{A} = \mathcal{K} \cup \{\theta_0\} \cup \bigcup_{e[\delta] \in S^{\pm}} \pi_{\mathsf{iv}(S)}(\delta)$, where θ_0 is an arbitrary assignment of $\mathsf{iv}(S)$ which does not occur in $\pi_{\mathsf{iv}(S)}(\delta)$ for any $e[\delta] \in S$ (if such an assignment exists). We claim that this set of assignments contains at least one representative for each equivalence class of $\mathsf{equiv}[\mathsf{join}(S), \mathsf{iv}(S)]$ (and possibly more than one for some classes).

To establish this claim we will show that any assignment θ of $\mathsf{iv}(S)$ that is not in $\mathcal{A}$ must be extension equivalent to some member of $\mathcal{A}$. Let θ be an assignment of $\mathsf{iv}(S)$ that is not in $\mathcal{A}$ (if such an assignment exists). If $S^{\pm}$ contains any positive constraints, then θ has an empty set of extensions to these constraints, and hence is extension equivalent to θ_0. Otherwise, any extension of θ will satisfy all negative constraints in $S^{\pm}$, so the extensions of θ that satisfy $\mathsf{join}(S)$ are completely determined by the counting function K_θ. In this case θ will be extension equivalent to some element of $\mathcal{K}$.

Moreover, the set of assignments $\mathcal{A}$ can be computed from S in polynomial time in the the size of $\mathsf{iv}(S)$ and the total size of the descriptions of the constraints in $S^{\pm}$. Therefore, Γ is a cooperating catalogue as described in Definition 5.9. ∎

Example 5.18 By Theorem 5.17, the constraints in Example 5.1 form a cooperating catalogue. □

5.2 From cooperation to structure

In any CSP instance, multiple variables may occur in exactly the same set of constraint scopes. We will show in this section that, for any constraint problem over a cooperating catalogue, a set of variables that all occur in exactly the same set of constraint scopes can be replaced by a single variable with an appropriate domain, to give a polynomial-time reduction to a smaller problem.

For classes of problems with suitable hypergraphs we can use such reductions to obtain an equivalent CSP instance with a lower treewidth, and hence obtain a solution more easily. We formalize this idea below.

Definition 5.19 (Dual of a hypergraph) Let $G = \langle V, H \rangle$ be a hypergraph. The *dual* G^* of G is a hypergraph with vertex set H and a hyperedge $\{h \in H \mid v \in h\}$ for every $v \in V$.

For a class $\mathcal{H}$ of hypergraphs, let $\mathcal{H}^* = \{G^* \mid G \in \mathcal{H}\}$. □

Example 5.20 Consider the hypergraph G in Figure 5.2. The dual, G^*, of this hypergraph has vertex set $\{C_1, C_2, C_3, C_4, C_5\}$, as well as five hyperedges $\{C_1, C_2\}$, $\{C_1, C_3\}$, $\{C_2, C_4\}$, $\{C_3, C_4\}$ and $\{C_4, C_5\}$. This transformation is illustrated in Figure 5.21. □

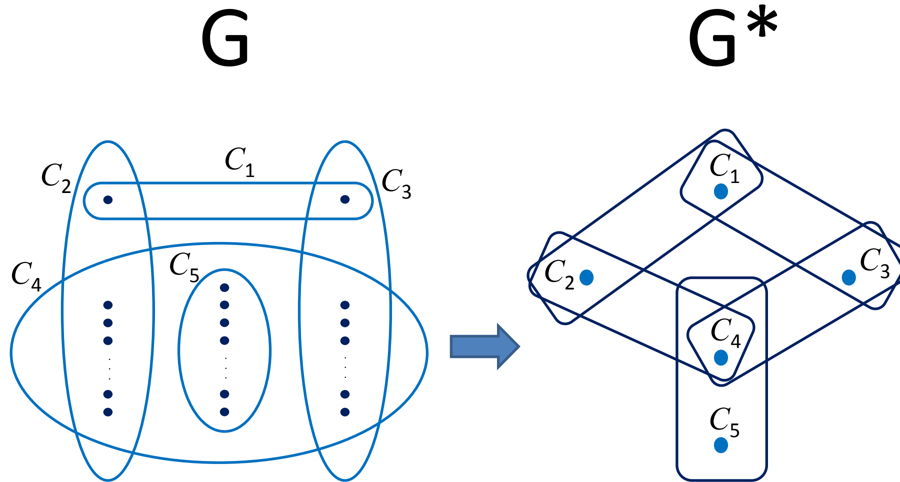

Figure 5.21: G and G^* from Example 5.20

Example 5.22 Consider the dual hypergraph G^* defined in Example 5.20. Taking the dual of this hypergraph yields G^{**}, with vertex set $\{h_1, \ldots, h_5\}$ (corresponding to the 5 hyperedges in G^*) and 5 distinct hyperedges, as shown in Figure 5.23. □

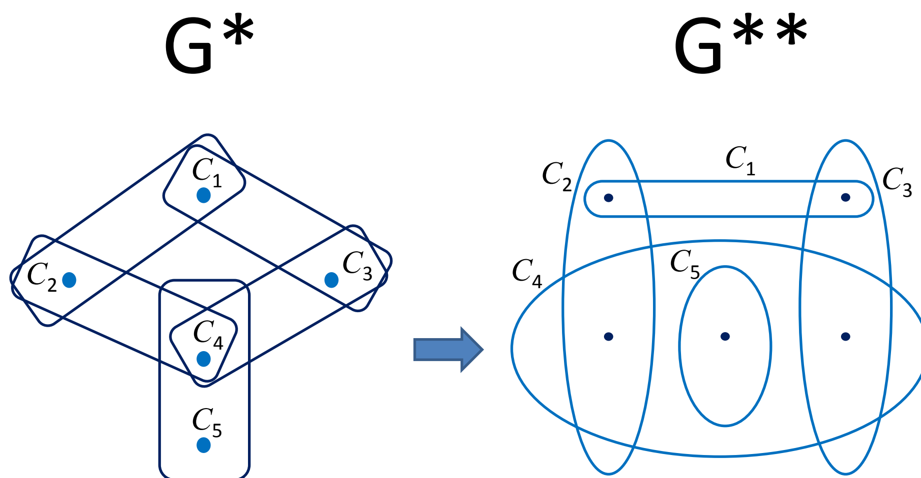

Figure 5.23: G^* and G^{**} from Example 5.22

Note that the dual of the dual of a hypergraph is not necessarily the original hypergraph. This is due to our definition of a hypergraph, since we do not allow multiple hyperedges over the same set of vertices. For hypergraphs G that do allow such edges, it is indeed the case that $G^{**} = G$.

In the example above, taking the dual of a hypergraph twice had the effect of merging precisely those sets of variables that occur in the same set of hyperedges. The lemma below shows that this is true in general: taking the dual of a hypergraph twice equates precisely those variables that occur in the same set of hyperedges.

Lemma 5.24 *For any hypergraph G, the hypergraph G^{**} has precisely one vertex corresponding to each maximal subset of vertices of G that occur in the same set of hyperedges.* □

Proof Let $G = \langle V, H \rangle$ be a hypergraph, and $v, w \in V$ be two vertices. If v, w occur in exactly the same hyperedges of G, then in G^* they give rise to the identical hyperedges $\{h \in H \mid v \in h\} = \{h \in H \mid w \in h\}$. Hence in G^{**} there is only one corresponding vertex.

In the other direction, if there is a hyperedge containing only one of v and w, then in G^* we have that the hyperedge $\{h \in H \mid v \in h\} \neq \{h \in H \mid w \in h\}$. Hence in G^{**} there are two distinct vertices that correspond to these distinct hyperedges. ■

Lemma 5.24 also provides us with an efficient way to compute the dual of the dual of a hypergraph, by simply replacing every set of vertices that occur in the same set of hyperedges with a single vertex.

Next, we combine the idea of the dual with the usual notion of treewidth (cf. Section 3.1.1) to create a new measure of width.

Definition 5.25 (twDD) Let G be a hypergraph. The treewidth of the dual of the dual (twDD) of G is $\mathrm{twDD}(G) = \mathrm{tw}(G^{**})$.

For a class of hypergraphs $\mathcal{H}$, we define $\mathrm{twDD}(\mathcal{H}) = \mathrm{tw}(\mathcal{H}^{**})$. □

Example 5.26 Consider the class $\mathcal{H}$ of hypergraphs of the family of problems described in Example 5.1. For every $G \in \mathcal{H}$, the dual hypergraph, G^*, is the same, as shown in Figure 5.21. Hence for all problems in this family the hypergraph G^{**} is as shown in Figure 5.23. It is easy to verify that this hypergraph has treewidth 3 (see Figure 5.27 for a tree decomposition). Hence $\mathrm{twDD}(\mathcal{H}) = 3$. □

Below, we demonstrate that twDD lies between treewidth and query width for every hypergraph. Furthermore, we show that twDD is incomparable with incidence width, that is, that neither $\mathrm{twDD}(G) \leq \mathrm{iw}(G)$ nor $\mathrm{twDD}(G) \geq \mathrm{iw}(G)$ hold for all hypergraphs G. Taken together, the two theorems below thus provide a precise placement of twDD among the structural restrictions discussed in Section 3.1.1.

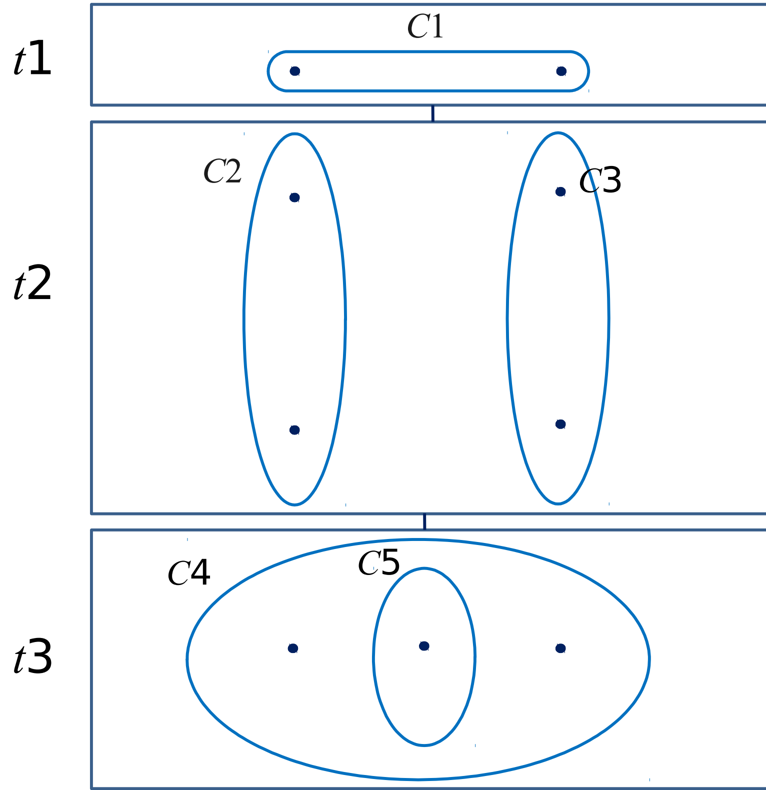

Figure 5.27: A tree decomposition of the hypergraph G^{**} in Figure 5.23

Theorem 5.28 *For every hypergraph G, we have* $\mathsf{tw}(G) \geq \mathsf{twDD}(G) \geq \mathsf{qw}(G)$. □

Proof That $\mathsf{tw}(G) \geq \mathsf{twDD}(G)$ is obvious. For $\mathsf{twDD}(G) \geq \mathsf{qw}(G)$, let G be a hypergraph. We will show that $\mathsf{qw}(G^{**}) \leq \mathsf{qw}(G)$. By Theorems 3.10 and 3.14, $\mathsf{qw}(G^{**}) \leq \mathsf{tw}(G^{**})$, and our claim follows.

Let $\langle T, \lambda \rangle$ be a query decomposition of G with width $\mathsf{qw}(G)$. Since every vertex of G^{**} corresponds to a set of vertices in G by Lemma 5.24, we can for every node t of T replace every vertex $v \in \bigcup \lambda(t)$ by the corresponding vertex $[v]$ in G^{**}. It is easy to verify that doing so preserves the conditions of Definition 3.12, and therefore the result is a query decomposition for G^{**} with width less than or equal to that of $\langle T, \lambda \rangle$. Hence $\mathsf{qw}(G^{**}) \leq \mathsf{qw}(G)$. ■

Theorem 5.29 *There exist infinite classes of hypergraphs $\mathcal{H}$ and $\mathcal{H}'$ such that*

1. *for every $G \in \mathcal{H}$,* $\mathsf{twDD}(G) > \mathsf{iw}(G)$*, and*
2. *for every $G' \in \mathcal{H}'$,* $\mathsf{twDD}(G') < \mathsf{iw}(G')$. □

Proof For the class $\mathcal{H}$, consider for every $n > 1$ the hypergraph G on the set of vertices $V = \{v_1, \ldots, v_{2n}\}$, with a hyperedge $\{V\}$ as well as a hyperedge $\{v_i, v_{i+1}\}$ for every $1 \leq i < n$. We have that $\mathsf{twDD}(G) = n$, while $\mathsf{iw}(G) = 2$.

For the class $\mathcal{H}'$, consider for every $n > 1$ the hypergraph G on the set of vertices $V = \{v_1, \ldots, v_{2n}\}$ with a hyperedge $\{v_i, v_{n+1}, \ldots, v_{2n}\}$ for every $1 \leq i \leq n$. We have that G^{**} is a star on $n+1$ vertices, and hence a tree, so $\mathsf{twDD}(G) = 1$. On the other hand, $\mathsf{inc}(G)$ contains a complete bipartite graph on n vertices, so $\mathsf{iw}(G) \geq n$. ∎

When replacing a set of variables in a CSP instance with a single variable, we will use the following definition.

Definition 5.30 (Quotient of a CSP instance) Let $P = \langle V, C \rangle$ be a CSP instance and $X \subseteq V$ be a non-empty subset of variables that all occur in the scopes of the same set S of constraints.

The *quotient* of P with respect to X, denoted P^X, is defined as follows.

- The variables of P^X are given by $V^X = (V - X) \cup \{v_X\}$, where v_X is a fresh variable, and the domain of v_X is the set of equivalence classes of $\mathsf{equiv}[\mathsf{join}(S), X]$.
- The constraints of P^X are unchanged, except that each constraint $e[\delta] \in S$ is replaced by a new constraint $e^X[\delta^X]$, where $\mathcal{V}(\delta^X) = (\mathcal{V}(\delta) - X) \cup \{v_X\}$. For any assignment θ of $\mathcal{V}(\delta^X)$, we define $e^X[\delta^X](\theta)$ to be 1 if and only if $\theta|_{\mathcal{V}(\delta)-X} \oplus \mu \in e[\delta]$, where μ is a representative of the equivalence class $\theta(v_X)$. □

We note that, by Definition 5.4, the value of $e^X[\delta^X]$ specified in Definition 5.30 is well-defined, that is, it does not depend on the specific representative chosen for the equivalence class $\theta(v_X)$, since each representative has the same set of possible extensions.

Computing the quotient of a CSP instance is simple if we are given a set of assignments that contains a representative for every equivalence class of $\mathsf{equiv}[\mathsf{join}(S), X]$.

Lemma 5.31 *Let $P = \langle V, C \rangle$ be a CSP instance and $X \subseteq V$ be a non-empty subset of variables that all occur in the scopes of the same set of constraints.*

The instance P^X has a solution if and only if P has a solution. □

Proof Let $P = \langle V, C \rangle$ and X be given, and let $S \subseteq C$ be the set of constraints $e[\delta]$ such that $X \subseteq \mathcal{V}(\delta)$.

Construct the instance P^X as specified in Definition 5.30. We will show that any solution to P can be converted into a corresponding solution for P^X, and vice versa. This conversion process just involves replacing the part of the solution assignment that gives values to the variables in the set X with an assignment that gives a suitable value to the new variable v_X.

The details are as follows. If θ is a solution to P, then $\theta|_X$ belongs to some equivalence class ξ of $\mathsf{equiv}[\mathsf{join}(S), X]$. Let θ^X be an assignment to the variables of P^X such that $\theta^X(v_X) = \xi$, and for every variable $v \notin X$, $\theta^X(v) = \theta(v)$. Since $\theta|_{\mathcal{V}(\mathsf{join}(S))}$ is in $\mathsf{join}(S)$, by Definition 5.30 the assignment θ^X satisfies the

constraints that replaced S in P^X. Since all the other constraints in P^X are unchanged, θ^X is a solution to P^X.

In the other direction, if θ^X is a solution to P^X, then it assigns to the variable v_X some equivalence class of $\mathsf{equiv}[\mathsf{join}(S), X]$. Let μ be a representative of this equivalence class, and set $\theta = \theta^X|_{V-v_X} \oplus \mu$. By Definition 5.30, we have that $\theta|_{\mathcal{V}(\mathsf{join}(S))}$ is in $\mathsf{join}(S)$. Therefore, θ satisfies the constraints in S, and since all other constraints in P^X and P are the same, θ is a solution to P. ∎

Combining the above with the notion of a cooperating catalogue (cf. Section 5.1), we can now prove the following.

Theorem 5.32 *Any CSP instance P can be converted to an instance P', with* $\mathsf{hyp}(P') = \mathsf{hyp}(P)^{**}$*, such that P' has a solution if and only if P does. Moreover, if P is over a cooperating catalogue, this conversion can be done in polynomial time.* □

Proof Let $P = \langle V, C\rangle$ be a CSP instance. For each variable $v \in V$ we define $S(v) = \{e[\delta] \in C \mid v \in \mathcal{V}(\delta)\}$ We then partition the vertices of P into subsets $X_1, \ldots, X_k$, where each X_i is a maximal subset of variables v that share the same value for $S(v)$.

We initially set $P_0 = P$. Then, for each X_i in turn, we set $P_i = (P_{i-1})^X$. Finally we set $P' = P_k$.

By Lemma 5.24, $\mathsf{hyp}(P') = \mathsf{hyp}(P)^{**}$.

By Lemma 5.31, P' has a solution if and only if P has a solution.

Finally, if P is over a cooperating catalogue, then by Definition 5.9, we can compute the domains of each new variable introduced in polynomial time in the size of each X_i and the total size of the constraints. Hence we can compute P' in polynomial time. ∎

Using Theorem 5.32, we can immediately get a new tractable CSP class by using the fact that bounded treewidth implies tractability for classic CSP instances (cf. Theorem 3.8). Furthermore, since SAT clauses form a cooperating constraint catalogue by Theorem 5.17, and twDD is incomparable with incidence width, this class is a new tractable class for SAT (cf. Theorems 3.11 and 5.29).

Theorem 5.33 *Let Γ be a constraint catalogue and $\mathcal{H}$ a class of hypergraphs.* $\mathrm{CSP}(\mathcal{H}, \Gamma)$ *is tractable if*

1. *Γ is a cooperating catalogue, and*
2. $\mathsf{twDD}(\mathcal{H}) < \infty$. □

Proof Let Γ be a cooperating catalogue, $\mathcal{H}$ a class of hypergraphs such that $\mathsf{twDD}(\mathcal{H}) < \infty$, and $P \in \mathrm{CSP}(\mathcal{H}, \Gamma)$. We will reduce P to an instance P' that has a solution if and only if P does, and which satisfies the conditions of Theorem 3.8.

Reduce P to a CSP instance P' using Theorem 5.32. By Definition 5.25, since $\mathsf{hyp}(P') = \mathsf{hyp}(P)^{**}$, $\mathsf{tw}(\mathsf{hyp}(P')) < \infty$. Therefore, P' belongs to a class of CSP

instances that satisfies the conditions of Theorem 3.8, and hence can be solved in polynomial time. ■

Recall the family of constraint problems described in Example 5.1 at the start of this paper. Since the constraints in this problem form a cooperating catalogue (cf. Example 5.18), and all instances have bounded twDD (cf. Example 5.26), this family of problems is tractable by Theorem 5.33.

5.3 Relational structures and cores

We can obtain a slightly more general tractable class by building on a more general version of Theorem 3.8. To do this we need to introduce a finer notion of structure for a CSP instance.

Definition 5.34 (Relational structure) A *relational structure* $\mathbf{S} = \langle V, R_1, \ldots, R_m \rangle$ is a set V, called the *universe* of S, together with a list of relations $R_1, \ldots, R_m$ over V. □

Example 5.35 A directed graph $G = \langle V, E \rangle$ can be seen as a relational structure, where the universe V is the vertices of the graph, and the relation E is a binary relation consisting of the directed edges, $\langle v_i, v_j \rangle$.

A directed graph with labelled edges can be seen as a relational structure, $\langle V, E_1, \ldots, E_m \rangle$, where each relation contains all the edges with a particular label. □

In fact, an arbitrary relational structure $\mathbf{S} = \langle V, R_1, \ldots, R_m \rangle$, can be seen as a hypergraph on the set of vertices V, where the hyperedges have been labelled, ordered to form tuples, and then grouped by their labels into relations.

The underlying hypergraph of a relational structure $\mathbf{S} = \langle V, R_1, \ldots, R_m \rangle$, denoted $\mathsf{hyp}(\mathbf{S})$, is the hypergraph with vertex set V and a hyperedge corresponding to each tuple of each of the relations in $\mathbf{S}$. Each of these hyperedges consists of the set of distinct elements in the corresponding tuple. For a class $\mathcal{A}$ of relational structures, we define $\mathsf{hyp}(\mathcal{A}) = \{\mathsf{hyp}(\mathbf{S}) \mid \mathbf{S} \in \mathcal{A}\}$.

Definition 5.36 We say that a CSP instance $\langle V, C \rangle$ *permits* a relational structure $\langle W, R_1, \ldots, R_m \rangle$ if $W = V$, and there is some linear ordering of V and some partition of the constraints C into sets $C_1, \ldots, C_m$, such that for each i, the following conditions hold:

- every constraint in C_i has the same extension; and
- the tuples of R_i correspond to the sets $\mathcal{V}(\delta)$, for each $e[\delta] \in C_i$, ordered by the ordering of V. □

Lemma 5.37 *For every relational structure* $\mathbf{S}$ *and CSP instance* P*, if* P *permits* $\mathbf{S}$ *then* $\mathsf{hyp}(\mathbf{S}) = \mathsf{hyp}(P)$. □

Proof Let P and $\mathbf{S}$ be given. If P permits $\mathbf{S}$, then $\mathsf{hyp}(\mathbf{S})$ and $\mathsf{hyp}(P)$ both have the variables of P as their vertices. Furthermore, $\mathsf{hyp}(P)$ has a hyperedge for every constraint scope in P. Every constraint scope in P corresponds to a tuple in a relation of $\mathbf{S}$, and $\mathsf{hyp}(\mathbf{S})$ has a hyperedge for every such tuple, so the equality follows. ∎

Note that the reverse does not hold in general: a relational structure may have the same underlying hypergraph as a CSP instance, but not be permitted by it due to the extensions of the constraints.

We can get a bigger tractable CSP class by combining Theorem 5.33 with a known result about relational structures.

Definition 5.38 (Core) A relational structure $\langle V, R_1, \ldots, R_m \rangle$ is a *substructure* of another relational structure $\langle V', R'_1, \ldots, R'_m \rangle$ if $V \subseteq V'$, and for every $i \in \{1, \ldots, m\}$, $R_i \subseteq R'_i$.

A *homomorphism* from a relational structure $\langle V, R_1, \ldots, R_m \rangle$ to a relational structure $\langle V', R'_1, \ldots, R'_m \rangle$ is a function $h : V \rightarrow V'$ such that, for every $i \in \{1, \ldots, m\}$ and $\mathbf{t} \in R_i$, we have $h(\mathbf{t}) \in R'_i$.

A relational structure $\mathbf{S}$ is a *core* if there is no homomorphism from $\mathbf{S}$ to a proper substructure of $\mathbf{S}$. A core of a relational structure $\mathbf{S}$ is a substructure $\mathbf{S}'$ of $\mathbf{S}$ such that there is a homomorphism from $\mathbf{S}$ to $\mathbf{S}'$, and $\mathbf{S}'$ is a core. As all cores of a relational structure $\mathbf{S}$ are isomorphic, we will speak of *the core* of $\mathbf{S}$, denoted $\mathsf{Core}(\mathbf{S})$. For a class of relational structures $\mathcal{A}$, we define $\mathsf{Core}(\mathcal{A}) = \{\mathsf{Core}(\mathbf{S}) \mid \mathbf{S} \in \mathcal{A}\}$. □

Example 5.39 The core of a relational structure $\mathbf{S}$ may have a much simpler hypergraph than $\mathbf{S}$ itself. For example, the core of any bipartite graph is a single edge. □

Theorem 5.40 ([DKV02, Prop. 1]) *Any CSP instance P over a constraint catalogue Γ that permits a relational structure $\mathbf{S}$ can be converted in polynomial-time to a CSP instance P' over Γ that permits the structure $\mathsf{Core}(\mathbf{S})$ and has a solution if and only if P does.* □

In the original, Theorem 5.40 is stated in terms of a constraint language, i.e. a set of relations, rather than a constraint catalogue. However, since the core of a relational structure is a substructure of it, this result also holds for an arbitrary constraint catalogue.

Theorem 5.41 *Let Γ be a constraint catalogue and $\mathcal{A}$ a class of relational structures.* $\mathsf{CSP}(\mathsf{hyp}(\mathcal{A}), \Gamma)$ *is tractable if*

1. *Γ is a cooperating catalogue, and*
2. $\mathsf{twDD}(\mathsf{hyp}(\mathsf{Core}(\mathcal{A}))) < \infty$. □

Proof Let Γ be a cooperating catalogue, $\mathcal{A}$ a class of relational structures such that $\mathsf{twDD}(\mathsf{hyp}(\mathsf{Core}(\mathcal{A}))) < \infty$, and $P \in \mathrm{CSP}(\mathsf{hyp}(\mathcal{A}), \Gamma)$. By Theorem 5.40, there exists a CSP instance P' over Γ that permits a structure in $\mathsf{Core}(\mathcal{A})$, and which has a solution if and only if P does.

Since P' permits a relational structure in $\mathsf{Core}(\mathcal{A})$, we have that $\mathsf{hyp}(P') \in \mathsf{hyp}(\mathsf{Core}(\mathcal{A}))$, and therefore that $P' \in \mathrm{CSP}(\mathsf{hyp}(\mathsf{Core}(\mathcal{A})), \Gamma)$. Since Γ is a cooperating catalogue and $\mathsf{twDD}(\mathsf{hyp}(\mathsf{Core}(\mathcal{A}))) < \infty$, P' satisfies the conditions of Theorem 5.33, and hence can be solved in polynomial time. ∎

5.4 Summary

In this chapter we have shown how the set of assignments to a set of variables can be split into a smaller number of equivalence classes with respect to the set of constraints in which all these variables occur. When this can be done in polynomial time, we obtain a tractable class of CSP instances using a new structural restriction that lies between treewidth and query width. We have also shown that splitting assignments into equivalence classes in this way can be done in polynomial time for a large class of global constraints.

We have also seen how this result may be augmented by considering the structure of a CSP instance as a relational structure rather than a hypergraph.

Chapter 6

Summary and future work

In this thesis, we have undertaken a systematic study of the complexity of constraint satisfaction problems with global constraints. As we have seen, global constraints are a natural and effective way to model a variety of problems. As such, the complexity of such problems is an area of interest. So far, however, this area has lagged behind in terms of techniques for establishing tractable classes of problems. In particular, techniques that have been successfully applied to classic constraints, and hence to conjunctive query evaluation, have so far not been lifted to the global case; more importantly, they have so far not been lifted to SAT, even though SAT solvers are widely used [Pet12]. We have therefore developed a framework for doing precisely that, and identified new ways of extending known tractability results for classic CSP instances to instances with global constraints.

To do so, we first gave a formal definition of global constraints that provides a better measure of size for a constraint's representation in Chapter 2. We also showed how global constraints can be used to model configuration problems, as well as previous formal systems for configuration, in a natural way.

In Chapter 3, we gave an overview of current research into the complexity and tractability of CSP instances, and observed that many structural restrictions that yield tractability for classic CSP instances rely on properties of the representation of classic constraints. In a sense, this makes them hybrid rather than structural properties, and leads us to try and identify these properties, so as to find classes of global constraints that possess them.

This we do in Chapter 4, by collecting several insights from the literature to conclude that classic CSP instances have few solutions in their size, and that structural restrictions exploit this fact. A formalization of this property leads to new tractable classes for instances with global constraints.

We also observe that our definition of global constraints allows us to treat a CSP instance as a single constraint. When all such instances have been drawn from known tractable classes, we obtain new tractability results by looking at the overlaps between instances. This result generalizes a previous result by Cohen and Green [CG06] using far simpler theoretical techniques.

Furthermore, we show that the above results also hold for weighted CSP instances, a variant of CSP where constraints assign costs to satisfying assignments, with no major changes.

Finally, in Chapter 5 we look at constraints that do not have the properties identified in Chapter 4. For such constraints, we apply a different idea, namely that many types of constraints can't distinguish between all possible assignments. This observation allows us to split assignments into equivalence classes, and obtain a new structural restriction that yields tractability when the number of equivalence classes is small. We also show that this result can be generalized by viewing the structure of a CSP instance as a relational structure rather than a hypergraph.

The major theme in the techniques presented in this thesis is that to obtain new tractability results for CSP instances with global constraints, we have to exploit not only properties of the problem's *structure* and of the *relations* specified by the problem's constraints, as has traditionally been done in hybrid tractability research, but also properties of the *representation* of those constraints. In retrospect, this is not surprising, since global constraints differ from classic constraints precisely in how they are represented. However, identifying the relevant properties of this difference has, as we have seen, required a combination of several pieces of technical machinery.

6.1 Open questions

The results presented in this thesis are not, by any means, the last word on tractability for CSP instances with global constraints. Below, we list some directions for future research in this area.

6.1.1 Subproblem decompositions

- Can other results that rely on a CSP instance having few solutions, such as those of Cohen et al. [CCGM11], be subsumed under the framework of subproblem decompositions? If not, can they be used to improve the framework?
- Subproblem decompositions allow us to combine known tractable classes. Which combinations are particularly useful or interesting?
- Can subproblem decompositions be used to work with the fully general valued CSP framework [SFV95], rather than just the special case of WCSP instances?
- Is there more to be said about back door sets for subproblem decompositions? In particular, what other consequences can we derive from Theorem 4.48 aside from the one mentioned in Section 4.3?

6.1.2 Equivalence classes

- What other types of global constraints form cooperating languages, besides those identified in Section 5.1.1?
- Is it possible, given a catalogue Γ and set of constraints $S \subseteq \Gamma$, to compute a set of assignments containing at least one representative of each equivalence class of $\mathsf{equiv}[\mathsf{join}(S), \mathsf{iv}(S)]$ (cf. Definition 5.9), if we know that the number of equivalence classes is small (that is, bounded by a fixed polynomial for every $S \subseteq \Gamma$), and the constraints in Γ allow partial assignment checking (cf. Definition 4.10)?
- Are there other restrictions, besides twDD, that lie between treewidth and query width, and yield tractability for interesting classes of global constraints?
- Can the results in Chapter 5 be applied to valued or weighted CSP instances? What about the problem of max CSP [GGS09], where we want to find a solution that satisfies as many constraints as possible?
- What happens to the constraint relations given by the constraints of a CSP instance when we use the reduction in Chapter 5, and replace sets of assignments with sets of equivalence classes? Does the constraint language gain, or keep, some useful properties?

References

[ABD07] A. Atserias, A. A. Bulatov, and V. Dalmau. On the power of k-consistency. In *Proceedings of the 34th International Colloquium on Automata, Languages and Programming (ICALP'07)*, volume 4596 of *Lecture Notes in Computer Science*, pp. 279–290. Springer, 2007.

↑ p. 43

[ACP87] S. Arnborg, D. G. Corneil, and A. Proskurowski. Complexity of finding embeddings in a k-tree. *SIAM Journal on Algebraic and Discrete Methods*, 8(2):277–284, 1987.

↑ p. 33

[ADF+11] M. Aschinger, C. Drescher, G. Friedrich, G. Gottlob, P. Jeavons, A. Ryabokon, and E. Thorstensen. Optimization methods for the partner units problem. In *Proceedings of the 8th International Conference on Integration of Artificial Intelligence and Operations Research Techniques in Constraint Programming for Combinatorial Optimization Problems (CPAIOR'11)*, volume 6697 of *Lecture Notes in Computer Science*, pp. 4–19. Springer, 2011.

↑ p. 18

[ADG+11a] M. Aschinger, C. Drescher, G. Gottlob, P. Jeavons, and E. Thorstensen. Structural decomposition methods and what they are good for. In *Proceedings of the 28th International Symposium on Theoretical Aspects of Computer Science (STACS'11)*, volume 9 of *Leibniz International Proceedings in Informatics*, pp. 12–28, 2011.

↑ pp. 31, 32, 33

[ADG+11b] M. Aschinger, C. Drescher, G. Gottlob, P. Jeavons, and E. Thorstensen. Tackling the partner units configuration problem. In *Proceedings of the 22nd International Joint Conference on Artificial Intelligence (IJCAI'11)*, pp. 497–503, 2011.

↑ pp. 18, 19, 20

[Adl06] I. Adler. *Width Functions for Hypertree Decompositions*. Doctoral dissertation, Albert-Ludwigs-Universität Freiburg, 2006.

↑ pp. 37, 38

[AGG07] I. Adler, G. Gottlob, and M. Grohe. Hypertree width and related hypergraph invariants. *European Journal of Combinatorics*, 28(8):2167–2181, 2007.

↑ p. 37

[AP89] S. Arnborg and A. Proskurowski. Linear time algorithms for NP-hard problems restricted to partial k-trees. *Discrete Applied Mathematics*, 23(1):11–24, 1989.

↑ p. 33

[BC94] N. Beldiceanu and E. Contejean. Introducing global constraints in CHIP. *Mathematical and Computer Modelling*, 20(12):97–123, 1994.

↑ p. 17

[BD06] A. A. Bulatov and V. Dalmau. A simple algorithm for Mal'tsev constraints. *SIAM Journal on Computing*, 36(1):16–27, 2006.

↑ p. 31

[Ber84] C. Berge. *Hypergraphs: Combinatorics of Finite Sets*, volume 45 of *North-Holland Mathematical Library*. Elsevier, 1984.

↑ p. 31

[Bes06] C. Bessiere. Constraint propagation. In *Handbook of Constraint Programming*, volume 2 of *Foundations of Artificial Intelligence*, pp. 169–208. Elsevier, 2006.

↑ p. 43

[BFMY83] C. Beeri, R. Fagin, D. Maier, and M. Yannakakis. On the desirability of acyclic database schemes. *Journal of the ACM*, 30:479–513, 1983.

↑ p. 31

[BHHW04] C. Bessiere, E. Hebrard, B. Hnich, and T. Walsh. The complexity of global constraints. In *Proceedings of the 19th National Conference on Artificial Intelligence (AAAI'04)*, pp. 112–117, 2004.

↑ pp. 17, 43

[BHHW07] C. Bessiere, E. Hebrard, B. Hnich, and T. Walsh. The complexity of reasoning with global constraints. *Constraints*, 12(2):239–259, 2007.

↑ pp. 17, 21

[BKN+10] C. Bessiere, G. Katsirelos, N. Narodytska, C.-G. Quimper, and T. Walsh. Decomposition of the NValue constraint. In *Proceedings of the 16th International Conference on Principles and Practice of Constraint Programming (CP'10)*, volume 6308 of *Lecture Notes in Computer Science*. Springer, 2010.

↑ pp. 17, 77

[BM10] A. A. Bulatov and D. Marx. The complexity of global cardinality constraints. *Logical Methods in Computer Science*, 6(4:4):1–27, 2010.

↑ p. 77

[Bod88] H. Bodlaender. Dynamic programming on graphs with bounded treewidth. In *Proceedings of the 15th International Colloquium on Automata, Languages and Programming (ICALP'88)*, volume 317 of *Lecture Notes in Computer Science*, pp. 105–118. Springer, 1988.

↑ p. 33

[CCGM11] D. A. Cohen, M. C. Cooper, M. J. Green, and D. Marx. On guaranteeing polynomially bounded search tree size. In *Proceedings of the 17th International Conference on Principles and Practice of Constraint Programming (CP'11)*, volume 6876 of *Lecture Notes in Computer Science*, pp. 160–171. Springer, 2011.

↑ p. 88

[CG06] D. Cohen and M. Green. Typed guarded decompositions for constraint satisfaction. In *Proceedings of the 12th International Conference on the Principles and Practice of Constraint Programming (CP'06)*, volume 4204 of *Lecture Notes in Computer Science*, pp. 122–136. Springer, 2006.

↑ pp. 11, 57, 58, 61, 87

[CG10] H. Chen and M. Grohe. Constraint satisfaction with succinctly specified relations. *Journal of Computer and System Sciences*, 76(8):847–860, 2010.

↑ pp. 35, 74

[CGFS86] F. Chung, R. Graham, P. Frankl, and J. Shearer. Some intersection theorems for ordered sets and graphs. *Journal of Combinatorial Theory, Series A*, 43(1):23–37, 1986.

↑ p. 51

[CJ06] D. Cohen and P. Jeavons. The complexity of constraint languages. In *Handbook of Constraint Programming*, volume 2 of *Foundations of Artificial Intelligence*, pp. 245 – 280. Elsevier, 2006.

↑ pp. 31, 41

[CJG08] D. Cohen, P. Jeavons, and M. Gyssens. A unified theory of structural tractability for constraint satisfaction problems. *Journal of Computer and System Sciences*, 74(5):721–743, 2008.

↑ pp. 32, 58

[CJS10] M. C. Cooper, P. G. Jeavons, and A. Z. Salamon. Generalizing constraint satisfaction on trees: Hybrid tractability and variable elimination. *Artificial Intelligence*, 174(9–10):570–584, 2010.

↑ pp. 31, 41

[Coo89] M. C. Cooper. An optimal k-consistency algorithm. *Artificial Intelligence*, 41(1):89–95, 1989.

↑ p. 43

[Cor88] G. Cornuéjols. General factors of graphs. *Journal of Combinatorial Theory, Series B*, 45(2):185–198, 1988.

↑ p. 44

[CR00] C. Chekuri and A. Rajaraman. Conjunctive query containment revisited. *Theoretical Computer Science*, 239(2):211–229, 2000.

↑ pp. 35, 36

[DF99] R. G. Downey and M. R. Fellows. *Parameterized Complexity*. Monographs in Computer Science. Springer, 1999.

↑ p. 40

[dGSV06] S. de Givry, T. Schiex, and G. Verfaillie. Exploiting Tree Decomposition and Soft Local Consistency in Weighted CSP. In *Proceedings of the 21st National Conference on Artificial Intelligence (AAAI'06)*, pp. 22–27, 2006.

↑ p. 64

[Die10] R. Diestel. *Graph Theory*, volume 173 of *Graduate Texts in Mathematics*. Springer, 4th edition, 2010.

↑ p. 33

[DKV02] V. Dalmau, P. G. Kolaitis, and M. Y. Vardi. Constraint satisfaction, bounded treewidth, and finite-variable logics. In *Proceedings of the 8th International Conference on Principles and Practice of Constraint Programming (CP'02)*, volume 2470 of *Lecture Notes in Computer Science*, pp. 223–254. Springer, 2002.

↑ pp. 31, 33, 85

[FFH+11] M. R. Fellows, T. Friedrich, D. Hermelin, N. Narodytska, and F. A. Rosamond. Constraint satisfaction problems: Convexity makes AllDifferent constraints tractable. In *Proceedings of the 22nd International Joint Conference on Artificial Intelligence (IJCAI'11)*, pp. 522–527, 2011.

↑ p. 17

[FG06] J. Flum and M. Grohe. *Parameterized Complexity Theory*. Texts in Theoretical Computer Science. Springer, 2006.

↑ p. 40

[FHS10] A. Falkner, A. Haselböck, and G. Schenner. Modeling technical product configuration problems. In *Proceedings of the Configuration Workshop at the 19th European Conference on Artificial Intelligence (ECAI'10)*, pp. 40–45, 2010.

↑ pp. 18, 19

[GGM07] G. Gottlob, G. Greco, and T. Mancini. Conditional constraint satisfaction: Logical foundations and complexity. In *Proceedings of the 20th International Joint Conference on Artificial Intelligence (IJCAI'07)*, pp. 88–93, 2007.

↑ pp. 25, 29

[GGS09] G. Gottlob, G. Greco, and F. Scarcello. Tractable optimization problems through hypergraph-based structural restrictions. In *Proceedings of the 36th International Colloquium on Automata, Languages and Programming (ICALP'09)*, volume 5556 of *Lecture Notes in Computer Science*, pp. 16–30. Springer, 2009.

↑ pp. 64, 89

[GJ79] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, 1979.

↑ pp. 9, 18, 19, 20, 21, 23, 45, 47, 64

[GJ08] M. J. Green and C. Jefferson. Structural tractability of propagated constraints. In *Proceedings of the 14th International Conference on Principles and Practice of Constraint Programming (CP'08)*, volume 5202 of *Lecture Notes in Computer Science*, pp. 372–386. Springer, 2008.

↑ pp. 10, 45, 46, 47

[GJC94] M. Gyssens, P. G. Jeavons, and D. A. Cohen. Decomposing constraint satisfaction problems using database techniques. *Artificial Intelligence*, 66(1):57–89, 1994.

↑ pp. 11, 29, 31

[GJM06] I. P. Gent, C. Jefferson, and I. Miguel. MINION: A fast, scalable constraint solver. In *Proceedings of the 17th European Conference on Artificial Intelligence (ECAI'06)*, pp. 98–102. IOS Press, 2006.

↑ pp. 10, 17

[GLS81] M. Grötschel, L. Lovász, and A. Schrijver. The ellipsoid method and its consequences in combinatorial optimization. *Combinatorica*, 1(2):169–197, 1981.

↑ p. 42

[GLS00] G. Gottlob, N. Leone, and F. Scarcello. A comparison of structural CSP decomposition methods. *Artificial Intelligence*, 124(2):243–282, 2000.

↑ p. 11

[GLS01] G. Gottlob, N. Leone, and F. Scarcello. The complexity of acyclic conjunctive queries. *Journal of the ACM*, 48(3):431–498, 2001.

↑ pp. 29, 35

[GLS02] G. Gottlob, N. Leone, and F. Scarcello. Hypertree decompositions and tractable queries. *Journal of Computer and System Sciences*, 64(3):579–627, 2002.

↑ pp. 11, 31, 33, 35, 36, 37

[GM06] M. Grohe and D. Marx. Constraint solving via fractional edge covers. In *Proceedings of the 17th ACM-SIAM symposium on discrete algorithms (SODA'06)*, pp. 289–298. ACM, 2006.

↑ pp. 33, 38, 39, 49, 51

[GPW06] G. Gottlob, R. Pichler, and F. Wei. Tractable database design through bounded treewidth. In *Proceedings of the 25th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems (PODS'06)*, pp. 124–133. ACM, 2006.

↑ p. 32

[Gro07] M. Grohe. The complexity of homomorphism and constraint satisfaction problems seen from the other side. *Journal of the ACM*, 54(1):1–24, 2007.

↑ pp. 32, 34

[GS11] S. Gaspers and S. Szeider. Kernels for global constraints. In *Proceedings of the 22nd International Joint Conference on Artificial Intelligence (IJCAI'11)*, pp. 540–545, 2011.

↑ p. 17

[GS12] S. Gaspers and S. Szeider. Backdoors to satisfaction. In *The Multivariate Algorithmic Revolution and Beyond*, volume 7370 of *Lecture Notes in Computer Science*, pp. 287–317. Springer, 2012.

↑ p. 67

[HK73] J. Hopcroft and R. Karp. An $n^{5/2}$ algorithm for maximum matchings in bipartite graphs. *SIAM Journal on Computing*, 2(4):225–231, 1973.

↑ p. 44

[JC95] P. G. Jeavons and M. C. Cooper. Tractable constraints on ordered domains. *Artificial Intelligence*, 79(2):327–339, 1995.

↑ p. 41

[JCP98] P. Jeavons, D. Cohen, and J. Pearson. Constraints and universal algebra. *Annals of Mathematics and Artificial Intelligence*, 24(1):51–67, 1998.

↑ p. 41

[Jég93] P. Jégou. Decomposition of domains based on the micro-structure of finite constraint-satisfaction problems. In *Proceedings of the 11th National Conference on Artificial Intelligence (AAAI'93)*, pp. 731–736, 1993.

↑ p. 41

[KEKM08] M. Kutz, K. Elbassioni, I. Katriel, and M. Mahajan. Simultaneous matchings: Hardness and approximation. *Journal of Computer and System Sciences*, 74(5):884–897, 2008.

↑ pp. 17, 45

[KNW12] G. Katsirelos, N. Narodytska, and T. Walsh. The SeqBin constraint revisited. In *Proceedings of the 18th International Conference on Principles and Practice of Constraint Programming (CP'12)*, volume 7514 of *Lecture Notes in Computer Science*, pp. 332–347. Springer, 2012.

↑ p. 43

[KV00] P. G. Kolaitis and M. Y. Vardi. Conjunctive-query containment and constraint satisfaction. *Journal of Computer and System Sciences*, 61(2):302–332, 2000.

↑ p. 34

[Mai98] D. Mailharro. A classification and constraint-based framework for configuration. *Artificial Intelligence for Engineering Design, Analysis and Manufacturing*, 12(4):383–397, 1998.

↑ p. 25

[Mar09a] D. Marx. Beyond fractional hypertree width. Talk given at Dagstuhl Seminar 09441, The Constraint Satisfaction Problem: Complexity and Approximability, 2009.

↑ pp. 51, 52, 60

[Mar09b] D. Marx. Tractable hypergraph properties for constraint satisfaction and conjunctive queries. *Computing Research Repository*, abs/0911.0801, 2009.

↑ pp. 11, 39, 40

[Mar10a] D. Marx. Approximating fractional hypertree width. *ACM Transactions on Algorithms*, 6(2):29:1–29:17, 2010.

↑ p. 39

[Mar10b] D. Marx. Tractable hypergraph properties for constraint satisfaction and conjunctive queries. In *Proceedings of the 42nd ACM Symposium on Theory of Computing, (STOC'10)*, pp. 735–744. ACM, 2010.

↑ p. 39

[MF89] S. Mittal and F. Frayman. Towards a generic model of configuraton tasks. In *Proceedings of the 11th International Joint Conference on Artificial Intelligence (IJCAI'89)*, pp. 1395–1401, 1989.

↑ p. 10

[MF90] S. Mittal and B. Falkenhainer. Dynamic constraint satisfaction problems. In *Proceedings of the 8th National Conference on Artificial Intelligence (AAAI'90)*, pp. 25–32, 1990.

↑ pp. 10, 25, 26

[Pap94] C. H. Papadimitriou. *Computational Complexity*. Addison Wesley, 1994.

↑ p. 9

[Pet12] J. Petke. *On the bridge between Constraint Satisfaction and Boolean Satisfiability*. Doctoral dissertation, University of Oxford, 2012.

↑ p. 87

[PJ97] J. Pearson and P. Jeavons. A survey of tractable constraint satisfaction problems. Technical Report CSD-TR-97-15, Royal Holloway, University of London, 1997.

↑ pp. 11, 31

[QLOvBG04] C.-G. Quimper, A. López-Ortiz, P. van Beek, and A. Golynski. Improved algorithms for the global cardinality constraint. In *Proceedings of the 10th International Conference on Principles and Practice of Constraint Programming (CP'04)*, volume 3258 of *Lecture Notes in Computer Science*, pp. 542–556. Springer, 2004.

↑ pp. 22, 43, 45

[Rég94] J.-C. Régin. A filtering algorithm for constraints of difference in CSPs. In *Proceedings of the 12th National Conference on Artificial Intelligence (AAAI'94)*, pp. 362–367. AAAI Press, 1994.

↑ p. 44

[Rég96] J.-C. Régin. Generalized Arc Consistency for Global Cardinality Constraint. In *Proceedings of the 13th National Conference on Artificial Intelligence (AAAI'96)*, pp. 209–215. AAAI Press, 1996.

↑ p. 44

[RMG+00] K. H. Rosen, J. G. Michaels, J. L. Gross, J. W. Grossman, and D. R. Shier, editors. *Handbook of Discrete and Combinatorial Mathematics*. Discrete Mathematics and Its Applications. CRC Press, 2000.

↑ p. 77

[RPD90] F. Rossi, C. Petrie, and V. Dhar. On the equivalence of constraint satisfaction problems. In *Proceedings of the 9th European Conference on Artificial Intelligence (ECAI'90)*, pp. 550–556, 1990.

↑ p. 35

[RS86] N. Robertson and P. D. Seymour. Graph minors. II. Algorithmic aspects of tree-width. *Journal of Algorithms*, 7(3):309–322, 1986.

↑ pp. 31, 33

[RvBW06] F. Rossi, P. van Beek, and T. Walsh, editors. *The Handbook of Constraint Programming*. Elsevier, 2006.

↑ p. 9

[Sch78] T. J. Schaefer. The complexity of satisfiability problems. In *Proceedings of the 10th ACM Symposium on Theory of Computing (STOC'78)*, pp. 216–226. ACM, 1978.

↑ p. 31

[SCKN07] R. Sirdey, J. Carlier, H. Kerivin, and D. Nace. On a resource-constrained scheduling problem with application to distributed systems reconfiguration. *European Journal of Operational Research*, 183(2):546 – 563, 2007.

↑ p. 10

[SF96] D. Sabin and E. C. Freuder. Configuration as composite constraint satisfaction. In *Proceedings of the Artificial Intelligence and Manufacturing Research Planning Workshop*, pp. 153–161, 1996.

↑ pp. 25, 27

[SF99a] D. Sabin and E. C. Freuder. Optimization methods for constraint resource problems. In *Proceedings of the AAAI Workshop on Configuration, technical report WS-99-05*, pp. 83–89, 1999.

↑ pp. 10, 25

[SF99b] M. Sabin and E. C. Freuder. Detecting and resolving inconsistency and redundancy in conditional constraint satisfaction problems. In *Proceedings of the AAAI Workshop on Configuration, technical report WS-99-05*, pp. 90–94, 1999.

↑ p. 26

[SFH98] M. Stumptner, G. E. Friedrich, and A. Haselböck. Generative constraint-based configuration of large technical systems. *Artificial Intelligence for Engineering Design, Analysis and Manufacturing*, 12(4):307–320, 1998.

↑ pp. 10, 25

[SFV95] T. Schiex, H. Fargier, and G. Verfaillie. Valued Constraint Satisfaction Problems: Hard and Easy Problems. In *Proceedings of the 14th International Joint Conference on Artificial Intelligence (IJCAI'95)*, pp. 631–639, 1995.

↑ pp. 64, 88

[SH93] M. Stumptner and A. Haselböck. A generative constraint formalism for configuration problems. In *Proceedings of the 3rd Congress of the Italian Association for Artificial Intelligence (AI*IA'93)*, volume 728 of *Lecture Notes in Computer Science*, pp. 302–313. Springer, 1993.

↑ p. 10

[SJ08] A. Z. Salamon and P. G. Jeavons. Perfect constraints are tractable. In *Proceedings of the 14th International Conference on Principles and Practice of Constraint Programming (CP'08)*, volume 5202 of *Lecture Notes in Computer Science*, pp. 524–528. Springer, 2008.

↑ pp. 31, 41, 42

[SS11] M. Samer and S. Szeider. Tractable cases of the extended global cardinality constraint. *Constraints*, 16(1):1–24, 2011.

↑ pp. 22, 24, 44

[Sze03] S. Szeider. On fixed-parameter tractable parameterizations of SAT. In *The 6th International Conference on Theory and Applications of Satisfiability Testing (SAT'03)*, volume 2919 of *Lecture Notes in Computer Science*, pp. 188–202. Springer, 2003.

↑ p. 35

[Tac09] G. Tack. *Constraint Propagation — Models, Techniques, Implementation*. Doctoral dissertation, Saarland University, 2009.

↑ pp. 10, 43

[TFF12] E. C. Teppan, G. Friedrich, and A. A. Falkner. QuickPup: A heuristic backtracking algorithm for the partner units configuration problem. In *Proceedings of the 24th Conference on Innovative Applications of Artificial Intelligence (IAAI'12)*. AAAI, 2012.

↑ p. 18

[Tod91] S. Toda. PP is as hard as the polynomial-time hierarchy. *SIAM Journal on Computing*, 20(5):865–877, 1991.

↑ p. 55

[Val79a] L. G. Valiant. The complexity of computing the permanent. *Theoretical Computer Science*, 8(2):189–201, 1979.

↑ p. 55

[Val79b] L. G. Valiant. The complexity of enumeration and reliability problems. *SIAM Journal on Computing*, 8(3):410–421, 1979.

↑ p. 55

[vHK06] W.-J. van Hoeve and I. Katriel. Global constraints. In *Handbook of Constraint Programming*, volume 2 of *Foundations of Artificial Intelligence*, pp. 169–208. Elsevier, 2006.

↑ pp. 10, 17, 21, 22

[VJ05] G. Verfaillie and N. Jussien. Constraint solving in uncertain and dynamic environments: A survey. *Constraints*, 10(3):253–281, 2005.

↑ p. 10

[VV86] L. Valiant and V. V. Vazirani. NP is as easy as detecting unique solutions. *Theoretical Computer Science*, 47(0):85–93, 1986.

↑ p. 55

[Wal96] M. Wallace. Practical applications of constraint programming. *Constraints*, 1:139–168, 1996.

↑ p. 9

[WGS03] R. Williams, C. P. Gomes, and B. Selman. Backdoors to typical case complexity. In *Proceedings of the 18th International Joint Conference on Artificial Intelligence (IJCAI'03)*, pp. 1173–1178, 2003.

↑ p. 67

[WNS97] M. Wallace, S. Novello, and J. Schimpf. ECLiPSe: A platform for constraint logic programming. *ICL Systems Journal*, 12(1):137–158, 1997.

↑ pp. 10, 17

[Živ09] S. Živný. *The complexity and expressive power of valued constraints*. Doctoral dissertation, University of Oxford, 2009.

↑ p. 64