Hybrid tractable classes of constraint problems*

Martin C. Cooper[1] and Stanislav Živný[2]

1 **IRIT, University of Toulouse III**
Toulouse, France
`cooper@irit.fr`

2 **Dept. of Computer Science, University of Oxford**
Oxford, UK
`standa.zivny@cs.ox.ac.uk`

Abstract

We present a survey of complexity results for *hybrid* constraint satisfaction problems (CSPs) and valued constraint satisfaction problems (VCSPs). These are classes of (V)CSPs defined by restrictions that are not exclusively language-based or structure-based.

1998 ACM Subject Classification F.4.1 Mathematical Logic, G.2.1 Combinatorics, G.2.2 Graph Theory, F.1.3 Complexity Measures and Classes, G.1.6 Optimization.

Keywords and phrases Constraint satisfaction problems, Optimisation, Tractability.

Digital Object Identifier 10.4230/DFU.Vol7.15301.113

1 Introduction

A fundamental challenge in computer science is to map out the frontier of the complexity class P, the class of decision problems that can be solved in polynomial time. The constraint satisfaction problem (CSP) is a generic combinatorial problem which includes in a natural way many important NP-complete problems such as SAT or graph-colouring. The valued constraint satisfaction problem (VCSP) can be seen as an even richer language than the CSP since it provides a general framework in which to express both constraint satisfaction problems and constrained optimisation problems. The identification of tractable classes of generic problems, such as the (V)CSP, has led not only to a deeper understanding of tractability, but also to wider application areas of well-known algorithms. Indeed, recent research has shown that very few algorithmic techniques suffice to solve all tractable language-based classes of (V)CSPs [1, 35, 40].

Many real-world problems can be modelled as classical and well-studied NP-complete problems, such as SAT, CSP or VCSP. This has the advantage that generic solvers exist which are efficient on many instances, but has the disadvantage of not taking into account specificities of the particular problem which could perhaps guarantee the existence of a polynomial-time algorithm. Obvious specificities include the type of constraints or cost functions that can occur or the structure of the hypergraph of constraint scopes. Much

* The authors were supported by EPSRC grant EP/L021226/1. Stanislav Živný was supported by a Royal Society University Research Fellowship. This work was partly done while the second author was visiting the Simons Institute for the Theory of Computing at UC Berkeley. This project has received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant agreement No 714532). The paper reflects only the authors' views and not the views of the ERC or the European Commission. The European Union is not liable for any use that may be made of the information contained therein.

© Martin C. Cooper and Stanislav Živný;
licensed under Creative Commons License BY

research effort has been devoted to identifying tractable language-based classes [2, 5] or tractable structural classes [32], with many notable successes. However, it is natural to ask whether interesting classes of instances can be defined in other ways.

One way of defining classes of instances is by simultaneously placing restrictions both on the language of possible constraints (or cost functions) and on the structure of the hypergraph of constraint (cost function) scopes. Such classes are known as hybrid classes, and by extension all classes which are not exclusively language-based nor structure-based are also known as hybrid [24]. This larger meaning is the one we apply in this chapter.

As an example of a hybrid tractable class, consider a company which wishes to give bonuses to its employees (chosen from a finite set of possible amounts). Each employee has a grade, with higher grades corresponding to more important posts. Some, but not all, employees have an immediate boss to whom they report. In this case, the employee and the immediate boss must receive bonuses such that the sum of the bonuses of the employee and the boss is bounded above and below by a specified amount. On the other hand, if an employee has no immediate boss, then the rule is that they must not receive a bigger bonus than anyone at a higher grade. We will see, in Example 1, that this bonus-assignment problem falls in a hybrid tractable class.

As another example, consider the same company which now wants to assign staff to a project, minimising total salary costs while respecting constraints concerning the minimum number of personnel from each section, the maximum total number of staff on the project, as well as the availability of each member of staff. Again, we will see, in Example 4, that this problem falls into a hybrid tractable class

An important way to classify work on hybrid tractability is how classes of instances are defined. We consider the following ways of defining a class of instances of (V)CSPs:

- independent restrictions on both the language of constraints (or cost functions) and on the structure of the instance.
- excluding generic sub-instances (known as forbidden patterns).
- properties that are required to hold *after* a preprocessing operation, such as establishing a certain level of consistency, has been performed.
- graph properties of the (weighted) microstructure of the instance.
- instances which are so strongly constrained that this implies a polynomial bound on search-tree size (or, on the contrary, so weakly constrained that there is always a solution).

Historically, different hybrid classes have been identified by attempting to generalise language or structural restrictions, or to determine a large class of instances solved by a particular algorithmic technique, or to translate known results from another field, usually graph theory, to (V)CSPs. The field of hybrid tractability has not yet reached maturity and is notably lacking a general theory which would allow us to express all the above types of restrictions, together with language and structural restrictions, in a common language. Such a unified language would no doubt lead to a greater understanding and new applications.

2 Independent language and structure restrictions

A *constraint satisfaction problem* (CSP) instance I is given by a triple $\langle X, D, C\rangle$, where $X = \{X_1, \dots, X_n\}$ is a finite set of variables, D is a finite set of values, and C is a finite set of constraints. Each constraint $c \in C$ is a pair $\langle \mathbf{v}, R\rangle$, where $\mathbf{v}$, the constraint *scope*, is a list of k variables from X and $R \subseteq D^k$, the constraint *relation*, is a k-ary relation on D. We call k the *arity* of the constraint. We note that different constraints within the same instance can have different arity. The question is whether there is an assignment of values to the variables that satisfies all the constraints. More formally, to decide whether there is an assignment $s : X \to D$ such that for every $c \in C$ with $c = (\mathbf{v}, R)$ and $\mathbf{v} = (X_{i_1}, \ldots, X_{i_k})$, we have $\langle s(X_{i_1}), \ldots, s(X_{i_k})\rangle \in R$.

In language-based classes of CSPs, one restricts the set of constraint relations R on D that can appear in any instance. A finite set of relations on a fixed finite set D is called a *constraint language* and we denote by $\mathrm{CSP}(\Gamma)$ the class of CSP instances in which all constraint relations belong to Γ.

In structure-based classes of CSPs, one restricts the type of interactions of the constraint scopes s (by restricting the hypergraph of the constraint scopes s on X) that can appear in any instance.

We remark that an equivalent definition of CSPs is the following: given two relational structures A and B, is there a homomorphism from A to B? Language-based CSPs correspond to fixing B to a single relational structure (corresponding to a constraint language) whereas structure-based CSPs correspond to requiring $A \in \mathcal{A}$ for some (infinite) class $\mathcal{A}$ of relational structures [34].

In this section we will discuss known results on the complexity of CSPs that impose independent restrictions on the structure of the instance and on the constraint language.

2.1 Planarity

A constraint language is called *Boolean* if the domain D is equal to $\{0, 1\}$.

The *incidence graph* of a CSP instance $I = \langle X, D, C\rangle$ has $X \cup C$ as its vertex set and (X_i, c) is an edge, for $X_i \in X$ and $c \in C$, if $X_i \in \mathbf{v}$ where $c = (\mathbf{v}, R)$.

For a Boolean constraint language Γ, we denote by $\mathrm{CSP}_p(\Gamma)$ the set of CSP instances from $\mathrm{CSP}(\Gamma)$ with *planar* incidence graphs and with the condition that, for each constraint in the instance, the variables in the scope of the constraint appear in the clockwise order (in some fixed planar embedding).

The complexity of Boolean planar language-restricted CSPs has recently been established. First, it was shown that, apart from Boolean constraint languages Γ where $\mathrm{CSP}(\Gamma)$ is tractable (and thus also $\mathrm{CSP}_p(\Gamma)$ is tractable), $\mathrm{CSP}_p(\Gamma)$ is intractable unless Γ is an *even* Δ*-matroid* [26]. Secondly, the tractability of $\mathrm{CSP}_p(\Gamma)$ for even Δ-matroids was shown [38]. In order to define even Δ-matroids we need some further notation.

We use $\oplus$ for the exclusive or. For a tuple t over $\{0, 1\}$, we denote by $\bar{t}$ the tuple obtained from t by flipping all values; i.e., $\bar{t} = t \oplus (1, \ldots, 1)$. We say that a relation R is *self-complementary* if for every $t \in R$ we have $\bar{t} \in R$. For a self-complementary R, we denote by dR the relation $\{(t_1 \oplus t_2, t_2 \oplus t_3, \ldots, t_k \oplus t_1) \mid (t_1, \ldots, t_k) \in R\}$. We say that Γ is self-complementary if every $R \in \Gamma$ is self-complementary. We define $d\Gamma = \{dR \mid R \in \Gamma\}$. A set $M \subseteq \{0, 1\}^k$ is a Δ*-matroid* if for all $\langle t_1, \ldots, t_k\rangle, \langle t'_1, \ldots, t'_k\rangle \in M$ and every $1 \leq i \leq k$ with $t_i \neq t'_i$ there is $1 \leq j \leq k$ with $t_j \neq t'_j$ such that t with t_i and t_j flipped also belongs to M. (We also allow the case of $i = j$, in which case the new tuple that is required to belong to M is obtained from t by flipping the ith position.) A Δ-matroid M is called *even* if all tuples in M have the same parity of the number of 1s. (Note that in this case i and j cannot be the same in the definition of Δ-matroids as this would change the parity of 1s.) Finally, a Boolean constraint language is called an even Δ-matroid if Γ is self-complementary and each relation in $d\Gamma$ is an even Δ-matroid.

We now give an example of a constraint language that can be used to model the perfect matching problem in graphs as a planar CSP [26]. Let $M_k \subseteq \{0, 1\}^k$ be the relation containing the k-tuples in which precisely one coordinate is set to one (and all others are set to zero). It is easy to check that M_k is an even Δ-matroid for every k. Let Γ_{pm} be the constraint language on $\{0,1\}$ with $d\Gamma = \cup_{k\geq 1} M_k$. It can be checked that Γ is self-complementary and by definition $d\Gamma$ is an even Δ-matroid. Hence Γ_{pm} is an even Δ-matroid. For a graph $G = (V,E)$, we construct an instance $I_G = \langle E, \{0,1\}, C\rangle$ of Boolean CSP(Γ_{pm}) with a constraint $\langle\langle e_1,\dots,e_k\rangle, M_k\rangle \in C$ for every degree-k vertex of G incident to edges $e_1,\dots,e_k$. It is clear that perfect matchings in G are in 1-to-1 correspondence with satisfying assignments to I_G.

2.2 Bounded occurrence

Another natural restriction is that of bounded occurrence of variables in the constraints. We denote by CSP$_k(\Gamma)$ the class of instances of CSP(Γ) in which every variable appears in at most k constraints. Feder showed that for Boolean constraint languages that contain constants, CSP$_3(\Gamma)$ is as hard as CSP(Γ) [27]. Here a constant is a unary singleton relation. On the Boolean domain $D = \{0,1\}$, there are only two constants $c_0 = \{(0)\}$ and $c_1 = \{(1)\}$.

Boolean CSPs in which every variable appears *exactly* in two constraints are called *edge* CSPs in [38] but have been long known as *Holant* problems in the counting community. We denote by CSP$_e(\Gamma)$ the set of CSP instances from CSP(Γ) in which every variable appears in exactly two constraints. (The case of a variable appearing in *at most* two constraints, CSP$_2(\Gamma)$, can be reduced to this case [27]). Feder showed that if Γ is a Boolean constraint language including constants such that CSP(Γ) is intractable then CSP$_e(\Gamma)$ is intractable unless Γ is a Δ-matroid. Tractability has been shown for several special classes of Δ-matroids [27, 21] and most recently for even Δ-matroids [38] (where the reader can find more references). For instance, CSP$_e(\Gamma_{\mathsf{pm}})$ captures precisely the perfect matching problem in graphs.

2.3 Lifted languages

Two recent papers [39, 52] study certain hybrid classes of CSPs in which the algebraic machinery developed for the computational complexity of constraint languages is (partially) applicable. In particular, it has been shown that the complexity of the class of CSP(Γ) instances in which the structure (hypergraph of constraint scopes) is closed under inverse homomorphisms can be shifted to the analysis of the complexity of a CSP(Γ') instance, where Γ' is a new, "lifted" language [39, 52]. A class of structures $\mathcal{H}$ is closed under inverse homomorphisms if whenever there is a homomorphism from R' to $R \in \mathcal{H}$ then $R' \in \mathcal{H}$. Examples of such structures include classes of acyclic or k-colourable graphs. Non-examples include classes of planar or perfect graphs.

3 Forbidden patterns

The notion of forbidden pattern is based on the idea that local properties of an instance may guarantee tractability and that a natural local property is the exclusion of those sub-instances which are obstructions to polynomial-time solution algorithms. Classifying graphs by excluding substructures is classical in graph theory and a CSP instance can be represented by a labelled graph, known as its microstructure, so this approach has the advantage that it sometimes allows us to use known results from graph theory.

For the moment, the study of forbidden patterns has been mostly limited to binary CSPs with no structure on the variables or domain values except possibly a total order. In this section, we begin with some formal definitions before presenting certain tractable classes defined by forbidden patterns. We then go on to consider extensions to non-binary CSPs and explore classes defined by excluding patterns as topological minors.

3.1 Definitions

A *binary CSP instance* requires the assignment of values from some specified finite domain D to a finite set X of variables $\{X_1, \ldots, X_n\}$. Each variable X_i has its own domain of possible values $\mathcal{D}(X_i) \subseteq D$. Without loss of generality, in this section we assume that each pair of variables, $X_i, X_j \in X$ is constrained by a constraint relation R_{ij}. A constraint is *non-trivial* if it is not the Cartesian product of the domains of the two variables. A *solution* to a binary CSP instance is an assignment s of values to variables, such that, for each constraint R_{ij}, $\langle s(X_i), s(X_j)\rangle \in R_{ij}$.

The *constraint graph* of an instance I is $G_I = \langle V_I, E_I\rangle$, where $V_I = X$ is the set of variables of I and E_I is the set of pairs $\{X_i, X_j\}$ for which R_{ij} is non-trivial. The instance I is *arc consistent* if $\forall i \neq j \in \{1, \ldots, n\}$, $\forall a \in \mathcal{D}(X_i)$, $\exists b \in \mathcal{D}(X_j)$ such that $\langle a, b\rangle \in R_{ij}$. The instance I is *directional arc consistent* if $\forall i < j \in \{1, \ldots, n\}$, $\forall a \in \mathcal{D}(X_i)$, $\exists b \in \mathcal{D}(X_j)$ such that $\langle a, b\rangle \in R_{ij}$.

One possible presentation of a binary CSP instance is as a labelled graph whose vertices are the set of possible variable-value assignments. This (labelled) graph is known as the (coloured) *microstructure* [37, 10, 50, 6, 43]. An n-variable binary CSP instance I in this microstructure presentation is an n-partite graph $\langle A_1, \ldots, A_n, E^+\rangle$, where the ith part A_i corresponds to the set of possible assignments $\langle X_i, a\rangle$ to variable X_i and there is an edge in E^+ between $\langle X_i, a\rangle \in A_i$ and $\langle X_j, b\rangle \in A_j$ if and only if $(a, b) \in R_{ij}$. We refer to individual variable-value assignments, such as $\langle X_i, a\rangle$ (i.e., the vertices of this n-partite graph) as *points*. If v is some variable of the instance we use the notation A_v to represent the set of possible assignments to v. (Sets A_v are sometimes called *potatoes*.) Thus A_i and A_{X_i} are synonyms.

An instance I can also be presented as a negative microstructure which is the n-partite labelled graph $\langle A_1, \ldots, A_n, E^-\rangle$, where there is an edge in E^- between points $p \in A_i$ and $q \in A_j$ (for $i \neq j$) if and only if there is no edge between p and q in E^+.

We refer to edges in E^+ as *positive* edges and edges in E^- as *negative* edges. We now generalise the notion of microstructure and negative microstructure to obtain *patterns*: a pattern is a labelled n-partite graph which has a set of positive edges, E^+, and a set of negative edges, E^-.

A binary CSP instance can be seen as a special kind of pattern where the parts correspond to the variables of the instance and there is exactly one positive or negative edge between each pair of possible assignments to each pair of distinct variables. Positive edges connect assignments that are allowed by the constraint on the corresponding pair of variables, and negative edges connect assignments that are disallowed by this constraint.

A *pattern* $P = \langle A_1, \ldots, A_n, E^+, E^-\rangle$ is a partially specified instance: there may be pairs of points $p \in A_i, q \in A_j$ (with $i \neq j$) such that there is neither a positive edge nor a negative edge between p and q. A point is said to be *isolated* if it does not belong to any edges in E^+ or E^-. We do not specifically disallow the possibility that two points in a pattern are joined by both positive and negative edges: such patterns cannot occur as a subpattern in an instance but can occur as a topological minor (c.f. Section 3.6).

In order that pattern exclusion be natural we define a pattern P as occurring in another pattern Q if, after arbitrary renaming and then possible merging of points, we get a substructure of Q.

A pattern $P' = \langle A'_1, \ldots, A'_n, E'^+, E'^-\rangle$ is a *homomorphic image* of a pattern $P = \langle A_1, \ldots, A_n, E^+, E^-\rangle$ if there exists a surjective mapping $f : \bigcup_{i=1}^n A_i \rightarrow \bigcup_{i=1}^n A'_i$ such that

- $\forall p, q \in \bigcup_{k=1}^n A_k$, p and q belong to the same part A_i if and only if $f(p)$ and $f(q)$ belong to the same part A'_j,

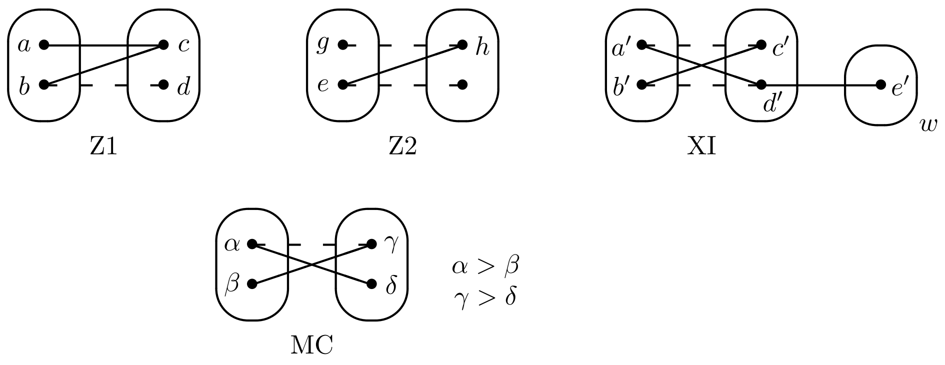

Figure 1 Four patterns.

- $\forall i, j \in \{1, \dots, n\}$ with $i \neq j$, $\forall p \in A_i$, $\forall q \in A_j$: $\{p, q\} \in E^+ \Rightarrow \{f(p), f(q)\} \in E'^+$ and $\{p, q\} \in E^- \Rightarrow \{f(p), f(q)\} \in E'^-$.

Note that forming a homomorphic image of a pattern allows the parts to be renamed, and points p, q within the same part to be merged (provided there is no third point r such that $\{p, r\}$ and $\{q, r\}$ are different types of edges).

We will say that a pattern P *occurs as a sub-pattern* of a pattern Q if Q can be transformed into a homomorphic image of P by a sequence of the following *substructure operations*:

- removal of (positive or negative) edges,
- removal of isolated points, and
- removal of empty parts.

A binary CSP instance is a pattern in which a part A_i corresponds to the set of assignments to a variable; hence elimination of a part corresponds to the elimination of a variable.

Consider the three patterns Z1, Z2 and XI shown in Figure 1. Points are represented by bullets, and points representing assignments to the same variable v are grouped together within an oval representing A_v. Solid lines represent positive edges (i.e., compatibility of the corresponding pair of points) and dashed lines negative (i.e., incompatibility) edges. For example, the pattern Z1 consists of 4 points $a, b \in A_{v_0}$, $c, d \in A_{v_1}$, two positive edges $E^+ = \{ac, bd\}$ and one negative edge $E^- = \{bd\}$. Z1 occurs in Z2 since Z2 can be transformed into a homomorphic image of Z1 by removal of edge gh and point g (under a homomorphism that maps a and b to the same point e in Z2). The pattern Z2 occurs in XI (as a subpattern) since XI can be transformed into a homomorphic image of Z2 by removal of the edges $a'd'$ and $d'e'$ followed by the removal of the (then) isolated point e' together with the (then) empty part w. By transitivity of the occurrence relation, Z1 also occurs in XI.

We also consider patterns with structure in the sense of relations between the points in a pattern. In this case the homomorphism f in the definition of homomorphic image, above, must preserve the structure of the pattern. This structure may be, for example, an order on the parts (i.e., variables) or an order on points within a part (i.e., domain values). Patterns (such as Z1, Z2 and XI in Figure 1) without any such structure are known as *flat* patterns [6]. The pattern MC in Figure 1 has the structure consisting of the partial order: $\alpha > \beta$, $\gamma > \delta$. Since XI is a flat pattern, MC does not occur in XI since this partial order clearly cannot be preserved by a homomorphism from MC to XI. On the other hand, Z1 occurs in MC (via a homomorphism which maps both a and b to α, c to δ and d to γ) since Z1 has no structure to be preserved.

A class of binary CSP instances can be defined by *forbidding* a pattern P. We use the

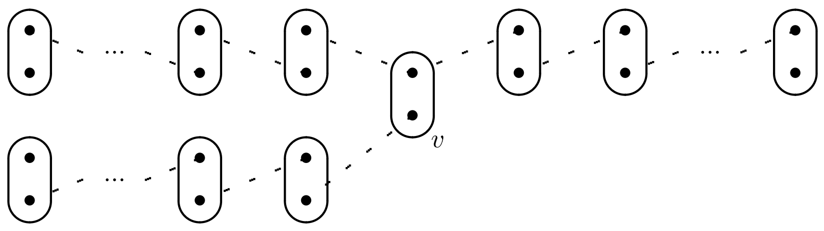

Figure 2 The negative pattern Pivot(k), where the number of edges in each of the three branches leaving the central variable v is k.

notation $\mathrm{CSP}_{\overline{\mathrm{SP}}}(P)$ to represent the set of binary CSP instances in which the pattern P does not occur as a subpattern. We say that a pattern P is *tractable* if there is a polynomial-time algorithm to solve $\mathrm{CSP}_{\overline{\mathrm{SP}}}(P)$ and *intractable* if $\mathrm{CSP}_{\overline{\mathrm{SP}}}(P)$ is NP-hard. The pattern MC is tractable since $\mathrm{CSP}_{\overline{\mathrm{SP}}}(\mathrm{MC})$ is the class of binary max-closed instances [36]. On the other hand, we know that Z2 is intractable since it does not satisfy a necessary condition for tractability described in Section 3.2. If P occurs as a subpattern of Q, then $\mathrm{CSP}_{\overline{\mathrm{SP}}}(P) \subseteq \mathrm{CSP}_{\overline{\mathrm{SP}}}(Q)$ and hence P is tractable if Q is tractable [6]. Thus we can immediately deduce that Z1 is tractable (since it occurs in MC) and that XI is intractable (since Z2 occurs in XI).

An important point is that applying any reduction operation which eliminates domain elements, such as arc consistency, SAC (Singleton Arc Consistency) or neighbourhood substitution [24], cannot introduce a pattern in an instance. On the other hand, reduction operations, such as 3-consistency, which modify constraints may introduce patterns.

A pattern P (such as pattern Z1 in Figure 1) is *mergeable* if there exists some pattern Q (such as the pattern Z1 without the edge ac or the point a) such that Q is a homomorphic image of P but P is not a homomorphic image of Q; otherwise, P is *unmergeable*. A point p (such as e' in pattern XI in Figure 1) is called *dangling* if p belongs to at most one positive edge $\{p, q\}$ and no negative edges (and p belongs to no other relation, such as a partial order, in the case of patterns with structure). The corresponding *dangling reduction* consists in removing both the edge $\{p, q\}$ and the point p (together with the part to which p belonged if this part becomes empty after removal of p). Dangling points provide no information in arc-consistent instances, in the sense that P occurs in an instance I if and only if the pattern P' occurs in I where P' is the result of applying a dangling reduction to P. Thus, using the fact that establishing arc consistency cannot introduce patterns, we have that P is tractable if and only if P' is tractable.

An unmergeable pattern with no dangling points is called *irreducible*. In Figure 1, pattern Z1 is mergeable, whereas patterns Z2 and XI are unmergeable. The point e' in pattern XI is dangling (as is the point a in the pattern Z1) but not β and δ in pattern MC (because of the partial order relation on these points). Thus of the four patterns in Figure 1, only Z2 and MC are irreducible.

3.2 Characterising tractable patterns

The theoretical tools necessary to provide a complete characterisation of tractable patterns have yet to be discovered. Indeed, characterising tractable patterns would appear to be, in general, even more difficult than characterising tractable constraint languages or tractable constraint-hypergraph structures. Nevertheless, certain characterisation results have been

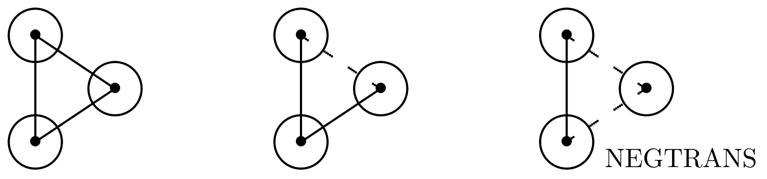

Figure 3 Tractable triangle patterns

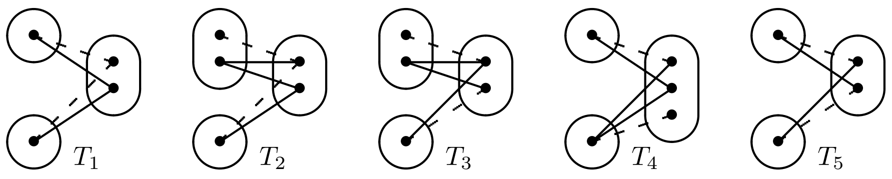

Figure 4 The five tractable irreducible flat patterns on 3 variables and 2 constraints.

proved. One important result concerns the negative edges in a tractable unmergeable pattern. It has been shown that the "skeleton" of a tractable unmergeable pattern, consisting of just the negative edges, must occur as a subpattern of (possibly multiple copies of) the pattern Pivot(k), shown in Figure 2, for some constant k [6]. Unfortunately, very little is known about the positive edges that can be added to such skeletons of negative edges.

Given that a general characterisation seems, for the moment, out of reach, certain special cases have been studied, such as triangle patterns and 2-constraint patterns [19, 15]. Figure 3 shows the three tractable flat patterns on a triangle of 3 points and 3 edges [19]. Although the first two patterns define fairly trivial classes, the third one, called NEGTRANS, is interesting since $\text{CSP}_{\overline{\text{SP}}}$(NEGTRANS) includes non-trivial instances composed of arbitrary unary constraints and non-overlapping All-Different constraints [48, 55]. The class of binary CSP instances satisfying this negative-transitivity property has been generalised to a large tractable class of optimisation problems involving cost functions of arbitrary arity which we will discuss in Section 7 [19]. Figure 4 shows the five tractable irreducible flat patterns on 3 variables and 2 constraints [15]. $\text{CSP}_{\overline{\text{SP}}}(T_4)$ is interesting since it includes all binary CSP instances with zero-one-all constraint relations [13] (which can be seen as a generalisation of 2SAT to non-Boolean domains).

Given the relatively modest successes in defining new and useful tractable classes by forbidding flat patterns, it is natural to consider possible extensions of patterns by studying structured patterns, non-binary patterns and other forms of occurrence than subpattern occurrence. These three extensions are the subject of the remainder of this section.

3.3 Partially-ordered patterns

The pattern BTP shown in Figure 5 is known as a broken triangle (since the positive edges can be said to form a triangle which is broken at variable z). Forbidding this pattern on all triples of variables $x < y < z$ defines a tractable class $\text{CSP}_{\overline{\text{SP}}}$(BTP) [16]. This class includes all binary CSP instances whose constraint graph is a tree $\mathcal{T}$ since, ordering the variables according to a pre-order of $\mathcal{T}$, each variable z is constrained by at most one variable $y < z$, its parent in $\mathcal{T}$, and hence the broken-triangle pattern cannot occur. If a variable ordering exists such that the broken-triangle pattern does not occur, then this order can be found in polynomial time: it suffices to establish arc consistency and then successively eliminate

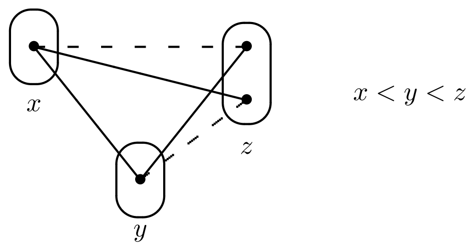

Figure 5 A binary CSP instance satisfies the broken triangle property if this pattern (known as a broken triangle or BTP) does not occur in the instance.

variables v which are not the right-hand variable z of a broken triangle (since we know that such a variable v can be the last variable in the ordering among the remaining variables). $\text{CSP}_{\overline{\text{SP}}}(\text{BTP})$ is solved by arc consistency since the BTP is exactly the obstruction which prevents an arc-consistent instance being backtrack-free. Indeed, even if the variable order is unknown, MAC (Maintaining Arc Consistency) solves $\text{CSP}_{\overline{\text{SP}}}(\text{BTP})$ [16]. Thus, most CSP solvers will automatically solve in polynomial time all instances in $\text{CSP}_{\overline{\text{SP}}}(\text{BTP})$.

▶ **Example 1.** Consider a company which wishes to give bonuses to its n employees. Each employee $i \in \{1, \dots, n\}$ has a grade $grade_i$, with higher grades corresponding to more important posts. Some, but not all, employees have an immediate boss to whom they report. The company wants to assign bonuses so that each employee's bonus is a multiple of 50 euros between 5% and 20% of their salary. If an employee i has an immediate boss b_i, then the sum of the bonuses of i and b_i must be no less than 10% and no more than 30% of the salary of b_i. On the other hand, if an employee i has no immediate boss then the rule is that they must not receive a bigger bonus than anyone at a higher grade.

Let the variable x_i be the bonus assigned to employee i. We assume that employees are numbered so that $(grade_i > grade_j) \Rightarrow (i < j)$. Thus, for example, employee number 1 is the CEO of the company. The domain of x_i is the multiples of 50 between 5% and 20% of the salary sal_i of employee i. If employee i has a boss b_i, then there is a binary constraint $0.1sal_{b_i} \leq x_i + x_{b_i} \leq 0.3sal_{b_i}$. Indeed, this is the only constraint between x_i and the variables x_j $(j < i)$. If employee i has no boss, then there are binary constraints $x_i \leq x_j$ for each $j \in \{1, \dots, i-1\}$ such that $grade_j > grade_i$. In either case, it is easy to verify that x_i cannot be the rightmost variable in the broken triangle pattern shown in Figure 5. Thus, this problem falls in $\text{CSP}_{\overline{\text{SP}}}(\text{BTP})$ and is solved by arc consistency. The constraint graph is of unbounded tree-width: for example, if no-one has a boss but everyone is at a different grade, then the constraint graph is the complete graph. Furthermore, the language of constraints is NP-hard. Thus, this bonus-assignment problem defines a truly hybrid tractable class.

We have seen that broken-triangle free instances are solved by arc consistency. Arc consistency is also a decision procedure for $\text{CSP}_{\overline{\text{SP}}}(\text{EMC})$ where EMC (Extended Max-Closed) is the pattern shown in the top left of Figure 6. EMC is particularly interesting because $\text{CSP}_{\overline{\text{SP}}}(\text{EMC})$ is a strict generalisation of binary max-closed CSPs [36] (since the pattern MC shown in Figure 1 is a subpattern of EMC).

▶ **Example 2.** Consider a binary CSP instance I with integer domains and in which all binary constraints are of the following form:

$$aX_i + bX_j \;\geq\; c$$

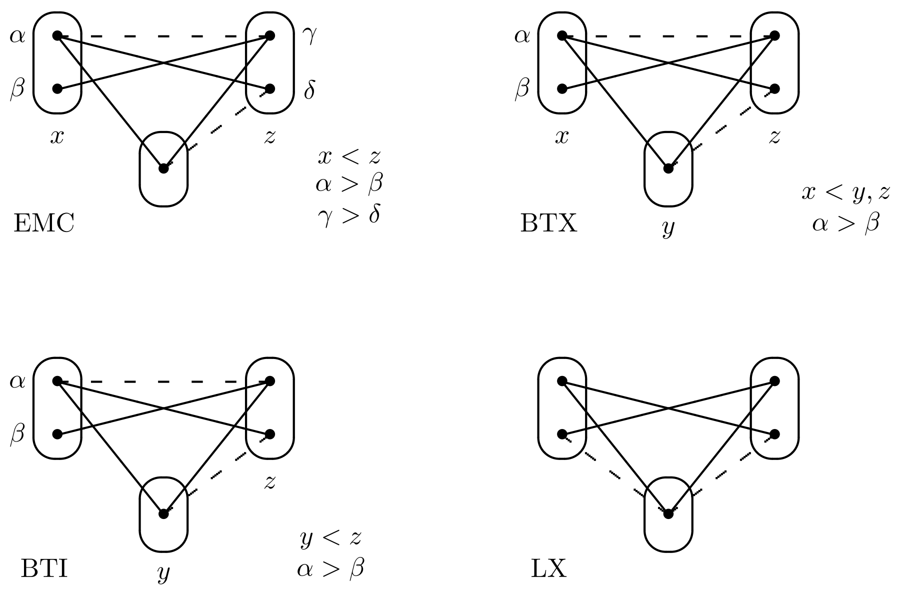

Figure 6 Partially-ordered patterns that are solved by arc consistency.

where a, b, c are non-zero constants. We say that X_i occurs *positively* (respectively, negatively) if $a > 0$ ($a < 0$). These constraints are max-closed if and only if at least one of the variables X_i, X_j occurs positively [36]. Suppose that in I, for all constraints on a pair of variables $X_i < X_j$ in which both variables occur negatively, the variable X_j only occurs negatively in other constraints. Then $I \in \text{CSP}_{\overline{\text{SP}}}(\text{EMC})$, and hence is solved by arc consistency.

In fact, if we consider only unmergeable patterns to which we then add a partial order to the variables and/or the domains, then there are just five patterns P such that arc consistency is a decision procedure for $\text{CSP}_{\overline{\text{SP}}}(P)$: BTP and the patterns EMC, BTX, BTI and LX shown in Figure 6 [20]. Given a fixed total order of the domain, there is a polynomial-time algorithm to find a total variable ordering such that any one of these patterns does not occur in an instance (or to determine that no such ordering exists). However, if the domain and variable orders are both unknown, then EMC, BTX and BTI become NP-complete to detect [20].

3.4 Non-binary CSPs

Most work on forbidden-pattern tractability has been restricted to binary CSPs. Indeed, notions such as microstructure and forbidden pattern do not (yet) have a widely-accepted generalisation to non-binary constraints. Nonetheless, the notion of arbitrary-arity patterns can be said to be already present in language classes defined by a polymorphism [11]. A polymorphism can be viewed as a forbidden pattern in each constraint relation R of the instance. The forbidden pattern corresponding to a polymorphism $f : D^r \to D$ consists of a set of r positive tuples and one negative tuple. The positive tuples $t_1, \ldots, t_r \in R$ are consistent assignments to the same variables and the negative tuple $f(t_1, \ldots, t_r) \notin R$ is the assignment to the same variables resulting from the pointwise application of f to the $t_1, \ldots, t_r$. By *forbidding* the pattern $(t_1, \ldots, t_r \in R,\ f(t_1, \ldots, t_r) \notin R)$, we impose the well-known polymorphism condition $t_1, \ldots, t_r \in R \implies f(t_1, \ldots, t_r) \in R$ [11].

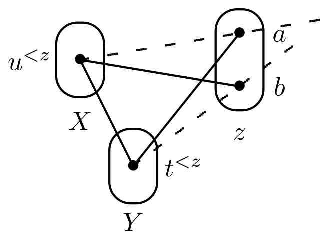

Figure 7 Illustration of a directional general-arity broken triangle.

As we have seen in Section 3.3, in the binary CSP, the broken-triangle property defines a tractable class $\mathrm{CSP}_{\overline{\mathrm{SP}}}(\mathrm{BTP})$. Although the definition of tractable classes by forbidden patterns in general-arity CSPs is an area which remains largely unexplored, a generalisation of the broken-triangle class $\mathrm{CSP}_{\overline{\mathrm{SP}}}(\mathrm{BTP})$ to general-arity CSPs has recently been given [14].

Purely for notational convenience we assume that a CSP instance I is given in the form of a set of negative (incompatible) tuples $\mathrm{NoGoods}(I)$, where a tuple is a set of variable-value assignments, and that the predicate $\mathrm{Good}(I, t)$ is true iff the tuple t does not contain any pair of distinct assignments to the same variable and $\nexists t' \subseteq t$ such that $t' \in \mathrm{NoGoods}(I)$. We write Good as a predicate whereas $\mathrm{Nogoods}(I)$ is a set to emphasize the asymmetry between the notions of positive and negative tuples. This asymmetry is not evident in the case of binary CSPs nor in the case of polymorphisms.

We suppose that a total ordering $<$ of the variables of a CSP instance I is given. We write $t^{<x}$ to represent the subset of the tuple t consisting of assignments to variables occurring before x in the order $<$, and $Vars(t)$ to denote the set of all variables assigned by t.

▶ **Definition 3.** A *directional general-arity broken triangle* (DGABTP) on assignments a, b to variable z in a CSP instance I is a pair of tuples t, u (containing no assignments to variable z) satisfying the following conditions:

1. $t^{<z}$ and $u^{<z}$ are non-empty,
2. $\mathrm{Good}(I, t^{<z} \cup u^{<z}) \quad \wedge \quad \mathrm{Good}(I, t^{<z} \cup \{\langle z, a\rangle\}) \quad \wedge \quad \mathrm{Good}(I, u^{<z} \cup \{\langle z, b\rangle\})$,
3. $t \cup \{\langle z, b\rangle\} \in \mathrm{NoGoods}(I) \quad \wedge \quad u \cup \{\langle z, a\rangle\} \in \mathrm{NoGoods}(I)$,
4. $\exists t'$ s.t. $Vars(t') = Vars(t) \quad \wedge \quad (t')^{<z} = t^{<z} \quad \wedge \quad t' \cup \{\langle z, a\rangle\} \notin \mathrm{NoGoods}(I)$,
5. $\exists u'$ s.t. $Vars(u') = Vars(u) \quad \wedge \quad (u')^{<z} = u^{<z} \quad \wedge \quad u' \cup \{\langle z, b\rangle\} \notin \mathrm{NoGoods}(I)$.

I satisfies the *directional general-arity broken-triangle property (DGABTP)* according to the variable ordering $<$ if no directional general-arity broken triangle occurs on any pair of values a, b for any variable z.

Points (1), (2) and (3) of Definition 3 are illustrated by Figure 7. This figure is similar to Figure 5 except that X,Y are sets of variables and $t^{<z}$,$u^{<z}$ are tuples. Note that the sets $X = Vars(u^{<z})$ and $Y = Vars(t^{<z})$ may overlap. Solid lines now represent partial solutions (i.e., consistent assignments to subsets of variables). The two dashed lines represent nogoods (i.e., tuples not in the constraint relation on its variables) $u \cup \{\langle z, a\rangle\}$ and $t \cup \{\langle z, b\rangle\}$ which possibly involve assignments to variables $w > z$. In the case of binary CSPs, a directional general-arity broken triangle is equivalent to a broken triangle as shown in Figure 5 (since nogoods, being binary, involve no other variables $w > z$ and the sets X, Y are necessarily singletons). Points (4) and (5) of Definition 3 are technical conditions (which always hold if a weak form of directional consistency holds) ensuring that the DGABTP can be tested in polynomial time for a given order whether constraints are given as tables of satisfying assignments or as nogoods.

Any instance I satisfying the DGABTP can be solved in polynomial time by repeatedly applying the following two operations: (i) merge two values in the last remaining variable (according to the order $<$); (ii) eliminate this variable when its domain becomes a singleton. *Merging* values $a, b \in \mathcal{D}(z)$ in a general-arity CSP instance I consists of replacing a, b in $\mathcal{D}(z)$ by a new value c which is compatible with all variable-value assignments compatible with at least one of the assignments $\langle z, a\rangle$ or $\langle z, b\rangle$, thus producing an instance I' with the new set of nogoods defined as follows:

$$\begin{aligned}
\mathrm{NoGoods}(I') \;=\;& \{t \in \mathrm{NoGoods}(I) \mid \langle z,a\rangle, \langle z,b\rangle \notin t\}\\
\cup\;& \{t \cup \{\langle z,c\rangle\} \mid t \cup \{\langle z,a\rangle\} \in \mathrm{NoGoods}(I) \;\wedge\\
& \quad \exists t' \in \mathrm{NoGoods}(I) \text{ s.t. } t' \subseteq t \cup \{\langle z,b\rangle\}\}\\
\cup\;& \{t \cup \{\langle z,c\rangle\} \mid t \cup \{\langle z,b\rangle\} \in \mathrm{NoGoods}(I) \;\wedge\\
& \quad \exists t' \in \mathrm{NoGoods}(I) \text{ s.t. } t' \subseteq t \cup \{\langle z,a\rangle\}\}\,.
\end{aligned}$$

In general, merging a pair of values in an instance I may produce an instance I' which is satisfiable even though I was not, but forbidding directional general-arity broken triangles prevents this from happening. Eliminating a variable z whose domain is a singleton $\{a\}$ consists in making the assignment $\langle z, a\rangle$ and eliminating $\langle z, a\rangle$ from all nogoods.

Unfortunately, when the variable order is not given, testing the existence of a variable ordering for which a CSP instance satisfies the DGABTP is NP-complete in the general-arity case [14]. This can be contrasted with the binary case in which this test is polytime.

Note that the set of general-arity CSP instances whose dual instance satisfies the BTP, denoted by DBTP, also defines a tractable class which can be recognised in polynomial time even if the ordering of the variables in the dual instance is unknown [44]. This DBTP class is incomparable with DGABTP (which is equivalent to BTP in binary CSP) since DBTP is known to be incomparable with the BTP class already in the special case of binary CSP [44]. A general-arity broken triangle can be said to be centred on a pair of *values* in the domain of a variable whereas a broken triangle in the dual instance is centred on a pair of *tuples* in a constraint relation. One consequence of this is that eliminating tuples from constraint relations cannot introduce broken triangles in the dual instance, whereas the DGABTP is only invariant under elimination of domain values. On the other hand, the DGABTP is invariant under adding a complete constraint (i.e., whose relation is the direct product of the domains of the variables in its scope) whereas this operation can introduce broken triangles in the dual instance. Another important difference is that DGABTP depends on an order on the variables whereas DBTP depends on an order on the constraints.

The generalisation of forbidden patterns to non-binary constraints is a largely unexplored area of research, but the generalisation of BTP to DGABTP has highlighted the asymmetry between positive and negative tuples when constraints are non-binary.

3.5 Quantified patterns

The notion of forbidding patterns has also been extended to rules based on applying a sequence of quantifiers to the variables and values in a pattern. This has led to the discovery of novel variable-elimination or value-elimination techniques [7, 12]. As an example, consider the broken triangle pattern shown in Figure 5. It is known that we can eliminate variables which are not the right-hand variable of a broken triangle: the resulting instance is satisfiable if and only if the original instance was satisfiable [16]. This variable-elimination rule can be strictly generalised to the following rule illustrated in Figure 8: a variable z can be eliminated from an instance I without changing the satisfiability of I if for all other variables

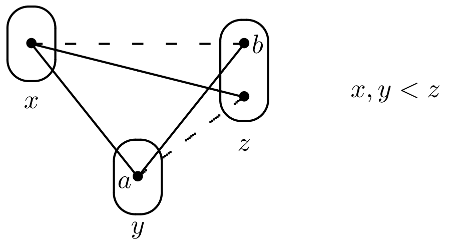

Figure 8 In the ∀∃BTP class, for all pairs of variables $y < z$, $\forall a \in \mathcal{D}(y)$, $\exists b \in \mathcal{D}(z)$ such that for all variables $x < z$, the broken triangle pattern shown does not occur.

y, $\forall a \in \mathcal{D}(y)$, $\exists b \in \mathcal{D}(z)$ with ab a positive edge in I such that no broken triangle exists including this edge ab [12]. The class of binary CSP instances, known as ∀∃BTP, which are such that all variables can be eliminated according to this rule strictly generalises the tractable class $\mathrm{CSP}_{\overline{\mathrm{SP}}}(\mathrm{BTP})$, since the BTP imposes the same condition but for *all* $b \in \mathcal{D}(X_k)$.

Let I be a binary arc-consistent CSP instance in the ∀∃BTP class and let s be a solution to the instance obtained by eliminating the last variable z from I. We will give a sketch proof that s can be extended to a solution for I. (We refer the reader to [12] for full details.) By assumption, $\forall y \in X \setminus \{z\}$, $\forall a \in \mathcal{D}(y)$, $\exists b^y_a \in \mathcal{D}(z)$ with ab^y_a a positive edge such that $\forall x \in X \setminus \{y, z\}$, $\forall c \in \mathcal{D}(x)$ with ac a positive edge and $b^y_a c$ a negative edge, $\forall d \in \mathcal{D}(z)$ with cd a positive edge, ad is also a positive edge. For $v \in X \setminus \{z\}$, let $\mathrm{Im}(v) := \{d \in \mathcal{D}(z) \mid s(v)d$ a positive edge$\}$, where $s(v)$ is the value assigned to variable v by s. If $x, y \in X \setminus \{z\}$ are such that $s(x)b^y_{s(y)}$ is a negative edge, then the ∀∃BTP property implies that $\mathrm{Im}(x) \subsetneq \mathrm{Im}(y)$ [12]. Now choose some $y \in X \setminus \{z\}$ such that $\mathrm{Im}(y)$ is minimal for inclusion among the sets $\mathrm{Im}(v)$ $(v \in X \setminus \{z\})$. Then the assignment $\langle z, b^y_{s(y)} \rangle$ is compatible with all the assignments $s(x)$ $(x \in X \setminus \{y, z\})$, otherwise we would have $\mathrm{Im}(x) \subsetneq \mathrm{Im}(y)$ (contradicting the minimality of $\mathrm{Im}(y)$). Therefore, s can be extended to a solution to I, by making the assignment $s(z) = b_{s(y)}$.

3.6 Topological minor patterns

We now present a new operation on patterns which allows us to define the notion of a topological minor of a pattern (and hence of a binary CSP instance). This new operation is analogous to the operation of eliminating subdivisions (vertices of degree 2) that is used to define a topological minor of a graph [25]. However, since patterns contain two kinds of edges, the definition is slightly more complicated.

This new operation, path reduction, will sometimes lead to the introduction of edges in $E^+ \cap E^-$ in a coloured microstructure $\langle A_1, \ldots, A_n, E^+, E^- \rangle$. This is why we do not impose the restriction that E^+ and E^- be disjoint in a pattern.

In a pattern $P = \langle A_1, \ldots, A_n, E^+, E^- \rangle$, we say that two parts A_i, A_j are *directly connected* if there is at least one (positive or negative) edge $\{p, q\} \in E^+ \cup E^-$ with $p \in A_i$ and $q \in A_j$.

If A_i, A_j are not directly connected and A_k is directly connected only to A_i and A_j, then the following operation can be performed, which is known as *path reduction*:

1. $\forall p \in A_i$, $\forall q \in A_j$: if $\exists r \in A_k$ such that $\{p, r\}, \{r, q\} \in E^+$, then introduce a new positive edge $\{p, q\}$,
2. $\forall p \in A_i$, $\forall q \in A_j$: if $\exists r, s \in A_k$ such that $\{p, r\}, \{s, q\} \in E^-$, then introduce a new negative edge $\{p, q\}$,
3. remove the part A_k and all edges containing points in A_k.

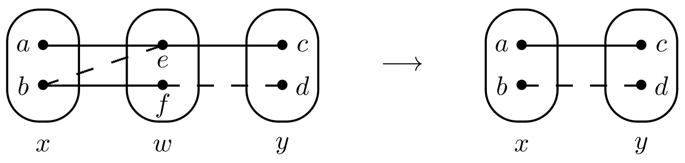

Figure 9 Path reduction removes the part w.

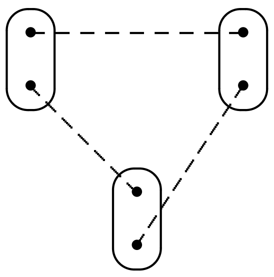

Figure 10 A pattern which defines the class of acyclic binary CSP instances when forbidden as a topological minor.

This operation is illustrated in Figure 9. Positive and negative edges are treated differently in this definition; this is because for $p \in A_i$ and $q \in A_j$ to be part of a solution to the sub-instance on variables X_i, X_j, X_k, the points p and q must both be compatible with some common point $r \in A_k$, whereas p and q may be incompatible if they are each incompatible with some point in A_k, not necessarily the same point. In Figure 9, after the path reduction operation which eliminates w, we have a positive edge ac (thanks to the edges ae and ec), but no positive edge bc. We also have a negative edge bd (thanks to the edges be and fd). As in the case of non-binary patterns (c.f. Section 3.4), it is essential to introduce an asymmetry between positive and negative edges in order to obtain a useful notion.

A pattern P occurs as a *topological minor* of a pattern Q if Q can be transformed into a homomorphic image of P by a sequence of substructure operations (listed above in Section 3.1) and path reductions.

We use the notation $\mathrm{CSP}_{\overline{\mathrm{TM}}}(P)$ to represent the set of binary CSP instances in which the pattern P does not occur as a topological minor. For each pattern P there are therefore two distinct notions of tractability: a pattern P is *sub-pattern tractable* if there is a polynomial-time algorithm to solve $\mathrm{CSP}_{\overline{\mathrm{SP}}}(P)$; a pattern P is *topological-minor tractable* if there is a polynomial-time algorithm to solve $\mathrm{CSP}_{\overline{\mathrm{TM}}}(P)$. A pattern which is sub-pattern tractable is topological-minor tractable since any pattern that occurs as a sub-pattern will also occur as a topological minor.

One important tractable class of binary CSP instances is the class of instances whose constraint graph is acyclic [28]. However, this class cannot be defined by a finite set of forbidden sub-patterns [9]. On the other hand, it is straightforward to characterise the class of acyclic instances by forbidding a single pattern as a topological minor. Forbidding the pattern shown in Figure 10 as a topological minor exactly defines the class of binary CSP instances whose constraint graph is acyclic. Indeed, this idea can easily be extended to any of the tractable classes of binary CSP instances defined by imposing any fixed bound on the treewidth of the constraint graph [29] using the graph minor theorem [49]. However, it remains to be seen whether the notion of patterns occurring as topological minors can be used to define a practically useful and genuinely novel tractable class.

4 Classes requiring a level of consistency

Some hybrid tractable classes have been defined which guarantee global consistency if some local property holds after establishing a certain level of local consistency. One example is that the constraints can be decomposed into the join of arity-r constraints after establishing strong $d(r-1)+1$ consistency, where d is the maximum domain size [23]. Of course, in general, establishing this level of consistency introduces constraints of order $d(r-1)$, so the assumption that constraints are of arity r is very strong. This class has been generalised to the class of arity-r CSP instances which are strongly $((m+1)(r-1)+1)$-consistent, where given an r-ary constraint and an instantiation of $r-1$ of the variables that participate in the constraint, the parameter m (called the tightness) is an upper bound on the number of instantiations of the rth variable that satisfy the constraint in the case that this is not the whole domain [54].

Naanaa [46] has proposed a generalisation of m-tightness. Let E be a finite set and let $\{E_i\}_{i\in I}$ be a finite family of subsets of E. The family $\{E_i\}_{i\in I}$ is said to be *independent* if and only if for all $J \subsetneq I$,

$$\bigcap_{i\in I} E_i \subsetneq \bigcap_{j\in J} E_j \,.$$

In particular, observe that $\{E_i\}_{i\in I}$ cannot be independent if $\exists j \neq j' \in I$ such that $E_j \subseteq E_{j'}$, since in this case and with $J = I \setminus \{j'\}$ we would have

$$\bigcap_{i\in I} E_i = \bigcap_{j\in J} E_j \,.$$

Let I be a CSP instance whose variables are totally ordered by $<$. Let $\langle \sigma, R\rangle$ be an r-ary constraint whose scope σ contains a variable x and let t be a tuple that instantiates the $r-1$ remaining variables of σ. Denote by $R_x(t)$ the set of values in $\mathcal{D}(x)$ that can extend t to form a tuple in the relation R. The *directional extension* of tuple t to variable x w.r.t. R and $<$ is defined to be $R_x(t)$ if x is the last (w.r.t. the order $<$) variable in σ, and $\mathcal{D}(x)$ otherwise. A family of extensions of tuples $t \in T$ is said to be consistent if and only if the tuple formed by the join $\bowtie_{t\in T} t$ of the corresponding tuples is consistent. With respect to the ordering $<$, the *directional rank* of x in I is the size of the largest independent and consistent family of directional extensions to x, and the *directional rank* κ of I is the maximum directional rank over all its variables. If I is a CSP instance with constraints of arity no greater than r which has directional rank no greater than κ and is directional strong $(\kappa(r-1)+1)$-consistent, then I is globally consistent [46]. In general, establishing this level of consistency introduces constraints of arity $\kappa(r-1)$ which is no greater than r only if $\kappa = 1$ or $(\kappa = 2 \ \wedge \ r = 2)$.

There is an interesting link between directional rank and forbidden patterns. Directional rank 1 is equivalent to the broken triangle property (i.e., forbidding as a subpattern the pattern shown in Figure 5) and directional rank $\kappa > 1$ subsumes an extension of the broken triangle property known as $(\kappa+1)$-BTP [17]. A binary CSP instance satisfies k-BTP if for all variables z and for all sets S of k variables occurring before z in the variable ordering, $\exists x, y \in S$ such that there are no broken triangles on variables x, y, z. We can see that 2-BTP corresponds exactly to the broken triangle property. If a binary CSP instance satisfies 3-BTP after establishment of directional strong 3-consistency, then it has directional rank $\kappa = 2$ and is directional strong $(\kappa(r-1)+1)$-consistent (since $r = 2$), and hence is globally consistent [46]. Directional rank 2 strictly subsumes 3-BTP since it is equivalent to forbidding (as a subpattern) the pattern shown in Figure 11. This is a natural generalisation of the broken-triangle pattern shown in Figure 5, but, unfortunately, we also require strong

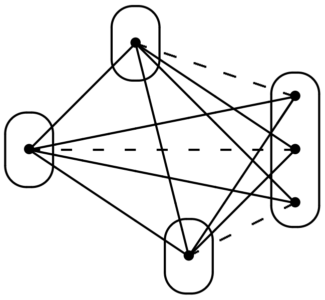

Figure 11 A binary CSP instance has directional rank 2 if this pattern does not occur in the instance.

directional 3-consistency to obtain a tractable class and establishing this level of consistency may introduce the pattern.

5 Microstructure-based classes

We recall the definition of microstructure. We have seen that a binary CSP instance on variables $X_1, \ldots, X_n$ can be represented by the domain $\mathcal{D}(X_i)$ of each variable X_i and a binary relation R_{ij} for each pair of variables X_i, X_j $(i \neq j)$ consisting of all possible consistent assignments to this pair of variables. If I is a binary CSP instance, then its *microstructure* is a graph $\langle A, E \rangle$ where $A = \{(X_i, a) \mid a \in \mathcal{D}(X_i)\}$ is the set of possible variable-value assignments and $E = \{\{(X_i, a), (X_j, b)\} \mid (a, b) \in R_{ij}\}$ [37]. The microstructure relies on both the structure and the relations of the instance I and so is a natural place to look for hybrid tractable classes. In this section we study properties of the graph $\langle A, E \rangle$ in which A is not partitioned into parts corresponding to variables (as was the case in patterns, studied in Section 3). Ignoring variable information has the obvious disadvantage that we lose possibly valuable information, but has the advantage that deep theorems from graph theory can be directly applied.

The *complement* of a graph $G = \langle V, E \rangle$ is the graph with vertices V and whose edges are the non-edges of G. The *microstructure complement* is the complement of the microstructure. Solutions to I are in one-to-one correspondence with the n-cliques of the microstructure of I and with the size-n independent sets of the microstructure complement of I.

The *chromatic number* of a graph is the smallest number of colours required to colour its vertices so that no two adjacent vertices have the same colour. A graph G is *perfect* if for every induced subgraph H of G, the chromatic number of H is equal to the size of the largest clique contained in H. Since a maximum clique in a perfect graph can be found in polynomial time [33], the class of binary CSP instances with a perfect microstructure is tractable as a direct consequence, as observed in [50]. Perfect graphs can also be recognized in polynomial time [3].

For a class of graphs $\mathcal{C}$, a graph G is $\mathcal{C}$-free if no induced subgraph of G is isomorphic to any graph in $\mathcal{C}$. The *cycle of order* k is the graph with vertices $v_1, \ldots, v_k$ and edges $\{v_k, v_1\}$ and $\{v_i, v_{i+1}\}$ for $i = 1, \ldots, k - 1$. A *hole* is a cycle of length $k \geq 5$. An *antihole* is the complement of a hole. An alternative definition of perfect graphs is that a graph is perfect if and only if it is (odd-hole,odd-antihole)-free [4]. Chordal graphs are examples of perfect graphs. Interesting examples of binary CSP instances whose microstructure is perfect are

- instances with unary constraints together with a global AllDifferent constraint [50],

- instances which are arc consistent and max-closed after independent (and possibly unknown) permutations of each domain [31].

6 Weakly or strongly constrained instances

One way to define a tractable class is to only allow a small number of weak constraints, in order to guarantee that the instance is always satisfiable. For example, if each variable is in the scope of at most t constraints and in each constraint relation the proportion of tuples that are disallowed is strictly less than $1/e(r(t-1)+1)$, where e is the base of natural logarithms and r the arity of the constraint, then the instance is necessarily satisfiable [47].

Another way to define a tractable class is to consider only instances which are sufficiently strongly constrained so that there is necessarily only a small number of partial solutions examined during search (and hence a small number of solutions). A simple example of a condition that guarantees a polynomial number of solutions is functionality. A constraint $\langle \sigma, R \rangle$ is *functional* on variable $X_i \in \sigma$ if the relation R contains no two assignments differing only at variable X_i. A CSP instance is *functional with root set of size* k if there exists a variable ordering $X_1 < \ldots < X_n$ such that, for all $i \in \{k+1, \ldots, n\}$, there is some constraint $\langle \sigma, R \rangle$ with $X_i \in \sigma \subseteq \{X_1, \ldots, X_i\}$ that is functional on X_i. (Note that this implies tractability.) In the case of binary CSP instances, a minimum root set can be found in polynomial time [22]. Unfortunately, determining the size of a minimum root set is NP-hard for ternary CSP instances [8].

Another condition that guarantees a backtracking search tree of polynomial size (assuming domain size bounded by a constant) is the k-Turan property [8] which we now define. Indeed, this property is very strong since it guarantees a polynomial-size search tree for all variable orderings. We say that a subset of variables S *represents* another set T if $S \subseteq T$. An (n,k)*-Turan system* is a pair $\langle X, B \rangle$ where B is a collection of subsets of the n-element set X such that every k-element subset of X is represented by some set in B. For example, let $\mathcal{C}_{4\text{Turan}}$ be the class of binary CSP instances over a set of variables X, each with a Boolean domain, in which each constraint is equivalent to a 3SAT clause and for each quadruple of distinct variables X_i, X_j, X_k, X_ℓ there is at least one ternary constraint whose scope is a subset of these variables. In this example, every 4-element subset of the set of n variables X is represented by the scope of some ternary constraint, and hence $\langle X, S \rangle$, where S is the set of constraint scopes, is an $(n,4)$-Turan system. An n-variable CSP instance over domain D and variables X is k*-Turan* if $\langle X, B \rangle$ forms an (n,k)-Turan system where B is the set of the scopes of the constraints $\langle \sigma, R \rangle$ for which

$$\forall a, b \in D, \ \{a,b\}^{|\sigma|} \not\subseteq R\,.$$

This condition says that at least one tuple is disallowed by the constraint over each Boolean subdomain $\{a,b\}$ of D. In the class $\mathcal{C}_{4\text{Turan}}$, all constraints satisfy this condition and hence $\mathcal{C}_{4\text{Turan}}$ is tractable since all instances in this class satisfy the 4-Turan property. Generalising this example, the class of k-SAT instances where every k'-tuple of variables, where $k' \geq k$, is restricted by a clause is k'-Turan and hence tractable.

It is an open question whether the k-Turan property can be relaxed in a way that guarantees a polynomial-size search tree for just one variable ordering, rather than all variable orderings, while imposing a weaker condition than functionality.

7 Valued CSPs

CSPs are inherently decision problems. In this section we discuss hybrid classes of valued CSPs, which is a generalisation of CSPs to problem that capture both decision and optimisation problems (and their combinations).

We denote by $\overline{\mathbb{Q}} = \mathbb{Q} \cup \{\infty\}$ the set of extended rationals. A *valued constraint satisfaction problem* (VCSP) instance I is given by a triple $\langle X, D, C\rangle$, where $X = \{X_1, \ldots, X_n\}$ is a finite set of variables, D is a finite set of values, and $C : D^n \to \overline{\mathbb{Q}}$ is an objective function expressed as a sum of valued constraints, i.e., $C(X_1, \ldots, X_n) = \sum_{i=1}^{q} \gamma_i(\mathbf{v}_i)$, where $\gamma_i : D^{k_i} \to \overline{\mathbb{Q}}$ is a *cost function* of arity k_i and $\mathbf{v}_i \in X^{k_i}$ is the scope of the valued constraint $\gamma_i(\mathbf{v}_i)$. The question is to find an assignment of values to the variables that minimises the objective function C.

The class of VCSP instances with $\{0, \infty\}$-valued cost functions corresponds to the class of CSP instances. VCSP instances with $\{0, 1\}$-valued cost functions are known as Min-CSPs. VCSP instances with $\mathbb{Q}$-valued cost functions are known as finite-valued CSPs [53].

Similarly to the case of CSPs, language-restricted VCSPs parameterised by the set of allowed cost functions in the instance have been studied [42]. In this section we will mention known results on the complexity of hybrid classes of VCSPs.

The idea of lifted languages briefly discussed in Section 2.3 has also been applied to certain VCSPs [39, 52].

7.1 JWP and generalisations

The study of hybrid classes of VCSPs was initiated in [18], where an interesting hybrid class called the *joint winner property* (JWP) was discovered. A class of *binary* VCSPs satisfies the JWP if for any three variable-value assignments (to three distinct variables), the multiset of pairwise costs imposed by the binary valued constraints does not have a unique minimum. If there is no valued constraint with the scope, say, $\mathbf{v} = \langle X_i, X_j\rangle$, then we view it as a 0-valued constraint $\gamma(\mathbf{v})$, where $\gamma : D^2 \to \overline{\mathbb{Q}}$ is the constant-0 binary cost function. Note that the unary valued constraints in a VCSP that satisfies the JWP can be arbitrary. JWP generalises the tractable pattern NEGTRANS discussed in Section 3.2, as the NEGTRANS pattern precisely forbids the combination of one 0 cost and two ∞ costs.

Following the discovery of JWP, Cooper and Živný classified classes of binary CSPs, Min-CSPs, finite-valued CSPs, and VCSPs parametrised by the allowed types of costs in triples of variable-value assignments (called triangles) [19]. In all studied cases, JWP was essentially the only interesting tractable case. Moreover, [19] generalised JWP to the tractable class of VCSPs with the *cross-free convex* (CFC) property.

A function $g : \{0, \ldots, s\} \to \overline{\mathbb{Q}}$ is called *convex on the interval* $[l, u]$ if g is $\mathbb{Q}$-valued on the interval $[l, u]$ and the derivative of g is nondecreasing on $[l, u]$, that is, $g(m + 2) - g(m + 1) \geq g(m + 1) - g(m)$ for all $m = l, \ldots, u - 2$.

Sets $A_1, \ldots, A_r \subseteq A$ are called *cross-free* if for all $1 \leq i, j \leq r$, either $A_i \subseteq A_j$, or $A_i \supseteq A_j$, or $A_i \cap A_j = \emptyset$, or $A_i \cup A_j = A$ [51].

We interpret a solution $s : X \to D$ to a VCSP instance $I = \langle X, D, C\rangle$ as its set of variable-value assignments $\{\langle X_i, s(X_i)\rangle \mid i = 1, \ldots, n\}$. If A_i is a set of variable-value assignments of a VCSP instance I and s a solution to I, then we use the notation $|s \cap A_i|$ to represent the number of variable-value assignments in the solution s that lie in A_i.

Finally we have everything we need to define the CFC property. Let I be a VCSP instance. Let $A_1, \ldots, A_r$ be cross-free sets of variable-value assignments of I. Let s_i be the number of distinct variables occurring in the set A_i. Instance I satisfies the *cross-free convexity* *property* if the objective function of I is $g(s) = g_1(|s \cap A_1|) + \ldots + g_r(|s \cap A_r|)$, where each $g_i : [0, s_i] \to \overline{\mathbb{Q}}$ $(i = 1, \ldots, r)$ is convex on an interval $[l_i, u_i] \subseteq [0, s_i]$ and $g_i(z) = \infty$ for $z \in [0, l_i - 1] \cup [u_i + 1, s_i]$.

We remark that the functions g_i above are not the cost functions associated with the valued constraints. Note that similarly to JWP, the addition of any unary cost function cannot destroy the cross-free convexity property because for each variable-value assignment $\langle X_i, a\rangle$ we can add the singleton $A_i = \{\langle X_i, a\rangle\}$, which is necessarily either disjoint from or a subset of any other set A_j (and furthermore the corresponding function $g_i : \{0, 1\} \to \overline{\mathbb{Q}}$ is trivially convex).

A special case of the CFC property are global cardinality constraints [48, 56] on cross-free sets.

▶ **Example 4.** To give a concrete example, consider a company which needs to assign staff to a project, minimising total salary cost while respecting constraints concerning the minimum number of personnel from each section, the maximum total number of staff on the project, as well as the availability of each member of staff. We can code this as a VCSP with a boolean variable X_i for each member of staff with X_i assigned the value true if the person in question is assigned to the project. The availability of each member of staff, as well as his/her salary, can be coded as a unary cost function. Assuming each member of staff belongs to a single section of the company, the remaining constraints are global cardinality constraints on cross-free sets.

The employees, numbered from 1 to n, are partitioned into sections S_j $(j = 1, \ldots, t)$. Let $A_j = \{\langle X_i, \mathit{true}\rangle \mid i \in S_j\}$ $(j = 1, \ldots, t)$, $A_0 = \bigcup_{j=1}^{t} A_j$, and $A_{t+i} = \{\langle X_i, \mathit{true}\rangle\}$ $(i = 1, \ldots, n)$. The sets A_i $(i = 0, \ldots, t + n)$ are cross-free. Let $g_{\leq m} : \{0, \ldots, n\} \to \overline{\mathbb{Q}}$ be the function given by $g_{\leq m}(x) = 0$ if $x \leq m$ and $g_{\leq m}(x) = \infty$ if $x > m$, with $g_{\geq m}$ defined similarly. For $i = 1, \ldots, n$, let $h_i : \{0, 1\} \to \overline{\mathbb{Q}}$ be the function given by $h_i(0) = 0$ and $h_i(1)$ equal to the salary of employee i if he/she is available to work on the project, ∞ if not. Suppose that the project requires at least n_i employees from section i, for each $i = 1, \ldots, t$, making a total of at most N staff members. Then the objective function is

$$g_{\leq N}(|s \cap A_0|) \; + \; \sum_{j=1}^{t} g_{\geq n_i}(|s \cap A_j|) \; + \; \sum_{i=1}^{n} h_i(|s \cap A_{t+i}|)\,.$$

The functions $g_{\leq N}$ and $g_{\geq n_i}$ are convex (indeed constant) on the interval on which they are finite, and each h_i is trivially convex on the interval $[0, 1]$.

It has recently been shown[1] that the class of convex cross-free VCSPs is a special case of M$^\natural$-convex functions studied in [45] and that JWP precisely captures binary M$^\natural$-completable functions.

7.2 Planarity

Similarly to the planar CSPs discussed in Section 2.1, one can define planar VCSPs. Fulla and Živný gave necessary conditions on the tractability of planar VCSPs [30]. In particular, they showed that if Γ is a Boolean valued constraint language such that VCSP(Γ) is intractable then VCSP$_p(\Gamma)$ is intractable unless Γ is self-complementary in the valued sense; i.e., for every $\gamma \in \Gamma$ and every tuple t, $\gamma(t) = \gamma(\bar{t})$. This is a generalisation of the self-complementarity

[1] Personal communication with Yuni Iwamasa and Kazuo Murota.

condition for planar CSPs from [26] discussed in Section 2.1. Furthermore, [30] obtained a dichotomy for planar VCSPs for conservative language (i.e., languages containing all $\{0,1\}$-valued unary cost functions) over arbitrary finite domains. As it turns out the planarity restriction does not give any new tractable languages in this setting, and the classification from [30] sharpens the classification of conservative valued constraint languages obtained in [41].

References

1 Libor Barto and Marcin Kozik. Constraint satisfaction problems solvable by local consistency methods. *J. ACM*, 61(1):3:1–3:19, 2014.

2 Andrei Bulatov, Andrei Krokhin, and Peter Jeavons. Classifying the Complexity of Constraints using Finite Algebras. *SIAM Journal on Computing*, 34(3):720–742, 2005.

3 Maria Chudnovsky, Gérard Cornuéjols, Xinming Liu, Paul Seymour, and Kristina Vuskovic. Recognizing Berge graphs. *Combinatorica*, 25(2):143–186, 2005.

4 Maria Chudnovsky, Neil Robertson, Paul Seymour, and Robin Thomas. The strong perfect graph theorem. *Annals of Math.*, 164(1):51–229, 2006.

5 David A. Cohen, Martin C. Cooper, Páidí Creed, Peter Jeavons, and Stanislav Živný. An algebraic theory of complexity for discrete optimisation. *SIAM Journal on Computing*, 42(5):915–1939, 2013.

6 David A. Cohen, Martin C. Cooper, Páidí Creed, Dániel Marx, and András Z. Salamon. The tractability of CSP classes defined by forbidden patterns. *J. Artif. Intell. Res. (JAIR)*, 45:47–78, 2012.

7 David A. Cohen, Martin C. Cooper, Guillaume Escamoche, and Stanislav Živný. Variable and value elimination in binary constraint satisfaction via forbidden patterns. *Journal of Computer and System Sciences*, 81(7):1127–1143, 2015.

8 David A. Cohen, Martin C. Cooper, Martin J. Green, and Dániel Marx. On guaranteeing polynomially bounded search tree size. In Jimmy Ho-Man Lee, editor, *CP 2011*, volume 6876 of *Lecture Notes in Computer Science*, pages 160–171. Springer, 2011.

9 David A. Cohen, Martin C. Cooper, Peter G. Jeavons, and Stanislav Živný. Tractable classes of binary CSPs defined by excluded topological minors. In Qiang Yang and Michael Wooldridge, editors, *IJCAI 2015*, pages 1945–1951. AAAI Press, 2015.

10 David A. Cohen, Peter Jeavons, Christopher Jefferson, Karen E. Petrie, and Barbara M. Smith. Symmetry definitions for constraint satisfaction problems. *Constraints*, 11(2-3):115–137, 2006.

11 David A. Cohen and Peter G. Jeavons. The complexity of constraint languages. In Francesca Rossi, Peter van Beek, and Toby Walsh, editors, *Handbook of Constraint Programming*, pages 245–280. Elsevier, 2006.

12 Martin C. Cooper. Beyond consistency and substitutability. In Barry O'Sullivan, editor, *CP 2014*, volume 8656 of *Lecture Notes in Computer Science*, pages 256–271. Springer, 2014.

13 Martin C. Cooper, David A. Cohen, and Peter Jeavons. Characterising tractable constraints. *Artif. Intell.*, 65(2):347–361, 1994.

14 Martin C. Cooper, Aymeric Duchein, Achref El Mouelhi, Guillaume Escamocher, Cyril Terrioux, and Bruno Zanuttini. Broken triangles: From value merging to a tractable class of general-arity constraint satisfaction problems. *Artificial Intelligence*, 234:196–218, 2016.

15 Martin C. Cooper and Guillaume Escamocher. Characterising the complexity of constraint satisfaction problems defined by 2-constraint forbidden patterns. *Discrete Applied Mathematics*, 184:89–113, 2015.

16 Martin C. Cooper, Peter G. Jeavons, and András Z. Salamon. Generalizing constraint satisfaction on trees: Hybrid tractability and variable elimination. *Artif. Intell.*, 174(9-10):570–584, 2010.

17 Martin C. Cooper, Philippe Jégou, and Cyril Terrioux. A microstructure-based family of tractable classes for CSPs. In Gilles Pesant, editor, *CP 2015*, volume 9255 of *Lecture Notes in Computer Science*, pages 74–88. Springer, 2015.

18 Martin C. Cooper and Stanislav Živný. Hybrid tractability of valued constraint problems. *Artificial Intelligence*, 175(9-10):1555–1569, 2011.

19 Martin C. Cooper and Stanislav Živný. Tractable triangles and cross-free convexity in discrete optimisation. *J. Artif. Intell. Res. (JAIR)*, 44:455–490, 2012.

20 Martin C. Cooper and Stanislav Živný. The power of arc consistency for CSPs defined by partially-ordered forbidden patterns. In *31st Annual ACM/IEEE Symposium on Logic in Computer Science, LICS*, 2016.

21 Víctor Dalmau and Daniel K. Ford. Generalized satisfability with limited occurrences per variable: A study through delta-matroid parity. In *Proceedings of the 28th International Symposium on Mathematical Foundations of Computer Science (MFCS'03)*, volume 2747 of *Lecture Notes in Computer Science*, pages 358–367. Springer, 2003.

22 Philippe David. Using pivot consistency to decompose and solve functional CSPs. *J. Artif. Intell. Res. (JAIR)*, 2:447–474, 1995.

23 Rina Dechter. From local to global consistency. *Artif. Intell.*, 55(1):87–108, 1992.

24 Rina Dechter. *Constraint processing*. Elsevier Morgan Kaufmann, 2003.

25 Reinhard Diestel. *Graph Theory*. Springer, fourth edition, 2010.

26 Zdeněk Dvořák and Martin Kupec. On Planar Boolean CSP. In *Proceedings of the 42nd International Colloquium on Automata, Languages and Programming (ICALP'15)*, volume 9134 of *Lecture Notes in Computer Science*, pages 432–443. Springer, 2015.

27 Tomás Feder. Fanout limitations on constraint systems. *Theoretical Computer Science*, 255(1-2):281–293, 2001.

28 Eugene C. Freuder. A sufficient condition for backtrack-free search. *J. ACM*, 29(1):24–32, 1982.

29 Eugene C. Freuder. A sufficient condition for backtrack-bounded search. *J. ACM*, 32:755–761, 1985.

30 Peter Fulla and Stanislav Živný. On Planar Valued CSPs. In *Proceedings of the 41st International Symposium on Mathematical Foundations of Computer Science (MFCS'16)*, 2016. Full version arXiv:1602.06323.

31 Martin J. Green and David A. Cohen. Domain permutation reduction for constraint satisfaction problems. *Artif. Intell.*, 172(8-9):1094–1118, 2008.

32 Martin Grohe. The complexity of homomorphism and constraint satisfaction problems seen from the other side. *J. ACM*, 54(1):1–24, March 2007.

33 Martin Grötschel, Laszlo Lovasz, and Alexander Schrijver. The ellipsoid method and its consequences in combinatorial optimization. *Combinatorica*, 1:169–198, 1981.

34 Pavol Hell and Jaroslav Nešetřil. Colouring, constraint satisfaction, and complexity. *Computer Science Review*, 2(3):143–163, 2008.

35 Pawel M. Idziak, Petar Markovic, Ralph McKenzie, Matthew Valeriote, and Ross Willard. Tractability and learnability arising from algebras with few subpowers. *SIAM J. Comput.*, 39(7):3023–3037, 2010.

36 Peter G. Jeavons and Martin C. Cooper. Tractable constraints on ordered domains. *Artif. Intell.*, 79(2):327–339, 1995.

37 Philippe Jégou. Decomposition of domains based on the micro-structure of finite constraint-satisfaction problems. In Richard Fikes and Wendy G. Lehnert, editors, *Proceedings of the*

11th National Conference on Artificial Intelligence (AAAI'93), pages 731–736. AAAI Press / The MIT Press, 1993.

38 Alexandr Kazda, Vladimir Kolmogorov, and Michal Rolínek. Even delta-matroids and the complexity of planar Boolean CSPs. In *Proceedings of the 28th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA'17)*, 2017.

39 Vladimir Kolmogorov, Michal Rolínek, and Rustem Takhanov. Effectiveness of structural restrictions for hybrid CSPs. In *Proceedings of the 26th International Symposium on Algorithms and Computation (ISAAC'15)*, volume 9472 of *Lecture Notes in Computer Science*, pages 566–577. Springer, 2015.

40 Vladimir Kolmogorov, Johan Thapper, and Stanislav Živný. The power of linear programming for general-valued csps. *SIAM J. Comput.*, 44(1):1–36, 2015.

41 Vladimir Kolmogorov and Stanislav Živný. The complexity of conservative valued CSPs. *Journal of the ACM*, 60(2), 2013. Article No. 10.

42 Andrei Krokhin and Stanislav Živný. The complexity of valued CSPs. In Andrei Krokhin and Stanislav Živný, editors, *The Constraint Satisfaction Problem: Complexity and Approximability*, volume 7 of *Dagstuhl Follow-Ups*, pages 229–261. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, Dagstuhl, Germany, 2017.

43 Achref El Mouelhi, Philippe Jégou, and Cyril Terrioux. Different classes of graphs to represent microstructures for CSPs. In Madalina Croitoru, Sebastian Rudolph, Stefan Woltran, and Christophe Gonzales, editors, *Graph Structures for Knowledge Representation and Reasoning - Third International Workshop, GKR 2013*, volume 8323 of *Lecture Notes in Computer Science*, pages 21–38. Springer, 2013.

44 Achref El Mouelhi, Philippe Jégou, and Cyril Terrioux. A hybrid tractable class for non-binary CSPs. *Constraints*, 20(4):383–413, 2015.

45 Kazuo Murota. *Discrete Convex Analysis.* SIAM, 2003.

46 Wady Naanaa. Unifying and extending hybrid tractable classes of CSPs. *J. Exp. Theor. Artif. Intell.*, 25(4):407–424, 2013.

47 Justin K. Pearson and Peter G. Jeavons. A survey of tractable constraint satisfaction problems. Technical Report CSD-TR-97-15, Royal Holloway, University of London, July 1997.

48 Jean-Charles Régin. A filtering algorithm for constraints of difference in CSPs. In *12th National Conference on Artificial Intelligence (AAAI'94)*, volume 1, pages 362–367, 1994.

49 Neil Robertson and Paul D. Seymour. Graph minors. XX. Wagner's conjecture. *J. Comb. Theory, Ser. B*, 92(2):325–357, 2004.

50 András Z. Salamon and Peter G. Jeavons. Perfect constraints are tractable. In Peter J. Stuckey, editor, *CP 2008*, volume 5202 of *Lecture Notes in Computer Science*, pages 524–528. Springer, 2008.

51 Alexander Schrijver. *Combinatorial Optimization: Polyhedra and Efficiency*, volume 24 of *Algorithms and Combinatorics.* Springer, 2003.

52 Rustem Takhanov. Hybrid (V)CSPs and algebraic reductions. *CoRR*, abs/1506.06540, 2015.

53 Johan Thapper and Stanislav Živný. The complexity of finite-valued CSPs. *Journal of the ACM*, 63(4), 2016. Article No. 37.

54 Peter van Beek and Rina Dechter. Constraint tightness and looseness versus local and global consistency. *J. ACM*, 44(4):549–566, 1997.

55 Willem-Jan van Hoeve and Irit Katriel. Global constraints. In Francesca Rossi, Peter van Beek, and Toby Walsh, editors, *Handbook of Constraint Programming*, chapter 6, pages 169–208. Elsevier, 2006.

56 Willem Jan van Hoeve, Gilles Pesant, and Louis-Martin Rousseau. On global warming: Flow-based soft global constraints. *Journal of Heuristics*, 12(4-5):347–373, 2006.