Structural Tractability of Propagated Constraints

Martin J. Green[1,⋆] and Christopher Jefferson[2,⋆⋆]

[1] Department of Computer Science, Royal Holloway, University of London, UK
[2] Computing Laboratory, Oxford University, UK

Abstract. Modern constraint solvers do not require constraints to be represented using any particular data structure. Instead, constraints are given as black boxes known as propagators. Propagators are given a list of current domains for variables and are allowed to prune values not consistent with these current domains.

Using propagation as the only primitive operation on constraints imposes restrictions on the operations that can be performed in polynomial time. In the extensional representation of constraints (so-called positive table constraints) join and project are primitive polynomial-time operations. This is not true for propagated constraints.

The question we pose in this paper is: If propagation is the only primitive operation, what are the structurally tractable classes of constraint programs (whose instances can be solved in polynomial time)?

We consider a hierarchy of propagators: arbitrary propagators, whose only ability is consistency checking; partial assignment membership propagators, which allow us to check partial assignments; and generalised arc consistency propagators, the strongest form of propagator.

In the first two cases, we answer the posed question by establishing dichotomies. In the case of generalised arc consistency propagators, we achieve a useful dichotomy in the restricted case of acyclic structures.

1 Background

Over recent years the research into the constraint satisfaction problem (CSP) has led to the development of many practically useful general purpose constraint solvers. Solving a *CSP instance*, or constraint program, involves assigning *values* to *variables* which are consistent with a set of restrictions, known as *constraints*. The identification of *tractable* classes of CSP instances (solvable in polynomial time) has become an important and fruitful area of research in the constraint community. Generally, tractable classes are described by limiting either the interaction of the constraints, called a *structural* restriction, or the type of constraint allowed, called a *relational* restriction.

Unfortunately, most of the identified classes [1,2] rely on a somewhat impractical *(positive) extensional* representation for the constraints: an explicit relation, or table, of allowed assignments. In this representation, *join* and *project*

⋆ This work was supported by EPSRC Grant EP/C525949/1.
⋆⋆ This work was supported by EPSRC Grant EP/D032636/1.

P.J. Stuckey (Ed.): CP 2008, LNCS 5202, pp. 372–386, 2008.

are primitive operations that can be performed in polynomial time. This notion of primitive operations allows large classes of CSP instances to be structurally tractable. For example, it is well-known that the class of CSP instances with *acyclic* structure is tractable in this representation [3]. *Structural decompositions*, such as *hypertrees* [1], aim to exploit this result, reducing CSP instances with bounded *width* to acyclic instances.

The advantage of the extensional representation is that it allows us to readily specify any constraint we wish on finite domains without issues of *expressibility*. Unfortunately, it also has a number of significant drawbacks in the context of constraint programming. Consider any constraint which forbids precisely one assignment on r variables each of domain size d. Such constraints can be used to represent clauses for SAT instances. However the (positive) extensional representation of this constraint requires listing all $d^r - 1$ allowed assignments.

This leads to the anomalous tractable class of all CSP instances which contain a *universal* (over all variables) constraint allowing all assignments. The extensional representation of this class is tractable: the size of the universal constraint dominates the number of solutions to the rest of the instance. However, it would be somewhat absurd to encode this universal constraint extensionally, and more succinct representations may affect the tractability of such classes.

Recently, researchers have started to determine those structural classes (derived from families of *hypergraphs* or *relational structures*) that remain tractable when we allow specific succinct representations to be used instead of the extensional representation. Both Houghton et al. [4] and Chen and Grohe [5] approach this task from above. Houghton et al. prove that a large class of structures, specified by two width parameters rather than just one, define a tractable structural class of CSP instances when we allow each constraint to be represented as the smaller of the allowed or forbidden assignments. Chen and Grohe specify two succinct representations: one based on a generalised form of DNF allowing larger domain sizes (called the GDNF representation), the other based on a form of decision diagrams (called the DD representation). In each case, they find a dichotomy between the tractable and intractable structural classes.

In this paper, we approach this task from below. Modern constraint solvers implement constraints as *oracles* known as *propagators*. Normally, only two primitive operations are allowed: consistency checking (membership) and propagation (pruning inconsistent values from domains). A propagator for a constraint is an algorithm. It takes as input a set of possible values for each variable and returns a subset of these values for each variable. Values may be removed from a set only when they cannot occur in any assignment to the constraint which is consistent with the given sets. In order to use a propagator as a complete representation of a constraint, we combine the two allowed operations into one, imposing a (natural) membership condition on the propagators we consider.

Propagators have proved to be very successful in practice. Efficient propagators have been designed for a large range of constraint types [6] and implementations are provided in many available solvers [7,8,9,10]. Unfortunately, it is often the case that propagators align poorly with existing tractability theory, simply

because they are able to provide very compact representations for many common constraint types. The standard assumption of polynomial-time join and project operators is no longer valid.

In this paper, we pose the question: If propagation is the only primitive operation, what are the structurally tractable classes of CSP instances?

We consider a hierarchy of propagators. In Sect. 3 we begin with arbitrary propagators, whose only ability is consistency checking (membership). In this representation, we find very small tractable structural classes. We then examine *partial assignment membership* (PAM) propagators in Sect. 4, which allow us to check partial assignments. In this representation we find that there are much larger, more useful tractable structural classes, defined by restricting the overlapping of constraints. Finally, in Sect. 5, we investigate *generalised arc consistency* propagators, the strongest form of propagator: here we find still larger tractable structural classes with less restrictive overlapping.

In the first two cases, we completely answer the posed question by establishing dichotomies. In the case of generalised arc consistency propagators, we achieve a useful dichotomy in the restricted case of acyclic structures.

2 Theoretical Introduction

In this section we introduce fundamental theory and definitions. We begin by considering a constraint satisfaction problem (CSP).

Definition 1. *A CSP instance is a triple $\langle V, D, C\rangle$, where V is a finite set of* ***variables****, D is a function which maps each element of V to a finite set of possible values, called its* ***domain****, and C is a finite set of* ***constraints****.*

Each constraint $c \in C$ is a pair, $\langle \sigma, \rho\rangle$, where σ is a sequence of variables from V, called the ***scope****. The length of σ, denoted $|\sigma|$, is called the* ***arity*** *of c. The* ***relation****, ρ, is a subset of $D(\sigma[1]) \times \cdots \times D(\sigma[r])$, where $r = |\sigma|$, and defines the allowed assignments for this constraint.*

A ***solution*** *to a CSP instance is a function which maps each variable to a value from its domain which is consistent with all of the constraints.*

A CSP is a decision problem. In this paper, we consider the *search problem* for a class of CSP instances. However, for any class of CSP instances for which we are allowed to add *constant constraints*, which define assignments to some subsets of the variables, these two notions coincide [11]: we find a solution (backtrack-free) in at most $|V| \times d$ steps (where V is the set of variables and d is the size of the largest domain) by adding unary constant constraints and calling the decision algorithm at each step. For all the classes we consider in this paper, we show that adding constant constraints does not affect complexity.

CSP instances are mathematical objects. They tell us what the instances are, but not how they should be *represented* for feeding to a constraint solver.

Definition 2. *The* ***extensional (Pos) representation*** *of a CSP instance $\langle V, D, C\rangle$ specifies the size of V, the size of the union of the domains, and then*

a list of constraints. Each constraint is represented extensionally as a list of the variables in the scope followed by a list of the allowed tuples. The ***size*** *of the extensional representation of the CSP instance* $\langle V, D, C\rangle$ *is*

$$\log|V| + \log d + \sum_{\langle\sigma,\rho\rangle\in C} \left(|\sigma|\left(|\rho|\log d + \log|V|\right)\right), \text{ where } d = \left|\bigcup_{v\in V} D(v)\right| .$$

It is this representation that has been assumed in most tractability results for classes of CSP instances [1,2]. However, this leads to anomalies since it imposes an artificial limitation on practitioners who use other representations such as forbidden assignments, equations, SAT clauses or *propagators*.

Definition 3. *Given a CSP instance* $\langle V, D, C\rangle$*, a* ***sub-domain*** *of a variable* $v \in V$ *is a subset of* $D(v)$*.*

Let $\langle\sigma, \rho\rangle \in C$ *be a constraint of arity* r*. A* ***list of sub-domains,*** S***, for*** σ *is a list containing a sub-domain for each element of* σ *in order. We denote by* $\times S$ *the product of the sets in* S*, so that* $\times S = S[1] \times \cdots \times S[r]$*.*

A ***propagator*** prop *for* $\langle\sigma, \rho\rangle$ *is a function which maps a list of sub-domains for* σ *to another list of sub-domains for* σ *satisfying the following conditions:*

1. $\times \text{prop}(S) \subseteq \times S$ ***(inclusion)***
2. $\times S \subseteq \times T \Rightarrow \times \text{prop}(S) \subseteq \times \text{prop}(T)$ ***(monotonicity)***
3. $a \in (\times S \cap \rho) \Rightarrow a \in \times \text{prop}(S)$ ***(validity)***
4. *If* $\times S = \{a\}$ *and* $a \notin \rho$*, then* $\times \text{prop}(S) = \emptyset$ ***(membership)***

The first three of these conditions are generally accepted requirements which ensure that applying a list of propagators results in a well-defined fixed point. The fourth condition is introduced to provide a method of checking if an assignment satisfies a constraint, allowing us to use a propagator as the complete representation of a constraint. In this paper, we will consider the class of all such propagators, Prop, as well as two restricted types: *partial assignment membership* (PAM) and *generalised arc consistency* (GAC).

Definition 4. *A* ***partial assignment membership (PAM) propagator*** *satisfies the condition that when all variables have either their complete sub-domain or a single value in their sub-domain, it will empty the sub-domains if there is no allowed assignment in the current sub-domains.*

A ***generalised arc consistency (GAC)*** *[12,13] propagator removes as many values as possible from the sub-domain of each variable.*

Propagators can naturally be intersected. As we will see in Proposition 2, it is not always possible to intersect (or join) propagators in polynomial time, even on the same scope. GAC propagators provide the greatest level of propagation that can be achieved for domain filtering: they can be seen as the intersection of all possible propagators for a constraint.

Example 1. The most famous GAC propagator is for the 'AllDiff' constraint, which imposes that a list of variables take different values [14].

Definition 5. *Let $\mathcal{R}$ be any class of propagators (such as Prop, PAM or GAC). An $\mathcal{R}$-**representation** of a constraint $\langle \sigma, \rho \rangle$ is a propagator for $\langle \sigma, \rho \rangle$ from $\mathcal{R}$.*

*An $\mathcal{R}$-**representation** of the CSP instance $\langle V = \{v_1, \ldots, v_n\}, D, C \rangle$ is a list of the indices of the variables in V, that is $1, \ldots, n$, followed by a list of the union of the domains (again, as indices), followed by a list of $\mathcal{R}$-representations of the constraints in C. The **size** of any $\mathcal{R}$-propagator representation of the CSP instance $\langle V, D, C \rangle$ is*

$$|V| \log |V| + d \log d + |C|, \text{ where } d = \left| \bigcup_{v \in V} D(v) \right| .$$

There are some differences between the size of a propagator representation of a CSP instance and its extensional representation (Definition 2). Firstly, we assume in a propagator representation that each variable is explicitly named (as the integers $1, \ldots, n$), as are the domain values (as the integers $1, \ldots, d$). This is because, unlike the extensional representation, we are not given the scopes and relations explicitly in the representation. We assume that constraints are represented in unit space (even their scopes). This is a very harsh property, but we make this assumption because we want to treat constraints as black boxes.

Definition 6. *Let $\mathcal{C}$ be a class of CSP instances and Θ be a method for representing CSP instances. We denote by $\Theta(\mathcal{C})$ the set of Θ representations of the instances in $\mathcal{C}$. We call the class of CSP instance representations $\Theta(\mathcal{C})$ **tractable** if there is a solution algorithm for $\Theta(\mathcal{C})$ that runs in time polynomial in the size of the representation: otherwise, we call the class **intractable**.*

In this paper, we are aiming for classes of propagator representations of CSP instances that are tractable. Strictly speaking, a class of propagator representations would be intractable if the propagators were not polynomial time. Instead, we assume that calling any propagator can be considered as one time step, which therefore limits any solution algorithm to a polynomial number of propagator calls in the size of the propagator representation (as given in Definition 5).

In order to prove intractability results in this paper, we will reduce intractable problems to propagator representations of CSP instances for which the propagators run in polynomial time. This will imply that the solution algorithm for these CSP instance representations must require exponential time (under a standard assumption from complexity theory). Using propagator representations allow us to drastically reduce the size of representation for some CSP instances.

Example 2. Recall Ex. 1. In any propagator representation, AllDiff is compact. However, we are unaware of any other representation that will succinctly represent an AllDiff constraint over r variables with $\geq r$ domain values. This includes Pos, forbidden tuples, GDNF [5] and DD [5].

Whilst it is possible to decompose an AllDiff constraint into a clique of binary inequalities, this decomposition does not allow for simple GAC propagation and is therefore less practical. Therefore, there is presently no method of identifying

practically applicable tractable classes of propagator representations that contain AllDiff constraints (of unbounded arity), despite the existence of such classes.

Definition 7. *A **multi-hypergraph** is a pair $H = \langle V, E\rangle$ where V is a set of vertices and E is a multi-set of subsets of V, called the hyperedges of H. We define a **list of structures**[1] to be a list of multi-hypergraphs. We say a list of structures is of **bounded arity** if there is a bound on the size of all hyperedges in all multi-hypergraphs in the list.*

*For any tuple t of arity r we define $\mathrm{set}(t) = \{t[1], \ldots, t[r]\}$. The **structure** of a CSP instance $P = \langle V, D, C\rangle$ is the multi-hypergraph $\langle V, E\rangle$ where E is the multi-set containing all of the scopes of the constraints in C, abstracted to be sets. That is, $E = \{\{\mathrm{set}(\sigma) \mid \langle \sigma, \rho\rangle \in C\}\}$, where $\{\{\ldots\}\}$ denotes a multiset.*

*A class of CSP instances is called **structural** if it is defined only by restricting the allowed structure of its instances to a given list of structures. For a list of structures $\mathcal{H}$ we denote by $\Psi(\mathcal{H})$ the structural class of CSP instances whose structures are all in $\mathcal{H}$.*

The definition of structure we give here differs from the two usual definitions in the literature. When constraints are represented extensionally, it is a cheap operation to join constraints occurring on the same scope. As such, instances may be treated as if they have only a single constraint on each scope, which leads to the *hypergraph structure* of a CSP instance. Many structural tractability results use this notion [1]. Our definition of structure as a multi-hypergraph does not alter complexity results for the extensional representation, since subsumed hyperedges (which includes multiple occurrences of the same hyperedge) are always removed as an initial step (called *normalisation*). Multi-hypergraphs are a natural generalisation of the hypergraph structure of CSP instances and are necessary for our complexity results. As Proposition 2 will show, there is no simple method of combining constraints defined using propagators — multiple constraints on the same scope must be handled with care.

Alternatively, some results on structural tractability use the notion of *relational structures*, which are more restrictive than multi-hypergraphs: they impose that a particular set of (ordered) hyperedges must take the same constraint relation. Use of relational structures is outside the scope of this paper. However, we can induce a relational structure from a multi-hypergraph by placing each occurrence of a hyperedge in a different relation of the relational structure.

An important result in the area of structural tractability for bounded arity CSP instances is reproduced in Theorem 1. Due to insufficient space, we will not define the terms used in this theorem, but instead refer the reader to [15].

Theorem 1 (Corollary 19 of Grohe [15]). *Given a list, $\mathcal{H}$, of (relational) structures of bounded arity, the class $\mathrm{Pos}(\Psi(\mathcal{H}))$ of extensionally represented CSP instances with structure in $\mathcal{H}$ is tractable if and only if the following requirement is satisfied: $\mathcal{H}$ has bounded treewidth modulo homomorphic equivalence.*

[1] We require a list of structures as there must be a mapping from the structures to the natural numbers, which arises naturally from a list. We also require this list of structures to be recursively enumerable.

Due to space limitations, we give the following result without proof.

Proposition 1. *Constraint solvers employing 2-way branching search cannot solve all tractable structural classes identified by Theorem 1 in polynomial time.*

The aim of this paper is to identify properties of families of structures which give tractability results for certain propagator representations. We rely heavily on Theorem 1 to produce complexity results by producing tractable CSP instances of bounded arity. Unfortunately, Proposition 1 shows that with current constraint solvers, employing only 2-way branching search, we cannot utilise these tractability results. We will be forced to modify the constraint solvers by providing new solution algorithms. However, the advantage of the results we give in this paper are that we need *only* provide new, simple, solution algorithms — we do not require *any* new representations of constraints to be implemented, instead utilising the current framework of propagated constraints.

For our hardness results we require some results from parameterised complexity theory. We limit ourselves to a very small explanation of the fundamental definitions here: for a more complete discussion we refer the reader to [16,17].

Definition 8. *A **parameterised problem** is a pair $(\mathcal{P}, \kappa)$ where $\mathcal{P}$ is a problem and κ is a function which maps each member of $\mathcal{P}$ to $\mathbb{N}$.*

Example 3. An instance of the parameterised problem p-CLIQUE is a pair (G, k) where G is a graph and k is an integer. The parameter of (G, k) is k. The question for (G, k) is: Does there exist a clique in G of size k?

Classes of CSP instances can be parameterised. In particular, the parameterisation we use throughout this paper is given in Ex. 4.

Example 4. Given a (possibly infinite) list of multi-hypergraphs $\mathcal{H}$, any class of CSP instances whose structures are all in $\mathcal{H}$ can be parameterised by mapping each CSP instance to the index of the first occurrence of its structure in $\mathcal{H}$.

Definition 9. *A parameterised problem $(\mathcal{P}, \kappa)$ is **fixed-parameter tractable (FPT)** if there is a (computable) function $f : \mathbb{N} \to \mathbb{N}$ and an algorithm which decides if the instance $p \in \mathcal{P}$ has a solution in time*

$$f(\kappa(p)).|p|^{O(1)} \ .$$

Definition 10. *An **FPT-reduction** from a parameterised problem $(\mathcal{P}, \kappa)$ to another parameterised problem $(\mathcal{P}', \kappa')$ is a mapping R from instances of $\mathcal{P}$ to instances of $\mathcal{P}'$ such that:*

- *$p \in \mathcal{P}$ has a solution if and only if $R(p)$ has a solution.*
- *There is a (computable) function $m : \mathbb{N} \to \mathbb{N}$ and an algorithm that, given $p \in \mathcal{P}$, computes $R(p)$ in time $m(\kappa(p)).|p|^{O(1)}$.*
- *There is a (computable) function $g : \mathbb{N} \to \mathbb{N}$ such that for all $p \in \mathcal{P}$ we have that $\kappa'(R(p)) \leq g(\kappa(p))$.*

Parameterised complexity is a large and complex field of research. Instead of the traditional distinction between complexity classes P and NP, there is a hierarchy FPT $\subseteq$ W[1] $\subseteq$ W[2] $\subseteq \ldots$ of complexity classes. Downey and Fellows [18] define this hierarchy and conjecture that it is strict. In this paper, we are only interested in W[1] and make the standard assumption that FPT $\neq$ W[1] throughout (including previously in Theorem 1). For our proofs, we will make extensive use of Theorem 2.

Theorem 2 (Corollary 3.2 of [19]). *The parameterised problem p-CLIQUE is W[1]-complete under FPT-reductions.*

3 Arbitrary Propagators

In this section, we will consider the question of tractable structural classes of CSP instances where each constraint is represented by an arbitrary propagator. Arbitrary propagators need only check if a complete assignment satisfies the constraint. As such, constant constraints can be integrated into a call to a propagator by intersecting them with the given sub-domains.

Example 5. Consider any CSP instance of arity bounded by k. We can convert any Prop representation of this instance into an extensional representation by testing the membership of all d^k possible assignments for each constraint, where d is the size of the union of the domains. Since the union of the domains is listed as part of the input and k is fixed, this is polynomial time. Alternatively, building a GAC propagator (which is the strongest possible propagator type) for an extensionally represented constraint is straightforward. It is equivalent to enforcing GAC on an extensionally represented constraint, which may be done in polynomial time [13], as can listing the variables and (used) domain values. Hence, for bounded arity, propagators are equivalent to extensionally represented constraints.

Lemma 1. *Let $\mathcal{H}$ be a list of structures of bounded arity. The class* Prop($\Psi(\mathcal{H})$) *is tractable if and only if $\mathcal{H}$ satisfies the requirement of Theorem 1.*

Proof. Ex. 5 shows that for bounded arity there exist polynomial-time reductions between Prop($\Psi(\mathcal{H})$) and Pos($\Psi(\mathcal{H})$), so Theorem 1 applies to Prop($\Psi(\mathcal{H})$). □

Unfortunately, as our next result shows, *any* list of structures of *unbounded* arity generates a class of CSP instances that is intractable when represented using arbitrary propagators.

Lemma 2. *Let $\mathcal{H}$ be any list of structures of unbounded arity. The search problem for the class* Prop($\Psi(\mathcal{H})$) *is W[1]-hard, and therefore intractable.*

Proof. We give an FPT-reduction from p-CLIQUE to Prop($\Psi(\mathcal{H})$). Given any instance (G, k) of p-CLIQUE, the parameter k is mapped to the index of the first element H of $\mathcal{H}$ which includes a hyperedge of arity at least k. We construct a Prop representation of a CSP instance with structure H as follows.

The domain of all variables is the vertices of G. For a single occurrence of a hyperedge of arity at least k, order this hyperedge and generate the Prop representation of the constraint with this scope such that the first k variables in its scope satisfy p-CLIQUE for (G, k). All other constraints are true for all assignments. The size of this mapping is the size of (G, k), plus a constant (polynomial in the size of H) to express the other domains and constraints, and is therefore a FPT-reduction. By Theorem 2, this proves the result. □

Lemmas 1 and 2 provide the following dichotomy for arbitrary propagator representations of structural classes of CSP instances.

Theorem 3. *Let $\mathcal{H}$ be any list of structures. The class* $\text{Prop}(\Psi(\mathcal{H}))$ *is tractable if and only if $\mathcal{H}$ is of bounded arity and satisfies the requirement of Theorem 1.*

4 Partial Assignment Membership Propagators

Partial assignment membership (PAM) propagators exist for many constraint types. For instance, most arithmetic constraints have polynomial-time PAM propagators but yet have no polynomial-time GAC propagator. PAM propagators provide a useful level of propagation. For example, all propagators provided by the Minion CSP solver [10] are PAM propagators.

One feature of the (positive) extensional representation of constraints is that it is possible to join two constraints in polynomial time with respect to the size of their representation. This does not in general hold for propagators.

Proposition 2. *Given two lists of constraints A and B with polynomial time PAM propagators (or GAC propagators), checking if the constraints $A[i] \wedge B[i]$ have a consistent assignment is, in general, an NP-complete problem.*

Proof. Consider the constraints $\sum_{i=1}^{n} a_i v_i \leq 0$ and $\sum_{i=1}^{n} a_i v_i \geq 0$ defined by integer constants a_i and variables v_i with domain $\{0, 1\}$. Both of these families of constraint have polynomial time GAC propagators [20], which are of course PAM propagators. Given a set of integers $S = \{s_1, \ldots, s_n\}$, the problem of finding a subset of S which sums to exactly 0 is a well-known NP-complete problem [21]. However, this problem is exactly equivalent to the intersection of the two constraints above by setting $a_i = s_i$ for each i. □

We observe that a PAM propagator exactly captures the notion of applying constant constraints to some subset of the variables. As such, we can freely integrate constant constraints into calls to PAM propagators.

Many more families of structures derive tractable classes for PAM propagators than with general propagators. For example, the class of CSP instances which contain only a single constraint is tractable for the PAM propagator representation. However, there are still simple classes which remain intractable.

Definition 11. *Given any constraint, c, the* ***Free-****c constraint is defined as follows: Free-c is a constraint over the same (number of) variables, each containing*

one extra domain value, denoted Free. An assignment to the scope of Free-c is true either if any variable is assigned Free or the assignment satisfies c.

For a set of constraints, C, the set of constraints $\{$*Free-c* $\mid c \in C\}$ *is denoted Free-C. For a problem,* $\mathcal{P}$*, and a set of constraints,* $C(\mathcal{P})$*, that model the instances of* $\mathcal{P}$*, we denote by Free-*$\mathcal{P}$ *the set of constraints Free-*$C(\mathcal{P})$*.*

PAM propagators for *Free-c* constraints only ever have to check if complete assignments satisfy c, because given a list of sub-domains with any unassigned variables, *Free-c* is satisfied by assigning any unassigned variable *Free*.

We assume the following model of p-CLIQUE as a set of constraints.

Definition 12. *Let* (G, k) *be an instance of p-CLIQUE, where* $G = \langle V, E\rangle$*. We generate a constraint on* k *variables each with domain* V *and satisfying assignments defined by cliques of size* k *in* G*.*

By imposing *Free*-p-CLIQUE, along with unary constraints of the type "variable is not assigned *Free*", we derive classes of CSP instances which model p-CLIQUE and are therefore W[1]-hard for the PAM representation.

Definition 13. *Let* $H = \langle V, E\rangle$ *be a multi-hypergraph. A vertex* $v \in V$ *is called* ***isolated*** *if it exists in (at most) one occurrence of a hyperedge in* E*.*

Lemma 3. *Given a list of structures* $\mathcal{H}$*, generate a new list of structures* $\mathcal{H}'$ *by removing from members of* $\mathcal{H}$ *all isolated vertices. The search problem for* $\mathrm{PAM}(\Psi(\mathcal{H}))$ *is W[1]-hard if and only if it is W[1]-hard for* $\mathrm{PAM}(\Psi(\mathcal{H}'))$*.*

Proof. Given a PAM propagator, prop, for a constraint c, over a list of variables $\langle x_1, \ldots, x_r\rangle$ each with domain D, consider the projection, c_k, of c onto the variables $\langle x_1, \ldots, x_k\rangle$, for $k \leq r$. A PAM propagator, prop_k, for c_k can be constructed from prop using at most $|D| \times (r - k)$ extra time: take the partial assignment to c_k and attach the complete sub-domain D for x_{k+1} to x_r.

Further, we observe that projecting out isolated variables preserves solutions. Therefore, we can solve CSP instances whose structure is $\mathcal{H}[i]$, for some i, by projecting down to $\mathcal{H}'[i]$ using the above transformation. Given a CSP instance whose structure is $\mathcal{H}'[i]$, we extend this to an instance on $\mathcal{H}[i]$ by giving all the extra variables a single domain value and have the constraints ignore the assignment to these variables. This gives FPT-reductions between $\mathrm{PAM}(\Psi(\mathcal{H}))$ and $\mathrm{PAM}(\Psi(\mathcal{H}'))$, proving the result. □

We may now prove the main result of this section: a complete dichotomy of the tractable structural classes of CSP instances for the PAM representation.

Theorem 4. *Given any list of structures* $\mathcal{H}$*, generate the list of structures* $\mathcal{H}'$ *by removing from members of* $\mathcal{H}$ *all isolated vertices. Then* $\mathrm{PAM}(\Psi(\mathcal{H}))$ *is tractable if and only if* $\mathcal{H}'$ *is of bounded arity and satisfies the requirement of Theorem 1.*

Proof. By Lemma 3 we know that the search problem for $\mathrm{PAM}(\Psi(\mathcal{H}))$ is W[1]-hard if and only if it is for $\mathrm{PAM}(\Psi(\mathcal{H}'))$. We therefore consider only $\mathrm{PAM}(\Psi(\mathcal{H}'))$.

If $\mathcal{H}'$ is not of bounded arity then we map p-CLIQUE to PAM($\Psi(\mathcal{H}')$) (using a similar mapping to that used in the proof of Lemma 2). Given any instance (G, k) of p-CLIQUE, the parameter k is mapped to the index of the first element H of $\mathcal{H}'$ which includes a hyperedge of arity at least k. Then, we construct a PAM representation of a CSP instance with structure H as follows.

For a single occurrence of a hyperedge of arity at least k, order this hyperedge and generate the PAM representation of the constraint with this scope such that the first k variables in its scope impose the *Free*-p-CLIQUE constraint for (G, k), extending the constraint to ignore the assignment to the other variables. The domain for these k variables is the vertices of G together with the value *Free*. By definition of $\mathcal{H}'$ no variable on which we imposed *Free*-p-CLIQUE can be isolated, so for each constraint meeting the first k variables of the *Free*-p-CLIQUE constraint we impose that no variable in the constraint is assigned *Free*. All other variables in the CSP instance are given a single value (which is not *Free*) in their domains, and all other constraints allow all assignments. This CSP instance has a solution if and only if G has a clique of size k. The size of this mapping is the size of (G, k), plus a constant (polynomial in the size of H) to express the other domains and constraints, and therefore is a FPT-reduction.

Otherwise, $\mathcal{H}'$ is of bounded arity. We use Ex. 5 to map in polynomial time between PAM($\Psi(\mathcal{H}')$) and Pos($\Psi(\mathcal{H}')$) and Lemma 3 completes the result. □

5 Generalised Arc Consistency Propagators

Generalised arc consistency (GAC) propagators provide the highest level of domain filtering. Many famous GAC propagators have been found for highly practical *global constraints*. For an overview, including the following constraint types, see, for example, [6].

- AllDiff($\langle v_1, \ldots, v_r \rangle$) : All of the v_i take distinct values.
- Element($\langle M_1, \ldots, M_n \rangle, x, y$) : $M_x = y$.
- GCC$\langle v_1, \ldots, v_r \rangle, \langle \langle l_1, u_1 \rangle, \ldots, \langle l_r, u_r \rangle \rangle$) : For constants $\langle l_i, u_i \rangle$, there are between l_i and u_i occurrences of i in $\langle v_1, \ldots, v_r \rangle$.

In this paper, we have not considered the cost of applying a propagator for a constraint, instead treating them as oracles taking constant time. That said, the tractability results generated in this paper would be less useful if low-cost propagation was not possible for many common constraint types. While there exist many constraint types for which GAC propagation is itself NP-hard, noticeably those involving arithmetic (such as bin packing), many constraints do have simple GAC propagators, including the previous examples. As with PAM propagators, calls to GAC propagators can integrate constant constraints without affecting complexity. In fact, they can integrate any unary constraints on their variables. This power of GAC propagation allows for significantly more tractable structural classes than with either arbitrary or PAM propagators.

In the special case of Boolean domains, Proposition 3 shows that PAM and GAC propagators have comparable complexity. However, for non-Boolean domains this is not the case, as our subsequent results show.

Proposition 3. *For a constraint c of arity r and Boolean domain, a GAC propagator for c can be constructed using $2r$ calls to a PAM propagator for c.*

Proof. We observe that non-empty sub-domains of a Boolean domain must either be complete or contain only a single value. This satisfies the condition for PAM propagators. GAC propagators must remove all values which cannot be extended to a solution. We can do this in at most $2r$ calls to the PAM propagator for c. □

Theorem 4 says that the structures which derive tractable classes for the PAM representation are essentially bounded arity: there must be a bounded number of non-isolated vertices in each hyperedge. However, for the GAC representation, this restriction is lifted. The following example seems to be known in the constraints community, though the authors failed to find a reference in the literature.

Example 6. We call a multi-hypergraph $\langle V, E\rangle$ a **tree with single vertex overlap** if the following algorithm creates an empty multi-hypergraph: Choose any hyperedge $e \in E$ for which $|e \cap \bigcup_{f\in E\setminus\{e\}} f| \leq 1$, remove it from E, and repeat whilst some hyperedge has been removed.

Let $\mathcal{T}$ be any list of trees with single vertex overlaps. The class GAC($\Psi(\mathcal{T})$) is tractable: GAC propagation decides. This result is a straightforward extension of (binary) tree-structured instances being decided by *arc consistency*.

In this section, we generalise this result to provide a dichotomy for lists of *acyclic* multi-hypergraphs.

Definition 14. *Given any constraint, c, with scope $\langle v_1, \ldots, v_r\rangle$, the **Pair-**$c$ constraint is defined as follows: Pair-c is a constraint over $\langle x_1, \ldots, x_r, y_1, \ldots, y_r\rangle$, where for all i, x_i and y_i have the same domain as v_i. Pair-c is true either if $x_i \neq y_i$, for any i, or the assignment to $\langle x_1, \ldots, x_r\rangle$ satisfies c.*

For a set of constraints, C, the set of constraints $\{$Pair-$c \mid c \in C\}$ is denoted Pair-C. For a problem, $\mathcal{P}$, and a set of constraints, $C(\mathcal{P})$, that model the instances of $\mathcal{P}$, we denote by Pair-$\mathcal{P}$ the set of constraints Pair-$C(\mathcal{P})$.

If assignments to a constraint c can be checked in polynomial time, then it is straightforward to show that *Pair-c* has a simple polynomial-time GAC propagator. The class of *Pair*-p-CLIQUE constraints will be used to construct intractable structural classes of CSP instances for the GAC representation.

Proposition 4. *The search problem for the following class of CSP instances, parameterised by structure, is W[1]-hard for the GAC representation:*

- *Variables labelled $x_1, \ldots, x_n$ and $y_1, \ldots, y_n$.*
- *An arbitrary Pair-p-CLIQUE constraint on scope $\langle x_1, \ldots, x_n, y_1, \ldots, y_n\rangle$.*
- *The constraint $x_i = y_i$ for $i = 1, \ldots, n$.*

Proof. There is a simple FPT-reduction from p-CLIQUE to this class. □

Proposition 4 shows one important feature of the tractability for the GAC representation is the *size* of overlaps with other constraints, rather than simply the number of overlaps or the total number of variables contained in overlaps.

Definition 15. *Given a multi-hypergraph H and a hyperedge e in H, a* **non-trivial overlap** *of e in H is a minimally sized set of vertices which, if removed from H, leaves every other hyperedge (including other occurrences of the same hyperedge) meeting e at no more than one vertex.*

Definition 16. *A multi-hypergraph H is* ***acyclic*** *if the following algorithm results in an empty multi-hypergraph: Remove all isolated vertices from (occurrences of) hyperedges, then remove all (occurrences of) hyperedges contained in some other hyperedge, and repeat whilst some change has been made.*

A ***join tree*** *of a multi-hypergraph $\langle V, E\rangle$ is a tree $J = \langle E, F\rangle$ which satisfies the condition: each $v \in V$ induces a connected subtree of J. Not all multi-hypergraphs have join trees. A multi-hypergraph is acyclic if it has a join tree.*

For any list, $\mathcal{H}$, of acyclic structures, it is well-known that $\mathrm{Pos}(\Psi(\mathcal{H}))$ is tractable [3]. In the extensional representation, we solve acyclic instances by identifying a join tree using the algorithm of Definition 16: the parent of a hyperedge is the hyperedge that subsumes it. We apply *pairwise consistency* along the join tree: two constraints are pairwise consistent if an assignment to their common variables can be extended to an assignment over their join. For the extensional representation, pairwise consistency can be achieved in polynomial time [22]. If each constraint still has allowed assignments, there is a solution, and we can find it in a backtrack-free way. This solution technique does not work for the GAC representation: for example, the intractable class of instances defined in Proposition 4 has only acyclic structures.

We are now in a position to provide the main result of this section: a dichotomy for the GAC representation in the restricted case of acyclic structures.

Theorem 5. *Let $\mathcal{H}$ be any list of acyclic structures.* $\mathrm{GAC}(\Psi(\mathcal{H}))$ *is tractable if and only if $\mathcal{H}$ has bounded non-trivial overlap.*

Proof (sketch). If $\mathcal{H}$ does not have bounded non-trivial overlap then, for every l, the following property holds: there exists a structure H in $\mathcal{H}$ for which there is an occurrence of a hyperedge e containing $2l$ distinct vertices which may be partitioned into l pairs such that each pair exists in the intersection of e with an occurrence of some other hyperedge (possibly another occurrence of e).

We give an FPT-reduction from p-CLIQUE to $\mathrm{GAC}(\Psi(\mathcal{H}))$. For any instance (G, k) of p-CLIQUE we choose the first structure H in $\mathcal{H}$ containing a hyperedge e with k pairs of vertices (as defined above). We generate an instance in $\mathrm{GAC}(\Psi(\mathcal{H}))$ as follows. We impose a *Pair*-p-CLIQUE constraint for (G, k) on the k pairs of variables of e. Other constraints allow all assignments except for enforcing that each of the selected pairs of variables are equal. This provides a FPT-reduction from p-CLIQUE to $\mathrm{GAC}(\Psi(\mathcal{H}))$.

Otherwise, $\mathcal{H}$ has bounded non-trivial overlap. Let k be this bound. We solve instances from $\mathrm{GAC}(\Psi(\mathcal{H}))$ in polynomial time in the following way. Identify a join tree, J, for the structure of the instance, labelling each node with the constraint on this (ordered) hyperedge. Then, we identify a non-trivial overlap for each hyperedge (node of J). We determine the set of allowed assignments to the non-trivial overlaps: at most d^k assignments for maximum domain size d.

Finally, we show that pairwise consistency can be achieved using GAC on these allowed assignments to the non-trivial overlaps. Consider any adjacent pair of nodes in J. Apart from their non-trivial overlaps, they must meet in at most one additional variable. We remove an assignment to a non-trivial overlap if it cannot extend to one of the two labels (constraints) of these hyperedges. This can be tested using the GAC propagators for these two constraints, since there is at most one additional variable in their overlap that is not assigned a constant.

Hence, we can use GAC to achieve pairwise consistency along the join tree, in polynomial time, just as with a positive representation. If at any point we empty a set of assignments to a non-trivial overlap, there is no solution. Otherwise, after pairwise consistency has been achieved, there is a solution which can be found in a backtrack-free way, testing unary extensions with GAC propagation. □

Theorem 5 includes the tractable class whose structures are trees with single vertex overlaps (Ex. 6): these structures have *empty* non-trivial overlaps. Unfortunately, we do not yet know what can be said about bounded width structural decompositions, such as hypertrees [1]. To finish, we observe a simple extension to the tractable structural classes of Theorem 5 for non-acyclic structures.

Corollary 1. *Let k be a constant and $\mathcal{H}$ be any list of structures, with bounded non-trivial overlap, that can be made acyclic by removal of at most k variables. The class* $\mathrm{GAC}(\Psi(\mathcal{H}))$ *is tractable.*

6 Conclusion

In this paper we have made a major step towards making the tractability of structural classes of CSP instances applicable to modern constraint solvers. We have shown the unfortunately weak tractability classes for general propagators and the much stronger tractability classes for partial assignment membership propagators. We have given a dichotomy for generalised arc consistency propagators in the restricted case of acyclic structures. While we have not yet provided a complete dichotomy for generalised arc consistency propagators, we have shown the existence of large classes, both tractable and intractable, providing immediately useful results and guidance towards where a future dichotomy may exist.

Acknowledgements. The authors are extremely grateful to Peter Jeavons and David Cohen for many insightful discussions about the contents of this paper.

References

1. Gottlob, G., Leone, N., Scarcello, F.: A comparison of structural CSP decomposition methods. Artif. Intell. 124(2), 243–282 (2000)
2. Cohen, D., Jeavons, P.: The complexity of constraint languages. In: Rossi, F., van Beek, P., Walsh, T. (eds.) Handbook of Constraint Programming. Elsevier, Amsterdam (2006)

3. Beeri, C., Fagin, R., Maier, D., Yannakakis, M.: On the desirability of acyclic database schemes. Journal of the ACM 30, 479–513 (1983)
4. Houghton, C., Cohen, D.A., Green, M.J.: The effect of constraint representation on structural tractability. In: Benhamou, F. (ed.) CP 2006. LNCS, vol. 4204, pp. 726–730. Springer, Heidelberg (2006)
5. Chen, H., Grohe, M.: Constraint satisfaction with succinctly specified relations. In: Creignou, N., Kolaitis, P., Vollmer, H. (eds.) Complexity of Constraints. Dagstuhl Seminar Proceedings, Dagstuhl, Germany, Internationales Begegnungs- und Forschungszentrum für Informatik (IBFI), vol. 06401. Schloss Dagstuhl, Germany, (2006)
6. van Hoeve, W.J., Katriel, I.: Global constraints. In: Rossi, F., van Beek, P., Walsh, T. (eds.) Handbook of Constraint Programming. Elsevier, Amsterdam (2006)
7. Cheadle, A., Harvery, W., Sadler, A.J., Schimpf, J., Shen, K., Wallace, M.: ECLiPSe: An introduction. Technical Report IC-Parc-03-1, Imperial College, London (2003)
8. The Gecode team: Generic constraint development environment, http://www.gecode.org
9. ILOG S.A.: ILOG Solver 5.3 Reference and User Manual (2002)
10. Gent, I.P., Jefferson, C., Miguel, I.: Minion: A fast scalable constraint solver. In: Brewka, G., Coradeschi, S., Perini, A., Traverso, P. (eds.) ECAI, pp. 98–102. IOS Press, Amsterdam (2006)
11. Cohen, D.: Tractable decision for a constraint language implies tractable search. Constraints 9(3), 219–229 (2004)
12. Mackworth, A.: Consistency in networks of relations. Artificial Intelligence 8, 99–118 (1977)
13. Bessière, C., Régin, J.C.: Arc consistency for general constraint networks: preliminary results. In: Proceedings of IJCAI 1997, Nagoya, Japan, pp. 398–404 (1997)
14. Régin, J.C.: A filtering algorithm for constraints of difference in CSPs. In: AAAI 1994: Proceedings of the twelfth national conference on Artificial intelligence, vol. 1, pp. 362–367. American Association for Artificial Intelligence, Menlo Park (1994)
15. Grohe, M.: The complexity of homomorphism and constraint satisfaction problems seen from the other side. J. ACM 54(1), 1 (2007)
16. Downey, R.G., Fellows, M.R.: Parameterized Complexity. Springer, Heidelberg (1999)
17. Flum, J., Grohe, M.: Parameterized Complexity Theory (Texts in Theoretical Computer Science. An EATCS Series). Springer, New York (2006)
18. Downey, R.G., Fellows, M.R.: Fixed-parameter tractability and completeness I: Basic results. SIAM J. Comput. 24(4), 873–921 (1995)
19. Downey, R.G., Fellows, M.R.: Fixed-parameter tractability and completeness II: On completeness for $W[1]$. Theoretical Computer Science 141(1–2), 109–131 (1995)
20. Schulte, C., Stuckey, P.J.: When do bounds and domain propagation lead to the same search space? ACM Trans. Program. Lang. Syst. 27(3), 388–425 (2005)
21. Rivest, R.L., Leiserson, C.E.: Introduction to Algorithms. McGraw-Hill, Inc., New York (1990)
22. Janssen, P., Jegou, P., Nouguier, B., Vilarem, M.: A filtering process for general constraint satisfaction problems: achieving pair-wise consistency using an associated binary representation. In: Proceedings of the IEEE Workshop on Tools for Artificial Intelligence, pp. 420–427 (1989)