Constraints (2016) 21:115–144
DOI 10.1007/s10601-015-9198-6

SURVEY

Tractability in constraint satisfaction problems: a survey

Clément Carbonnel[1,2] · **Martin C. Cooper**[3]

Published online: 21 July 2015

Abstract Even though the Constraint Satisfaction Problem (CSP) is NP-complete, many tractable classes of CSP instances have been identified. After discussing different forms and uses of tractability, we describe some landmark tractable classes and survey recent theoretical results. Although we concentrate on the classical CSP, we also cover its important extensions to infinite domains and optimisation, as well as #CSP and QCSP.

Keywords Computational complexity · Polynomial-time · Dichotomy · Tractable language · Polymorphism · Microstructure · Forbidden pattern · Relaxation

1 What is tractability?

The idea that an algorithm is efficient if its time complexity is a polynomial function of the size of its input can be traced back to pioneering work of Cobham [45] and Edmonds [83], but the foundations of complexity theory are based on the seminal work of Cook [52] and Karp [118]. A computational decision problem (such as the constraint satisfaction problem or CSP) consists of a generic instance (in the case of the CSP, a set of variables, their

Supported by ANR Project ANR-10-BLAN-0210 and EPSRC grant EP/L021226/1.

✉ Martin C. Cooper
cooper@irit.fr

Clément Carbonnel
carbonnel@laas.fr

[1] CNRS, LAAS, 7 avenue du colonel Roche, 31400 Toulouse, France

[2] University of Toulouse, INP Toulouse, LAAS, 31400 Toulouse, France

[3] IRIT, University of Toulouse III, 31062 Toulouse, France

domains and a set of constraints) together with a yes-no question (Is there an assignment of values to the variables which simultaneously satisfies all the constraints?). A problem Q is NP-hard if all problems P in NP are polynomially reducible to Q (and NP-complete if we also have $Q \in$ NP). On the other hand, the class P consists of all decision problems that can be decided in polynomial time. Although P$\neq$NP is still an open question, it is generally assumed to be true, and hence proving that a problem is NP-hard is universally accepted as a proof of computational hardness. The CSP is NP-hard since it includes as a subproblem 3SAT which is a canonical example of an NP-complete problem [52].

The essential property of a tractable class $\mathscr{C}$ of CSP instances is that there is a polynomial-time algorithm to test the satisfiability of instances $I \in \mathscr{C}$. However, depending on the application in which the tractability of $\mathscr{C}$ is used, other properties may be desirable or even required. For example, one such property is that the recognition problem (Is $I \in \mathscr{C}$?) is solvable in polynomial time.

We can identify several distinct uses of tractable classes:

1. automatic recognition and resolution of easy instances within general-purpose solvers,
2. as a target problem after instantiation of certain variables during exhaustive search,
3. construction of a polynomial-time solvable relaxation to prune exhaustive search, where given an instance I, a *relaxation* (or *lower-bound instance*) I_{lb} is an instance such that I_{lb} is unsatisfiable implies that I is unsatisfiable,
4. proof (by a human being) that a subproblem of CSP (for example, encountered in a real application) can be solved in polynomial time.

To be useful, cases (1) and (2) require polynomial-time recognition, whereas (3) and (4) do not. We therefore use the term *tractable* in its largest interpretation: we only require that the class be solvable in polynomial time. It should be noted, however, that not requiring that a tractable class be recognisable in polynomial time can lead to pathological cases of no interest, such as the class of all satisfiable CSP instances [140].

An example of a particularly versatile tractable class is the class of instances whose constraint graph has treewidth bounded by a constant k. This class can be recognized in polynomial time (indeed in linear time [21]), can provide a target class after instantiation of some variables and lower-bound instances can easily be produced in this class [73]. However, in the important case $k = 1$, a lower-bound instance I_{lb} with a constraint graph in the form of a tree provides no more information than we would obtain by applying arc consistency to the original instance I.

Many tractable classes of CSP are automatically solved in polynomial time by any algorithm which maintains (generalised) arc consistency during search: we can notably cite the class of instances with max-closed constraints [112], the class of instances whose constraints are max-closed after independent permutations of each domain [94] and the class of binary instances satisfying the broken-triangle property [58]. Similarly, Valued CSPs with submodular constraints are automatically solved by establishing OSAC (Optimal Soft Arc Consistency) [57]. Present-day solvers do not explicitly look for tractable classes, but by analysis of the algorithms they use it is sometimes possible to show that they automatically solve certain tractable classes. For instance, translating CSP instances with max-closed constraints [112] or CSP instances with connected row-convex constraints [77] into SAT instances using the order encoding produces instances that fall into known tractable classes of SAT which are solved efficiently by modern clause-learning SAT-solvers [108, 143]. Tractable classes that are automatically solved by standard algorithms are nevertheless useful since proving that the solver will always execute in polynomial time in a given application provides a potentially important guarantee of efficiency.

The identification of certain interesting subproblems of CSP as tractable has led to both practical and theoretical applications. For example, some of the results on tractable constraint classes were used by British Telecom in the design of their Work Manager scheduling package [131] and by Rank Xerox in the analysis of schedulers for printing systems [145]. The tractability of the interpretation of line drawings of curved objects follows from results on tractable constraint languages [53] and a novel relaxation of planning problems follows from the tractability of simple temporal constraints [59]. Indeed, the tractability of classes of temporal constraints have found many applications, including the scheduling of agile satellites [144].

A global constraint is a constraint on a non-fixed number of variables. A global constraint often provides an expressive and concise way of modelling a condition which could otherwise only be expressed by a large number of simpler constraints. Arguably the most common and useful tractable classes of CSP instances consist of instances comprising a single global constraint together with arbitrary unary constraints. The fact that such instances are tractable for certain types of global constraints is the basis of domain filtering methods for global constraints used in solvers [153]. Such filtering can be viewed as the use of a polynomial-time solvable relaxation consisting of a single global constraint together with the current domains of each variable.

A *tractable class* of a computational problem P is a set of instances of P for which there exists a polynomial-time solution algorithm. Sets of CSP instances can be described in different ways, including:

1. By restricting the language of constraint relations. Such classes are called *language classes* and are discussed in Sections 2, 3 and 4.
2. By restricting the constraint (hyper)graph. Such classes are called *structural classes* and are discussed in Section 5
3. By placing restrictions which are not exclusively language-based nor exclusively structural. Such classes, sometimes called *hybrid classes*, are described in Sections 6 and 7.

The problem of identifying all tractable languages has been the inspiration for a rich literature at the boundary of theoretical computer science and algebra. Indeed, several surveys have already been published [39, 51, 65, 102, 113]. For the moment, the Feder and Vardi dichotomy conjecture that all finitelanguages of constraint relations either define a polynomial-time solvable class or an NP-complete class remains open. On the other hand, the problem of identifying all structural tractable classes has been solved in the case of bounded arity constraints, since it has been shown that bounded treewidth is the only reason for tractability [97]. In the unbounded arity case, a fixed-parameter tractability dichotomy has been given when the parameter is the number of variables [137].

An algorithm is polynomial-time if its worst-case time complexity is bounded by a polynomial function of the size of its input. This seemingly unambiguous definition can nevertheless lead to subtleties in the definition of a tractable class of CSP instances. For example, Chen and Grohe study tractability under assumptions of succinct representations of the constraint relations, in the form of DNF formulas or (non-deterministic) decision diagrams. With a more succinct representation it is more difficult to find polynomial-time algorithms, since complexity has to be a polynomial function of a smaller input size, but interesting structural and language-based tractable classes still exist [41].

In the rest of the paper we consider *CSP instances* $I = (X, D, C)$ consisting of a set X of n variables $X_1, \ldots, X_n$ with domains $\mathscr{D}(X_i) \subseteq D$ and a set C of constraints of the form $\langle \sigma, R \rangle$, where $\sigma \subseteq X$ is the *scope* of the constraint and R the *relation* containing all allowed assignments to the variables in σ. A *solution* is an assignment to the n variables such that, for each constraint $\langle \sigma, R \rangle$, the corresponding partial assignment to the variables σ belongs to R. The *arity* of a constraint $\langle \sigma, R \rangle$ is $|\sigma|$. A CSP instance is *binary* if all its constraints have arity at most two. We suppose that domains are *finite*. The case of infinite domains is covered in Section 9.1. Unless stated otherwise, we assume that the relations are given in the input as lists of tuples, every variable belongs to at least one scope, and every domain value belongs to at least one tuple. The *size* of a CSP instance $I = (X, D, C)$ is $|I| = \sum_{\langle \sigma, R \rangle \in C} |\sigma| \times |R|$, where $|R|$ is the number of tuples in R. We will use the following examples of tractable classes as illustrations throughout the paper. The reasons for tractability will be given when the examples are referred to in the following sections.

Example 1 Let $\mathscr{C}_{\text{mc}}$ be the class of binary CSP instances over the domain $\{1, \ldots, d\}$ in which unary constraints are arbitrary and all binary or ternary constraints are of one of the following forms

1. $a \leq X_i - X_j \leq b$,
2. $X_i \leq X_j + X_k + c$,
3. $X_k < \max(X_i, X_j)$

where a, b, c are arbitrary constants. The class $\mathscr{C}_{\text{mc}}$ contains some simple scheduling problems over a discrete time axis.

Example 2 Let $\mathscr{C}_{\text{aff2}}$ be the class of binary CSP instances over the domain $\{0, 1\}$ in which all constraints are linear equations modulo 2. A simple example of such a problem is the recovery of missing bits from a message using parity bits.

Example 3 Let $\mathscr{C}_{\text{tw1}}$ be the class of binary CSP instances over the domain $\{1, \ldots, d\}$ with arbitrary unary and binary constraints except for the fact that the constraint graph (i.e. the graph $\langle X, E \rangle$, where X is the set of variables and E the set of scopes of the binary constraints) is a forest.

Example 4 Let $\mathscr{C}_{\text{AllDiff}}$ be the class of CSP instances over the domain $\{1, \ldots, d\}$ consisting of arbitrary unary constraints a binary constraint of the form $X_i \neq X_j$ on each pair of distinct variables X_i, X_j. The class $\mathscr{C}_{\text{AllDiff}}$ contains some simple assignment problems, including the problem of finding a bijection between employees and tasks to be performed with the unary constraints coding the competences of each employee to perform each task.

Example 5 Let $\mathscr{C}_{\text{4Turan}}$ be the class of binary CSP instances over the 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 these variables.

The paper is structured as follows. Section 2 introduces the important theoretical notion of polymorphism and then discusses the two main algorithmic techniques for solving language-based tractable classes. We give language-based tractable classes which could

be potentially important in terms of practical applications as illustrations of these notions. Sections 3 and 4 cover the more theoretical problems of characterising all tractable languages and the complexity of recognizing tractable languages. Sections 5 and 6 cover tractable classes defined by placing restrictions, respectively, on the constraint graph or the microstructure of the instance. Other tractable classes, which are not defined by a restriction on the language, the constraint graph or the microstructure of the instance, are covered in Section 7. Fixed-parameter tractability results are grouped together in Section 8. Although this article is mainly concerned with the traditional finite-domain decision-problem version of the constraint satisfaction problem, we also briefly cover in Section 9 the case of infinite domains and other versions of the CSP, such as the Valued CSP, Quantified CSP, Uncertain CSP and #QCSP.

2 Tractable languages

The potential importance of tractable languages of constraint relations is illustrated by the importance of linear programming in Mathematical Programming or Horn clauses in SAT.

If Γ is a set of relations, then CSP(Γ) denotes the class of CSP instances all of whose constraint relations belong to the language Γ. If CSP(Γ')$\in$P, for all finite subsets of Γ' of Γ, then Γ is said to be a *tractable language*. An important consequence of this definition in terms of finite subsets of Γ is that we can assume that the arity of constraints is bounded by a constant, the maximum arity of relations in the finite subset Γ'. This constant bound on the arity of constraints ensures that tractability does not depend on the way in which constraint relations are encoded: although, in practice, constraint relations are often stored implicitly, in this section, we can make the simplifying assumption that all constraint relations are stored as an explicit list of tuples.

2.1 Polymorphisms

Initial research in the identification of language-based tractable classes uncovered interesting but disparate examples of tractable languages. The algebraic approach provided a unifying framework in which to study tractability of languages [107]. It also allowed bridges to be built between the complexity of constraint languages and the mathematics of clone theory and universal algebra, leading to important cross-fertilisation. Indeed, this approach has helped to produce results which strongly support the longstanding dichotomy conjecture of Feder and Vardi [86] that all such classes are either solvable in polynomial time or else NP-hard.

A foundational theoretical result is that a necessary condition for tractability of Γ is the existence of a (non-trivial) componentwise closure operation, known as a polymorphism [107, 110]. For simplicity, it is usually assumed that there is a unique domain D such that all variable-domains $\mathscr{D}(X_i)$ are subsets of D. A function $f : D^k \rightarrow D$ can be extended to a function from D^{rk} to D^r by applying it componentwise: given k tuples $\mathbf{x_1}, \ldots, \mathbf{x_k} \in D^r$,

$$f(\mathbf{x_1}, \ldots, \mathbf{x_k}) = \langle f(\mathbf{x_1}[1], \ldots, \mathbf{x_k}[1]), \ldots, f(\mathbf{x_1}[r], \ldots, \mathbf{x_k}[r])\rangle,$$

where $\mathbf{x_i}[j]$ is the jth component of the tuple $\mathbf{x_i}$. Then f is a *polymorphism* of a language Γ (and Γ is *preserved* by the operation f) if for all relations $R \in \Gamma$, $\forall \mathbf{x_1}, \ldots, \mathbf{x_k} \in R$, $f(\mathbf{x_1}, \ldots, \mathbf{x_k}) \in R$.

Thus all tractable languages have a nontrivial polymorphism f, where nontrivial means that f is not a projection (i.e. f is not a function of the form $f(x_1, \ldots, x_t) = x_i$ for some i). For example, any Boolean relation which is equivalent to a conjunction of Horn clauses has the polymorphism min, as does any arithmetical constraint of the form $a_1x_1+a_2x_2+\ldots+a_{r-1}x_{r-1} \leq a_rx_r$ for any positive constants $a_1, \ldots, a_r$ or any conjunction of disjunctions of the form $(x_1 < b_1) \vee \ldots \vee (x_r < b_r) \vee (x_i > c)$ where $b_1, \ldots, b_r, c$ are constants [112]. As another example, the language of zero-one-all relations [55] (a generalization of 2SAT clauses to multi-valued logics) has the *dual discriminator* polymorphism given by $f(x, y, y) = y$, $f(x, y, z) = x$ if $y \neq z$.

2.2 Language classes solved by local consistency

Local consistency methods are the most common and well-studied polynomial-time algorithmic techniques for CSP. In general they are incomplete and thus only used to prune inconsistent values, but in presence of some particular polymorphisms they may become decision procedures. A CSP instance is *k-consistent* if every consistent assignment to $k-1$ variables can be extended to a consistent assignment of any kth variable, and strongly k-consistent if it is j-consistent for all $j \leq k$. (Strong) k-consistency can be easily enforced in polynomial time if k is fixed. If enforcing strong k-consistency empties the domain of at least one variable for every unsatisfiable instance of CSP(Γ), we say that Γ has *strict width* k.

Some results are stated using a different form of local consistency, called *relational consistency*. A CSP instance is (k, l)-minimal if for every subset L of l variables there is a constraint with scope $\sigma \supseteq L$ and for every set K of at most k variables, the projections of any two constraints whose scope contain K are identical. For instance, generalised arc-consistency (GAC) is equivalent to (1, 1)-minimality. For fixed (k, l), enforcing (k, l)-minimality is polynomial time. We say that a language Γ has *(relational) width* (k, l) if the (k, l)-minimality algorithm acts as a decision procedure for CSP(Γ), width k if it has width (k, k), and bounded width if it has width (k, l) for some fixed (k, l).

Tractable classes, such as the min-closed class, can be generalised by determining which properties of the polymorphism are required by the solution algorithm. A *semilattice operation* f is a binary operation that is associative, commutative, and idempotent, i.e. $\forall x, y, z$, $f(x, f(y, z)) = f(f(x, y), z)$, $f(x, y) = f(y, x)$ and $f(x, x) = x$. If Γ has a semilattice polymorphism, then CSP(Γ) has width 1 [107]. The unary, binary and ternary constraint relations given in Example 1 all have the polymorphism max which is a semilattice operation. It follows that all instances in the corresponding class $\mathscr{C}_{\text{mc}}$ can be solved by GAC.

It was later shown that CSP(Γ) has width 1 if and only if Γ has *totally symmetric* (TS) polymorphisms of all arities, that is, polymorphisms f_i such that $f_i(x_1, \ldots, x_r) = f_i(y_1, \ldots, y_r)$ whenever $\{x_1, \ldots, x_r\} = \{y_1, \ldots, y_r\}$ [68].

Considering more powerful solution algorithms is another way to generalise tractable classes. A *near-unaminity* (NU) polymorphism is a k-ary operation f, where $k \geq 3$, satisfying $\forall x, y$, $f(y, x, \ldots, x) = f(x, y, x, \ldots, x) = \ldots = f(x, \ldots, x, y) = x$. It is known that a language Γ has strict width k if and only if Γ has a k-ary NU polymorphism and for any such language, strong k-consistency implies global consistency [111]. A *majority* operation is a ternary near-unanimity operation. The dual discriminator operation, defined

above, is an example of a majority operation. Connected row-convex constraints [77] are closed under the ternary median operation which is another example of a majority operation [111]. It has been recently shown that singleton arc-consistency solves CSP(Γ) when Γ admits a majority polymorphism [40].

A binary operation f is a 2-Semilattice if it is idempotent, commutative and satisfies the identity $\forall x, y, f(x, f(x, y)) = f(x, y)$. A language Γ that has a 2-Semilattice polymorphism is tractable since CSP(Γ) has bounded width [22]. A k-ary ($k \geq 2$) operation $f : D^k \rightarrow D$ is a *weak near-unanimity operation* (WNU) if, $\forall x, y \in D$, $f(y, x, x, \ldots, x) = f(x, y, x, \ldots, x) = f(x, x, \ldots, x, y)$. Any language Γ preserved by WNU polymorphisms of all arities greater than or equal to 3 has bounded width [9]; this class is extremely large and encompasses all tractable classes discussed above. Finally, it has been shown that every language with bounded width has width (2, 3) [8, 24], which allows for fairly efficient solving.

2.3 Polynomial-sized representation of all solutions

Another standard algorithmic technique for solving tractable languages, apart from local consistency operations, is based on the property of having a polynomial-sized representation for the solution set of any instance. A *Mal'tsev operation* is a ternary operation which satisfies $\forall x, y, f(y, y, x) = f(x, y, y) = x$. If Γ is preserved by a Mal'tsev operation, then this property holds and CSP(Γ) can be solved by an algorithm based on a generalization of Gaussian elimination [28]. More precisely, if a relation R admits a Mal'tsev polymorphism f then there exists a relation R_c of arity r such that R_c has $O(rd)$ tuples and R is the closure of R_c under f [81]. R_c is called a *frame* for R. If $I = (X, D, C)$ is a CSP instance with $C = (C_i, \ldots, C_q)$, the algorithm for Mal'tsev languages iteratively computes a frame F for the solution set of $(X, D, C_1, \ldots, C_{i+1})$ from a frame for $(X, D, C_1, \ldots, C_i)$. Once $i = q - 1$ is reached, F is a frame for the set of solutions of I, and F is empty if and only if the instance has no solution.

An *affine operation* is an example of a Mal'tsev operation and is of the form $f(x, y, z) = x - y + z$ where $(D, +, -)$ is an Abelian group [107]. Any constraint satisfaction problem over a prime domain of size d, with constraint relations that are closed under an affine operation, corresponds to a set of simultaneous linear equations over the integers modulo d [106]. In the case $d = 2$, as illustrated by the class $\mathscr{C}_{\text{aff2}}$ given in Example 2, the affine operation is equivalent to $f(x, y, z) = x + y + z$ and is known as the minority operation, since $\forall x, y \in \{0, 1\}$ $f(x, x, y) = f(x, y, x) = f(y, x, x) = y$.

A *generalized majority-minority* operation is such that for all pairs of domain elements a,b either $f(x, y, \ldots, y) = f(y, x, y, \ldots, y) = \ldots = f(y, \ldots, y, x) = y$ for all $x, y \in \{a, b\}$ or $f(x, y, \ldots, y) = f(y, \ldots, y, x) = x$ for all $x, y \in \{a, b\}$. Generalized majority-minority (GMM) operations are a tractable generalization of both near-unanimity operations and Mal'tsev operations [67]. Finally, it was shown by Idziak et al. [104] that a necessary and sufficient condition for Gaussian-like algorithms to solve CSP(Γ) is that Γ has an *edge* polymorphism, which is an operation f such that $\forall x, y \in D$, $f(y, y, x, x, \ldots, x) = f(y, x, y, x, x, \ldots, x) = x$ and $f(x, x, x, y, x, \ldots, x) = f(x, x, x, x, y, x, \ldots, x) = f(x, \ldots, x, y) = x$. The solution algorithm is usually referred to as the *few subpowers algorithm*. Note that a k-edge operation has arity $k + 1$, so Mal'tsev operations correspond to 2-edge operations.

3 On characterizing tractable languages

This section covers the theoretical problem of characterizing those tractable classes defined by a language of relations. We begin by surveying the progress that has been made towards a proof of Feder and Vardi's dichotomy conjecture [86] which can be stated in the following form:

Conjecture 1 (Feder and Vardi) For every finite language Γ, either Γ is tractable or CSP(Γ) is NP-complete.

A landmark result is Schaefer's characterization of tractable languages for the special case of CSP instances over the Boolean domain [149]: a language Γ is tractable if Γ is 0-valid, 1-valid, Horn, dual Horn, affine or bijunctive; otherwise CSP(Γ) is NP-complete. A language is *c-valid* if it has the constant unary polymorphism $f(x) = c$. This is a trivial case, since assigning the value c to all variables necessarily satisfies all constraints. A Boolean language Γ is *Horn* (*dual Horn*) if all relations in Γ are logically equivalent to a conjunction of (dual) Horn clauses, which is equivalent to Γ having the polymorphism min (max) [112]. A Boolean language Γ is *affine* if all relations in Γ are equivalent to a system of linear equations (modulo 2), which is equivalent to Γ having the ternary affine polymorphism $f(x, y, z) = x + y - z$. Recall that over the Boolean domain, the affine polymorphism is also known as the *minority polymorphism* since $\forall x, y \in \{0, 1\}$, $f(x, x, y) = f(x, y, x) = f(y, x, x) = y$ and $f(x, x, x) = x$. A Boolean language Γ is *bijunctive* if all relations in Γ are equivalent to a set of 2SAT clauses, which is equivalent to Γ having the unique Boolean majority polymorphism given by $\forall x, y \in \{0, 1\}$, $f(x, x, y) = f(x, y, x) = f(y, x, x) = x$. If the solution to the CSP instance must be surjective, in the sense that each domain value appears at least once, then the tractable languages are Horn, dual Horn, affine or bijunctive [64].

In Section 2 we saw that the existence of a non-trivial polymorphism is a necessary condition for a language Γ to be tractable. This can be refined if we restrict attention to reduced languages, known as cores, which we now define. A *squashing function* is a unary polymorphism $f : D \to D' \subset D$. If Γ has a squashing function then if an instance $I \in$ CSP(Γ) has a solution over D, then it has a solution over D' (obtained by applying f component-wise to the solution). A language Γ over a domain D is a *core* if it has no squashing function. A necessary (but not sufficient) condition for a reduced language (a core) to be tractable is that it has a polymorphism which is a constant function, a majority function, an idempotent binary function (which is not a projection), a Malt'sev function, or a semi-projection [51]. A language Γ is a *rigid core* if its only unary polymorphism is identity. It is known that the search for tractable languages can be restricted to languages that are rigid cores [32].

3.1 Dichotomies

Although the Feder and Vardi conjecture is still open, several dichotomies have been found under special conditions:

- languages containing a single binary symmetric relation [101],
- languages closed under all domain permutations [55, 66],
- languages closed under disjunctions [103],

- maximal languages (where a language Γ is *maximal* if there is a relation $R \notin \langle\Gamma\rangle$ and each proper extension of $\langle\Gamma\rangle$ contains all relations on D) [33],
- languages over a size-3 domain [23],
- conservative languages (i.e. languages containing all unary relations) [6, 25, 27],

A binary relation over a domain D can be viewed as a graph $\langle D, R\rangle$. In the case of languages Γ containing a single binary symmetric relation R, the Hell-Nešetřil theorem shows that Γ is tractable if R viewed as a graph is bipartite or contains a loop, and is NP-complete otherwise [101].

The tractable class of languages preserved by WNU operations of all arities greater than or equal to 3 characterizes precisely the languages that can be solved by local consistency [9]. Recall that the existence of an edge polymorphism is a necessary and sufficient condition for CSP(Γ) to be solved by the few subpowers algorithm [104]. Thus, the languages Γ such that CSP(Γ) can be solved by either of the two most important algorithmic techniques (namely, consistency methods and Gaussian-like algorithms) have been completely characterised by the polymorphisms of Γ.

3.2 More specific conjectures

Feder and Vardi showed that the CSP dichotomy conjecture is equivalent to the CSP dichotomy conjecture restricted to digraphs (i.e languages consisting of a single binary relation) [86]. Therefore constraint satisfaction problems can be reduced to (di)graph homomorphism problems studied in graph theory for over thirty years. It has been shown that the CSP dichotomy holds for digraphs with no sources and no sinks [10]. It has also been shown that the truth of the following conjecture implies the truth of that of Feder and Vardi: If Γ has a Siggers operation as a polymorphism, then CSP(Γ) is tractable, where a *Siggers operation* f is an arity-4 operation satisfying the following identities: $\forall x, y, z,\ f(x, x, x, x) = x$ and $f(y, x, y, z) = f(x, y, z, x)$ [36]. The hierarchy of the main tractable classes discussed in Section 2 (plus the class of languages with a Siggers polymorphism which is conjectured to be tractable) is summarized in Figure 1 for the special case of rigid cores, whose polymorphisms are all idempotent. This hierarchy is no longer correct if we consider arbitrary languages instead.

Given a language Γ, $\langle\Gamma\rangle$ is the language of relations that can be generated from Γ by combinations of equality, join and project operations (and is known as the *relational clone* generated by Γ). Since for any finite set $\Gamma' \subseteq \langle\Gamma\rangle$ there is a polynomial reduction from CSP(Γ') to CSP(Γ), Γ is tractable if and only if $\langle\Gamma\rangle$ is tractable [107]. A set of operations containing all projections and closed under composition is known as a *clone*. Since Γ and $\langle\Gamma\rangle$ have the same clone of polymorphisms, it is possible to establish a Galois connection between relational clones (of relations) and clones (of polymorphisms) [110]. Indeed, by going beyond clones to finite algebras and varieties of algebras it has been shown that the complexity of a language is determined by the identities satisfied by its polymorphisms [32].

Bulatov et al. [32] have given a more specific version of the Feder and Vardi conjecture from the perspective of universal algebra (the field of mathematics that studies algebras themselves). An algebra is a set of operations on some fixed set. For each algebra $\mathscr{A}$, there is a corresponding language consisting of the set of relations closed under the operations of $\mathscr{A}$. An algebra is defined as tractable if its corresponding language is tractable. The *algebraic dichotomy conjecture* is that a finite idempotent algebra $\mathscr{A}$ is NP-complete if it has a nontrivial factor $\mathscr{B}$ (a homomorphic image of a sub-algebra of $\mathscr{A}$) all of whose operations

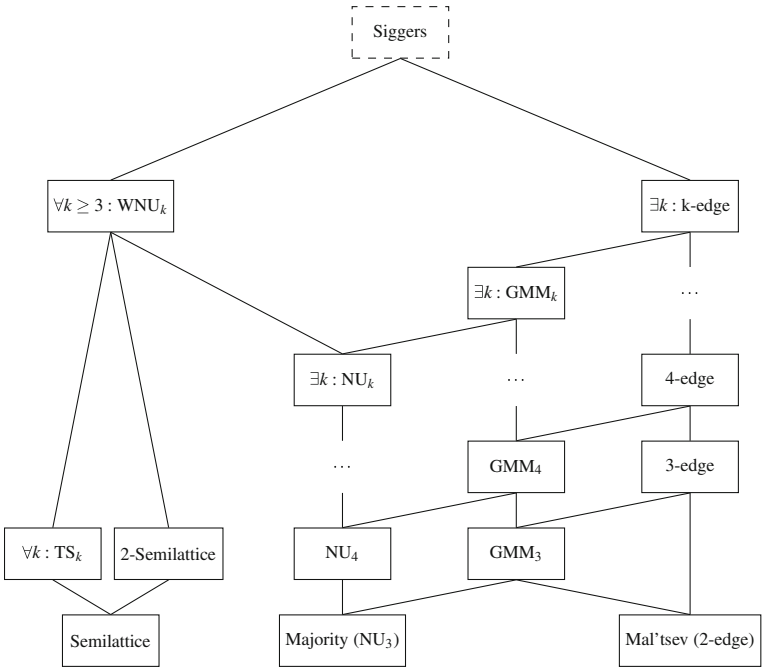

Fig. 1 Hasse diagram of tractable classes for rigid cores, ordered by inclusion. The subscript after the name of a polymorphism denotes its arity

are projections, otherwise it is tractable. This conjecture, if true, would completely solve the question of the complexity of any constraint language. Kun and Szegedy [127] have linked the dichotomy conjecture to the theory of Probabilistically Checkable Proofs.

3.3 Related problems

In a probabilistic analysis of Feder and Vardi's dichotomy conjecture, Luczak and Nešetřil showed that for a randomly chosen language Γ the probability that CSP(Γ) is tractable tends to zero as either the domain size or the maximum relation arity tends to infinity (under the assumption that all relations R are loop-free, i.e. for all domain values a, $(a, \ldots, a) \notin R$) [133].

Larose and Zadori studied the tractability of determining whether a system of polynomial equations over a finite algebra admits a solution [130]. They characterized, within various families of algebras, which of them give rise to an NP-complete problem and which yield a problem solvable in polynomial time. In particular, they prove a dichotomy result which encompasses the cases of lattices, rings, modules and quasigroups.

Feder and Hell studied the tractability of languages on W-full structures CSP$_W(\Gamma)$ (where W is any set of positive integers), representing the set of CSP instances in CSP(Γ) in which $\forall w \in W$, any w variables are involved in a w-ary constraint. For example, if $2 \in W$, then there is a binary constraint between each pair of variables. For conservative languages Γ (and, indeed, for the more general case of languages containing all unary relations of size 3), the W-full problem CSP$_W(\Gamma)$ is NP-complete or solvable in quasi-polynomial time $n^{O(\log n)}$ (where n is the number of variables) [84].

4 The recognition problem for tractable languages

In this section we detail the complexity of the recognition problem for the most important tractable languages, assuming we are given a finite language Γ as input. For any particular operation $f : D^k \to D$, such as min (for a given domain ordering) or the dual discriminator operation, we can exhaustively test in polynomial-time that f preserves every relation in Γ. We therefore study the recognition of classes of tractable languages defined by the identities satisfied by their polymorphisms, for example, the class of languages with a semilattice polymorphism or the class of languages with a majority polymorphism.

4.1 Recognition of different families of polymorphisms

In the literature on tractable languages, the recognition problem is often considered under the assumption that the domain size, the maximum number of tuples or the maximum arity is bounded [24, 25]. Here we only detail the complexity of the basic recognition problem which has no restriction on the input language.

It was observed in [86] that fixed-arity near-unanimity polymorphisms can be detected in polynomial time. When the arity is not fixed, the problem is still decidable but the exact complexity is unknown [7]. Interestingly, deciding whether a language admits the median polymorphism (which is a particular case of majority polymorphism) with respect to an unknown ordering of the domain is NP-hard [94]. This highlights the fact that more general polymorphisms are not necessarily harder to detect.

In the case of rigid cores, the class of languages with WNU polymorphisms of all arities greater than or equal to 3 can be recognized in polynomial time [8] by combining an alternative characterization involving only two WNU polymorphisms of fixed arity [124] and a variant of the recognition algorithm used for near-unanimity polymorphisms. However, deciding whether an arbitrary language has bounded width is NP-hard [34]. Deciding whether a language has width 1 is also NP-hard in the general case [129], and membership in NP is unknown. A result of [94] implies that semilattices are NP-hard to detect even in the special case of conservative languages.

The complexity of detecting edge and Mal'tsev polymorphisms is an interesting open problem, although Mal'tsev polymorphisms can be detected in polynomial time in digraphs (languages consisting of a single binary relation) [38] and conservative binary languages [14]. Finally, it was shown in [119] that all tractable binary conservative languages have bounded width, thus in this case the dichotomy is polynomially decidable; however, if the algebraic dichotomy conjecture holds, deciding if an arbitrary language is tractable is NP-hard [31].

4.2 Expressibility

A related computational problem is the expressibility problem: given a finite language Γ over a finite domain and a relation R, is $R \in \langle\Gamma\rangle$? This problem is decidable using an indicator problem which either provides a gadget for expressing the relation R using relations from Γ or a polymorphism of Γ that does not preserve R (which is a proof of inexpressibility) [109]. Unfortunately, Willard proved the co-NEXPTIME-hardness of deciding expressibility. Indeed, not only gadgets demonstrating expressibility but also polymorphisms demonstrating inexpressibility may be required to be of exponential size [156]. Kozik proved the EXPTIME-completeness of the expressibility problem for operations: is the operation f in the clone generated by operations $f_1, \ldots, f_k$? [123].

5 Structural tractable classes

Recall that a constraint is a pair $\langle \sigma, R \rangle$, where σ is a set of variables (the scope of the constraint) and R is the relation containing all allowed assignments to the variables in the scope. The *constraint hypergraph* of a CSP instance I is $H_I = \langle X, S \rangle$, where X is the set of variables of I and S is the set of constraint scopes of I. The *constraint graph* of I is the primal graph of H_I (i.e. the graph with vertices X and an edge joining a pair of variables if the two variables occur together in the scope of a constraint). A graph $G_1 = \langle V_1, E_1 \rangle$ is a *subgraph* of graph $G_2 = \langle V_2, E_2 \rangle$ if $V_1 \subseteq V_2$ and $E_1 \subseteq E_2$. The graph G_1 is an *induced subgraph* of G_2 if $V_1 \subseteq V_2$ and E_1 is the set of edges of E_2 that have both endpoints in V_1. Given a total order $<$ on the vertices of a graph $G = \langle V, E \rangle$, we use Parents(v) to denote the set of vertices $u < v$ such that $\{u, v\} \in E$. A *k-tree* is a graph G such that for some order on the vertices (without loss of generality $v_1 < \ldots < v_n$): for each $i = k + 1, \ldots, n$, the induced subgraph of G on Parents(v_i) $\cup \{v_i\}$ is the complete graph K_{k+1}. Trees are 1-trees.

It is well known that the class of instances whose constraint graph has bounded treewidth is tractable [87]. Graphs with *treewidth bounded by k* are also known as partial k-trees, because they are exactly the subgraphs of k-trees [4]. In fact, this general phenomenon of structural tractability is also true for a number of problems in PSPACE, including all properties expressible in Monadic Second-Order logic [63]. Constraint-based problems that can be solved in polynomial time when the constraint graph has bounded treewidth include the Valued CSP [13], #CSP (the problem of counting the number of solutions), calculating partition functions and the Polynomial CSP (in which, by replacing complex numbers in the partition function by generalised polynomials over a small number of formal variables, it is possible to express problems such as finding a maximum balanced cut of a graph) [150].

In the bounded-arity case, bounded treewidth is the only possible reason for structural tractability of CSPs, under the very reasonable assumptions that FPT≠W[1] and that the class of structures is recursively enumerable [96, 97]. Determining the treewidth and a corresponding tree decomposition of an arbitrary graph is NP-hard [5]. Furthermore, in random problems treewidth is unlikely to be small [89]. Nevertheless, finding a suboptimal, but still useful order can be practical in real applications [70, 75].

In the unbounded-arity case, new tractable classes appear compared to the bounded-arity case. As a trivial example, any CSP instance which contains at least one constraint with scope of size $\Omega(n)$, given in extension, together with other arbitrary constraints, can be solved in time which is a polynomial function of its input (which may already be exponential in the number of variables). A *fractional edge cover* of a hypergraph is a weight assignment to its hyperedges such that every vertex is covered by hyperedges of total weight at least 1. The *fractional edge cover number* is the smallest total weight of a fractional edge cover. Using Shearer's lemma, Grohe and Marx [99] showed that if the hypergraph H_I of a CSP instance I has fractional edge cover number w, then there are at most $|I|^w$ satisfying assignments. Thus a simple backtracking algorithm applied to an instance with bounded fractional edge cover number will find all solutions in polynomial time.

Several notions of bounded hypertree width have been shown to define equivalent tractable classes [2]. These tractable classes have been strictly generalised by Grohe and Marx [99] who combined the notion of fractional edge cover with hypertree decomposition [93] to define a fractional hypertree decomposition of weight w. This is identical to a hypertree decomposition (bags of vertices arranged in a tree structure such that (1) if two vertices are connected by a hyperedge then there is a bag containing both of them and (2) for every vertex v, the bags containing v form a connected subtree) except that instead of there being, for each bag, w hyperedges covering that bag, there is now a fractional edge

cover of weight w for each bag. The *fractional hypertree width* of H_I is defined to be the width w of its best decomposition. Marx [135] has shown that for any constant w, if H_I has fractional hypertree width w then it is possible to find in polynomial time a fractional hypertree decomposition of width $O(w^3)$.

Marx [136] introduced the notion of adaptive width (a strict generalisation of the notion of fractional hypertree width [99]) and showed that it is the only reason for structural tractability if the constraints are stored as truth tables, unless CSP instances with bounded arity and treewidth bounded by t can be solved in time which is subexponential in t. Note that the size of a truth table constraint of arity r over a domain of size d is always $O(d^r)$ regardless of its number of tuples.

6 Microstructure-based tractable classes

A binary CSP instance on variables $X_1, \ldots, X_n$ can be represented by the domain $\mathscr{D}(X_i)$ 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 $A = \{(X_i, a) \mid a \in \mathscr{D}(X_i)\}$ is the set of possible variable-value assignments and $E = \{\{(X_i, a), (X_j, b)\} \mid (a, b) \in R_{ij}\}$ [114]. The microstructure relies on both the structure and the relations of the instance I and so is a natural place to look for tractable classes which are neither purely structural nor purely language-based. The *complement* of a graph $G = \langle V, E\rangle$ is the graph with vertices V and whose edges arethe 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 [100], the class of binary CSP instances with a perfect microstructure is tractable [147]. Perfect graphs can also be recognized in polynomial time [43].

For a class of graphs $\mathscr{C}$, a graph G is $\mathscr{C}$-free if no induced subgraph of G is isomorphic to any graph in $\mathscr{C}$. 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 [44]. Interesting examples of binary CSP instances whose microstructure is perfect are

- instances with arbitrary domains and a constraint $X_i \neq X_j$ between each pair of variables (or, alternatively, unary constraints together with a global ALL-DIFFERENT$(X_1, \ldots, X_n)$ constraint) [147],
- instances which are arc consistent and max-closed after independent (and possibly unknown) permutations of each domain [94].

In the *coloured microstructure*, the vertices of the microstructure representing an assignment to variable X_i are labelled by a colour representing variable X_i, thus maintaining the distinction between assignments to different variables. Indeed, since it retains all the information contained in the original instance, we can identify a binary CSP instance with its

coloured microstructure. A *pattern* is a CSP instance except that it has three types of tuple in its constraint relations, tuples which are explicitly allowed/disallowed and tuples which are labelled as *unknown*. It can be identified with its coloured microstructure which can be viewed as a graph over a three-valued logic in which each pair of vertices is an edge, a non-edge or unknown. A pattern P *occurs* in a binary instance I if there is a homomorphism from P to an induced subgraph of the coloured microstructure which preserves edges and non-edges [47]. A class of binary CSP instance can be defined by *forbidding* a pattern P: CSP($\overline{P}$) is the set of instances in which P does not occur. Any class of instances defined by a forbidden pattern is necessarily recognisable in polynomial time by a simple exhaustive search for the pattern.

One simple example of a forbidden pattern P which defines a tractable classes of binary CSP instances is shown in Fig. 2a: it is based on the transitivity of non-edges [62]. The class CSP($\overline{P}$) consists of all binary CSP instances in which for all triples of assignments $a_1 = \langle X_i, a\rangle$, $a_2 = \langle X_j, b\rangle$, $a_3 = \langle X_k, c\rangle$ to three distinct variables, whenever the pairs (a_1, a_2), (a_2, a_3) are both disallowed ($\langle a, b\rangle \notin R_{ij}$ and $\langle b, c\rangle \notin R_{jk}$), the third pair (a_1, a_3) is also disallowed ($\langle a, c\rangle \notin R_{ik}$). This property is satisfied, for example, by instances consisting of unary constraints and AllDifferent constraints on non-overlapping subsets of the variables, and in particular by the class $\mathscr{C}_{\text{AllDiff}}$ given in Example 4. 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 [61, 62]. Figure 2b shows another example of a pattern Q defining a tractable class. The class CSP($\overline{Q}$) includes as a proper subset all instances with zero-one-all constraints [55].

Dichotomies for tractable classes of binary CSP instances defined by forbidding a pattern P have been discovered in the following special cases:

- negative patterns P (i.e. patterns in which all tuples are either disallowed or unknown) [47],
- 2-constraint patterns [56].

Recently, the notion of forbidding patterns has been extended to rules based on applying a sequence of quantifiers to the variables and values in a pattern. As an example, consider the pattern BTP shown in Fig. 2c, known as a broken-triangle. It is known that forbidding this pattern for all triples of variables $X_i, X_j < X_k$ (according to some variable ordering $<$) defines a tractable class which includes all binary CSP instances whose constraint graph is a forest (the class $\mathscr{C}_{\text{tw1}}$ given in Example 3) [58]. If such a variable ordering exists,

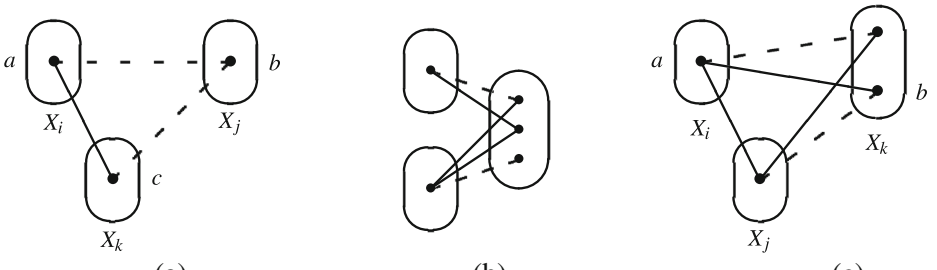

Fig. 2 **a** A binary CSP instance I satisfies the negative transitive property if this pattern P does not occur in I. **b** A pattern Q. **c** The broken triangle pattern BTP

then it can be found in polynomial time: it suffices to establish arc consistency and then successively eliminate variables X_k which are not the right-hand variable of a broken triangle. This variable-elimination rule can be strictly generalised to the following rule: X_k can be eliminated without changing the satisfiability of the instance if $\forall i \neq k$, $\forall a \in \mathscr{D}(X_i)$, $\exists b \in \mathscr{D}(X_k)$ with $(a, b) \in R_{ik}$ such that no broken triangle exists including the edge $\{\langle X_i, a\rangle, \langle X_k, b\rangle\}$ [54]. The class of binary CSP instances which are such that all variables can be eliminated according to this rule strictly generalises the tractable class CSP($\overline{\text{BTP}}$).

Most work on microstructure-based tractability has been restricted to binary CSPs, for the simple reason that the traditional definition of microstructure assumes binary constraints. It is well known that any general-arity CSP instance can be coded (indeed in several different ways) as a binary instance, for example via the dual encoding. It is then possible to define tractable classes of general-arity CSPs as those instances whose equivalent binary instance have a microstructure satisfying certain conditions guaranteeing tractability [138]. A direct generalisation of the broken-triangle class CSP($\overline{\text{BTP}}$) to general-arity CSPs has also recently been given [60], but the definition of tractable classes by forbidden patterns in general-arity CSPs is an area which remains largely unexplored.

7 Other tractable classes

Some 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 constraint relations of a binary CSP instance are row-convex after establishing strong 3-consistency [11]. The class of binary CSP instances which are min-of-max extendable after establishing arc consistency is another example (which actually includes all strong 3-consistent row-convex instances) [58]. Another 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 [72]. 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 [12].

Naanaa [139] 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 $J \subset I$,

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

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 $\mathscr{D}(x)$ 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 $\mathscr{D}(x)$ otherwise. A family of extentions 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 which has directional rank no greater than κ and is directional strong $(\kappa(r-1)+1)$-consistent then I is globally consistent [139].

Two other ways to guarantee tractability are to have so few disallowed tuples that the instance is necessarily satisfiable, or, on the contrary, to have so many disallowed tuples that any subproblem necessarily has at most a polynomial number of solutions. 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 [142].

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. In the case of binary CSP instances, a minimum root set can be found in polynomial time [69]. Unfortunately, determining the size of a minimum root set is NP-hard for ternary CSP instances [48].

Another condition that guarantees a backtracking search tree of polynomial size is the k-Turan property [48]. Indeed, this property is very strong since it guarantees a polynomial-size search tree for all variable orderings. 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, in the class $\mathscr{C}_{4\text{Turan}}$ given in Example 5, every 4-element subset of the set of n variables X is represented of 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 $\mathscr{C}_{4\text{Turan}}$, all constraints satisfy this condition and hence $\mathscr{C}_{4\text{Turan}}$ is tractable since all instances in this class satisfy the 4-Turan property.

8 Parameterized and subexponential complexity

8.1 Parameterized complexity of CSP

The framework of parameterized complexity was introduced by Downey and Fellows [79] to allow a more fine-grained analysis of computational hardness than classical complexity theory [90]. This area of research has been very successful in the last two decades and several attempts have been made to study the complexity of CSP from this angle.

A problem is parameterized if each instance is paired with a nonnegative parameter k. A parameterized problem is fixed-parameter tractable (FPT) if it can be solved in time

$O(f(k)|x|^{O(1)})$, where $|x|$ is the size of the instance x and f is any computable function of the parameter. For instance, k-Vertex Cover (the problem of deciding whether a graph with n vertices contains a set S of at most k vertices such that each edge is incident to at least one vertex in S) parameterized by k is FPT as it can be solved in time $O(1.2738^k + kn)$ [42]. FPT is a strict subset of XP, which contains all parameterized problems solvable in time $O(|x|^{g(k)})$. Note that for a fixed k any problem in XP (or FPT) is polynomial-time solvable; however the asymptotic complexity of FPT problems is independent of the value to which k is fixed, while this is not the case in general for XP. Between FPT and XP, Downey and Fellows [78] proposed a hierarchy FPT = W[0] $\subseteq$ W[1] $\subseteq$ W[2] $\subseteq \ldots \subseteq$ XP. These classes are closed under fpt-reductions, which map each instance (x, k) of a parameterized problem P to an instance $\phi(x, k) = (x', k')$ of a parameterized problem Q such that: (i) x' is a yes-instance if and only if x is, (ii) there exists a computable function g such that $k' \leq g(k)$ and (iii) x' can be computed from x in time $O(f(k)x^{O(1)})$ for some computable function f. For every t, W[$t+1$] is believed to be strictly larger than W[t]; in particular, no W[1]-hard problem can be FPT unless the Exponential Time Hypothesis (ETH) fails [1].

A large number of parameters have been considered for the CSP. When the parameter is the number of variables n, the CSP is in general W[1]-complete [141]. However, Marx has identified the structural restrictions which ensure FPT tractability when the parameter is n: these are exactly the classes of hypergraphs with bounded *submodular width*, under the assumptions that the ETH holds and the class is recursively enumerable [137]. Bounded submodular width is equivalent to bounded adaptive width, discussed in Section 5. A similar result holds when the parameter is the length l of the instance (that is, the sum of the sizes of the constraint scopes): the problem is W[1]-complete [141], but if $\mathscr{G}$ is a recursively enumerable set of graphs and CSP($\mathscr{G}$,_) denotes the set of all instances whose primal graph is in $\mathscr{G}$, CSP($\mathscr{G}$,_) is FPT for the parameter l (and in fact in P) if and only if $\mathscr{G}$ has bounded treewidth, unless FPT = W[1] [97].

If we consider the case when the primal graph of the language of relations is restricted instead of the constraint graph, some FPT results about CSP can be inferred from the recent progress in the parameterized complexity of first-order logic. Given a language Γ of relations, the first-order formulas over Γ are built from the atoms $R(x_1, \ldots, x_r)$ for every $R \in \Gamma$ and $x = y$ by using the logical connectives $\neg$, $\wedge$, $\vee$ and the quantifiers $\forall$ and $\exists$. It follows that a CSP instance of length l over a language Γ is described by a first-order formula ϕ over Γ, with $|\phi| = O(l)$. It has been recently shown that if a class of graphs $\mathscr{G}$ is *effectively nowhere dense*, then the evaluation of first-order formulas on languages whose primal is in $\mathscr{G}$ is FPT when the parameter is the size of the formula [98]. Nowhere dense classes of graphs include many sparse graph classes; in particular, all minor-closed classes of graphs and graphs of bounded expansion are nowhere dense. As a direct consequence, CSP(_,$\mathscr{G}$) (the restriction of CSP to instances such that the primal of the language of relations is in $\mathscr{G}$) is FPT when the parameter is l if $\mathscr{G}$ is nowhere dense. Another well-known result about first-order logic [95] implies that CSP(_,$\mathscr{G}$) is FPT when the parameter is l if there is a computable function $f : \mathbb{N} \to \mathbb{N}$ such that $f(x) = o(x^\epsilon)$ for all fixed $\epsilon > 0$ and every $G = (V, E) \in \mathscr{G}$ has degree at most $f(|V|)$. An example of such a function is $f(x) = \lceil log(x) \rceil$.

A set S of parameters *dominates* $S' = (p_1, \ldots, p_s)$ if there is a function f which is monotonically increasing in each argument such that $\forall p \in S$ and each instance I, $p(I) \leq f(p_1(I), \ldots, p_s(I))$. In [148], Samer and Szeider considered 11 parameters (treewidth of the primal/dual/incidence graph $tw/tw^d/tw^*$, number of variables n, domain size d, number of constraints c, maximum arity r, maximum relation size t, maximum number of occurences of a variable deg, and the maximum overlap/difference between scopes

ovl/diff) and classified the complexity of CSP with respect to any subset of these parameters: it is FPT if the subset is dominated by either $\{tw^*, t\}$, $\{tw^*, d, \text{diff}\}$ or $\{d, c, \text{ovl}\}$ and W[1]-hard otherwise.

Bulatov and Marx have studied the complexity of OCSP (CSP in which an assignment must have exactly k nonzero variables to be a valid solution) and the more general CCSP (CSP in which each value $v_i \in D$ must appear exactly k_i times in the solution) with respect to the language of constraint relations Γ. The parameters considered are k for OCSP and $\sum_i k_i$ for CCSP. Generalizing previous results in the Boolean domain [134], they have shown that assuming Γ contains all constants (unary relations with a single tuple), both OCSP and CCSP exhibit a dichotomy as they are either FPT or W[1]-hard [35, 132].

8.2 Backdoors

The notion of *backdoor*, introduced in [157] in the context of SAT, is closely related to parameterized complexity. If $I = (X, D, C)$ is an instance of CSP, $S \subseteq X$ is a subset of the variables and $\phi : S \to D$ is an assignment to S, the residual instance produced by ϕ is the instance $I_\phi = (X \backslash S, D', C')$ obtained from I by removing from the constraints the tuples inconsistent with the assignment ϕ and then removing the variables in S from every constraint scope. Given a CSP instance I and a subproblem T of CSP, a strong T-backdoor is a subset B of the variables of I such that every complete assignment to B is guaranteed to produce a residual instance in T. In some sense, a strong backdoor is a measure of the distance between I and the target class T: if a small strong backdoor exists, then I can be decomposed into a small number of instances in T. It follows that CSP parameterized by domain size and strong backdoor size (with respect to an arbitrary tractable subproblem) is FPT, provided the strong backdoor is known.

The parameterized complexity of strong backdoor detection when the target class is only defined by restrictions on the constraint language is W[2]-hard for every tractable class defined by idempotent polymorphisms even if the parameter is $d+k$, where d is the domain size and k is the size of the strong backdoor [91]. Nonetheless, if r is the arity of the instance, the problem becomes FPT for the parameter $d + k + r$ [91] and even $k + r$ if the tractable class is defined by finitely many polymorphisms [37]. However, the classes for which strong backdoor detection is FPT in $k + r$ are quite small and for all tractable classes represented in Fig. 1 the problem is W[2]-hard for $k + r$ [37].

A particular form of backdoor, called *partition backdoor*, was recently introduced [14] in the case where the target tractable class T is conservative. The construction is based on the observation that if we have a partition (C_1, C_2) of the constraints such that $C_1 \in T$, then the vertex cover of the primal graph of C_2 is a strong backdoor to T; the minimum-size partition backdoor is then defined as the minimum-size such backdoor over every possible partition of the constraints. This kind of backdoor is easier to compute (FPT in $k + l$, where k is the size of the partition backdoor and l is the size of the language of the input instance), but can be arbitrarily larger than the minimum-size strong backdoor [91].

On the structural side, it is known that in the binary case finding a minimum-size strong backdoor to acyclic CSP (i.e. $\mathscr{C}_{\text{tw1}}$ the class of binary CSP instances whose constraint graph is a forest) is NP-hard in general but FPT when the parameter is k [76]. This type of backdoor is usually referred to as a *cycle cutset* [73] in the CSP literature. More generally, for every constant c, finding a minimum-size strong backdoor to CSPs whose constraint graph has treewidth at most c is FPT. This can be seen by observing that the class of graphs

obtained by adding at most k vertices to a graph of treewidth at most c is minor-closed and hence can be recognized in cubic time [146]. The corresponding algorithm is however very impractical.

For microstructure-based classes, if the target class T is defined by a finite set of forbidden patterns of size (in terms of the number of variables covered) bounded by some constant s then finding a minimum-size strong backdoor to T is FPT with respect to the size of the backdoor: the algorithm would simply generate a list of all minimal subsets of variables on which a pattern occurs (in time $O(n^s)$) and return the minimum-size HITTING SET over these sets (which is FPT when the size of each set is bounded [79]). Finally, it is worth noting that the notion of root set discussed in Section 7 in the context of functional networks is a particular case of strong backdoor.

8.3 Subexponential complexity of CSP

A subproblem of CSP is solvable in *uniform subexponential time* if it can be solved in time $d^{o(n)}|I|^{O(1)}$, where d is the domain size, n is the number of variables and $|I|$ is the instance size. The yet unproven Exponential Time Hypothesis (ETH) states that 3-SAT cannot be solved in subexponential time.

If both d and the maximum arity r are bounded, CSP is solvable in subexponential time if and only if the ETH does not hold. Furthermore, if either d or r is bounded but not the other, the subexponential time solvability of the corresponding subproblem of CSP implies the falsity of the ETH but the converse is not believed to be true [71]. Aside from the restrictions on d and r, de Haan, Kanj and Szeider have provided a more fine-grained analysis of the subexponential time complexity of CSP by considering restrictions on the number of tuples t, the number of constraints c, the treewidth of the constraint graph tw and the treewidth of the incidence graph tw^* [71]. An overview of their results can be found in Table 1. Their results are complemented by a study of the subexponential complexity of CSP when all constraints are global constraints of the same type T, where $T \in$ {AllDiff, NValue, AtMostNValue, AtLeastNValue, cTable}.

9 Tractable classes of other constraint-based problems

9.1 CSPs over infinite domains

Ladner's theorem tells us that there are problems in NP that are neither in P nor NP-complete, assuming P≠NP [128]. Bodirsky and Grohe showed that every computational decision problem is polynomial-time equivalent to a problem CSP(Γ) where Γ is an infinite

Table 1 Subexponential time complexity of CSP under restrictions on the number of tuples, number of constraints, treewidth of the primal of the constraint graph and incidence treewidth of the constraint graph

Solvable in subexponential time	Not solvable in subexponential time (assuming the ETH)
$t = o(n)$	$t = \Omega(n)$
$c = O(1)$	$c = \omega(1)$
$tw = o(n)$	$tw = \Omega(n)$
$tw^* = O(1)$	$tw^* = \omega(1)$

language over an infinite domain, and adapted the proof of Ladner's theorem to show that no dichotomy can exist for infinite constraint languages over infinite domains [18].

It is nevertheless possible to characterise the complexity of certain restricted versions of infinite-domain CSPs. For example, Bodirsky and Kára studied the computational complexity of CSPs over countably infinite domains in which all constraint relations can be defined by a Boolean combination of the equality relation (e.g. $(X_1 = X_2) \vee (X_2 \neq X_3)$) [19]. They proved a dichotomy theorem for the problems CSP(Γ), the set of CSP instances in which all relations belong to a set Γ. The only non-trivial tractable classes Γ (assuming P≠NP) consist of Horn =-formulas: those formulas which are defined by a Boolean formula in conjunctive normal form in which each clause contains at most one positive literal (i.e. $X_i = X_j$ rather than $X_i \neq X_j$). A necessary condition has also been given for tractability of maximal constraint languages over infinite domains [17].

Infinite and continuous domains, such as the real numbers or the rationals, are often used to model time in temporal reasoning problems or space in spatial reasoning problems. A landmark tractable class of temporal problems are the so-called simple temporal problems (STP). An STP consists of a set of real-valued variables and a set of *simple temporal constraints*, that is constraints of the form $c \leq X_i - X_j \leq d$ for constants c, d. An STP can be solved in $O(n^3)$ time by using Floyd-Warshall's all-pairs shortest paths algorithm [74]. A simple temporal problem with difference constraints (in which we may also have constraints of the form $X_i - X_j \neq d$) can be solved in $O(n^3 + k)$ time and $O(n^2 + k)$ space, where n is the number of variables and k the number of difference constraints [92]. In another generalisation, simple temporal constraints have been extended to *shift monotonic constraints*: $X_i - X_j \in [f(X_j, X_i), g(X_j, X_i)]$, where both $f(x, y)$ and $g(x, y)$ are monotone increasing functions of x and monotone decreasing functions of y. The consistency of a set of shift-monotonic constraints can again be tested in cubic time [144].

Simple temporal constraints have been generalised to the larger tractable class of linear Horn constraints (also known as Horn DLR or Horn disjunctive linear relations). A *linear Horn relation* is a disjunction of linear relations in which all but at most one of the disjuncts is of the form $p_i(X_1, \ldots, X_n) \neq q_i(X_1, \ldots, X_n)$ and at most one disjunct is of the form $p_i(X_1, \ldots, X_n)\ R\ q_i(X_1, \ldots, X_n)$ where R is $<$, $\leq$ or $=$ and the polynomials p_i, q_i are all linear [122]. The satisfiability of a set of linear Horn constraints can be tested in polynomial time using an algorithm based on linear programming [115]. Linear Horn constraints generalise several previously-published tractable classes [80] and are an example of a general method of building tractable languages using disjunctions [50].

Given that no dichotomy can exist for constraint languages over infinite domains, it is natural to restrict attention to classes of constraints which commonly occur in real applications. A *temporal relation* is a relation $R \subseteq \mathbb{Q}_k$, for some finite k, with a first-order definition in $(\mathbb{Q}; <)$, the ordered rational numbers. A *temporal constraint language* is a set of temporal relations. Bodirsky and Kára showed that there are exactly nine tractable temporal constraint languages [20]. Allen's interval algebra consists of binary relations between intervals which are disjunctions of 13 basic interval relations (such as before, meets, includes, overlaps, starts, finishes or equals) [3]. For the problem of deciding whether there exist intervals on the real line satisfying a set of relations, a dichotomy has been given for all tractable subalgebras of Allen's algebra [125, 126]. In the Region Containment Calculus (RCC-5) used in spatial reasoning, variables denote non-empty regions and the basic relations in the calculus express containment, disjointness, overlap or equality of pairs of regions. A complete classification of all tractable fragments of the region connection calculus RCC-5 has been given [116].

9.2 Optimisation versions of the CSP

In optimisation versions of the CSP, constraints are replaced by *soft constraints* $\langle \sigma, \phi \rangle$, where ϕ is a function from the Cartesian product of the domains of the variables in the scope σ to a set of costs. This set of costs is totally ordered in the case of Valued CSPs and partially-ordered in the case of semi-ring CSPs [15]. The objective function to minimise is the aggregation of the cost functions. For example, whereas financial costs or log-probabilities are usually added, fuzzy values may be aggregated using an idempotent operator or, in the leximin model, by multiset union of costs [15]. Since many applications can be modelled by minimising the sum of costs in $\mathbb{Q}^{\geq 0} \cup \{\infty\}$, we will only cover this important special case. Cost functions which only take values in $\{c, \infty\}$, for some finite constant c are known as *crisp* and are equivalent to classical constraints.

State-of-the-art results concerning the tractability of languages of cost functions are covered in detail in a recent comprehensive survey article [113]. A complete dichotomy is impossible without resolving the Feder-Vardi conjecture since the general-valued VCSP exhibits a dichotomy if and only if the CSP does [120]. However, tractable languages have been characterised in the following important cases:

- Boolean domain [49],
- finite-valued cost functions (including MAX-CSP as a special case) [152],
- conservative languages of cost functions (i.e. languages of cost functions containing all finite-valued unary cost functions) [121].

Furthermore, an algebraic theory has recently been developed for soft constraints which generalises polymorphisms to weighted polymorphisms, relational clones to weighted relational clones and clones to weighted clones [46]. The expressibility of a language Γ of cost functions is precisely characterised by its weighted polymorphisms. The indicator problem for languages of relations (see Section 3) has been generalised to the weighted indicator problem for languages of cost functions [159].

In the case of the Boolean domain [49], the eight maximal tractable languages include the six tractable languages in Schaefer's dichotomy for Boolean crisp constraints given in Section 3. One of the other two languages is almost trivial since it consists of crisp bijective binary constraints together with arbitrary unary cost functions. The remaining tractable language is the set of submodular functions which are well-known to be tractable in the Operations Research community [105]. A cost function $\phi : D^r \to \mathbb{R}^{\geq 0} \cup \{\infty\}$ is *submodular* if

$$\forall \mathbf{x}, \mathbf{y} \in D^r, \quad \phi(\min(\mathbf{x}, \mathbf{y})) + \phi(\max(\mathbf{x}, \mathbf{y})) \;\leq\; \phi(\mathbf{x}) + \phi(\mathbf{y}).$$

where min and max are applied componentwise. Examples of submodular functions include all unary functions, binary functions such as $\sqrt{x^2 + y^2}$, the cut function of a graph, the rank function of a matroid and the function $\eta_a^\rho : D^s \to \mathbb{R}^{\geq 0} \cup \{\infty\}$ (for $\rho > 0$), where

$$\eta_a^\rho(\mathbf{x}) \;=\; \begin{cases} 0 \text{ if } (x_1 < a_1 \wedge \ldots \wedge x_r < a_r) \vee (x_{r+1} > a_{r+1} \wedge \ldots \wedge x_s > a_s) \\ \rho \text{ otherwise.} \end{cases}$$

If $G = \langle V, E \rangle$ is an undirected graph with non-negative weighted edges, then a *cut* $S \in 2^V$ is a partition of the vertices of G into two disjoint subsets, the corresponding cut-set is the set of edges that have one endpoint in each subset of the partition and the *cut function* $f_G(S)$ is the sum of the weights of the edges in the cut-set of S. The cut function of a graph can

be expressed as the sum of functions of the form η_a^ρ, as can generalisations such as the cut function of a hypergraph or a directed graph.

In the case of finite-valued cost functions, the tractable languages Γ which are cores are exactly those that have a binary symmetric weighted polymorphism [152]. Such languages Γ, which include submodular functions, are solvable by linear programming. The VCSP can be coded as as an integer programming problem (whose variables include $v_{ia} \in \{0, 1\}$ which is equal to 1 iff $X_i = a$ in the original VCSP instance). The linear relaxation of this integer programming problem (in which v_{ia} is now a real number in the interval [0, 1]) has integer solutions for such languages Γ, meaning that VCSP(Γ) can be solved by linear programming. The dual of this relaxation is the linear program used by OSAC [57] to transform the original instances into an equivalent instance with an explicit lower bound on cost [155]; for instances in VCSP(Γ) this explicit lower bound is thus equal to the cost of an optimal solution. Indeed, even if the language Γ of finite-valued cost functions is not a core, Γ is still tractable if and only if VCSP(Γ) is solved by this linear program [152].

We now consider the more general case of cost functions which are not necessarily finite-valued. A *conservative* language Γ contains all $\{0, 1\}$-valued unary cost functions. Assuming P≠NP, for all tractable conservative languages Γ, VCSP(Γ) can be solved by first establishing local consistency and then solving the linear relaxation, mentioned above, of the resulting instance [121]. Thapper and Živný [151] have obtained a precise algebraic characterisation of (not necessarily conservative) languages Γ for which VCSP(Γ) can be solved exactly by the linear programming relaxation. This notably includes languages of cost functions that are submodular on arbitrary lattices.

9.3 Quantified CSP, Uncertain CSP, #CSP and related problems

In this section we briefly consider tractability of extensions of the CSP to quantified variables, uncertainty or counting the number of solutions.

In the *Quantified CSP* (QCSP) variables are existentially or universally quantified. This is therefore a generalisation of the classical CSP, which can be viewed as a QCSP in which the question of the existence of a solution corresponds to placing an existential quantifier on each variable. QCSP is PSPACE-complete.

There is a dichotomy for the complexity of constraint languages for the *relatively quantified CSP* (RQCSP), the version of the CSP in which variables may be universally and existentially quantified over arbitrary subsets of the finite domain: RQCSP(Γ) is either in P or is PSPACE-complete. In fact, RQCSP(Γ) $\in$ P if and only if for every 2-element subset B of the domain D, Γ has a polymorphism f such that the restriction $f|_B$ of f to B is a semilattice, majority or minority operation. This result was discovered independently by two different groups of researchers [16, 85]. Recall that over the Boolean domain, relations equivalent to Horn clauses, relations equivalent to 2SAT clauses and affine constraints have, respectively, semilattice, majority and minority polymorphisms.

The tractable class defined by forbidding the broken-triangle pattern shown in Fig. 2c for a given variable order can be extended to a tractable class of QCSP [88]. The corresponding property is called the broken-angle property.

The *Uncertain CSP* (UCSP) is a general framework allowing either discrete or continuous domains and which does not impose any particular representation of uncertainty (which can be, for example, sets, intervals, ellipsoids). It is a nonprobabilistic framework which extends the CSP to allow incomplete or erroneous data. Known tractable classes of the UCSP include linear, monotone and simple temporal constraints [158].

In the #CSP the aim is to count the number of solutions to a classical CSP. As we have already pointed out, bounded treewidth leads to a structural tractable class for #CSP. It is again the polymorphisms of a language Γ which determine tractability of #CSP(Γ). Bulatov has characterised all languages for which #CSP(Γ) can be solved in polynomial time and proved that for all other languages the problem is #P-complete [26]. All tractable languages have a Mal'tsev polymorphism. This includes the important tractable problem of counting the number of solutions to a system of linear equations. The criterion of Bulatov's dichotomy involves finding a defect in any of a potentially infinite class of structures built on Γ and it was unclear whether this criterion is algorithmically checkable. It was recently shown that this criterion is not only decidable, but actually in NP and unlikely to be NP-complete since it can be reduced to graph isomorphism [81].

An important tool in the study of the complexity of classical CSPs is the notion of a relational clone, which is the set $\langle\Gamma\rangle$ of all relations expressible using primitive positive formulas over a particular set of relations Γ. An analogous notion of functional clones has been developed to study the expressibility and computational complexity of languages for the (weighted) #CSP [30]. The problem of computing the partition function (the sum of the weights of all possible variable assignments) is a generalisation of #CSP which has applications in statistical physics [154]. Complexity dichotomies are known for different versions of the problem of computing the partition function of a weighted CSP, in the case of the Boolean domain [29, 82].

10 Discussion

This survey of tractable classes has allowed us to highlight certain long-standing open theoretical questions:

1. Is there a dichotomy for tractable languages, as conjectured by Feder and Vardi [86]? This is equivalent to determining the tractablity of Siggers operations.
2. Which classes of tractable languages can be recognised in polynomial time? In particular, it is presently not known whether the class of languages with a Mal'tsev polymorphism (and, more generally, a k-edge polymorphism) is recognisable in polynomial time.
3. The finite-infinite question: is CSP(Γ) $\in$ P for all tractable languages Γ? Recall that Γ is tractable if CSP(Γ') $\in$ P for all finite $\Gamma' \subseteq \Gamma$. For a finite language Γ', we can consider the maximum arity and the domain size as constants. Relaxing these strong assumptions may lead to the identification of tractable languages involving so-called global constraints and/or unbounded domains (as commonly encountered in database and genetic applications [160]).

Looking beyond the problem of determining the exact borderline between P and NP-complete, it is also possible to study the relative complexity of NP-complete problems. For example, over the Boolean domain, the language Γ consisting of the single relation $R_{1/3}^{\neq\neq\neq}$ containing the three tuples $\langle 1,0,0,0,1,1\rangle$, $\langle 0,1,0,1,0,1\rangle$ and $\langle 0,0,1,1,1,0\rangle$, has been shown to be the computationally easiest NP-complete language [117]. *Computationally easiest* means that if any NP-complete CSP(Γ') can be solved in $O(c^n)$ time, then CSP(Γ) can also be solved in $O(c^n)$ time. We also note that while the parameterized complexity of CSPs has been relatively well studied, the fixed-parameter tractability results obtained have yet to

be used in practical applications. It would be interesting to see if this approach can provide competitive tools to improve the efficiency of constraint solvers.

In Section 1 we highlighted the possible use of tractable classes as lower bound instances (relaxations). It is always possible, by eliminating constraints, to find a lower-bound instance with treewidth bounded by any given constant k. For a given polymorphism f which defines a tractable language, we can always obtain a lower-bound instance of an instance I in this language by successively adding tuples to each constraint relation R in I until the resulting relation satisfies the polymorphism f. The resulting relation is simply the closure of R under componentwise application of f to sets of tuples until convergence, and can be calculated in polynomial time for constraints of arity bounded by a constant. For a given forbidden pattern defining a tractable class, we can again find a lower-bound instance using a similar technique of successively adding tuples to relations until the relaxed instance falls in the class.

There is a danger that the concentration of effort on identifying language-based and structural tractable classes may deflect attention from the possibility of alternative definitions of tractable classes. We are convinced that there remain many useful tractable classes to be discovered by exploring different definitions of sets of instances. It is also interesting to note that two very important relaxations, namely local consistency in the case of the CSP and linear relaxation in the case of the VCSP, are obtained not by restricting the set of instances but rather by changing the problem statement. An interesting open question is whether there exist other useful tractable relaxations of constraint problems that can be obtained by changing the problem statement.

Acknowledgments We are grateful to Peter Jeavons and Stanislav Živný for their detailed comments on a first draft of this paper, and to the reviewers for their constructive comments.

References

1. Abrahamson, K.A., Downey, R.G., & Fellows, M.R. (1995). Fixed-parameter tractability and completeness IV: On completeness for W[P] and PSPACE analogues. *Annals of Pure and Applied Logic*, *73*(3), 235–276.
2. Adler, I., Gottlob, G., & Grohe, M. (2007). Hypertree width and related hypergraph invariants. *European Journal of Combinatorics*, *28*(8), 2167–2181.
3. Allen, J.F. (1983). Maintaining knowledge about temporal intervals. *Communications of the ACM*, *26*, 832–843.
4. Arnborg, S. (1985). Efficient algorithms for combinatorial problems with bounded decomposability - a survey. *BIT*, *25*(1), 2–23.
5. Arnborg, S., Corneil, D.G., & Proskurowski, A. (1987). Complexity of finding an embedding in k-trees. *SIAM journal of Algebraic and Discrete Methods*, *8*, 277–284.
6. Barto, L. (2011). The dichotomy for conservative constraint satisfaction problems revisited. In *LICS* (pp. 301–310): IEEE Computer Society.
7. Barto, L. (2011). Finitely related algebras in congruence distributive varieties have near unanimity terms. *Canadian Journal of Mathematics*.
8. Barto, L. (2015). The collapse of the bounded width hierarchy. *Journal of Logic and Computation*. doi:10.1093/logcom/exu070.
9. Barto, L., & Kozik, M. (2014). Constraint satisfaction problems solvable by local consistency methods. *Journal of the ACM*, *61*(1), 3.
10. Barto, L., Kozik, M., & Niven, T. (2009). The CSP dichotomy holds for digraphs with no sources and no sinks (a positive answer to a conjecture of bang-jensen and hell). *SIAM Journal on Computing*, *38*(5), 1782–1802.
11. van Beek, P., & Dechter, R. (1995). On the minimality and decomposability of row-convex constraint networks. *Journal of the ACM*, *42*(3), 543–561.

12. van Beek, P., & Dechter, R. (1997). Constraint tightness and looseness versus local and global consistency. *Journal of the ACM*, *44*(4), 549–566.
13. Bertelè, U., & Brioshi, F. (1972). *Nonserial dynamic programming*. Academic Press.
14. Bessière, C., Carbonnel, C., Hébrard, E., Katsirelos, G., & Walsh, T. (2013). Detecting and exploiting subproblem tractability. In *Proceedings of the 23rd international joint conference on Artificial Intelligence* (pp. 468–474). AAAI Press.
15. Bistarelli, S., Montanari, U., Rossi, F., Schiex, T., Verfaillie, G., & Fargier, H. (1999). Semiring-based CSPs and valued CSPs: Frameworks, properties, and comparison. *Constraints*, *4*, 199–240.
16. Bodirsky, M., & Chen, H. (2009). Relatively quantified constraint satisfaction. *Constraints*, *14*(1), 3–15.
17. Bodirsky, M., Chen, H., Kára, J., & von Oertzen, T. (2009). Maximal infinite-valued constraint languages. *Theoretical Computer Science*, *410*(18), 1684–1693.
18. Bodirsky, M., & Grohe, M. (2008). Non-dichotomies in constraint satisfaction complexity. In *ICALP* (pp. 184–196).
19. Bodirsky, M., & Kára, J. (2008). The complexity of equality constraint languages. *Theory Computing Systems*, *43*(2), 136–158.
20. Bodirsky, M., & Kára, J. (2010). The complexity of temporal constraint satisfaction problems. *Journal of the ACM*, *57*(2), 9:1–9:41.
21. Bodlaender, H.L. (1996). A linear-time algorithm for finding tree-decompositions of small treewidth. *SIAM Journal on Computing*, *25*(6), 1305–1317.
22. Bulatov, A.A. (2006). Combinatorial problems raised from 2-semilattices. *Journal of Algebra*, *298*(2), 321–339.
23. Bulatov, A.A. (2006). A dichotomy theorem for constraint satisfaction problems on a 3-element set. *Journal of the ACM*, *53*(1), 66–120.
24. Bulatov, A.A. (2010). *Bounded relational width*. Tech. rep., School of Computer Science, Simon Fraser University.
25. Bulatov, A.A. (2011). Complexity of conservative constraint satisfaction problems. *ACM Transactions on Computational Logic*, *12*(4), 24.
26. Bulatov, A.A. (2013). The complexity of the counting constraint satisfaction problem. *Journal of the ACM*, *60*(5), 34.
27. Bulatov, A.A. (2014). Conservative constraint satisfaction re-revisited. preprint arXiv:1408.3690v1.
28. Bulatov, A.A., & Dalmau, V. (2006). A simple algorithm for Mal'tsev constraints. *SIAM Journal on Computing*, *36*(1), 16–27.
29. Bulatov, A.A., Dyer, M.E., Goldberg, L.A., Jalsenius, M., & Richerby, D. (2009). The complexity of weighted Boolean #CSP with mixed signs. *Theoretical Computer Science*, *410*(38–40), 3949–3961.
30. Bulatov, A.A., Dyer, M.E., Goldberg, L.A., Jerrum, M., & McQuillan, C. (2013). The expressibility of functions on the boolean domain, with applications to counting CSPs. *Journal of the ACM*, *60*(5), 32.
31. Bulatov, A.A., & Jeavons, P.G. (2001). Algebraic structures in combinatorial problems. Tech. Rep. MATH-AL-4-2001, Technische Universität Dresden.
32. Bulatov, A.A., Jeavons, P.G., & Krokhin, A.A. (2005). Classifying the complexity of constraints using finite algebras. *SIAM Journal on Computing*, *34*, 720–742.
33. Bulatov, A.A., Krokhin, A.A., & Jeavons, P. (2001). The complexity of maximal constraint languages. In J.S. Vitter, P.G. Spirakis, & M. Yannakakis (Eds.), *STOC* (pp. 667–674). ACM.
34. Bulatov, A.A., Krokhin, A.A., & Larose, B. (2008). Dualities for constraint satisfaction problems. In N. Creignou, P.G. Kolaitis, & H. Vollmer (Eds.), *Complexity of Constraints - An Overview of Current Research Themes [Result of a Dagstuhl Seminar]., Lecture Notes in Computer Science*, (Vol. 5250 pp. 93–124). Springer.
35. Bulatov, A.A., & Marx, D. (2014). Constraint satisfaction parameterized by solution size. *SIAM Journal on Computing*, *43*(2), 573–616.
36. Bulatov, A.A., Valeriote, M., P. Kolaitis, & H. Vollmer (2008). Recent results on the algebraic approach to the CSP. In N. Creignou (Ed.), *Complexity of Constraints, Lecture Notes in Computer Science*, (Vol. 5250 pp. 68–92). Berlin / Heidelberg: Springer.
37. Carbonnel, C., Cooper, M.C., & Hebrard, E. (2014). On backdoors to tractable constraint languages. In B. O'Sullivan (Ed.), *Principles and Practice of Constraint Programming - 20th International Conference, CP 2014, Lyon, France, September 8-12, 2014. Proceedings, Lecture Notes in Computer Science*, (Vol. 8656 pp. 224–239). Springer.
38. Carvalho, C., Egri, L., Jackson, M., & Niven, T. (2011). On Maltsev digraphs. In *Computer Science–Theory and Applications* (pp. 181–194). Springer.
39. Chen, H. (2009). A rendezvous of logic, complexity, and algebra. *ACM Computing Surveys*, *42*(1).

40. Chen, H., Dalmau, V., & Grußien, B. (2013). Arc consistency and friends. *Journal of Logic and Computation*, *23*(1), 87–108.
41. Chen, H., & Grohe, M. (2010). Constraint satisfaction with succinctly specified relations. *Journal of Computer and System Sciences*, *76*(8), 847–860.
42. Chen, J., Kanj, I.A., & Xia, G. (2010). Improved upper bounds for vertex cover. *Theoretical Computer Science*, *411*(40), 3736–3756.
43. Chudnovsky, M., Cornuéjols, G., Liu, X., Seymour, P., & Vuskovic, K. (2005). Recognizing Berge graphs. *Combinatorica*, *25*(2), 143–186.
44. Chudnovsky, M., Robertson, N., Seymour, P., & Thomas, R. (2006). The strong perfect graph theorem. *Annals of Math*, *164*(1), 51–229.
45. Cobham, A. (1964). The intrinsic computational difficulty of functions. In *Proceedings of International Congress for Logic, Methodology, and Philosophy of Science* (pp. 24–30). North-Holland.
46. Cohen, D.A., Cooper, M.C., Creed, P., Jeavons, P.G., & Živný, S. (2013). An algebraic theory of complexity for discrete optimisation. *SIAM Journal on Computing*, *42*(5), 1915–1939.
47. Cohen, D.A., Cooper, M.C., Creed, P., Marx, D., & Salamon, A.Z. (2012). The tractability of CSP classes defined by forbidden patterns. *Journal of Artificial Intelligence Research (JAIR)*, *45*, 47–78.
48. Cohen, D.A., Cooper, M.C., Green, M.J., & Marx, D. (2011). On guaranteeing polynomially bounded search tree size. In J.H.M. Lee (Ed.), *CP, Lecture Notes in Computer Science*, (Vol. 6876 pp. 160–171). Springer.
49. Cohen, D.A., Cooper, M.C., Jeavons, P., & Krokhin, A.A. (2006). The complexity of soft constraint satisfaction. *Artificial Intelligence*, *170*(11), 983–1016.
50. Cohen, D.A., Jeavons, P., Jonsson, P., & Koubarakis, M. (2000). Building tractable disjunctive constraints. *Journal of the ACM*, *47*(5), 826–853.
51. Cohen, D.A., & Jeavons, P.G. (2006). The complexity of constraint languages. In F. Rossi, P. van Beek, & T. Walsh (Eds.), *Handbook of constraint programming* (pp. 245–280). Elsevier.
52. Cook, S.A. (1971). The complexity of theorem-proving procedures. In M.A. Harrison, R.B. Banerji, & J.D. Ullman (Eds.), *STOC* (pp. 151–158). ACM.
53. Cooper, M.C. (1999). Linear-time algorithms for testing the realisability of line drawings of curved objects. *Artificial Intelligence*, *108*, 31–67.
54. Cooper, M.C. (2014). Beyond consistency and substitutability. In *CP* (pp. 256–271).
55. Cooper, M.C., Cohen, D.A., & Jeavons, P. (1994). Characterising tractable constraints. *Artificial Intelligence*, *65*(2), 347–361.
56. Cooper, M.C., & Escamocher, G. (2012). A dichotomy for 2-constraint forbidden CSP patterns. In J. Hoffmann, & B. Selman (Eds.), *AAAI*. AAAI Press.
57. Cooper, M.C., de Givry, S., Sanchez, M., Schiex, T., Zytnicki, M., & Werner, T. (2010). Soft arc consistency revisited. *Artificial Intelligence*, *174*(7–8), 449–478.
58. 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), 570–584.
59. Cooper, M.C., Maris, F., & Régnier, P. (2014). Monotone temporal planning: Tractability, extensions and applications. *Journal of Artificial Intelligence Research (JAIR)*, *50*, 447–485.
60. Cooper, M.C., Mouelhi, A.E., Terrioux, C., & Zanuttini, B. (2014). On broken triangles. In O'Sullivan, B. (Ed.), *Principles and Practice of Constraint Programming - 20th International Conference, CP 2014, Lyon, France, September 8-12, 2014. Proceedings, Lecture Notes in Computer Science*, (Vol. 8656 pp. 9–24). Springer.
61. Cooper, M.C., & Živný, S. (2011). Hybrid tractability of valued constraint problems. *Artificial Intelligence*, *175*(9-10), 1555–1569.
62. Cooper, M.C., & Živný, S. (2012). Tractable triangles and cross-free convexity in discrete optimisation. *Journal of Artificial Intelligence Research (JAIR)*, *44*, 455–490.
63. Courcelle, B. (1990). Graph rewriting: An algebraic and logic approach. In *Handbook of theoretical computer science, Volume B: formal models and sematics (B)* (pp. 193–242).
64. Creignou, N., & Hébrard, J.J. (1997). On generating all solutions of generalized satisfiability problems. *Informatique Théorique et Applications*, *31*(6), 499–511.
65. Creignou, N., Kolaitis, P., & Vollmer, H. (Eds.) (2008). *Complexity of Constraints, Lecture Notes in Computer Science*, Vol. 5250. Springer.
66. Dalmau, V. (2000). A new tractable class of constraint satisfaction problems., *AMAI*.
67. Dalmau, V. (2006). Generalized majority-minority operations are tractable. *Logical Methods in Computer Science*, *2*(4), 438–447.
68. Dalmau, V., & Pearson, J. (1999). Closure functions and width 1 problems. In *Principles and Practice of Constraint Programming–CP 99* (pp. 159–173). Springer.

69. David, P. (1995). Using pivot consistency to decompose and solve functional CSPs. *Journal of Artificial Intelligence Research (JAIR)*, *2*, 447–474.
70. de Givry, S., Schiex, T., & Verfaillie, G. (2006). Exploiting tree decomposition and soft local consistency in weighted CSP. In *AAAI* (pp. 22–27).
71. de Haan, R. Kanj (2015). On the subexponential-time complexity of CSP. *Journal of Artificial Intelligence Research*, *52*, 203–234.
72. Dechter, R. (1992). From local to global consistency. *Artificial Intelligence*, *55*(1), 87–108.
73. Dechter, R. (2003). *Constraint processing*, (pp. 94104–3205). San Francisco: Morgan Kaufmann Publishers.
74. Dechter, R., Meiri, I., & Pearl, J. (1991). Temporal constraint networks. *Artificial Intelligence*, *49*(1-3), 61–95.
75. Dechter, R., & Pearl, J. (1987). Network-based heuristics for constraint-satisfaction problems. *Artificial Intelligence*, *34*(1), 1–38.
76. Dehne, F.K.H.A., Fellows, M.R., Langston, M.A., Rosamond, F.A., & Stevens, K. (2005). An $o(2^{o(k)}n^3)$ FPT algorithm for the undirected feedback vertex set problem. In L. Wang (Ed.), *Computing and Combinatorics, 11th Annual International Conference, COCOON 2005, Kunming, China, August 16–29, 2005, Proceedings, Lecture Notes in Computer Science*, (Vol. 3595 pp. 859–869). Springer.
77. Deville, Y., Barette, O., & Hentenryck, P.V. (1999). Constraint satisfaction over connected row convex constraints. *Artificial Intelligence*, *109*(1-2), 243–271.
78. Downey, R.G., & Fellows, M.R. (1995). Fixed-parameter tractability and completeness I: Basic results. *SIAM Journal on Computing*, *24*(4), 873–921.
79. Downey, R.G., Fellows, M.R., & Stege, U. (1999). Parameterized complexity: A framework for systematically confronting computational intractability. In *Contemporary trends in discrete mathematics: From DIMACS and DIMATIA to the future*, (Vol. 49 pp. 49–99). AMS-DIMACS Proceedings Series.
80. Drakengren, T., & Jonsson, P. (2005). Computational complexity of temporal constraint problems. In M. Fisher, D. Gabbay, & L. Vila (Eds.), *Handbook of temporal reasoning in artificial intelligence* (pp. 197–218). Elsevier.
81. Dyer, M., & Richerby, D. (2013). An effective dichotomy for the counting constraint satisfaction problem. *SIAM Journal on Computing*, *42*(3), 1245–1274.
82. Dyer, M.E., Goldberg, L.A., & Jerrum, M. (2009). The complexity of weighted boolean #CSP. *SIAM Journal on Computing*, *38*(5), 1970–1986.
83. Edmonds, J. (1965). Paths, trees and flowers. *Canadian Journal of Mathematics*, *17*, 449–467.
84. Feder, T., & Hell, P. (2006). Full constraint satisfaction problems. *SIAM Journal on Computing*, *36*(1), 230–246.
85. Feder, T., & Kolaitis, P.G. (2006). Closures and dichotomies for quantified constraints. *Electronic Colloquium on Computational Complexity (ECCC)*, *13*(160).
86. Feder, T., & Vardi, M.Y. (1998). The computational structure of monotone monadic SNP and constraint satisfaction: A study through Datalog and group theory. *SIAM Journal of Computing*, *28*(1), 57–104.
87. Freuder, E.C. (1990). Complexity of k-tree structured constraint satisfaction problems. In H.E. Shrobe, T.G. Dietterich, & W.R. Swartout (Eds.), *AAAI* (pp. 4–9). AAAI Press / The MIT Press.
88. Gao, J., Yin, M., & Zhou, J. (2011). Hybrid tractable classes of binary quantified constraint satisfaction problems. In W. Burgard, & D. Roth (Eds.), *AAAI*. AAAI Press.
89. Gao, Y. (2003). Phase transition of tractability in constraint satisfaction and bayesian network inference. In *UAI* (pp. 265–271).
90. Garey, M., & Johnson, D. (1979). *Computers and intractability: A guide to the theory of NP-completeness*. San Francisco: W.H.Freeman and Company.
91. Gaspers, S., Misra, N., Ordyniak, S., Szeider, S., & Zivny, S. (2014). Backdoors into heterogeneous classes of SAT and CSP. In C.E. Brodley, & P. Stone (Eds.), *Proceedings of the Twenty-Eighth AAAI Conference on Artificial Intelligence, July 27 -31, 2014, Québec City, Québec, Canada* (pp. 2652–2658). AAAI Press. http://www.aaai.org/Library/AAAI/aaai14contents.php.
92. Gerevini, A., & Cristani, M. (1997). On finding a solution in temporal constraint satisfaction problems. In *IJCAI* (pp. 1460–1465).
93. Gottlob, G., Leone, N., & Scarcello, F. (2002). Hypertree decomposition and tractable queries. *Journal of Computer and System Sciences*, *64*(3), 579–627.
94. Green, M.J., & Cohen, D.A. (2008). Domain permutation reduction for constraint satisfaction problems. *Artificial Intelligence*, *172*(8-9), 1094–1118.
95. Grohe, M. (2001). Generalized model-checking problems for first-order logic. In *STACS 2001* (pp. 12–26). Springer.
96. Grohe, M. (2006). The structure of tractable constraint satisfaction problems. In *MFCS* (pp. 58–72).
97. Grohe, M. (2007). The complexity of homomorphism and constraint satisfaction problems seen from the other side. *Journal of the ACM*, *54*(1), 1–24.

98. Grohe, M., Kreutzer, S., & Siebertz, S. (2014). Deciding first-order properties of nowhere dense graphs. In *Proceedings of the 46th Annual ACM Symposium on Theory of Computing* (pp. 89–98). ACM.
99. Grohe, M. (2014). Marx, D.: Constraint solving via fractional edge covers. *ACM Transactions on Algorithms*, *11*(1), 4.
100. Grötschel, M., Lovasz, L., & Schrijver, A. (1981). The ellipsoid method and its consequences in combinatorial optimization. *Combinatorica*, *1*, 169–198.
101. Hell, P., & Nešetřil, J. (1990). On the complexity of h-coloring. *Journal of Combinatorial Theory, Series B*, *48*(1), 92–110.
102. Hell, P., & Nešetřil, J. (2008). Colouring, constraint satisfaction, and complexity. *Computer Science Review*, *2*(3), 143–163.
103. Hermann, M., & Richoux, F. (2009). On the computational complexity of monotone constraint satisfaction problems. In S. Das, & R. Uehara (Eds.), *WALCOM, Lecture Notes in Computer Science*, (Vol. 5431 pp. 286–297). Springer.
104. Idziak, P.M., Markovic, P., McKenzie, R., Valeriote, M., & Willard, R. (2010). Tractability and learnability arising from algebras with few subpowers. *SIAM Journal on Computing*, *39*(7), 3023–3037.
105. Iwata, S., & Orlin, J.B. (2009). A simple combinatorial algorithm for submodular function minimization. In *SODA* (pp. 1230–1237).
106. Jeavons, P., Cohen, D., & Gyssens, M. (1995). A unifying framework for tractable constraints. In *Proceedings 1st international conference on constraint programming, CP'95*, (Vol. 976 pp. 276–291). Springer-Verlag.
107. Jeavons, P., Cohen, D.A., & Gyssens, M. (1997). Closure properties of constraints. *Journal of the ACM*, *44*(4), 527–548.
108. Jeavons, P., & Petke, J. (2012). Local consistency and SAT-solvers. *Journal of Artificial Intelligence Research (JAIR)*, *43*, 329–351.
109. Jeavons, P.G. (1998). Constructing constraints. In *Proceedings 4th International Conference on Constraint Programming—CP'98 (Pisa, October 1998), Lecture Notes in Computer Science*, (Vol. 1520 pp. 2–16). Springer-Verlag.
110. Jeavons, P.G. (1998). On the algebraic structure of combinatorial problems. *Theoretical Computer Science*, *200*, 185–204.
111. Jeavons, P.G., Cohen, D.A., & Cooper, M.C. (1998). Constraints, consistency and closure. *Artificial Intelligence*, *101*(1–2), 251–265.
112. Jeavons, P.G., & Cooper, M.C. (1995). Tractable constraints on ordered domains. *Artificial Intelligence*, *79*(2), 327–339.
113. Jeavons, P.G., Krokhin, A.A., & Živný, S. (2014). *The complexity of valued constraint satisfaction*: Bulletin of EATCS.
114. Jégou, P. (1993). Decomposition of domains based on the micro-structure of finite constraint-satisfaction problems. In *AAAI* (pp. 731–736). Menlo Park: AAAI Press.
115. Jonsson, P., & Bäckström, C. (1998). A unifying approach to temporal constraint reasoning. *Artificial Intelligence*, *102*(1), 143–155.
116. Jonsson, P., & Drakengren, T. (1997). A complete classification of tractability in RCC-5. *Journal of Artificial Intelligence Research*, *6*, 211–221.
117. Jonsson, P., Lagerkvist, V., Nordh, G., & Zanuttini, B. (2013). Complexity of SAT problems, clone theory and the exponential time hypothesis. In S. Khanna (Ed.), *SODA* (pp. 1264–1277). SIAM.
118. Karp, R.M. (1972). Reducibility among combinatorial problems. In R.E. Miller, & J.W. Thatcher (Eds.), *Complexity of computer computations* (pp. 85–103). Plenum Press.
119. Kazda, A. (2011). CSP for binary conservative relational structures. preprint arXiv:1112.1099.
120. Kolmogorov, V., Krokhin, A., & Rolinek, M. (2015). The complexity of general-valued CSPs. arXiv preprint arXiv:1502.07327.
121. Kolmogorov, V., & Živný, S. (2013). The complexity of conservative valued CSPs. *Journal of the ACM*, *60*(2), 10.
122. Koubarakis, M. (1996). Tractable disjunctions of linear constraints. In E.C. Freuder (Ed.), *CP, Lecture Notes in Computer Science*, (Vol. 1118 pp. 297–307). Springer.
123. Kozik, M. (2008). A finite set of functions with an EXPTIME-complete composition problem. *Theoretical Computer Science*, *407*(1–3), 330–341.
124. Kozik, M., Krokhin, A., Valeriote, M., & Willard, R. Characterizations of several Maltsev conditions. preprint (2013). (to appear in Algebra Universalis).
125. Krokhin, A., Jeavons, P., & Jonsson, P. (2003). Reasoning about temporal relations: The tractable subalgebras of Allen's interval algebra. *Journal of the ACM*, *50*, 591–640.
126. Krokhin, A., Jeavons, P., & Jonsson, P. (2004). Constraint satisfaction problems on intervals and lengths. *SIAM Journal on Discrete Mathematics*, *17*(3), 453–477.

127. Kun, G., & Szegedy, M. (2009). A new line of attack on the dichotomy conjecture. In M. Mitzenmacher (Ed.), *Proceedings of the 41st Annual ACM Symposium on Theory of Computing, STOC 2009, Bethesda, MD, USA, May 31 - June 2, 2009* (pp. 725–734). ACM.
128. Ladner, R.E. (1975). On the structure of polynomial time reducibility. *Journal of the ACM*, *22*(1), 155–171.
129. Larose, B., Loten, C., & Tardif, C. (2007). A characterisation of first-order constraint satisfaction problems. preprint arXiv:0707.2562.
130. Larose, B., & Zádori, L. (2006). Taylor terms, constraint satisfaction and the complexity of polynomial equations over finite algebras. *IJAC*, *16*(3), 563–582.
131. Lesaint, D., Azarmi, N., Laithwaite, R., & Walker, P. (1998). Engineering dynamic scheduler for Work Manager. *BT Technology Journal*, *16*, 16–29.
132. Lin, B. (2015). The parameterized complexity of k-biclique. In *Proceedings of the Twenty-Sixth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2015, San Diego, CA, USA, January 4-6, 2015* (pp. 605–615).
133. Luczak, T., & Nesetril, J. (2006). A probabilistic approach to the dichotomy problem. *SIAM Journal on Computing*, *36*(3), 835–843.
134. Marx, D. (2005). Parameterized complexity of constraint satisfaction problems. *Computational Complexity*, *14*(2), 153–183.
135. Marx, D. (2010). Approximating fractional hypertree width. *ACM Transactions on Algorithms*, *6*(2), 29:1–29:17.
136. Marx, D. (2011). Tractable structures for constraint satisfaction with truth tables. *Theory Computing System*, *48*(3), 444–464.
137. Marx, D. (2013). Tractable hypergraph properties for constraint satisfaction and conjunctive queries. *Journal of the ACM*, *60*(6), 42.
138. Mouelhi, A.E., Jégou, P., & Terrioux, C. (2014). *Different classes of graphs to represent microstructures for CSPs* (Vol. 8323, pp. 21–38).
139. Naanaa, W. (2013). Unifying and extending hybrid tractable classes of CSPs. *Journal of Experimental & Theoretical Artificial Intelligence*, *25*(4), 407–424.
140. Ordyniak, S., Paulusma, D., & Szeider, S. (2013). Satisfiability of acyclic and almost acyclic CNF formulas. *Theoretical Computer Science*, *481*, 85–99. doi:10.1016/j.tcs.2012.12.039.
141. Papadimitriou, C.H., & Yannakakis, M. (1997). On the complexity of database queries. In *Proceedings of the 16th ACM SIGACT-SIGMOD-SIGART symposium on Principles of database systems* (pp. 12–19). ACM.
142. Pearson, J.K., & Jeavons, P.G. (1997). A survey of tractable constraint satisfaction problems. Tech. Rep. CSD-TR-97-15, Royal Holloway, University of London.
143. Petke, J., & Jeavons, P. (2011). The order encoding: From tractable CSP to tractable SAT. In K.A. Sakallah, & L. Simon (Eds.), *SAT, Lecture Notes in Computer Science*, (Vol. 6695 pp. 371–372). Springer.
144. Pralet, C., & Verfaillie, G. (2012). Time-dependent simple temporal networks. In *CP* (pp. 608–623).
145. Purvis, L., & Jeavons, P. (1999). Constraint tractability theory and its application to the product development process for a constraint-based scheduler. In *Proceedings of 1st International Conference on The Practical Application of Constraint Technologies and Logic Programming - PACLP'99* (pp. 63–79). Practical Applications Company.
146. Robertson, N., & Seymour, P.D. (1995). Graph minors XIII. The disjoint paths problem. *Journal of Combinatorial Theory Series B*, *63*(1), 65–110.
147. Salamon, A.Z., & Jeavons, P.G. (2008). Perfect constraints are tractable. In *CP, Lecture Notes in Computer Science*, (Vol. 5202 pp. 524–528): Springer.
148. Samer, M., & Szeider, S. (2010). Constraint satisfaction with bounded treewidth revisited. *Journal of Computer and System Sciences*, *76*(2), 103–114.
149. Schaefer, T.J. (1978). The complexity of satisfiability problems. In *Proceedings 10th ACM Symposium on Theory of Computing, STOC'78* (pp. 216–226).
150. Scott, A.D., & Sorkin, G.B. (2009). Polynomial constraint satisfaction problems, graph bisection, and the Ising partition function. *ACM Transactions on Algorithms*, *5*(4), 45:1–45:27.
151. Thapper, J., & Živný, S. (2012). The power of linear programming for valued CSPs. In *53rd Annual IEEE Symposium on Foundations of Computer Science, FOCS 2012, New Brunswick, NJ, USA, October 20-23, 2012* (pp. 669–678). IEEE Computer Society.
152. Thapper, J., & Živný, S. (2013). The complexity of finite-valued CSPs. In D. Boneh, T. Roughgarden, & J. Feigenbaum (Eds.), *STOC* (pp. 695–704). ACM.
153. van Hoeve, W.J., & Katriel, I. (2006). Global constraints. In F. Rossi, P. van Beek, & T. Walsh (Eds.), *Handbook of Constraint Programming* (chap. 6, pp. 169–208). Elsevier.

154. Welsh, D. (1993). *Complexity: Knots, Colourings and Counting*. Cambridge University Press.
155. Werner, T. (2007). A linear programming approach to max-sum problem: A review. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *29*(7), 1165–1179.
156. Willard, R. (2010). Testing expressibility is hard. In D. Cohen (Ed.), *CP, Lecture Notes in Computer Science*, (Vol. 6308 pp. 9–23). Springer.
157. Williams, R., Gomes, C.P., & Selman, B. (2003). Backdoors to typical case complexity. In G. Gottlob, & T. Walsh (Eds.), *IJCAI* (pp. 1173–1178). Morgan Kaufmann.
158. Yorke-Smith, N., & Gervet, C. (2009). Certainty closure: Reliable constraint reasoning with incomplete or erroneous data. *ACM Transactions on Computational Logic*, *10*(1).
159. Živný, S. (2012). *The complexity of valued constraint satisfaction problems. Cognitive technologies*. Springer.
160. Zytnicki, M., Gaspin, C., & Schiex, T. (2008). DARN! A weighted constraint solver for RNA motif localization. *Constraints*, *13*(1–2), 91–109.