Journal of Artificial Intelligence Research 45 (2012) 47–78 Submitted 03/12; published 09/12

The Tractability of CSP Classes Defined by Forbidden Patterns

David A. Cohen DAVE@CS.RHUL.AC.UK
Department of Computer Science
Royal Holloway, University of London
Egham, Surrey, UK

Martin C. Cooper COOPER@IRIT.FR
IRIT
University of Toulouse III, 31062 Toulouse, France

Páidí Creed P.CREED@QMUL.AC.UK
School of Mathematical Sciences
Queen Mary, University of London
Mile End, London, UK

Dániel Marx DMARX@CS.BME.HU
Computer and Automation Research Institute
Hungarian Academy of Sciences (MTA SZTAKI)
Budapest, Hungary

András Z. Salamon ANDRAS.SALAMON@ED.AC.UK
Laboratory for Foundations of Computer Science
School of Informatics, University of Edinburgh, UK

Abstract

The constraint satisfaction problem (CSP) is a general problem central to computer science and artificial intelligence. Although the CSP is NP-hard in general, considerable effort has been spent on identifying tractable subclasses. The main two approaches consider *structural* properties (restrictions on the hypergraph of constraint scopes) and *relational* properties (restrictions on the language of constraint relations). Recently, some authors have considered *hybrid* properties that restrict the constraint hypergraph and the relations simultaneously.

Our key contribution is the novel concept of a *CSP pattern* and classes of problems defined by *forbidden patterns* (which can be viewed as forbidding generic sub-problems). We describe the theoretical framework which can be used to reason about classes of problems defined by forbidden patterns. We show that this framework generalises certain known hybrid tractable classes.

Although we are not close to obtaining a complete characterisation concerning the tractability of general forbidden patterns, we prove a dichotomy in a special case: classes of problems that arise when we can only forbid binary negative patterns (generic sub-problems in which only disallowed tuples are specified). In this case we show that all (finite sets of) forbidden patterns define either polynomial-time solvable or NP-complete classes of instances.

1. Introduction

In the constraint satisfaction paradigm we consider computational problems in which we have to assign values (from a *domain*) to *variables*, under some *constraints*. Each constraint limits the (simultaneous) values that a list of variables (its *scope*) can be assigned. In a typical situation some pair of variables might represent the starting times of two jobs in a machine shop scheduling problem. A reasonable constraint would require a minimum time gap between the values assigned to these two variables.

Constraint satisfaction has proved to be a useful modelling tool in a variety of contexts, such as scheduling, timetabling, planning, bio-informatics and computer vision. This has led to the development of a number of successful constraint solvers. Unfortunately, solving general constraint satisfaction problem (CSP) instances is NP-hard and so there has been significant research effort into finding tractable fragments of the CSP.

In principle we can stratify the CSP in two quite distinct and natural ways. The structure of the constraint scopes of an instance of the CSP can be thought of as a hypergraph where the variables are the vertices, or more generally as a relational structure. We can find tractable classes by restricting this relational structure, while allowing arbitrary constraints on the resulting scopes (Dechter & Pearl, 1987). Sub-problems of the general constraint problem obtained by such restrictions are called structural. Alternatively, the set of allowed assignments to the variables in the scope can be seen as a relation. We can choose to allow only specified kinds of constraint relations, but allow these to interact in an arbitrary structure (Jeavons, Cohen, & Gyssens, 1997). Such restrictions are called relational or language-based.

Structural subclasses are defined by specifying a set of hypergraphs (or relational structures) which are the allowed structures for CSP instances. It has been shown that tractable structural classes are characterised by limiting appropriate (structural) width measures (Dechter & Pearl, 1989; Freuder, 1990; Gyssens, Jeavons, & Cohen, 1994; Gottlob, Leone, & Scarcello, 2002; Marx, 2010a, 2010b). For example, a tractable structural class of binary CSPs is obtained whenever we restrict the constraint structure (which is a graph in this case) to have bounded tree width (Dechter & Pearl, 1989; Freuder, 1990). In fact, it has been shown that, subject to certain complexity-theoretic assumptions, the only structures which give rise to tractable CSPs are those with bounded (hyper-)tree width (Dalmau, Kolaitis, & Vardi, 2002; Grohe, 2006, 2007; Marx, 2010a, 2010b).

Relational subclasses are defined by specifying a set of constraint relations. The complexity of the subclass arising from any such restriction is precisely determined by the so called *polymorphisms* of the set of relations (Bulatov, Jeavons, & Krokhin, 2005; Cohen & Jeavons, 2006). The polymorphisms specify that, whenever some set of tuples is in a constraint relation, then it cannot be the case that a particular tuple (the result of applying the polymorphism) is not in the constraint relation. It is thus the relationship between allowed tuples and disallowed tuples inside the constraint relations that is of key importance to the relational tractability of any given class of instances. Whilst a general dichotomy has not yet been proven for the relational case, many dichotomies on sub-problems have been obtained, for instance those by Bulatov (2003), Bulatov et al. (2005) or Bulatov (2006).

Using only structural or only relational restrictions limits the possible subclasses that can be defined. By allowing restrictions on both the structure and the relations we are able to identify new tractable classes. We call these restrictions *hybrid* reasons for tractability.

Several hybrid results have been published for binary CSPs (Jégou, 1993; Weigel & Bliek, 1998; Cohen, 2003; Salamon & Jeavons, 2008; Cooper, Jeavons, & Salamon, 2010; Cooper & Živný, 2011b). Instead of looking at the set of constraint scopes or the constraint language, these results captured tractability based on the properties of the (coloured) microstructure of CSP instances. The *microstructure* of a binary CSP instance is the graph $\langle V, E \rangle$ where V is the set of possible assignments of values to variables and E is the set of pairs of mutually consistent variable-value assignments (Jégou, 1993). In the *coloured microstructure*, the vertices representing an assignment to variable v_i are labelled by a colour representing variable v_i. This maintains the distinction between assignments to different variables.

The coloured microstructure of a CSP instance captures both the structure and the relations of a CSP instance and so it is a natural place to look for tractable classes which are neither purely structural nor purely relational. Of the results on (coloured) microstructure properties, three are of particular note. First it was observed that the class of instances with a perfect microstructure is tractable (Salamon & Jeavons, 2008). This is a proper generalisation of the well known hybrid tractable CSP class whose instances allow arbitrary unary constraints and in which every pair of variables is constrained to be not equal (Régin, 1994; van Hoeve, 2001), and of the hybrid class whose microstructure is triangulated (Jégou, 1993; Weigel & Bliek, 1998; Cohen, 2003). The perfect microstructure property excludes an infinite set of induced subgraphs from the microstructure.

Secondly, the Joint Winner Property (JWP) (Cooper & Živný, 2011b) applied to CSPs provides a different hybrid class that also strictly generalises the class of CSP instances with a disequality constraint ($\neq$) between every pair of variables and an arbitrary set of unary constraints, but does so by forbidding a single pattern (a subgraph) in the coloured microstructure. The JWP has been generalized to hierarchies of soft non-binary constraints (Cooper & Živný, 2011a), including, for example, soft hierarchical global cardinality constraints, by reduction to a minimum convex cost flow problem.

Thirdly, the so called broken-triangle property properly extends the structural notion of acyclicity to a more interesting hybrid class (Cooper et al., 2010). The broken triangle property is specified by excluding a particular pattern in the coloured microstructure. It is the notion of forbidden pattern that we study in this paper. We therefore work directly with the CSP instance (or equivalently its coloured microstructure) rather than its microstructure abstraction which is a simple graph. This allows us to introduce a language for expressing hybrid classes in terms of forbidden patterns, providing a framework in which to search for novel hybrid tractable classes. In the case of binary negative patterns we are able to characterise all tractable (finite sets of) forbidden patterns. We also state a necessary condition for the tractability of a (finite set of) general patterns.

1.1 Contributions

In this paper we generalise the definition of a CSP instance to that of a CSP pattern which has three types of tuple in its constraint relations, tuples which are explicitly allowed/disallowed and tuples which are labelled as *unknown*[1]. By defining a natural notion of containment of patterns in a CSP, we are able to describe problems defined by *forbidden patterns*: a class of CSP instances defined by forbidding a particular pattern χ are exactly those instances that do not contain χ. We use this framework to capture tractability by identifying local patterns of allowed *and* disallowed tuples (within small groups of connected constraints) whose absence is enough to guarantee tractability.

Using the concept of forbidden patterns, we lay foundations for a theory that can be used to reason about classes of CSPs defined by hybrid properties. Since this is the first work of this kind, we primarily focus on the simplest case: binary patterns in which tuples are either disallowed or unknown (called *negative patterns*). We give a large class of binary negative patterns which give rise to intractable classes of problems and, using this, show that any negative pattern that defines a tractable class of problems must have a certain structure. We are able to prove that this structure is also enough to guarantee tractability thus providing a dichotomy for tractability defined by forbidding binary negative patterns. Importantly, our intractability results also allow us to give a necessary condition on the form of general tractable patterns.

The remainder of the paper is structured as follows. In Section 2 we define constraint satisfaction problems, and give other definitions used in the paper. Then, in Section 3, we define the notion of a CSP pattern and describe classes of problems defined by forbidden patterns. We give some examples of tractable classes defined by forbidden patterns on three variables. In Section 4 we show that one must take the size of patterns into account to have a notion of maximal classes defined by forbidding patterns. In general, we are not yet able to make any conjecture concerning a dichotomy for hybrid tractability defined by general forbidden patterns. However, in Section 5 we are able to give a necessary condition for such a class to be tractable and in Section 6 prove the dichotomy for negative patterns. Finally, in Section 7 we summarise our results and discuss directions for future research.

2. Preliminaries

Definition 2.1. *A CSP instance is a triple $\langle V, D, C\rangle$ where:*

- *V is a finite set of **variables** (with $n = |V|$).*
- *D is a finite set called the **domain** (with $d = |D|$).*
- *C is a set of **constraints**. Each constraint $c \in C$ is a pair $c = \langle \sigma, \rho\rangle$ where:*
  - *σ is a list of distinct variables called the **scope** of c.*
  - *ρ is a relation over D of arity $|\sigma|$ called the **relation** of c. It is the set of tuples allowed by c.*

*A **solution** to the CSP instance $P = \langle V, D, C\rangle$ is a mapping $s : V \to D$ where, for each $\langle \sigma, \rho\rangle \in C$ we have $s(\sigma) \in \rho$ (where $s(\sigma)$ represents the tuple resulting from the application of s component-wise to the list of variables σ).*

1. This can be viewed as the natural generalisation of the CSP to a three-valued logic.

For simplicity of presentation, we assume that all variables have the same domains. Unary constraints can be used to impose different domains for different variables.

The arity of a CSP is the largest arity of any of its constraint scopes. Our long-term aim is to identify all tractable subclasses of the CSP problem which can be detected in polynomial time. In this paper we describe a general theory of forbidden patterns for arbitrary arity but only consider the implications of the new theory for tractable classes of arity two (binary) problems specified by finite sets of forbidden patterns. In such cases we are certain that class membership can be decided in polynomial time.

The CSP decision problem, which asks whether a particular CSP instance has a solution, is already NP-complete for binary CSPs. For example, there is a straightforward reduction from graph colouring to this problem in which the set of colours is used as the domain of the CSP instance, vertices i of the graph map to CSP variables v_i, and edges $\{i, j\}$ map to disequality constraints $v_i \neq v_j$.

It will sometimes be convenient in this paper to use an equivalent functional formulation of a constraint. In this alternative formulation the scope σ of the constraint $\langle \sigma, \rho \rangle$ is abstracted to a set of variables and each possible assignment is seen as a function $f : \sigma \to D$. The constraint relation in this alternative view is then a function from the set of possible assignments, D^σ, into the set $\{T, F\}$ where, by convention, the tuples which occur in the constraint relation are those which map to T. It follows that any assignment to the set of all variables is allowed by $\langle \sigma, \rho \rangle$ when its restriction to σ is mapped to T by ρ.

Definition 2.2. *For any function $f : X \to Y$ and $S \subset X$, the notation $f|_S$ means the function with domain S satisfying $f|_S(x) = f(x)$ for all $x \in S$.*

*Given a set V of variables and a domain D, a **constraint** in functional representation is a pair $\langle \sigma, \rho \rangle$ where $\sigma \subseteq V$ and $\rho : D^\sigma \to \{T, F\}$. A **CSP instance** in functional representation is a triple $\langle V, D, C \rangle$ where C is a set of constraints in functional representation.*

*A **solution** (to a CSP instance $\langle V, D, C \rangle$ in functional representation) is a mapping $s : V \to D$ where, for each $\langle \sigma, \rho \rangle \in C$ we have $\rho(s|_\sigma) = T$.*

The functional formulation is clearly equivalent to the relational formulation and we will use whichever seems more appropriate throughout the paper. The choice will always be clear from the context.

The following notions are standard in the study of the CSP. A **binary** CSP instance is one where the maximum arity of any constraint scope is two. The **subproblem** of I on variables $U \subseteq V$ is the instance $\langle U, D, C_U \rangle$ where C_U is the set of constraints $\langle \sigma, \rho \rangle \in C$ such that $\sigma \subseteq U$. The instance I is **arc-consistent** if $\forall v_1, v_2 \in V$, each solution to the subproblem of I on $\{v_1\}$ can be extended to a solution to the subproblem of I on $\{v_1, v_2\}$. The **constraint graph** of a binary CSP instance $I = \langle V, D, C \rangle$ is the graph with vertices V and edges the set of scopes of binary constraints in C. Since it is often convenient to consider that a (possibly irrelevant) constraint exists between every pair of variables, we introduce the refined notion of true constraint graph.

Definition 2.3. *A binary constraint between v_1 and v_2 is **improper** if it allows every pair of values allowed by the unary constraints on v_1 and v_2, and **proper** otherwise.*

*The **true constraint graph** of a binary CSP instance is the constraint graph of the instance after removing any improper binary constraints.*

We may also sometimes need to disregard unary constraints so we have the following.

Definition 2.4. *The **binary reduction** of a CSP instance is obtained by removing from the constraint set all those constraints whose scope does not have arity two.*

3. Forbidden Patterns in CSP

In this paper we explain how we can define classes of CSP instances by forbidding the occurrence of certain patterns. A CSP pattern is a generalisation of a CSP instance. In a CSP pattern we define the relations relative to a three-valued logic on $\{T, F, U\}$, meaning that the pattern can be seen as representing the set of CSP instances in which each undefined value U is replaced by either T or F. Forbidding a CSP pattern is equivalent to simultaneously forbidding all these instances as sub-problems.

Definition 3.1. *We define a three-valued logic on $\{T, F, U\}$, where U stands for* unknown *or* undefined*. The set $\{T, F, U\}$ is partially ordered so that $U < T$ and $U < F$ but T and F are incomparable. Let D be a finite set. A k-ary **three-valued relation** on D is a function $\rho : D^k \to \{T, F, U\}$. Given k-ary three-valued relations ρ and ρ', we say ρ **realises** ρ' if*

$$\forall x \in D^k \ \rho(x) \geq \rho'(x).$$

We can extend the definition of a CSP or constraint pattern to include additional structure on the set of variable names or the set of domain values, as a set of relations on the set in question. Adding structure makes patterns more specific. We can therefore capture larger, and hence more interesting, tractable classes. For example, when the domain is totally ordered we can define the tractable max-closed class (Jeavons & Cooper, 1995); when we have an independent total order for the domain of each variable we can capture the renamable Horn class (Green & Cohen, 2003); and placing an order on variables in a pattern will allow us to define the class of tree-structured CSP instances.

Definition 3.2. *A **CSP pattern** is a quadruple $\chi = \langle V, D, C, S\rangle$, where:*

- *V is the set of **variables**, with an associated relational structure with universe V.*
- *D is the **domain**, with an associated relational structure with universe D.*
- *C is a set of **constraint patterns**. Each constraint pattern $c \in C$ is a pair $c = \langle\sigma, \rho\rangle$, where $\sigma \subseteq V$, the **scope** σ of c, is a list of distinct variables and $\rho : D^\sigma \to \{T, F, U\}$ is the **three-valued relation** (in functional representation) of c. A constraint pattern is **non-trivial** if its three-valued relation maps at least one tuple to $\{T, F\}$.*
- *S is the **structure**, a set consisting of the relational structures associated with its variable set and its domain.*

*The **arity** of a CSP pattern χ is the maximum arity of any constraint pattern $\langle\sigma, \rho\rangle$ of χ.*

Our most basic type of pattern is one which employs no structure, with S empty. We also frequently require patterns which use a disequality relation $\neq$, applied to every pair from some specified subset of variables, and we allow several such subsets of variables in the structure.

In this paper the relations occurring in the structure all have arity two, and their interpretation is limited to a few selected binary relations representing disequality or a partial order. When the structure of a variable set or domain is clear from the context, we will not explicitly mention it. Different kinds of structure can be imposed on CSP patterns; indeed structures specified by more general relations would be an interesting area for future study.

The weakest structure that we will consider only allows us to say when two variables are distinct. Thus the structure S of a CSP pattern is then simply a set of disequalities between subsets of variables. In this paper we denote such disequalities by $\mathrm{NEQ}(v_1, \ldots, v_r)$ meaning that variables $v_1, v_2, \ldots, v_r$ are all pairwise distinct. A pattern with such a structure will be called **flat**. Indeed, in this paper we are mostly concerned with flat patterns. If two variables occur together in the scope of some constraint pattern, then we also assume that S implicitly includes the disequality $\mathrm{NEQ}(v_1, v_2)$.

Thus CSP patterns are defined using relational structures with three sorts: for variables, for domain values, and for variable-value assignments. The constraint patterns of a CSP pattern are then three-valued relations over the sort of variable-value assignments. If a CSP pattern is flat then its structure specifies relations over the sort of variables. A partial order over the variables is also a relation over the sort of variables, and partial orders over domain values are relations over the sort of domain values.

For simplicity of presentation, we assume throughout this paper that no two constraint patterns in C have the same scope (and that, in the case of CSP instances, that no two constraints have the same scope). We will represent binary CSP patterns by simple diagrams. Each oval represents the domain of a variable, each dot a domain value. The tuples in constraint patterns with value F are shown as dashed lines, those with value T as solid lines and those with value U are not depicted at all.

Definition 3.3. *A constraint pattern $\langle \sigma, \rho \rangle$ will be called **negative** if ρ never takes the value T. A CSP pattern χ is negative if every constraint pattern in χ is negative.*

3.1 Patterns, CSPs and Occurrence

In a CSP instance it is implicitly assumed that all variables and all domain values are distinct. This is equivalent to the existence of implicit disequalities NEQ between all variable names and all domain values. A CSP instance is just a CSP pattern (with a structure that all variables and all domain values are distinct) in which the three-valued relations of the constraint patterns never take the value U. That is, we decide for each possible tuple whether it is in the relation or not. Furthermore, in a CSP instance, for each pair of variables we assume that a constraint exists with this scope; if no explicit constraint is given on this scope, then we assume that the relation is complete, i.e. it contains all tuples. This can be contrasted with CSP patterns for which the absence of an explicit constraint on a pair of variables implies that the truth value of each tuple is undefined.

In order to define classes of CSP instances by forbidding patterns, we require a formal definition of an occurrence (containment) of a pattern within an instance. We define the more general notion of containment of one CSP pattern within another pattern. Informally, the names of the variables and domain elements of a CSP pattern are inconsequential and a containment allows a renaming of the variables and the domain values of each variable. Thus, in order to define the containment of patterns, we firstly require a formal definition of a renaming. In an arbitrary renaming, unless explicitly prohibited by a disequality in the structure, two distinct variables may map to the same variable and two distinct domain values may map to the same domain value. Furthermore, when a pattern occurs in another, it may use only a subset of the variables of the second pattern; hence the notion we require is known as a renaming-extension.

A **domain labelling** of a set of variables is just an assignment of domain values to those variables. Variable and domain renaming induces a mapping on the domain labellings of scopes of constraints: we simply assign the renamed domain values to the renamed variables. There is a natural way to extend this mapping of domain labellings to a mapping of a constraint pattern: the truth-value of each mapped domain labelling is the same as the truth-value of the original domain labelling. However, it may occur that two domain labellings of some scope map to the same domain labelling, so instead the resulting value is taken to be the greatest of the original truth-values. (In order for this process to be well-defined, if two domain labellings of a constraint are mapped to the same domain labelling, then their original truth-values must be comparable.) This leads to the following formal definition of a renaming-extension which is the first step towards the definition of containment.

Definition 3.4. *Let $\chi = \langle V, D, C, S\rangle$ and $\chi' = \langle V', D', C', S'\rangle$ be CSP patterns.*

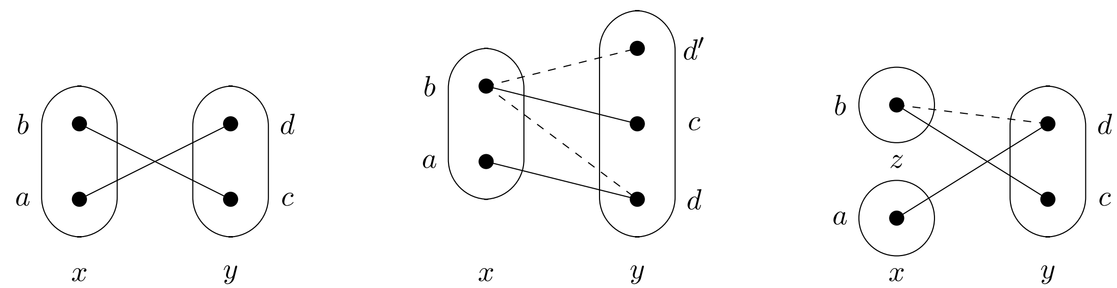
*We say that χ' is a **renaming-extension** of χ if there exist a variable-renaming function $s : V \to V'$ and a domain-renaming function $t : V \times D \to D'$ such that s, t and the assignment-renaming function $F : V \times D \to V' \times D'$ induced by (s, t) and defined by $F(\langle v, a\rangle) = \langle s(v), t(v, a)\rangle$ satisfy:*

- *For each constraint pattern $\langle \sigma, \rho\rangle \in C$, for any two domain labellings $\ell, \ell' \in D^\sigma$ for which $F(\ell) = F(\ell')$, we have that $\rho(\ell)$ and $\rho(\ell')$ are comparable, where $F(\ell)$ denotes the assignment $f : s(\sigma) \to D'$ such that $\forall v \in \sigma,\ f(s(v)) = t(v, \ell(v))$.*
- *$C' = \{\langle s(\sigma), \rho'\rangle \mid \langle \sigma, \rho\rangle \in C\}$, where, for each assignment $f : s(\sigma) \to D'$, $\rho'(f) = U$ if $F(\ell) \neq f$ for every $\ell \in D^\sigma$, and $\rho'(f) = \max\{\rho(\ell) \mid F(\ell) = f\}$ otherwise.*
- *If χ has any structure, then s, t and F preserve this structure. The mapping s induces a homomorphism between the relational structures of the variable-sets, and mapping t induces a homomorphism between the relational structures of the domains. (In particular, if $\mathrm{NEQ}(v_1, v_2) \in S$, then $s(v_1) \neq s(v_2)$ and $\mathrm{NEQ}(s(v_1), s(v_2)) \in S'$.)*

We will use patterns to define sets of CSP instances by forbidding the occurrence (containment) of the patterns in the CSP instances. In this way we will be able to characterise tractable subclasses of the CSP. Informally, a pattern χ is said to occur in a CSP instance P if we can find a sub-problem Q of P (formed by taking subsets of variables and domains) which realises χ. Q realises χ if, after renaming of variables and domain values in χ, each constraint pattern in χ is realised by the corresponding constraint in Q. By Definition 3.4, during a renaming-extension, extra variables, domain values and disequalities can be introduced. Thus we only need to combine the notions of renaming-extension and realisation to formally define what we mean by a pattern occurring in another pattern (and, in particular, in a CSP instance).

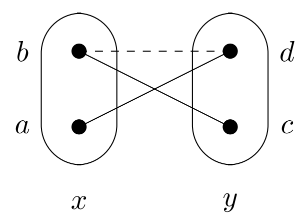
Definition 3.5. *We say that a CSP pattern χ* ***occurs*** *in a CSP pattern $P = \langle V, D, C, S\rangle$ (or that P* ***contains*** *χ), denoted $\chi \rightarrow P$, if there is a renaming-extension $\langle V, D, C', S\rangle$ of χ where, for every constraint pattern $\langle \sigma, \rho'\rangle \in C'$ there is a constraint pattern $\langle \sigma, \rho\rangle \in C$ and, furthermore, ρ realises ρ'.*

Pattern 1



(i) (ii) (iii)

Pattern 2



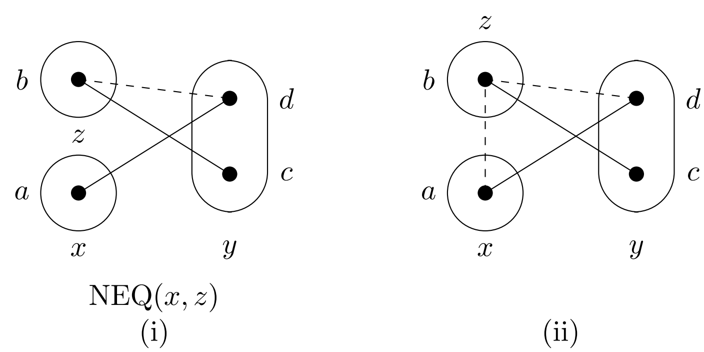
Example 3.6. This example describes three simple containments. Consider the three CSP patterns, Pattern 1(i)–(iii). These patterns occur in, or are contained in, Pattern 2 by the mappings F_1, F_2, and F_3, respectively, which we will now describe.

F_1 is simply a bijection. Although the patterns are different, this is a valid containment of Pattern 1(i) into Pattern 2 because the three-valued relation of Pattern 2 is a realisation of the three-valued relation in Pattern 1(i): we are replacing $(b, d) \mapsto U$ by $(b, d) \mapsto F$.

F_2 maps (x, a), (x, b), and (y, c) to themselves, and maps both (y, d) and (y, d') to (y, d). This merging of domain elements is possible because the values of the three-valued constraint relation of Pattern 1(ii) are comparable on tuples involving the assignments (y, d) and (y, d') and, furthermore, the restriction of the three-valued relation of Pattern 1(ii) to either of these two assignments is realised by the three-valued constraint relation of Pattern 2: $(b, d) \mapsto F$ and $(a, d) \mapsto T$. For example, we are replacing $(a, d') \mapsto U$ by $(a, d) \mapsto T$. In a similar manner, Pattern 1(i) is also contained in Pattern 2 by the simple mapping F'_1 which maps both of (x, b), (x, a) to (x, b) and both of (y, c), (y, d) to (y, c).

Finally, F_3 maps (y, c) and (y, d) to themselves, and maps (x, a) and (z, b) in Pattern 1(iii) to (x, a) and (x, b), respectively, in Pattern 2. This merging of variables is possible because the three-valued relations agree and because there is no NEQ(x, z) structure in Pattern 1(iii). $\square$

Pattern 3



Throughout this paper, we use the notation NEQ$(v_1, \ldots, v_r)$ to denote the fact that the variables $v_1, \ldots, v_r$ of a CSP pattern are distinct. It is worth discussing what this structure implies as far as Definition 3.4 is concerned. Structure in the source pattern must be preserved in the target pattern. Thus Pattern 1(iii) occurs in Pattern 3(i), but Pattern 3(i) is not contained in Pattern 1(iii) since the structure NEQ(x, z) is not preserved in the target pattern. The structure NEQ(v_1, v_2) is considered to be preserved by a renaming-extension χ' of χ even if it is not explicitly given in χ' but is implicit, for example, due to the existence of a non-trivial constraint pattern $\langle \sigma, \rho \rangle$ in χ' such that $v_1, v_2 \in \sigma$. As an example, consider the two CSP patterns, Pattern 3(i)–(ii). Pattern 3(i) can be mapped to Pattern 3(ii) by a simple bijection so that the three-valued relation of Pattern 3(ii) is a realisation of the three-valued relation in Pattern 3(i). The structure NEQ(x, z) is considered to be preserved by this mapping due to the existence of a non-trivial constraint pattern between variables x and z in Pattern 3(ii). Hence, Pattern 3(i) occurs in Pattern 3(ii).

Before continuing we need to define what we mean when we say that a class of CSP instances is definable by forbidden patterns.

Definition 3.7. *Let C be any class of CSP instances with maximum arity k. We say that C is **definable by forbidden patterns** if there is some set of patterns $\mathcal{X}$ for which the set of CSP instances of maximum arity k in which none of the patterns in $\mathcal{X}$ occur are precisely the instances in C.*

Notation: Let $\mathcal{X}$ be a set of CSP patterns with maximum arity k. We will use CSP$(\overline{\mathcal{X}})$ to denote the set of CSP instances in which no element $\chi \in \mathcal{X}$ occurs. When $\mathcal{X}$ is a singleton $\{\chi\}$ we will use CSP$(\overline{\chi})$ to denote CSP$(\overline{\{\chi\}})$.

In this paper, we only consider classes CSP$(\overline{\mathcal{X}})$ for sets $\mathcal{X}$ of CSP patterns which are binary in the sense that all constraint patterns have scope of size exactly two.

For all $\mathcal{X}$ such that all patterns in $\mathcal{X}$ are binary, CSP$(\overline{\mathcal{X}})$ is closed under arc consistency (in the sense that the arc consistency closure of any instance $I \in$ CSP$(\overline{\mathcal{X}})$ belongs to

CSP($\overline{\mathcal{X}}$)) and any other operation which only updates unary constraints. Indeed, changing unary constraints cannot introduce any of the patterns $\mathcal{X}$ in any instance $I \in$ CSP($\overline{\mathcal{X}}$).

3.2 Tractable Patterns

In this paper we will define, by forbidding certain patterns, tractable subclasses of the CSP. Furthermore, we will give examples of truly hybrid classes (i.e. classes not definable by purely relational or purely structural properties).

Definition 3.8. *A finite set of patterns $\mathcal{X}$ is **intractable** if* CSP($\overline{\mathcal{X}}$) *is NP-hard. It is **tractable** if there is a polynomial-time algorithm to solve* CSP($\overline{\mathcal{X}}$). *A single pattern χ is tractable (intractable) if $\{\chi\}$ is tractable (intractable). (We assume throughout this paper that P$\neq$ NP, and therefore that the sets of tractable and intractable patterns are disjoint.)*

It is worth observing that classes of CSP instances defined by forbidding patterns do not have a fixed domain. Recall, however, that each CSP instance has a finite domain. Any structure present in a CSP instance is assumed given as part of the instance. In particular, all variables in a CSP instance are assumed to be distinct. For finite sets of patterns $\mathcal{X}$, the number of possible renaming-extensions into a particular instance P is polynomial in the size of P. Hence we can determine whether an instance lies in CSP($\overline{\mathcal{X}}$) by exhaustive search in polynomial time.

We will need the following simple lemmas for our proofs of intractability results in later sections of this paper.

Lemma 3.9. *If $\chi_1 \to \chi_2$ and $\chi_2 \to \chi_3$, then $\chi_1 \to \chi_3$.*

Proof. If $\chi \to \chi'$, then each constraint pattern $\langle \sigma, \rho \rangle$ of χ maps to a constraint pattern $\langle \sigma', \rho' \rangle$ such that ρ' realises ρ. The transitivity of $\to$ follows from the following facts:

- The realisation operation is transitive.
- If $\chi_1 \to \chi_2$ and $\chi_2 \to \chi_3$, then by Definition 3.4, any structure in χ_1 is preserved in χ_2 and hence in χ_3. □

Lemma 3.10. *Let $\mathcal{X}$ and $\mathcal{T}$ be sets of CSP patterns and suppose that for every pattern $\tau \in \mathcal{T}$, there is some pattern $\chi \in \mathcal{X}$ for which $\chi \to \tau$. Then* CSP($\overline{\mathcal{X}}$) $\subseteq$ CSP($\overline{\mathcal{T}}$).

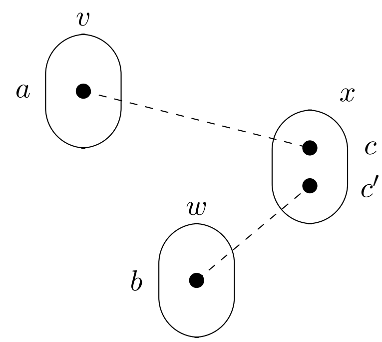
Proof. Let $P \in$ CSP($\overline{\mathcal{X}}$), so $\chi \not\to P$ for each $\chi \in \mathcal{X}$. Then we cannot have $\tau \to P$ for any $\tau \in \mathcal{T}$, since this would imply that there exists some $\chi \in \mathcal{X}$ such that $\chi \to \tau \to P$ and hence that $\chi \to P$ by Lemma 3.9. Hence, $P \in$ CSP($\overline{\mathcal{T}}$). □

Corollary 3.11. *Let $\mathcal{X}$ and $\mathcal{T}$ be sets of CSP patterns and suppose that for every pattern $\tau \in \mathcal{T}$, there is some pattern $\chi \in \mathcal{X}$ for which $\chi \to \tau$.*

We then have that CSP($\overline{\mathcal{T}}$) *is intractable if* CSP($\overline{\mathcal{X}}$) *is intractable and conversely, that* CSP($\overline{\mathcal{X}}$) *is tractable whenever* CSP($\overline{\mathcal{T}}$) *is tractable.*

Finally, we give some examples of tractable patterns. The first example is a *negative* pattern since the only truth-values in the relations are F and U.

Pattern 4 A very simple negative pattern.

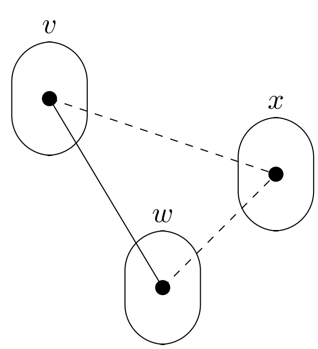


NEQ(v, w, x)

Example 3.12. Consider Pattern 4. This defines a class of CSPs which is trivially tractable. Forbidding Pattern 4 ensures that there are no paths of more than two variables in the true constraint graph. Thus, any problem forbidding Pattern 4 can be decomposed into a set of independent sub-problems, each with at most two variables. □

Example 3.13. Cooper and Živný (2011b) showed that forbidding the pattern Negtrans shown in Pattern 5 describes a tractable class of CSP instances. This can be seen as a generalisation of the well-known tractable class of problems, AllDifferent+unary (Costa, 1994; Régin, 1994; van Hoeve, 2001): an instance of this class consists of a set of variables V, a set of arbitrary unary constraints on V, and the constraint $v \neq w$ defined on each pair of distinct variables $v, w \in V$. Forbidding Negtrans is equivalent to saying that disallowed tuples form a transitive relation, i.e. if $(\langle v, a\rangle, \langle x, b\rangle)$ and $(\langle x, b\rangle, \langle w, c\rangle)$ are disallowed then $(\langle v, a\rangle, \langle w, c\rangle)$ must also be disallowed. Thus Negtrans does not occur in any binary CSP instance in the class AllDifferent+unary by the transitivity of equality (equality being exactly what is disallowed). □

Pattern 5 Negative transitive pattern (Negtrans)



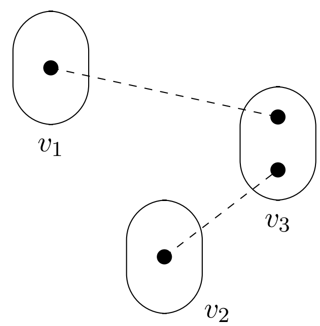
NEQ(v, w, x)

Cooper and Živný (2011b) also recently showed that the tractable class defined by forbidding Pattern 5 (Negtrans) can be extended to soft constraint problems.

3.3 Tractable Patterns with Structure

This paper primarily studies patterns with a weak structure in that the only conditions that are imposed are that variables are distinct. However, it is worth pointing out that adding structure to a pattern allows us to capture larger classes of instances. In Example 3.14 below we show that a forbidden pattern can capture the class of CSPs with tree width 1 by adding a variable-ordering to Pattern 4. In this case pattern containment must preserve the total order. For an ordered pattern χ, we will consider an unordered CSP P to be in CSP($\overline{\chi}$) if there exists some ordering of the variable set of P such that χ is forbidden. In order for χ to define a tractable class, it must be possible to find this ordering in polynomial time. This is the case for the patterns in Examples 3.14 and 3.15.

Pattern 6 Tree structure pattern ($\textsc{Tree}$)



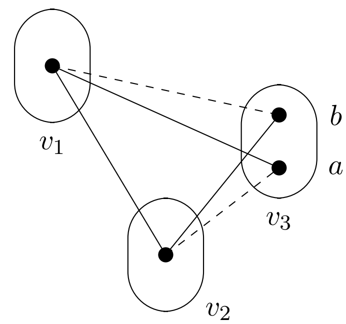
$v_1 < v_2 < v_3$

Example 3.14. Consider the pattern $\textsc{Tree}$, given as Pattern 6. We will show that the class CSP($\overline{\textsc{Tree}}$) is exactly the set of CSPs whose true constraint graph is a forest (i.e. has tree width 1). First, suppose $P \in$ CSP($\overline{\textsc{Tree}}$). Then, there exists some ordering $\pi = (v_1, \ldots, v_n)$ such that each variable shares a proper constraint with at most one variable preceding it in the ordering. On the other hand, suppose P is a CSP whose true constraint graph is a tree. By ordering the vertices according to a pre-order traversal, we obtain an ordering in which each variable shares a proper constraint with at most one variable preceding it in the ordering (its parent); thus, $P \in$ CSP($\overline{\textsc{Tree}}$). $\square$

Example 3.15. Forbidding the pattern BTP shown in Pattern 7 is known as the **broken-triangle property** (Cooper et al., 2010). In order to capture this class by a forbidden pattern we again have to impose a total order on pattern variables. Cooper et al. (2010) proved that the class of CSP instances CSP($\overline{\text{BTP}}$) can be solved in polynomial time and, indeed, that CSP instances in CSP($\overline{\text{BTP}}$) for some unknown total ordering of the variables can be recognised and solved in polynomial time. $\square$

It is easy to see that $\textsc{Tree}$ (shown in Pattern 6) occurs in BTP (with some truth-values U being changed to T). It follows from Lemma 3.10 that CSP($\overline{\textsc{Tree}}$) $\subseteq$ CSP($\overline{\text{BTP}}$). Hence the class CSP($\overline{\text{BTP}}$) includes all CSP instances whose true constraint graph is a tree. However, CSP($\overline{\text{BTP}}$) also includes certain CSP instances whose true constraint graph has

Pattern 7 Broken triangle pattern (BTP)



$v_1 < v_2 < v_3$

tree width r for any value of r: consider, for example, a CSP instance with $r + 1$ variables and an identical constraint between every pair of variables which simply disallows the single tuple $\langle 0, 0\rangle$.

For any tractable forbidden pattern with an order imposed on the variables, we can obtain another tractable class by considering problems forbidding the pattern without this ordering condition. The class obtained is generally smaller, because it is easier to establish containment of the flat pattern. For example, consider Pattern 4 which is the flat version of Pattern 6. We have seen that forbidding Pattern 4 gives rise to the class of CSP instances in which there are no paths of length greater than two in the true constraint graph. On the other hand, forbidding Pattern 6 gives the much larger class of CSP instances in which the true constraint graph has tree width 1.

In the case of the broken-triangle property, we also obtain a strictly smaller tractable class by forbidding Pattern 7 for all triples of variables v_1, v_2, v_3 irrespective of their order. We can easily exhibit a CSP instance that shows this inclusion to be strict: for example, the 3-variable CSP instance over Boolean domains consisting of the two constraints $v_1 = v_2$, $v_1 = v_3$ with the variable ordering $v_1 < v_2 < v_3$. This unordered version of BTP was recently used to obtain a dichotomy for patterns consisting of 2 constraints (Cooper & Escamocher, 2012).

4. On Maximal Tractable Classes Defined by Forbidden Patterns

In relational tractability we can define a maximal tractable sub-problem of the CSP problem given by a set Γ of possible relations. Such a class of relations is maximal if it is not possible to add even one more relation to Γ without sacrificing tractability.

In the case of structural tractability the picture is less clear, since here we measure the complexity of an infinite set of hypergraphs (or, more generally, relational structures). We obtain tractability if we have a bound on some width measure of these structures. Whatever width measure is chosen we have a containment of the class with width bounded by k inside that of the class of width bounded by $k+1$ and so no maximal class is possible (although for each k there is a unique maximal class of structurally tractable instances). In this section, we show that in the case of forbidden patterns the situation is similar.

Definition 4.1. *Let $\chi = \langle V, D, C, S\rangle$ and $\tau = \langle V', D', C', S'\rangle$ be any two flat CSP patterns. We can form the disjoint unions $V \uplus V'$ and $D \uplus D'$. Now, extend each constraint pattern in C to be over the domain $D \uplus D'$ by setting the value of any tuple including elements of D' to be U, and extend similarly the constraint patterns in C': in this way we can define $C \uplus C'$. Also define the structure $S \uplus S'$ by forming the disjoint union of S and S' and adding all disequalities $\mathrm{NEQ}(v, v')$ for all $v \in V$ and $v' \in V'$. Then we set the* ***disjoint union*** *of χ and τ to be $\chi \uplus \tau = \langle V \uplus V', D \uplus D', C \uplus C', S \uplus S'\rangle$.*

Lemma 4.2. *Let χ and τ be flat non-empty (i.e. containing at least one variable) binary CSP patterns. Then*

$$\mathrm{CSP}(\overline{\chi}) \cup \mathrm{CSP}(\overline{\tau}) \subsetneq \mathrm{CSP}(\overline{\chi \uplus \tau})\,.$$

Moreover, we have that $\mathrm{CSP}(\overline{\chi \uplus \tau})$ is tractable whenever $\mathrm{CSP}(\overline{\chi})$ and $\mathrm{CSP}(\overline{\tau})$ are tractable.

Proof. We begin by showing the strict inclusion

$$\mathrm{CSP}(\overline{\chi}) \cup \mathrm{CSP}(\overline{\tau}) \subsetneq \mathrm{CSP}(\overline{\chi \uplus \tau})\,.$$

That the inclusion holds follows directly from Lemma 3.10. Among all patterns in which χ occurs, let χ^- be a pattern with the smallest number of variables. We define τ^- similarly. To see that the inclusion is strict, observe that χ and τ occur in a CSP pattern whose domain is the disjoint union of those for χ^- and τ^-, but whose variable set has size equal to the larger of the variable sets of χ^- and τ^-. Any CSP instance containing this pattern is neither in $\mathrm{CSP}(\overline{\chi})$ nor in $\mathrm{CSP}(\overline{\tau})$. However, we can construct a CSP instance containing this pattern which is contained in $\mathrm{CSP}(\overline{\chi \uplus \tau})$, as the structure of $\chi \uplus \tau$ imposing disequalities between variables of χ and τ means that $\chi \uplus \tau$ is not contained in this pattern: there are simply not enough variables.

Suppose $P \in \mathrm{CSP}(\overline{\chi \uplus \tau})$. If $P \in \mathrm{CSP}(\overline{\chi}) \cup \mathrm{CSP}(\overline{\tau})$ then P can be solved in polynomial time, by the tractability of $\mathrm{CSP}(\overline{\chi})$ and $\mathrm{CSP}(\overline{\tau})$.

So we may suppose that $\chi \rightarrow P$. Choose a particular occurrence of χ in P and let σ denote the set of variables used in the containment. Consider any assignment $t : \sigma \rightarrow D$. Let P_t denote the problem obtained by making this assignment and then enforcing arc-consistency on the resulting problem. This corresponds to adding some new unary constraints to P.

We will show that if τ occurs in P_t then $\chi \uplus \tau$ must occur in P. To see this, observe that any containment of τ in P_m naturally induces a containment of τ in P that extends to a containment of $\chi \uplus \tau$ in P, by considering the occurrence of χ in σ. Thus, we can conclude that $P_t \in \mathrm{CSP}(\overline{\tau})$, and so can be solved in polynomial time.

By construction, any solution to P_t extends to a solution to P by adding the assignment t to the variables σ. Moreover, every solution to P corresponds to a solution to P_t for some $t : \sigma \rightarrow D$. Since the size of χ is fixed, we can iterate over the solutions to χ in polynomial time. If P has a solution, then we will find it as the solution to some P_t. If we find that no P_t has a solution, then we know P does not have a solution. Thus, since we can solve each P_t in polynomial time, we can also solve P in polynomial time. ∎

Corollary 4.3. *No tractable class defined by forbidding a flat pattern is maximal.*

Proof. Let χ be any tractable flat pattern. Consider the pattern $\chi \dot\cup \chi$ defined by the disjoint union of two copies of χ. By Lemma 4.2 we have that $\mathrm{CSP}(\overline{\chi \dot\cup \chi})$ is tractable but also that

$$\mathrm{CSP}(\overline{\chi}) \subsetneq \mathrm{CSP}(\overline{\chi \dot\cup \chi}),$$

and hence $\mathrm{CSP}(\overline{\chi})$ is not a maximal tractable class. □

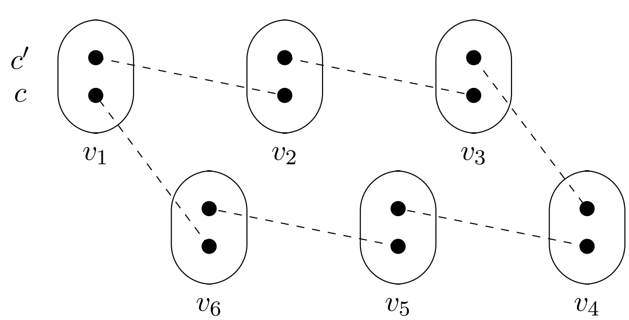
It follows that we cannot characterise tractable forbidden patterns by exhibiting all maximal tractable classes defined by forbidding a pattern (or any finite set of patterns, since by Lemma 4.2 such a finite set can be replaced by a single pattern). Indeed, a consequence of Lemma 4.2 is that we can construct an infinite chain of patterns, such that forbidding each one gives rise to a slightly larger tractable class. Naturally, if we place an upper bound on the size of the patterns then there are only finitely many patterns that we can consider, so maximal tractable classes defined by forbidden patterns of bounded size necessarily exist.

5. Binary Flat Negative Patterns

For the moment, we are not able to make a conjecture concerning the complete characterisation of the complexity of general forbidden patterns, although we conjecture that a dichotomy exists. Nonetheless, by restricting our attention to a special case, forbidden *binary flat negative* patterns, we are able to obtain a dichotomy. Recall that a pattern is flat if the only structure that can be imposed is that variables are distinct, and that it is negative if in all of its constraint patterns $\langle \sigma, \rho \rangle$, ρ never takes the value T.

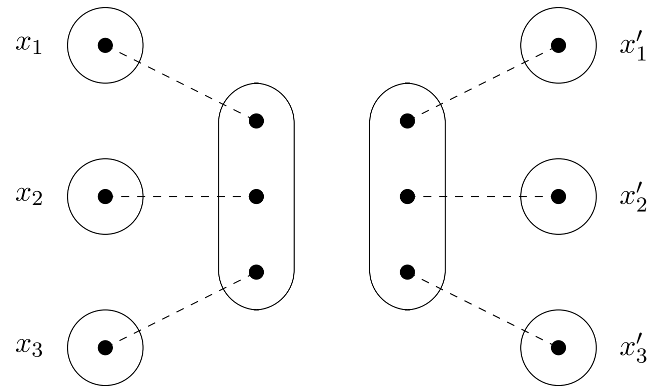
We begin by defining three particular patterns and one infinite class of patterns. We then use these patterns to characterise a very large class of intractable patterns. We prove that any finite set of flat negative patterns not in this class has a simple structure: one of the patterns must be contained in one of a particular set of patterns, which we call *pivots*. This means that any tractable such set of patterns must include a pattern which occurs in a pivot pattern. Furthermore, we demonstrate that forbidding any pivot pattern gives rise to a tractable class. This then leads to a simple characterisation of the tractability of finite sets of binary flat negative patterns.

Pattern 8 Cycle(6)



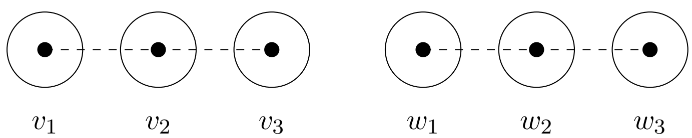
$\mathrm{NEQ}(v_1, \ldots, v_6)$

Pattern 9 VALENCY



$$\mathrm{NEQ}(x_1, x_2, x_3, x'_1) \wedge \mathrm{NEQ}(x'_1, x'_2, x'_3)$$

Pattern 10 PATH



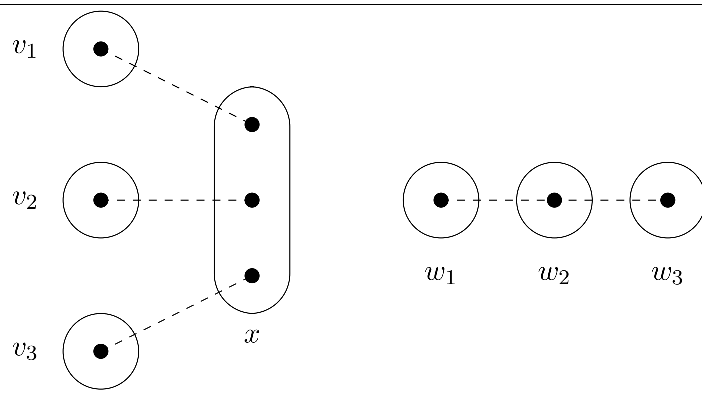
$$\mathrm{NEQ}(v_1, v_2, v_3, w_1) \wedge \mathrm{NEQ}(w_1, w_2, w_3)$$

In Definition 5.1 below, we define the concept of a neg-connected binary pattern. These correspond to binary patterns χ such that the true constraint graph of every realisation of χ as a binary CSP instance is a connected graph. We first generalise the notion of true constraint graph to CSP patterns. We call the resulting graph the negative structure graph.

Definition 5.1. *Let χ be any binary pattern. The vertices of the* ***negative structure graph*** *G are the variables of χ. A pair of vertices is an edge in G if and only if they form a scope in χ whose constraint pattern assigns at least one tuple the value F. We say that a pattern χ is* ***neg-connected*** *if its negative structure graph is connected. In the case of negative patterns, we use the simpler term* ***connected*** *instead of neg-connected.*

Pattern 9 (VALENCY), Pattern 10 (PATH) and Pattern 11 (VALENCY+PATH) are not connected. Note that a pattern which is not connected may occur in a connected pattern (and vice versa). Pattern 8 shows CYCLE(6) which is connected. This is just one example of the generic pattern CYCLE(k) where $k \geq 2$. The only structure for CYCLE(k) is that all variables are distinct, except for the special case $k = 2$ for which the structure also includes NEQ(c, c'). This additional requirement means that CYCLE(2) is composed of a single binary constraint pattern containing two *distinct* disallowed tuples. The following theorem uses these patterns to show that most patterns are intractable.

Pattern 11 Valency+Path



NEQ(v_1, v_2, v_3), NEQ(w_1, w_2, w_3), and NEQ(x, w_2)

Theorem 5.2. *Let $\mathcal{X}$ be any finite set of neg-connected binary patterns. If, for each $\chi \in \mathcal{X}$, at least one of* Cycle(k) *(for some $k \geq 2$),* Valency, Path, *or* Valency+Path *occurs in χ, then $\mathcal{X}$ is intractable.*

Proof. Let $\mathcal{X}$ be a finite set of neg-connected negative binary patterns and let ℓ be the number of variables in the largest element of $\mathcal{X}$.

Assuming at least one of the four patterns occurs in each $\chi \in \mathcal{X}$, we can construct a class of CSPs in which no element of $\mathcal{X}$ occurs and to which we have a polynomial-time reduction from the well-known NP-complete problem 3SAT (Garey & Johnson, 1979).

The construction will involve three gadgets, examples of which are shown in Figure 1. These gadgets each serve a particular purpose:

1. The *cycle gadget*, shown in Figure 1(a) for the special case of 4 variables, enforces that a cycle of Boolean variables $(v_1, v_2, \ldots, v_r)$ all take the same value.

2. The *clause gadget* in Figure 1(b) is equivalent to the clause $v_1 \vee v_2 \vee v_3$, since v_C has a value in its domain if and only if one of the three v_i variables is set to true. We can obtain all other 3-clauses on these three variables by inverting the domains of the v_i variables.

3. The *line gadget* in Figure 1(c), imposes the constraint $v_1 \Rightarrow v_2$. It can also be used to impose the logically equivalent constraint $\neg v_2 \Rightarrow \neg v_1$.

The cycle gadget will be connected to the clause gadget via line gadgets. These three types of gadgets have been specified to ensure that at most one negative edge is adjacent to any vertex in the coloured microstructure, except when the cycle gadget is connected to a line gadget.

Now, suppose that we have an instance Ψ of 3SAT with n propositional variables $X_1, \ldots, X_n$ and m clauses $C_1, \ldots, C_m$.

We begin our construction of a CSP instance P_Ψ to solve the 3SAT instance Ψ by using n copies of the cycle gadget (Figure 1(a)), each with $m(\ell+1)$ variables. For $i = 1, \ldots, n$, the variables along the ith copy of this cycle are denoted by $(v_i^1, v_i^2, \ldots, v_i^{m(\ell+1)})$. In any

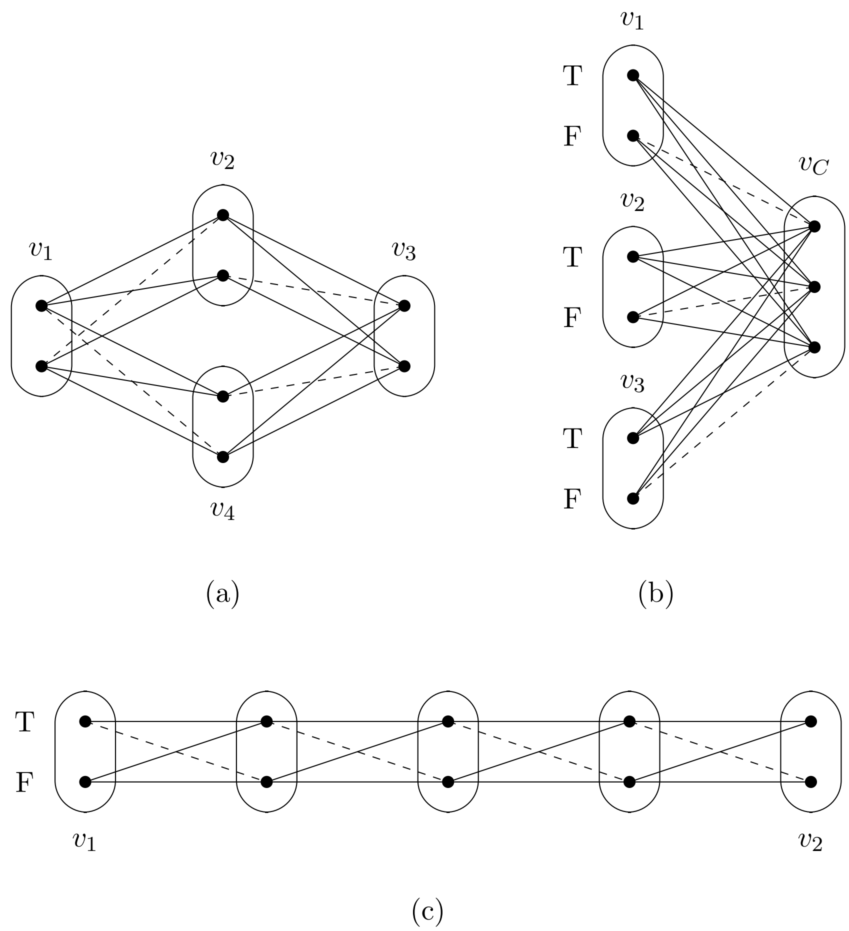


Figure 1: (a) Making copies of the same variable ($v_1 = v_2 = v_3 = v_4$). (b) Imposing the ternary constraint $v_C = v_1 \vee v_2 \vee v_3$. (c) A line of constraints of length 4 which imposes $v_1 \Rightarrow v_2$.

solution to a CSP instance P_Ψ with these and other constraints, we will have that the variables $v_i^j, j = 1, \ldots, m(\ell+1)$ must all have the same value, d_i. We can therefore consider each v_i^j as a copy of X_i.

Consider the clause C_w. There are eight cases to consider but they are all very similar so we will show the details for just one case. Suppose that $C_w \equiv X_i \vee X_j \vee \neg X_k$. We build the clause gadget (Figure 1(b)) with the three Boolean variables being c_w^i, c_w^j and c_w^k and invert the domain of c_w^k since it occurs negatively in C_w. Then any solution s to our constructed CSP must satisfy $s(c_w^i) \vee s(c_w^j) \vee \neg s(c_w^k) = T$.

We complete the insertion of C_w into the CSP instance by adding some line gadgets of length $\ell + 1$ (Figure 1(c)). We connect the cycle gadgets corresponding to X_i, X_j and X_k to the clause gadget for clause C_w since X_i, X_j and X_k occur in C_w. We connect $v_i^{w(\ell+1)}$ to c_w^i since X_i is positive in C_w, so $s(c_w^i) = T$ is only possible when $s(v_i^{w(\ell+1)}) = T$, for any solution s. Similarly, we connect $v_j^{w(\ell+1)}$ to c_w^j. Finally, since X_k occurs negatively in C_w, we impose the line constraints in the other direction. This ensures that $s(c_w^k) = F$ is only possible when $s(v_k^{w(\ell+1)}) = F$. Imposing these constraints ensures that a solution is only possible when at least one of the cycles corresponding to variables X_i, X_j, and X_k is assigned a value that would make the corresponding literal in C_w true.

We continue this construction for each clause of the 3SAT instance. Since ℓ is a constant, this is clearly a polynomial reduction from 3SAT.

We now show that any CSP instance P_Ψ constructed in the manner we have just described cannot contain any pattern in $\mathcal{X}$. We do this by showing that no neg-connected pattern containing Cycle(k) (for $2 \leq k \leq \ell$), Valency, Path, or Valency+Path can occur in the instance. This is sufficient to show that the CSP instance P_Ψ does not contain any of the patterns in $\mathcal{X}$.

In the CSP instance P_Ψ no constraint contains more than one disallowed tuple. Thus, any $\chi \in \mathcal{X}$ for which $\textsc{Cycle}(2) \to \chi$ cannot occur in P_Ψ. Furthermore, P_Ψ is built from cycles of length $m(\ell+1)$ and paths of length $\ell+1$, and so cannot contain any cycles on less than $\ell+1$ vertices. Thus, since ℓ is the maximum number of vertices in any element of $\mathcal{X}$, it follows that no $\chi \in \mathcal{X}$ for which $\textsc{Cycle}(k) \to \chi$, for any $k \geq 3$, can occur in P_Ψ.

We define the *valency* of a variable x to be the number of distinct variables which share a constraint pattern with x. Suppose $\textsc{Valency} \to \chi$, where $\chi \in \mathcal{X}$ is neg-connected. For this to be possible we require that there is a variable of valency four in χ, or a pair of variables of valency three connected by a path of length at most ℓ in the negative structure graph of χ. Certainly P_Ψ has no variables of valency four. Moreover, the fact that P_Ψ was built using paths of length $\ell+1$ means that no two of its valency three variables are joined by a path of length at most ℓ. Thus, $\chi \in \mathcal{X}$ does not occur in P_Ψ if $\textsc{Valency} \to \chi$.

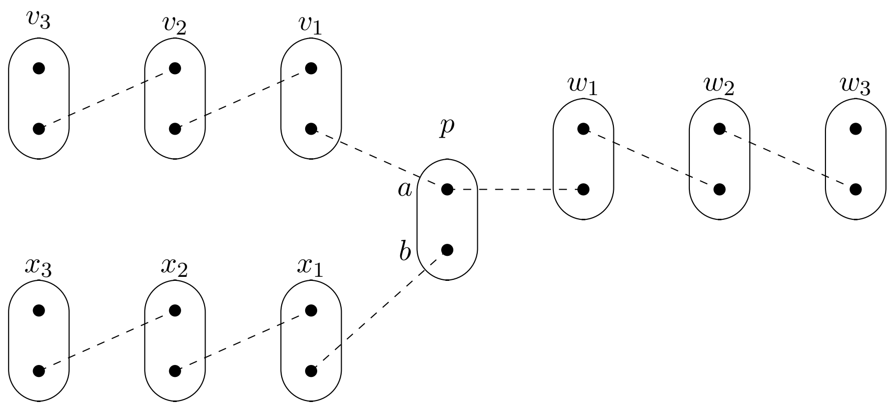
Next, consider the case when $\textsc{Path} \to \chi$, where $\chi \in \mathcal{X}$ is neg-connected. Here χ must have two distinct (but possibly overlapping) three-variable lines (with disallowed tuples in these constraint patterns that match at domain values) separated by at most ℓ variables. The only place where disallowed tuples can meet in P_Ψ is when we connect the line gadget to the cycle gadget. These connection sites are always at distance greater than ℓ, so we can conclude that $\chi \not\to P_\psi$ whenever $\textsc{Path} \to \chi$.

Finally, consider the case where $\textsc{Valency+Path} \to \chi$, where $\chi \in \mathcal{X}$ is neg-connected. Here, χ must have a variable of valency at least 3 and a path of constraint patterns on three variables with intersecting disallowed tuples, and these must be connected by a path of less than ℓ variables in the negative structure graph of χ. As observed above, the only places in P_Ψ where we can have disallowed tuples meeting is where the line gadget meets the cycle gadget, and there is a path of at least ℓ variables between each one of these points and every other variable of valency 3. Thus, $\chi \not\to P_\psi$ whenever $\textsc{Valency+Path} \to \chi$. □

It remains to consider which sets of negative binary patterns could be tractable. For this, we need to define the *pivot* patterns, $\textsc{Pivot}(r)$, which contain every tractable negative binary pattern.

Definition 5.3. *Let* $V = \{p\} \cup \{v_1, \ldots, v_r\} \cup \{w_1, \ldots, w_r\} \cup \{x_1, \ldots, x_r\}$, $D = \{a, b\}$ *and* $S = \{\text{NEQ}(p, v_1, \ldots, v_r, w_1, \ldots, w_r, x_1, \ldots, x_r)\}$. *We define the pattern* $\textsc{Pivot}(r) =$

Pattern 12 Pivot(3)



$\mathrm{NEQ}(p, v_1, v_2, v_3, w_1, w_2, w_3, x_1, x_2, x_3)$

$\langle V, D, C_p \cup C_v \cup C_w \cup C_x, S\rangle$, *where*

$$\begin{aligned}
C_p &= \{\langle (p, v_1), \rho_{ab}\rangle, \langle (p, w_1), \rho_{ab}\rangle, \langle (p, x_1), \rho_{bb}\rangle\}\\
C_v &= \{\langle (v_i, v_{i+1}), \rho_{ab}\rangle \mid i = 1, \ldots, r-1\}\\
C_w &= \{\langle (w_i, w_{i+1}), \rho_{ab}\rangle \mid i = 1, \ldots, r-1\}\\
C_x &= \{\langle (x_i, x_{i+1}), \rho_{ab}\rangle \mid i = 1, \ldots, r-1\}
\end{aligned}$$

and where $\rho_{ab}(a, b) = F$, $\rho_{ab}(s, t) = U$ *(for all* $(s, t) \neq (a, b)$*),* $\rho_{bb}(b, b) = F$, $\rho_{bb}(s, t) = U$ *(for all* $(s, t) \neq (b, b)$*). The pattern* Pivot(r) *has the structure* S *that all its variables are distinct. See Pattern 12 for an example,* Pivot(3)*.*

We say that a pattern χ on variables $v_1, \ldots, v_r$ is a **distinct-variable** pattern if its structure includes $\mathrm{NEQ}(v_1, \ldots, v_r)$. The following proposition characterises those sets of connected binary flat negative distinct-variable patterns which Theorem 5.2 does not prove intractable.

Proposition 5.4. *Any connected binary flat negative distinct-variable pattern* χ *either contains* Cycle(k) *(for some* $k \geq 3$*),* Valency*,* Path*, or* Valency+Path*, or itself occurs in* Pivot(r) *for some integer* $r \leq |\chi|$*.*

Proof. Suppose χ does not contain any of the patterns Valency, Cycle(k) (for any $k \geq 3$), Path, or Valency+Path. Recall that the *valency* of a variable x is the number of distinct variables which share a constraint pattern with x. Since χ does not contain Valency it can only contain one variable of valency three and all other variables must have valency at most two. Moreover, since Cycle$(k) \not\to \chi$ for $k \geq 3$, the negative structure graph of χ does not contain any cycles. Thus, since χ is connected, the negative structure graph of χ consists of up to three disjoint paths joined at a single vertex. If two disallowed tuples over distinct scopes intersect, then we call the union of the scopes the **footprint** of the intersection. The fact that the negative structure graph of χ is acyclic and that χ does not contain PATH means that all such pairs of intersecting disallowed tuples in χ must have the same footprint. Moreover, the fact that χ does not contain VALENCY+PATH means that all such intersections must occur at the variable with valency 3, if it exists. The fact that χ is flat and negative means that in a renaming-extension any pair of disallowed tuples $\langle a, b\rangle$, $\langle c, d\rangle$ over the same scope $\langle u, v\rangle$ in χ can be merged by the domain-renaming function t, i.e. $t(\langle u, a\rangle) = t(\langle u, c\rangle)$ and $t(\langle v, b\rangle) = t(\langle v, d\rangle)$. It then follows that χ occurs in PIVOT(r), for some $r \leq |\chi|$. $\square$

Corollary 5.5. *Let $\mathcal{X}$ be a finite set of connected binary flat negative distinct-variable patterns. Then* CSP$(\overline{\mathcal{X}})$ *is tractable only if there is some $\chi \in \mathcal{X}$ that occurs in* PIVOT(r)*, for some integer $r \leq |\chi|$.*

We now prove the same result for patterns which are not necessarily distinct-variable.

Corollary 5.6. *Let $\mathcal{X}$ be a finite set of connected binary flat negative patterns. Then* CSP$(\overline{\mathcal{X}})$ *is tractable only if there is some $\chi \in \mathcal{X}$ that occurs in* PIVOT(r)*, for some integer $r \leq |\chi|$.*

Proof. For χ a connected binary flat negative pattern, let $dv(\chi)$ denote the set of connected binary flat negative distinct-variable patterns in which χ occurs, which have the same domain as χ and at most $|\chi|$ variables. We use $dv(\mathcal{X})$ to denote the union of the sets $dv(\chi)$ for $\chi \in \mathcal{X}$.

By Lemma 3.10, CSP$(\overline{\chi}) \subseteq$ CSP$(\overline{dv(\chi)})$. For every CSP instance P such that $\chi \to P$, we have $\tau \to P$ for some $\tau \in dv(\chi)$. It follows that CSP$(\overline{dv(\chi)}) \subseteq$ CSP$(\overline{\chi})$, and hence CSP$(\overline{\chi}) =$ CSP$(\overline{dv(\chi)})$. Since CSP$(\overline{\mathcal{X}})$ is just the intersection of the CSP$(\overline{\chi})$ for $\chi \in \mathcal{X}$ and CSP$(\overline{dv(\mathcal{X})})$ the intersection of the CSP$(\overline{dv(\chi)})$ for $\chi \in \mathcal{X}$, we have that CSP$(\overline{\mathcal{X}}) =$ CSP$(\overline{dv(\mathcal{X})})$.

By Corollary 5.5, CSP$(\overline{dv(\mathcal{X})})$ is tractable only if some pattern $\tau \in dv(\chi)$, for some $\chi \in \mathcal{X}$, occurs in PIVOT(r) for some $r \leq |\tau|$. But, by definition of $dv(\chi)$, χ occurs in τ and $|\tau| \leq |\chi|$. Therefore, CSP$(\overline{\mathcal{X}})$ is tractable only if χ occurs in PIVOT(r) for some integer $r \leq |\chi|$. $\square$

For an arbitrary (not necessarily flat or negative) binary CSP pattern χ, we denote by NEG(χ) the flat negative pattern obtained from χ by replacing all truth-values T by U in all constraint patterns in χ and ignoring any structure beyond disequalities between variables. Recall that the structure of a flat pattern only contains disequality relations between variables, so NEG(χ) is a flat pattern by definition. For a set of patterns $\mathcal{X}$, NEG$(\mathcal{X})$ is naturally defined as the set NEG$(\mathcal{X}) = \{$NEG$(\chi) : \chi \in \mathcal{X}\}$. Clearly CSP$(\overline{\text{NEG}(\mathcal{X})})$ $\subseteq$ CSP$(\overline{\mathcal{X}})$. The following result follows immediately from Corollary 5.6. It provides a necessary condition for tractability of general patterns.

Corollary 5.7. *Let $\mathcal{X}$ be a finite set of binary patterns such that for each $\chi \in \mathcal{X}$,* NEG(χ) *is connected. Then* CSP$(\overline{\mathcal{X}})$ *is tractable only if there is some $\chi \in \mathcal{X}$ such that* NEG(χ) *occurs in* PIVOT(r)*, for some integer $r \leq |\chi|$.*

6. The Pivot Theorem

Theorem 6.1. Pivot(r) *is tractable for all* $r \geq 1$.

This theorem together with Corollary 5.6 immediately provides a dichotomy for finite sets of connected binary flat negative patterns. Most of this section is devoted to the proof of this theorem (which we call the pivot theorem). We conclude this section by giving a dichotomy for finite sets of flat negative patterns which are not necessarily connected.

We will need some definitions from graph theory.

Definition 6.2. *A **subdivision** of a graph G is a graph obtained by replacing some edges of G with simple paths.*

*A **minor** of a graph G is any graph obtained from G by deleting edges, contracting edges and removing isolated vertices. A graph H is a **topological minor** of a graph G if a subdivision of H is a subgraph of G.*

We will need to use the following well-known theorem of Robertson and Seymour (1986).

Theorem 6.3. *For every planar graph* H *there is an integer* $k > 0$ *such that if a graph does not contain* H *as a minor, then its tree width is at most* k.

In particular, if a graph has large tree width, then it contains a large grid minor. In this section we will consider hexagonal grid minors instead (see Figure 2). The reason for this is the well-known fact that if a graph of maximum degree three is a minor of another graph, then it is a topological minor and the latter notion is more convenient for our proofs. As illustrated in Figure 2, an h hexagonal grid is a graph composed of hexagons in a honeycomb pattern: its width h is the number of hexagons in both horizontal and vertical directions.

Definition 6.4. *Let* $g : r \to \mathbb{N}$ *be such that every graph of tree width at least* $g(r)$ *contains the* $3(r+4)$ *hexagonal grid as a topological minor.*

Let us observe the following simple property first:

Lemma 6.5. *Any three degree three vertices of the hexagonal grid of width* $3r$ *begin disjoint paths of length* r.

Proof. If each vertex is in a different row then we can simply choose a path for each of them along their row (in the direction away from the nearest boundary of the grid). See vertices a, b and c in Figure 2 to visualise this typical situation.

Otherwise it may be possible, by rotating the grid through 120 or 240 degrees to get all three vertices to lie on different rows. If we cannot separate the vertices by rotating then they are the corners of an equilateral triangle, such as x, y, z or p, q, r in the diagram.

If the triangle has an interior row, such as Row 4 for x, y, z in the diagram, then we can extend two vertices along their row and drop the third to the interior row and then along that row. Thus, for example, the path beginning at y would drop down one row and continue along Row 4.

The only remaining case is when the three vertices form an equilateral triangle occupying just two adjacent rows, like p, q, r in the diagram. In this case there is some orientation for which the two vertices that are on the same row *do not both* lie along the edge of the grid. In the diagram we can rotate either 120 or 240 degrees to achieve this for p, q, r. Now we can extend two of the three vertices along their row and the third can shift away from the centre of the triangle in order to find an empty row along which the path can be extended. $\square$

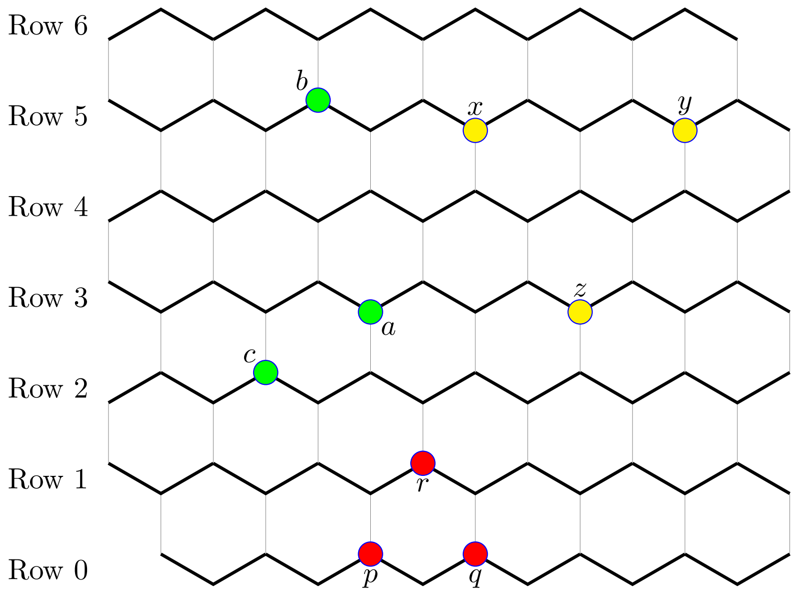


Figure 2: A hexagonal grid of width 6 with the rows picked out in bold and numbered.

The following combinatorial result is crucial for our algorithm, but it is interesting in its own right:

Lemma 6.6. *Let G be a 3-connected graph of tree width at least $g(r)$ and let a, b, c be distinct vertices of G. Then G contains 3 pairwise vertex disjoint paths starting at a, b, c, respectively, and each having length r.*

Proof. Let H be the $3(r+4)$ hexagonal grid. By the definition of $g(r)$, graph G contains H as a topological minor. Note that H contains some vertices of degree 2 on the boundary, which will cause some complications in the proof. To avoid this complication, we observe that H is a subdivision of a graph H^- whose every vertex has degree 3 and we will focus on the graph H^- instead.

Let H^-_G denote the subdivision of H^- appearing in G and let S denote the vertices of degree three in H^-_G. By Menger's theorem (Dirac, 1966) there are vertex-disjoint paths P_a, P_b, P_c in G from a, b, c to distinct vertices s_a, s_b, s_c in S, respectively. Choose the paths in such a way that the total number of edges used by them that are *not* in H^-_G is minimized.

Let $x, y \in S$ be two vertices that correspond to adjacent vertices in H^-. This means that H^-_G contains a path Q with endpoints x and y whose internal vertices are disjoint from S. Suppose that, say, P_a contains an internal vertex of Q. We claim that either (1) $x, y \in \{s_a, s_b, s_c\}$ or (2) $s_a \in \{x, y\}$ and P_b, P_c are disjoint from Q. Suppose that (1) does not hold, say, $x \notin \{s_a, s_b, s_c\}$. Consider the internal vertex q of Q closest to x that is used by one of the paths. We can reroute this path from q to x without using any further edges outside H_G^-. This would create a new set of paths with smaller number of edges outside H_G^-, unless the rerouted path did not use any further edges outside H_G^- after q. This is only possible if the path goes from q to y on Q. But this means that there is only one path intersecting Q: no path intersects Q between q and x by the definition of q and the same path uses all the vertices from q to y. Thus case (2) holds.

By Lemma 6.5 there are three independent paths of length $r+4$ from the vertices s_a, s_b and s_c in the (non subdivided) hexagonal grid H, which correspond to paths X_a, X_b, X_c in H_G^-. We will use these paths to create three independent paths of length at least r from a, b and c in G. By definition, the path X_a does not go through s_b or s_c. Therefore, by the claim in the previous paragraph, X_a is disjoint from P_b and P_c: if X_a uses the path Q between x and y, then $s_b, s_c \notin \{x, y\}$ means that neither (1) or (2) can happen if P_b or P_c intersects Q. If X_a is disjoint from P_a as well, then we create a new path T_a by simply concatenating P_a and X_a. Otherwise, the only way X_a can intersect P_a is on the first subdivided edge of H^- where X_a goes (this is the only place where case (2) of the claim can happen). In this case, we create a new path T_a by following P_a until it meets this subdivided edge and then following X_a. As each edge of H^- corresponds to up to 4 edges of H, path T_a could meet up to 4 edges of H fewer than what X_a does. The path T_a will, in either case, meet at least r subdivided edges of H and so has length at least r. We can build T_b and T_c in an analogous fashion. □

We will require the following technical lemma in our proofs.

Lemma 6.7. *Let P be any binary CSP instance. Suppose that the assignment of d to x, or of d' to y both extend to a solution of P but that there is no solution assigning both d to x and d' to y. Then there is a path of proper binary constraints between x and y. Furthermore, there is such a path such that the first constraint along this path disallows some tuple $\langle d, d_1\rangle$ and the last constraint disallows some tuple $\langle d_2, d'\rangle$.*

Proof. Let S_x be any solution to P including the assignment of d to x and similarly let S_y be any solution to P including the assignment of d' to y.

Define a graph G on the variables of P. There is an edge from x to a variable z if and only if the assignment of d to x is incompatible with some domain value for z. Similarly, there is an edge from y to a variable z if and only if the assignment of d' to y is incompatible with some domain value for z. Finally there is an edge between any two variables other than x and y if and only if the constraint between them is proper.

Let C_x be the component of G containing x. Define an assignment S to the variables of P by setting $S(z) = S_x(z)$ if $z \in C_x$ and $S(z) = S_y(z)$ otherwise. This is a solution to P since the only possible unsatisfied constraint would have to be between a variable of C_x and a variable not in C_x, but by the choice of C_x this cannot happen.

By hypothesis, we know that $S(y) \neq d'$ and so we have the required path of proper binary constraints. □

Note that if in Lemma 6.7 there is a binary constraint between x and y that forbids assigning d to x and d' to y, then setting $d_1 = d'$ and $d_2 = d$ yields the required path of proper binary constraints, of length one.

We first show that $\mathrm{CSP}(\overline{\textsc{Pivot}(r)})$ is tractable in a special case with restricted structure.

Lemma 6.8. *The subclass of* $\mathrm{CSP}(\overline{\textsc{Pivot}(r)})$ *consisting of instances:*

- *which have an arc-consistent binary reduction;*
- *which have no unary constraints on variables of degree two in the constraint graph;*
- *where the true constraint graph is a subdivided three-connected graph,*

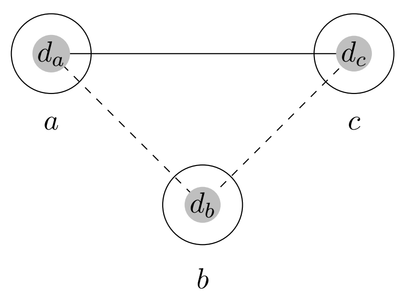
has time complexity $O\left(n^3 d^{g(r)+1}\right)$.

Proof. Let P be an instance satisfying the conditions of the lemma. In time $O\left(nd^2\right)$, we join the binary constraints along each subdivided edge eliminating intermediate variables as we go, to obtain the instance P', whose constraint graph is three-connected, but which may have some improper binary constraints and arbitrary unary constraints. Let G denote the true constraint graph of P and G' the three-connected graph obtained from G by contracting subdivided edges. The true constraint graph of P' is a subgraph of G' on the same vertex-set.

We can solve $P' \in \mathrm{CSP}(\overline{\textsc{Negtrans}})$ in time $O\left(n^2 d^3 (n+d)\right)$ (Cooper & Živný, 2011b). This is clearly $O\left(n^3 d^{g(r)+1}\right)$ since $g(r) \geq 3$ for all r. Furthermore, if G' has tree width at most $g(r)$ then P' can be solved in time $O\left(nd^{g(r)+1}\right)$ (Dechter & Pearl, 1989). In either case this solution can be extended to a solution of the original instance P in time $O(nd)$.

The only remaining case to consider is when $\textsc{Negtrans}$ occurs in P' and the tree width of G' is also at least $g(r)$. We will complete the proof by deriving the contradiction $P \notin \mathrm{CSP}(\overline{\textsc{Pivot}(r)})$, in order to show that this case cannot occur.

Suppose $\textsc{Negtrans}$ occurs in P' on variables a, b, c and values d_a, d_b, d_c:



By Lemma 6.6, G' contains 3 vertex-disjoint paths T_a, T_b, T_c starting at a, b, c, respectively, each having length at least r. Recall that the true constraint graph G of the original instance P was a subdivided three-connected graph and that P' was obtained from P by joining binary constraints along subdivided edges. Let $\tilde{T}_a, \tilde{T}_b, \tilde{T}_c$ denote the paths in G corresponding to T_a, T_b, T_c in G'. Recall also that $\textsc{Negtrans}$ occurs in P' on variables a, b, c and values d_a, d_b, d_c.

Now let $\tilde{a}$ and $\tilde{c}$ be the first vertices along the subdivided edges from b to a and c in G. The embedding of $\textsc{Negtrans}$ in P' shows that $\langle d_b, d_a \rangle$ is disallowed by the join of the arc-consistent path from b to a. Since this path is, by construction, a subdivision of an edge of P' we know that no unary constraints occur on any internal vertices. We also know, by arc consistency of the binary constraints, that the assignments $a = d_a$ or $b = d_b$ both extend to a consistent assignment to the path between a and b. So, by Lemma 6.7 we know that there is some value $\tilde{d}_a$ in the domain of $\tilde{a}$ such that $\langle d_b, \tilde{d}_a \rangle$ is disallowed in P by the constraint between b and $\tilde{a}$. Similarly there is a value $\tilde{d}_c$ for which $\langle d_b, \tilde{d}_c \rangle$ is disallowed in P by the constraint between a and $\tilde{c}$. By appending the path from b to a to the path $\tilde{T}_a$ and the path from b to c to $\tilde{T}_c$, together with $\tilde{T}_b$, we obtain three independent paths of length at least r of proper constraints in P, beginning at variable b, two beginning with constraints disallowing a tuple with value d_b at b. So we have shown that Pivot(r) does indeed occur in P and we are done. ☐

CSP($\overline{\text{Pivot}(r)}$) places an upper bound on the length of the chain of dependencies that may have to be followed to discard a partial solution that cannot be extended to a solution. Informally speaking, forbidding the Pivot(r) pattern bounds the amount of local search that may have to be done when extending a partial solution to a larger partial solution. The amount of effort that may be required increases with the length of such chains of inference, and this worst-case behaviour is quantified more precisely in the following result. We first require some definitions.

Definition 6.9. *Let G be a graph and U be a subset of the vertices of G. The induced graph $G[U]$ of G on U is the graph with vertex set U and whose edges are those edges of G which connect two vertices of U.*

For graphs $G = \langle V, E \rangle$ and $G' = \langle V', E' \rangle$ define $G \cup G' = \langle V \cup V', E \cup E' \rangle$.

*In a graph G we say that $\langle U_1, U_2 \rangle$ is a **separation** if $G = G[U_1] \cup G[U_2]$ and neither of U_1, U_2 is a subset of the other. The **separator** of the separation $\langle U_1, U_2 \rangle$ is $U_1 \cap U_2$ and its **order** is $|U_1 \cap U_2|$. A **minimal** separator is one of minimal order.*

*The **torso** of U_1 in the separation $\langle U_1, U_2 \rangle$ is obtained from the induced graph $G[U_1]$ by adding every edge between the vertices of the separator of $\langle U_1, U_2 \rangle$.*

Theorem 6.10. *The class of* Pivot(r)*-free instances is solvable in time $O\left(n^3 d^{g(r)+3}\right)$.*

Proof. We will prove the result by induction on the number of variables.

The base case is straightforward. When the instance has fewer than $g(r)+3$ variables, we can clearly solve it by exhaustive search in time $O\left(n^3 d^{g(r)+3}\right)$. For the inductive case we can assume that we can solve all smaller instances with $n < k$ variables in time $O\left(n^3 d^{g(r)+3}\right)$.

Let P be any Pivot(r)-free instance with $n = k$ variables. First make P arc-consistent in time $O(n^2 d^2)$ (Bessière, Régin, Yap, & Zhang, 2005). Since P is now arc-consistent, unary constraints no longer have any effect. Remove all unary and improper binary constraints from P in time $O(n^2 d^2)$. Let G be the true constraint graph of the resulting instance, P'.

If G has no separation of order two then it is either three-connected or has at most three vertices. In the three-connected case we can solve P', and hence P, in time $O\left(n^3 d^{g(r)+1}\right)$ by Lemma 6.8. If P has at most three variables then it is trivial to solve in time $O(d^3)$.

So, we can assume that G has a separation of order two. By definition of the torso any size-2 separator of a torso is a size-2 separator of G. Hence we can find a size-2 separation $\langle U_1, U_2 \rangle$ where the torso of U_1 has no separation of order two. So we can assume that the torso of U_1 is either three-connected or has at most three vertices.

Now consider the separator $M = U_1 \cap U_2$ of $\langle U_1, U_2 \rangle$. If M is empty then P' is composed of two smaller independent Pivot(r)-free instances and so can be solved in time $O\left(n_1^3 d^{g(r)+3}\right) + O\left(n_2^3 d^{g(r)+3}\right)$ where $n_1 + n_2 = n$ and $1 \leq n_1, n_2 < n$. It follows that we can solve P in time $O\left(n^3 d^{g(r)+3}\right)$, so we are done.

If $M = \{m\}$ then we consider the structure of $G[U_1]$. It is three-connected, so m vertex has degree at least three. In this case we can add any unary constraint on m and, by Lemma 6.8, solve the instance in time $O\left(n^3 d^{g(r)+1}\right)$. Hence we can find, in time $O\left(dn^3 d^{g(r)+1}\right)$, which values of the variable m extend to all variables of U_1. Adding this restriction as a unary constraint on variable m leaves the induced instance on U_2 PIVOT(r)-free and we see, by induction, that we can solve P' in time $O\left(dn^3 d^{g(r)+1} + (n-1)^3 d^{g(r)+3}\right) = O\left(n^3 d^{g(r)+3}\right)$, so we are done.

Finally we must consider $M = \{x, y\}$. Since M is minimal we know that $G[U_2]$ is connected and there is a path between x and y in U_2. Denote by Q the CSP instance induced on $G[U_1]$, together with some path in U_2 from x to y. The constraint graph of Q is a subdivision of the torso of U_1 which is either three-connected or has at most three vertices. In the latter case Q has tree width at most two so, after the addition of unary constraints on x and y, can be solved in time $O(n^2d^3)$ (Dechter & Pearl, 1989). If the torso of U_1 is three-connected then the degrees of x and of y in Q are at least three. After the addition of any unary constraints on x and y we can, by Lemma 6.8, solve this case in time $O\left(n^3 d^{g(r)+1}\right)$. Hence we can solve Q with all possible unary constraints on x and y which only allow one value for each of x and y, in time $O\left(d^2 n^3 d^{g(r)+1}\right)$.

For each value of variable x we now know whether it extends to some solution on all the variables in Q. Similarly, for each value of variable y we know whether it extends to a solution on all variables in Q. We can express these two restrictions as unary constraints, $u(x)$ and $u(y)$ on x and y. Lastly we find the binary constraint $c(x, y)$ on x and y which specifies precisely which pairs of values, allowed by $u(x)$ and $u(y)$, extend to all variables of U_1. This we obtain by solving the subdivided three-connected instance and seeing which pairs are disallowed by the subdivided edge in U_2 – for such pairs we do not set the constraint relation in $c(x, y)$ to F.

If $u(x)$ allows no values for x then P has no solution and we stop.

Now consider the instance R, which is induced by P' on U_2 together with the constraints $u(x), u(y)$ and $c(x, y)$. By construction, P has a solution if and only if R has a solution. Any PIVOT(t) occurring in R must use a pair of values disallowed by $c(x, y)$ since it cannot occur in P'. Suppose that $\langle d, d' \rangle$ is disallowed by $c(x, y)$. It follows that the assignment of d to x, or of d' to y both extend to a solution of Q but assigning both d to x and d' to y does not extend to a solution of the problem induced by P' on U_1. By Lemma 6.7 there is a path of proper constraints between x and y in $G[U_1]$. Furthermore, the first constraint along this path disallows some tuple $\langle d, d_1 \rangle$ and the last constraint disallows some tuple $\langle d_2, d' \rangle$. It follows that we cannot embed PIVOT(r) in the instance R induced on U_2 together with the constraint $c(x, y)$ (otherwise we would have been able to embed it in the instance P).

Since $R \in \mathrm{CSP}(\overline{\textsc{Pivot}(r)})$, we can solve it in time $O\left(n^3 d^{g(r)+3}\right)$ by our inductive hypothesis. Thus, in this final case, the complexity is $O\left(d^2 n^3 d^{g(r)+1} + n^3 d^{g(r)+3}\right) = O\left(n^3 d^{g(r)+3}\right)$ and we are done. $\square$

Theorem 6.1 is important as it gives us a tractable class of CSPs defined by forbidding a negative pattern which, unlike $\mathrm{CSP}(\overline{\textsc{Tree}})$, contains problems of unbounded tree width, and so cannot be captured by structural tractability. This is true even for PIVOT(1). As an example of a class of CSP instances in $\mathrm{CSP}(\overline{\textsc{Pivot}(1)})$ with unbounded tree width, consider the n-variable CSP instance P_n with domain $\{1, \ldots, n\}$ whose constraint graph is the complete graph and, for each pair of distinct values $i, j \in \{1, \ldots, n\}$, the constraint on variables v_i, v_j disallows a single pair of assignments $(\langle v_i, j\rangle, \langle v_j, i\rangle)$. Since each assignment $\langle v_i, j\rangle$ occurs in a single disallowed tuple, Pivot(1) does not occur in P_n, and hence $P_n \in \mathrm{CSP}(\overline{\text{Pivot}(1)})$. To produce an example of a class of instances in $\mathrm{CSP}(\overline{\text{Pivot}(1)})$ with unbounded tree width and which are not in $\mathrm{CSP}(\overline{\text{Negtrans}})$, we can modify P_n by introducing a Boolean variable v_{ij} for each pair $i < j$ and by replacing the constraint on variables v_i, v_j by constraints on v_i, v_{ij} and on v_j, v_{ij}: the former disallowing the single pair of assignments $(\langle v_i, j\rangle, \langle v_{ij}, 0\rangle)$ and the latter the pair of assignments $(\langle v_j, i\rangle, \langle v_{ij}, 0\rangle)$. The pattern Negtrans occurs on each triple of assignments $(\langle v_i, j\rangle, \langle v_{ij}, 0\rangle, \langle v_j, i\rangle)$.

The dichotomy for finite sets of connected binary flat negative patterns now follows directly from Theorem 6.1 and Corollary 5.6.

Theorem 6.11. *Let $\mathcal{X}$ be a finite set of connected binary flat negative patterns. Then $\mathcal{X}$ is tractable if and only if there is some $\chi \in \mathcal{X}$ that is contained in* Pivot(r)*, for some integer $r \leq |\chi|$.*

Informally speaking, this dichotomy states that bounding the length of problematic Pivot(r)-style inference chains leads to tractability, and moreover that when a class of instances defined by a finite set of forbidden flat patterns is tractable, then it must avoid problematic inference chains of this form.

This dichotomy easily extends to patterns which are not necessarily connected. When a negative pattern χ is not connected, it can be decomposed into connected patterns corresponding to the connected components of the negative structure graph of χ. We call these patterns the connected components of χ.

Corollary 6.12. *Let $\mathcal{X}$ be a finite set of binary flat negative patterns. Then $\mathcal{X}$ is tractable if and only if for some $\chi \in \mathcal{X}$, each of the connected components of χ is contained in* Pivot(r)*, for some integer $r \leq |\chi|$.*

Proof. Let $\mathcal{X}$ be a finite set of binary flat negative patterns. Let $CC(\chi)$ represent the set of connected components of a pattern χ, and $CC(\mathcal{X})$ the union of all the sets $CC(\chi)$ $(\chi \in \mathcal{X})$.

Suppose that $\mathcal{X}$ is tractable. Consider an arbitrary subset $\mathcal{X}'$ of $CC(\mathcal{X})$ such that the set $\mathcal{X}'$ contains exactly one connected component from each pattern $\chi \in \mathcal{X}$. By Lemma 3.10 $\mathrm{CSP}(\overline{\mathcal{X}'}) \subseteq \mathrm{CSP}(\overline{\mathcal{X}})$, and hence $\mathcal{X}'$ is also tractable. Therefore, by Corollary 5.6, there is some pattern $\chi' \in \mathcal{X}'$ that occurs in Pivot(r), for some integer $r \leq |\chi'|$. The only way this can be true for all possible choices of $\mathcal{X}'$ is if there is some $\chi \in \mathcal{X}$ such that *all* connected components of χ occur in Pivot(r), for some integer $r \leq |\chi|$.

On the other hand, suppose that for some $\chi \in \mathcal{X}$, each of the connected components of $\chi \in \mathcal{X}$ occurs in Pivot(r), where $r \leq |\chi|$. Let k be the number of connected components of χ. Then χ occurs in the disjoint union of k copies of Pivot(r). This is tractable by Theorem 6.1 and $k - 1$ applications of Lemma 4.2. It follows that χ, and hence $\mathcal{X}$, is tractable. □

7. Conclusion

In this paper we described a framework for identifying classes of CSPs in terms of *forbidden patterns*, to be used as a tool for identifying tractable classes of the CSP. We gave several examples of small patterns that can be used to define tractable classes of CSPs.

In the search for a general result, we restricted ourselves to the special case of binary patterns and binary CSPs. In Theorem 5.2 we showed that CSP($\overline{\mathcal{X}}$) is NP-hard if every pattern in a set $\mathcal{X}$ contains at least one of four patterns (Patterns 8, 9, 10, and 11). Moreover, we showed that any binary flat negative pattern χ that does not contain any of these patterns must itself be contained within (possibly several copies of) a special type of pattern called a *pivot*. Hence, being contained in (several copies of) a pivot is a necessary condition for pattern χ to be tractable. We then showed that forbidding the pivot pattern defines a tractable class.

Beyond this dichotomy for binary flat negative patterns, it will be interesting to see what new tractable classes can be defined by more general binary patterns or by non-binary patterns. In particular, an important area of future research is determining all maximal tractable classes of problems defined by patterns of some fixed size (given by the number of variables or the number of variable-value assignments). A further avenue for future research is the characterisation of the complexity of patterns involving structure that uses more than just disequalities between groups of variables, such as a total ordering on its variables.

Acknowledgments

The authors acknowledge support from ANR Project ANR-10-BLAN-0210, EPSRC grants EP/F011776/1 and EP/I011935/1, ERC Starting Grant PARAMTIGHT (No. 280152), and EPSRC platform grant EP/F028288/1.

References

Bessière, C., Régin, J.-C., Yap, R. H. C., & Zhang, Y. (2005). An optimal coarse-grained arc consistency algorithm. *Artificial Intelligence*, *165*(2), pp. 165–185. doi:10.1016/j.artint.2005.02.004.

Bulatov, A., Jeavons, P., & Krokhin, A. (2005). Classifying the complexity of constraints using finite algebras. *SIAM Journal on Computing*, *34*(3), pp. 720–742. doi:10.1137/S0097539700376676.

Bulatov, A. A. (2003). Tractable conservative constraint satisfaction problems. In *LICS '03: Proceedings of 18th IEEE Symposium on Logic in Computer Science*, pp. 321–330. doi:10.1109/LICS.2003.1210072.

Bulatov, A. A. (2006). A dichotomy theorem for constraint satisfaction problems on a 3-element set. *Journal of the ACM*, *53*(1), pp. 66–120. doi:10.1145/1120582.1120584.

Cohen, D., & Jeavons, P. (2006). The complexity of constraint languages. In Rossi et al. (Rossi et al., 2006), chap. 8, pp. 245–280.

Cohen, D. A. (2003). A new class of binary CSPs for which arc-consistency is a decision procedure. In *CP '03: Proceedings of the 9th International Conference on Principles and Practice of Constraint Programming*, No. 2833 in Lecture Notes in Computer Science, pp. 807–811. Springer-Verlag. doi:10.1007/978-3-540-45193-8_57.

Cooper, M. C., & Escamocher, G. (2012). A Dichotomy for 2-Constraint Forbidden CSP Patterns. In *AAAI '12: Proceedings of the Twenty-*

Sixth AAAI Conference on Artificial Intelligence. Available from: https://www.aaai.org/ocs/index.php/AAAI/AAAI12/paper/view/4960/5225.

Cooper, M. C., Jeavons, P. G., & Salamon, A. Z. (2010). Generalizing constraint satisfaction on trees: Hybrid tractability and variable elimination. *Artificial Intelligence*, *174*(9–10), pp. 570–584. doi:10.1016/j.artint.2010.03.002.

Cooper, M. C., & Živný, S. (2011a). Hierarchically nested convex VCSP. In *CP '11: Proceedings of the 17th International Conference on Principles and Practice of Constraint Programming*, pp. 187–194. Springer-Verlag. doi:10.1007/978-3-642-23786-7_16.

Cooper, M. C., & Živný, S. (2011b). Hybrid tractability of valued constraint problems. *Artificial Intelligence*, *175*(9–10), pp. 1555–1569. doi:10.1016/j.artint.2011.02.003.

Costa, M.-C. (1994). Persistency in maximum cardinality bipartite matchings. *Operations Research Letters*, *15*(3), pp. 143–149. doi:10.1016/0167-6377(94)90049-3.

Dalmau, V., Kolaitis, P. G., & Vardi, M. Y. (2002). Constraint satisfaction, bounded treewidth, and finite-variable logics. In *CP '02: Proceedings of the 8th International Conference on Principles and Practice of Constraint Programming*, No. 2470 in Lecture Notes in Computer Science, pp. 310–326. Springer-Verlag. doi:10.1007/3-540-46135-3_21.

Dechter, R., & Pearl, J. (1987). Network-based heuristics for constraint-satisfaction problems. *Artificial Intelligence*, *34*(1), pp. 1–38. doi:10.1016/0004-3702(87)90002-6.

Dechter, R., & Pearl, J. (1989). Tree clustering for constraint networks. *Artificial Intelligence*, *38*(3), pp. 353–366. doi:10.1016/0004-3702(89)90037-4.

Dirac, G. A. (1966). Short proof of Menger's graph theorem. *Mathematika*, *13*(1), pp. 42–44. doi:10.1112/S0025579300004162.

Freuder, E. C. (1990). Complexity of K-Tree Structured Constraint Satisfaction Problems. In *AAAI '90: Proceedings of the Eighth National Conference on Artificial Intelligence*, pp. 4–9. Available from: http://www.aaai.org/Library/AAAI/1990/aaai90-001.php.

Garey, M. R., & Johnson, D. S. (1979). *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, San Francisco, CA.

Gottlob, G., Leone, N., & Scarcello, F. (2002). Hypertree decompositions and tractable queries. *Journal of Computer and System Sciences*, *64*(3), pp. 579–627. doi:10.1006/jcss.2001.1809.

Green, M. J., & Cohen, D. A. (2003). Tractability by approximating constraint languages. In *CP '03: Proceedings of the 9th International Conference on Principles and Practice of Constraint Programming*, Vol. 2833 of *Lecture Notes in Computer Science*, pp. 392–406. Springer-Verlag. doi:10.1007/978-3-540-45193-8_27.

Grohe, M. (2006). The structure of tractable constraint satisfaction problems. In *MFCS '06: Proceedings of the 31st Symposium on Mathematical Foundations of Computer Science*, Vol. 4162 of *Lecture Notes in Computer Science*, pp. 58–72. Springer-Verlag. doi:10.1007/11821069_5.

Grohe, M. (2007). The complexity of homomorphism and constraint satisfaction problems seen from the other side. *Journal of the ACM*, *54*(1), pp. 1–24. doi:10.1145/1206035.1206036.

Gyssens, M., Jeavons, P. G., & Cohen, D. A. (1994). Decomposing constraint satisfaction problems using database techniques. *Artificial Intelligence*, *66*(1), pp. 57–89. doi:10.1016/0004-3702(94)90003-5.

Jeavons, P., Cohen, D., & Gyssens, M. (1997). Closure properties of constraints. *Journal of the ACM*, *44*(4), pp. 527–548. doi:10.1145/263867.263489.

Jeavons, P. G., & Cooper, M. C. (1995). Tractable constraints on ordered domains. *Artificial Intelligence*, *79*(2), pp. 327–339. doi:10.1016/0004-3702(95)00107-7.

Jégou, P. (1993). Decomposition of domains based on the micro-structure of finite constraint-satisfaction problems. In *AAAI '93: Proceedings of the Eleventh National Conference on Artificial Intelligence*, pp. 731–736. Available from: http://www.aaai.org/Library/AAAI/1993/aaai93-109.php.

Marx, D. (2010a). Can you beat treewidth?. *Theory of Computing*, *6*(1), pp. 85–112. doi:10.4086/toc.2010.v006a005.

Marx, D. (2010b). Tractable hypergraph properties for constraint satisfaction and conjunctive queries. In *STOC '10: Proceedings of the 42nd ACM symposium on Theory of computing*, pp. 735–744. ACM. doi:10.1145/1806689.1806790.

Régin, J.-C. (1994). A filtering algorithm for constraints of difference in CSPs. In *AAAI '94: Proceedings of the Twelfth National Conference on Artificial Intelligence*, Vol. 1, pp. 362–367. Available from: http://www.aaai.org/Library/AAAI/1994/aaai94-055.php.

Robertson, N., & Seymour, P. D. (1986). Graph minors. V. Excluding a planar graph. *Journal of Combinatorial Theory, Series B*, *41*, pp. 92–114. doi:10.1016/0095-8956(86)90030-4.

Rossi, F., van Beek, P., & Walsh, T. (Eds.). (2006). *Handbook of Constraint Programming*. Foundations of Artificial Intelligence. Elsevier.

Salamon, A. Z., & Jeavons, P. G. (2008). Perfect constraints are tractable. In *CP '08: Proceedings of the 14th International Conference on Principles and Practice of Constraint Programming*, Vol. 5202 of *Lecture Notes in Computer Science*, pp. 524–528. Springer-Verlag. doi:10.1007/978-3-540-85958-1_35.

van Hoeve, W. J. (2001). The alldifferent Constraint: A Survey. In *Proceedings of the 6th Annual Workshop of the ERCIM Working Group on Constraints*. Available from: http://arxiv.org/abs/cs/0105015v1.

Weigel, R., & Bliek, C. (1998). On reformulation of constraint satisfaction problems. In *ECAI '98: Proceedings of the 13th European Conference on Artificial Intelligence*, pp. 254–258.