Available online at www.sciencedirect.com

ScienceDirect

Journal of Computer and System Sciences 74 (2008) 721–743

JOURNAL OF COMPUTER AND SYSTEM SCIENCES

www.elsevier.com/locate/jcss

A unified theory of structural tractability for constraint satisfaction problems

David Cohen [a,*], Peter Jeavons [b], Marc Gyssens [c]

[a] *Department of Computer Science, Royal Holloway, University of London, Egham, Surrey, UK*
[b] *Oxford University Computing Laboratory, Wolfson Building, Parks Road, Oxford, UK*
[c] *Department WNI, Hasselt University and Transnational University of Limburg, Agoralaan, Building D, B-3590 Diepenbeek, Belgium*

Received 23 March 2007; received in revised form 14 August 2007

Available online 28 August 2007

Abstract

In this paper we derive a generic form of structural decomposition for the constraint satisfaction problem, which we call a guarded decomposition. We show that many existing decomposition methods can be characterised in terms of finding guarded decompositions satisfying certain specified additional conditions.

Using the guarded decomposition framework we are also able to define a new form of decomposition, which we call a spread-cut. We show that the discovery of width-k spread-cut decompositions is tractable for each k, and that spread-cut decompositions strongly generalise many existing decomposition methods. Finally we exhibit a family of hypergraphs H_n, for $n = 1, 2, 3 \ldots$, where the minimum width of any hypertree decomposition of each H_n is $3n$, but the width of the best spread-cut decomposition is only $2n + 1$.

Keywords: Constraints; Complexity; Structural decomposition; Hypertree

1. Introduction

A constraint satisfaction problem consists of a set of variables that must be assigned values subject to certain constraints. These constraints restrict the simultaneous assignments to certain specified subsets of the variables. Many real-world problems can be represented very naturally in this framework [6,17,18].

Similar problems arise in the field of relational databases, where they are referred to as conjunctive query evaluation problems [15]. Many of the techniques developed in this paper can also be applied directly to the conjunctive query evaluation problem, but we shall not pursue this application here.

The decision problem for the general constraint satisfaction problem is NP-hard [16]. This motivates the search for more restricted subproblems which are *tractable*, that is, solvable in polynomial-time.

* Corresponding author.
E-mail addresses: d.cohen@rhul.ac.uk (D. Cohen), peter.jeavons@comlab.ox.ac.uk (P. Jeavons), marc.gyssens@uhasselt.be (M. Gyssens).

0022-0000/$ – see front matter
doi:10.1016/j.jcss.2007.08.001

This paper considers subproblems of the constraint satisfaction problem which can be shown to be tractable using so-called *structural* methods or *decomposition* methods. These methods rely exclusively on using the structural properties of instances, in other words the way in which the constraints overlap each other.

A variety of such decomposition methods have been developed and applied in both the constraint satisfaction community and the database community. Examples include methods based on the use of treewidth [9], cycle cutsets [7], tree-clustering [8], hinges [14], cycle hypercutsets and hypertrees [10,11]. All of these methods rely on reducing a given problem instance to an equivalent instance with a simpler structure, which can then be solved efficiently. The maximum number of edges of the original structure which are combined to form a single element of this equivalent simpler structure is called the *width* of the decomposition.

The main contribution of this paper is to present a generic, abstract form of decomposition, which we call a *guarded decomposition*, together with a generic algorithm to compute guarded decompositions. We show that many of the earlier decomposition methods can be viewed as special cases of guarded decomposition, each characterised by some simple additional conditions. In this way we unify much of the existing theory of structural decompositions. Furthermore, by presenting simple sufficient conditions which ensure that a guarded decomposition is tractable, we create the possibility of a systematic search for new tractable structural classes.

One existing decomposition method, based on the use of hypertrees [10–12], is of particular importance as it has been shown to be strictly more general than many other decomposition methods. It has also been shown that it is tractable to discover hypertree decompositions of width at most k, for each fixed choice of k. Our work draws heavily on the ideas introduced by Gottlob, Leone and Scarcello in their work on hypertrees [10–12], including the notion of a guarded block, the recursive decomposition strategy, and the tools for comparison of different methods.

Another contribution of this work is that we are able to use the general framework and algorithm presented here to define a new decomposition method, which we call *spread-cut* decomposition. We show that spread-cut decomposition generalises many of the methods previously shown to be generalised by hypertree decomposition. We also show that it is tractable to discover spread-cut decompositions of width at most k, for each fixed choice of k. Finally we exhibit a family of hypergraphs H_n, for $n = 1, 2, 3 \ldots$, where the width of the best hypertree decomposition of each H_n is $3n$, but the width of the best spread-cut decomposition is only $2n + 1$.

2. Constraint satisfaction problems and hypergraphs

A constraint satisfaction problem (CSP) consists of a collection of variables that must be assigned values from some given domain. The values that the variables can take are not independent; they are restricted by *constraints*. Each constraint restricts the allowed simultaneous assignments to a certain subset of the variables.

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

- V is a finite set of *variables*;
- D is a set called the *domain* of P;
- C is a set of *constraints*. Each constraint $c \in C$ is a pair $c = \langle \chi, \rho \rangle$ where $\chi \subseteq V$ is a set of variables, called the *scope* of c, and $\rho \subseteq D^{\chi}$ is a set of functions from the scope of c to the domain of P, called the *relation* of c.

A *solution* to the CSP instance $P = \langle V, D, C \rangle$ is a function from V to D whose restriction[1] to the scope of any constraint $c \in C$ is one of the functions in the relation of c.

Example 2.2. Consider the CSP instance $P_{AG} = \langle V, D, C \rangle$ where $V = \{1, 2, \ldots, 10\}$, $D = \{0, 1\}$, and $C = \{c_1, c_2, \ldots, c_8\}$.

[1] To simplify the presentation we assume, throughout this paper, that every variable of a CSP instance is constrained; that is, every variable occurs in the scope of some constraint.

This instance has ten variables which must each be assigned the value 0 or 1, subject to 8 constraints. The constraints in C are defined as follows:

$$c_1 = \langle \{1,2\}, D^{\{1,2\}} \rangle, \qquad c_2 = \langle \{2,3,9\}, D^{\{2,3,9\}} \rangle,$$
$$c_3 = \langle \{3,4,10\}, D^{\{3,4,10\}} \rangle, \qquad c_4 = \langle \{4,5\}, D^{\{4,5\}} \rangle,$$
$$c_5 = \langle \{5,6,9\}, D^{\{5,6,9\}} \rangle, \qquad c_6 = \langle \{6,7,10\}, D^{\{6,7,10\}} \rangle,$$
$$c_7 = \langle \{7,8,9\}, D^{\{7,8,9\}} \rangle,$$
$$c_8 = \langle \{1,8,10\}, \{ f \in D^{\{1,8,10\}} \mid f(1) = f(8) = 0 \Rightarrow f(10) = 1 \} \rangle.$$

Note that each constraint except c_8 allows *every* assignment to the variables of its scope. The constraint c_8 does not allow the three variables of its scope all to take the value 0 simultaneously, but does allow all other assignments.

A straightforward calculation shows that this instance has exactly $7 * 2^7 = 896$ solutions.

In this paper we study the complexity of solving certain classes of CSP instances. For this reason we need to specify precisely what we mean by the *size* of a CSP instance. We adopt the usual convention for theoretical work: we define the size of an instance to be the sum of the sizes of the *explicit* constraints.

Definition 2.3. Given an instance $\langle V, D, C \rangle$ we represent each constraint scope as a list of variables from V, and each constraint relation as a list of tuples over D. The *size* of the instance $\langle V, D, C \rangle$ is then

$$\sum_{\langle \chi, \rho \rangle \in C} |\chi| \big(\log |V| + |\rho| \log |D| \big).$$

In Example 2.2 we used a shorthand notation to represent the constraint relations. Here and elsewhere this does not imply that this is a legal way to formally express a relation for an instance of the CSP in a succinct form (and hence one cannot have exponentially large relations as part of a small instance).

In order to study the *structural* properties of CSP instances, that is, the way in which the constraint scopes overlap each other, we need the standard notion of a *hypergraph*.

Definition 2.4. A *hypergraph* is a pair $H = \langle V, E \rangle$, where V is an arbitrary set, called the *vertices* of H, and E is a set of subsets of V, called the *hyperedges* of H.

Definition 2.5. For any CSP instance $P = \langle V, D, C \rangle$, the *structure* of P, denoted $\sigma(P)$, is the hypergraph $\langle V, \{\chi \mid \langle \chi, \rho \rangle \in C\} \rangle$.

Example 2.6. Recall the CSP instance P_{AG} defined in Example 2.2. The structure of P_{AG} is the hypergraph[2] H_{AG} illustrated in Fig. 1.

The set of vertices of H_{AG} is the set $\{1, 2, \ldots, 10\}$, and the eight hyperedges of H_{AG} are the following subsets of these vertices: $e_1 = \{1,2\}$, $e_2 = \{2,3,9\}$, $e_3 = \{3,4,10\}$, $e_4 = \{4,5\}$, $e_5 = \{5,6,9\}$, $e_6 = \{6,7,10\}$, $e_7 = \{7,8,9\}$, $e_8 = \{1,8,10\}$.

3. Guarded decompositions

Two CSP instances with the same set of variables are called *solution-equivalent* if they have the same set of solutions. As we will argue in Section 5, all known structural decomposition methods take CSP instances and transform them into solution-equivalent instances with simpler structure. The constraints of these transformed instances are obtained by calculating the relational joins of certain constraint relations in the original instances, and then projecting these onto new scopes. To describe this general transformation scheme we introduce the following terminology.

[2] This hypergraph was originally described in Example 3 of Adler et al. [2].

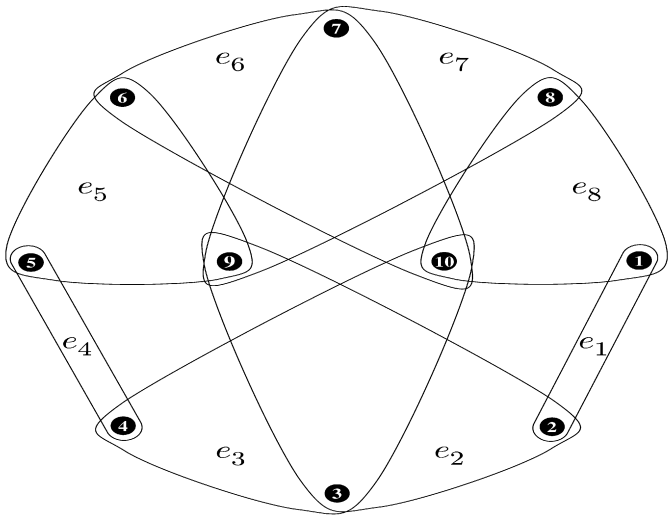

Fig. 1. The hypergraph H_{AG}, which is the structure of the CSP instance P_{AG}, defined in Example 2.2.

Definition 3.1. A *guarded block* of a hypergraph $H = \langle V, E\rangle$ is a pair $\langle \lambda, \chi\rangle$ where the *guard*, λ, is a subset of the hyperedges of H, and the *block*, χ, is a subset of the vertices of the guard. That is, $\lambda \subseteq E$ and $\chi \subseteq \bigcup_{e\in\lambda} e$.

For any guarded block b, the guard of b will be denoted $\lambda(b)$ and the block of b will be denoted $\chi(b)$. For any set of hyperedges λ, we will write $(\bigcup \lambda)$ to refer to the set of all vertices contained in edges of λ; that is, $(\bigcup \lambda) = \bigcup_{e\in\lambda} e$.

Definition 3.2. For any CSP instance P, and any guarded block b of $\sigma(P)$, the constraint *generated* by P on b is the constraint $\langle \chi(b), \rho\rangle$, where ρ is the projection onto $\chi(b)$ of the relational join of all the constraints of P whose scopes are elements of $\lambda(b)$.

Given a CSP instance P, with structure H, and a collection of guarded blocks of H, we can generate constraints on each of these guarded blocks to obtain a new collection of constraints, and hence a new CSP instance. In some cases, if the guarded blocks are carefully chosen, this new instance will be solution-equivalent to P, and so can be used to solve P. If this property holds for *any* CSP instance with structure H, then the collection of guarded blocks will be called a *guarded decomposition* of H, which is formally defined as follows:

Definition 3.3. A set of guarded blocks Ξ of a hypergraph H is called a *guarded decomposition* of H if, for every CSP instance $P = \langle V, D, C\rangle$ with structure H, the instance $P' = \langle V, D, C'\rangle$, where C' is the set of constraints generated by P on the members of Ξ, is solution-equivalent to P.

Example 3.4. For any hypergraph $H = \langle V, E\rangle$ there are two trivial guarded decompositions.

The set $\{\langle\{e\}, e\rangle \mid e \in E\}$ is a guarded decomposition. For every instance P with structure H, the generated instance is P itself, where constraints over the same scope are merged by taking the intersection of their relations. Thus, the generated instance is clearly solution-equivalent to the original one.

The set $\{\langle E, V\rangle\}$ is also a guarded decomposition. For every instance P with structure H, the tuples of the single constraint in the generated instance are exactly the solutions to the instance P.

Theorem 3.6 below gives a simple and efficient way to determine whether a given collection of guarded blocks is in fact a guarded decomposition, based on the following properties.

Definition 3.5. A guarded block $\langle \lambda, \chi\rangle$ of a hypergraph H *covers* a hyperedge e of H if $e \subseteq \chi$.

A set of guarded blocks Ξ of a hypergraph H is called a *guarded cover* for H if each hyperedge of H is covered by some guarded block of Ξ.

A set of guarded blocks Ξ of a hypergraph H is called a *complete guarded cover* for H if each hyperedge e of H occurs in the guard of some guarded block of Ξ which covers e.

Theorem 3.6. *A set of guarded blocks Ξ of a hypergraph H is a guarded decomposition of H if and only if it is a complete guarded cover for H.*

Proof. Let Ξ be a set of guarded blocks of a hypergraph $H = \langle V, E\rangle$.

Suppose that Ξ is *not* a complete guarded cover for H. Choose $e \in E$ such that there is no guarded block $\langle \lambda, \chi\rangle \in \Xi$ for which $e \in \lambda$ and $e \subseteq \chi$. We will construct an instance P_e with structure H which will witness the fact that Ξ is not a guarded decomposition of H.

Let the domain of P_e be $D = \{0, 1\}$. For each edge $f \in E$, $f \neq e$, let the constraint of P_e with scope f allow all mappings from f to D. Finally let the constraint of P_e with scope e, c_e, allow all mappings from e to D except for the mapping that assigns the value 0 to all the vertices of e simultaneously.

Now let $\langle \lambda, \chi\rangle$ be any guarded block in the set Ξ. If $e \notin \lambda$ then the constraint generated by P_e on $\langle \lambda, \chi\rangle$ allows every assignment. On the other hand, if $e \in \lambda$, then, by the choice of e, we know that $e \not\subseteq \chi$. However, any projection of c_e onto any proper subset of e allows all assignments, so again the constraint generated by P_e on $\langle \lambda, \chi\rangle$ allows every assignment.

Since all constraints generated by P_e on all elements of Ξ allow every assignment, the resulting CSP instance is not solution-equivalent to P_e, and hence Ξ is not a guarded decomposition of H.

For the converse, suppose that Ξ is a complete guarded cover for H. Let P be an arbitrary CSP instance with structure H. We have to show that the CSP instance P_Ξ obtained by taking the constraints generated by P on each element of Ξ is solution-equivalent to P.

Clearly, any solution to P is a solution to P_Ξ. On the other hand, by the completeness of Ξ, and the construction of P_Ξ, the projection of any solution to P_Ξ onto any edge $e \in E$ must be allowed by the constraint of P with scope e. Hence any solution to P_Ξ is also a solution to P. □

4. Tractability

The classes of CSP instances that we shall be identifying in this paper are classes which are defined purely in terms of the structure of their instances, without imposing any restrictions on the constraint relations.

Definition 4.1. A class I of CSP instances is called *structural* if there is some class of hypergraphs $\mathcal{H}$ for which $I = \{P \mid \sigma(P) \in \mathcal{H}\}$.

We have shown in the previous section that a guarded decomposition for a hypergraph H can be used to associate any CSP instance whose structure is H with a solution-equivalent instance having a different structure, which may be easier to solve.

We shall now consider structural classes of CSP instances which are defined as the class of all instances whose structure has a guarded decomposition of a certain kind. The most important structural classes of this kind are those which are *tractable*, in the following sense.

Definition 4.2. A class I of CSP instances is called *tractable* if there exists

- a polynomial-time algorithm to decide membership in I; and
- a polynomial-time algorithm to solve all members of I.

To ensure that a structural class defined using a class of guarded decompositions is tractable, it is sufficient to ensure that the class of guarded decompositions used in the definition has the following properties.

Tractable discovery: For any given hypergraph it must be possible to decide in polynomial time whether it has a guarded decomposition of the type we are considering, and to obtain such a decomposition in polynomial time if it exists.

Tractable construction: Given such a guarded decomposition, it must be possible to generate each of the new constraints in the corresponding solution-equivalent instance in polynomial time.

Tractable solution: Given such a solution-equivalent instance, it must be possible to solve the resulting instance in polynomial time.

We will now examine what conditions can be imposed on a class of guarded decompositions to ensure that it has each of these properties, starting with the most straightforward property.

4.1. Tractable construction

The time complexity of a relational join operation is $O(r^k)$, where k is the number of relations being joined, and r is the maximum number of tuples in any of these relations. Hence to ensure that we have the *tractable-construction* property, it is sufficient to bound the number of constraint relations that need to be combined using the relational join operation. This can be done by bounding the number of hyperedges in the guard of any guarded block used in the decomposition.

Definition 4.3. The *width* of a set of guarded blocks is the maximum number of hyperedges in any of its guards.

For any fixed value of k, the class of guarded decompositions of width at most k has the tractable-construction property.

4.2. Tractable solution

For a class of guarded decompositions to have the *tractable-solution* property we require that the new instance obtained by using any guarded decomposition in the class can be solved in polynomial time. One way to achieve this is to ensure that the structure of these new instances is *acyclic* [3]. The property of being acyclic can be defined as follows:

Definition 4.4. A *join tree of a hypergraph* H is a tree, T, whose nodes are the hyperedges of H, such that, whenever the vertex x of H occurs in two hyperedges e_1 and e_2 of H, then x occurs in each node of the unique path connecting e_1 and e_2 in T. In other words, the set of nodes of T in which x occurs induces a (connected) subtree of T.

A hypergraph is called *acyclic* if it has a join tree.

Theorem 4.5. *(See [14].) Any CSP instance whose structure is acyclic can be solved in polynomial time.*

Definition 4.6. A *join tree of a set of guarded blocks* Ξ of H is a tree, T, whose nodes are the elements of Ξ, such that, whenever the vertex x of H occurs in the blocks of two elements of Ξ, then x occurs in the block of each node of the unique path connecting them in T. In other words, the set of nodes of T for which x occurs in the block induces a (connected) subtree of T.

A set of guarded blocks is called *acyclic* if it has a join tree.

By Theorem 4.5, any class of *acyclic* guarded decompositions has the tractable-solution property.

A rooted join tree for an arbitrary acyclic guarded cover is often called a *generalised hypertree* [1,2,13]. We have avoided this terminology here, and instead used the term "acyclic guarded cover," for two reasons: in order to emphasise the set of guarded blocks itself (rather than a particular choice of rooted join tree for it) and in order to indicate that a hypertree decomposition (see Example 5.13) is just one of several possible specialisations of the general concept of acyclic guarded cover.

Using Theorem 3.6 and Definition 4.6 we now show that an acyclic guarded decomposition can be obtained from any acyclic guarded cover without increasing the width, or significantly increasing the number of guarded blocks, by simply adding appropriate additional guarded blocks to make a complete guarded cover.

Theorem 4.7. *If the set of guarded blocks* Ξ *is an acyclic guarded cover for* H *then the set* $\Xi \cup \{\langle\{e\}, e\rangle \mid e \in E\}$ *is an acyclic guarded decomposition of* H.

Proof. The set $\Xi \cup \{\langle\{e\}, e\rangle \mid e \in E\}$ is clearly a complete guarded cover of H, by Definition 3.5, and hence a guarded decomposition of H, by Theorem 3.6. It only remains to show that it is acyclic.

Since Ξ is a cover of H, for each hyperedge e of H there is some $b_e \in \Xi$ which covers e. Now consider any join tree for Ξ. For each guarded block $\langle\{e\}, e\rangle$ which is not in Ξ, add it to this tree as an additional node, and connect it to the existing node b_e. The resulting graph is a join tree for $\Xi \cup \{\langle\{e\}, e\rangle \mid e \in E\}$. □

In view of this result, we shall focus in the rest of the paper on methods to find acyclic guarded covers.

4.3. Tractable discovery

It follows from the results above that, for any fixed choice of k, any CSP instance whose structure has an acyclic guarded cover with width at most k can be solved in polynomial time by generating an associated solution-equivalent instance whose structure is acyclic.

However, to define a tractable structural class we still need some way to determine in polynomial time whether a given instance has such a guarded cover or not. We now introduce a very general algorithmic approach to this problem and identify conditions which are sufficient to ensure that this approach is effective.

Before describing our algorithmic approach we need some further definitions.

Definition 4.8. (See [12].) Let $H = \langle V, E\rangle$ be a hypergraph and $\chi \subseteq V$ be any subset of vertices.

A pair of vertices x, y is *χ-connected* if there is a sequence of hyperedges $e_0, \ldots, e_m$ such that $x \in e_0 - \chi$, $y \in e_m - \chi$ and $e_i \cap e_{i+1} \not\subseteq \chi$, for $i = 0, \ldots, m-1$.

A *χ-component* of H is a maximal non-empty set of vertices C such that each pair of vertices in C is χ-connected.

Example 4.9. Consider again the hypergraph H_{AG} defined in Example 2.6 and illustrated in Fig. 1.

If we set $\chi = \{3, 7, 10\}$, then there is just one χ-component: $\{1, 2, 4, 5, 6, 8, 9\}$. If we set $\chi = \{3, 6, 7, 9, 10\}$, then there are two χ-components: $\{4, 5\}$ and $\{1, 2, 8\}$. If we set $\chi = \{2, 3, 6, 7, 9, 10\}$, then there are again two χ-components: $\{4, 5\}$ and $\{1, 8\}$.

Definition 4.10. Let H be a hypergraph and let T be a tree whose nodes N are guarded blocks of H. For any pair of adjacent nodes n and n' of T, we define the *n'-branch of T with respect to n*, denoted $\mathrm{br}_n(n')$, to be the set of nodes of T whose (unique) path to n includes n'.

We define the vertices of $\mathrm{br}_n(n')$, denoted $\chi(\mathrm{br}_n(n'))$, to be the vertices in the blocks of the elements of $\mathrm{br}_n(n')$ which are not in $\chi(n)$. That is,

$$\chi\big(\mathrm{br}_n(n')\big) \stackrel{\mathrm{def}}{=} \bigcup_{b \in \mathrm{br}_n(n')} \chi(b) - \chi(n).$$

Proposition 4.11. *Let H be a hypergraph, and let T be a join tree of a guarded cover of H. Then T satisfies the following conditions*:

JT1 *for every arc $\langle n, n'\rangle$ of T, and every edge e of H, if $e \cap \chi(\mathrm{br}_n(n')) \neq \emptyset$, then e is covered by some node of* $\mathrm{br}_n(n')$;
JT2 *for every arc $\langle n, n'\rangle$ of T, $\chi(\mathrm{br}_n(n'))$ is a union of $\chi(n)$-components of H.*

Proof. To see that JT1 holds, choose an arbitrary $v \in e \cap \chi(\mathrm{br}_n(n')) \neq \emptyset$. By Definition 4.10, $v \notin \chi(n)$. Hence, by the join-tree property, v does not belong to the block of any node in any other branch of T with respect to n. Since T is a join tree of a guarded cover, there must be a node b of T such that $e \subseteq \chi(b)$. In particular, $v \in \chi(b)$, whence $b \in \mathrm{br}_n(n')$.

To see that JT2 holds, let $v \in \chi(\mathrm{br}_n(n'))$. It suffices to show that $\chi(\mathrm{br}_n(n'))$ contains the entire $\chi(n)$-component to which v belongs. Thereto, let w be another vertex in this $\chi(n)$-component. Hence, there exist edges $e_0, \ldots, e_m$ for which $v \in e_0 - \chi(n)$, $w \in e_m - \chi(n)$ and, for $i = 1, \ldots, m$, $e_{i-1} \cap e_i \not\subseteq \chi(n)$. We show that, for $i = 0, \ldots, m$, $(e_i - \chi(n)) \subseteq \chi(\mathrm{br}_n(n'))$. For $i = 0$, this follows from JT1. Now let $i > 0$. As inductive hypothesis, assume that our claim holds for $j = 0, \ldots, i-1$. Since $(e_{i-1} - \chi(n)) \subseteq \chi(\mathrm{br}_n(n'))$ and $e_{i-1} \cap e_i \not\subseteq \chi(n)$, $(e_i - \chi(n)) \cap \chi(\mathrm{br}_n(n')) \neq \emptyset$.

Types:

$H = \langle V, E\rangle$ is a hypergraph;
b is a guarded block of H;
C is a $\chi(b)$-component of H;
β is a predicate on pairs of guarded blocks of H.

β-DECOMPOSE$(H, b, C) :=$
There exists a guarded block b', such that
(1) $\beta(b, b')$ holds;
(2) $\chi(b') \subseteq \chi(b) \cup C$;
(3) For every edge e of H, if $e \cap C \neq \emptyset$, then $e \subseteq C \cup \chi(b')$
(4) For every $\chi(b')$-component C', if $C' \cap C \neq \emptyset$, then
  (a) $C' \cap \chi(b) = \emptyset$; and
  (b) β-DECOMPOSE(H, b', C').

Fig. 2. Definition of the predicate β-DECOMPOSE.

Hence, by JT1, $(e_i - \chi(n)) \subseteq \chi(\mathrm{br}_n(n'))$. We may thus conclude that $(e_m - \chi(n)) \subseteq \chi(\mathrm{br}_n(n'))$, whence $w \in \chi(\mathrm{br}_n(n'))$. □

For our present purpose of ensuring the tractable discovery property, condition JT2 is not strong enough. Therefore, we now define a rooted variation of a join tree for a set of guarded blocks, which we call a *decomposition tree*. For a decomposition tree, we require that $\chi(\mathrm{br}_n(n'))$ is a *single* $\chi(n)$-component whenever n' is a child of n.

Definition 4.12. A *decomposition tree*, T, of a hypergraph H is a rooted join tree of a set of guarded blocks of H satisfying the following conditions:

DT1 for every arc $\langle n, n'\rangle$ of T, and every edge e of H, if $e \cap \chi(\mathrm{br}_n(n')) \neq \emptyset$, then e is covered by some node of $\mathrm{br}_n(n')$;
DT2 for every arc $\langle n, n'\rangle$ of T, there exists a single $\chi(n)$-component, $C_{\langle n,n'\rangle}$, of H such that $\chi(\mathrm{br}_n(n')) = C_{\langle n,n'\rangle}$.

We are now ready to state and prove the main result of this section.

Theorem 4.13. *Let H be a hypergraph, and let β be a binary predicate on pairs of guarded blocks of H.*

For any guarded block b of H, and any $\chi(b)$-component C of H, the predicate β-DECOMPOSE(H, b, C) defined in Fig. 2 holds if and only if H has a decomposition tree T with root b such that $C = \chi(\mathrm{br}_b(b'))$ for the unique child b' of b in T and every arc $\langle n, n'\rangle$ of T satisfies β.

Proof. ($\Rightarrow$) First assume that β-DECOMPOSE(H, b, C) holds. We will prove that there exists a corresponding decomposition tree $T_{b,C}$ with root b such that $C = \chi(\mathrm{br}_b(b'))$ for the unique child b' of b in $T_{b,C}$ and every arc $\langle n, n'\rangle$ of T satisfies β. The proof is by induction on the depth of recursion required to establish that β-DECOMPOSE(H, b, C) holds.

If this depth is 0, then by the definition of β-DECOMPOSE, there exists a guarded block b' of H such that:

(1) $\beta(b, b')$ holds;
(2) $\chi(b') \subseteq \chi(b) \cup C$;
(3) for every edge e of H, if $e \cap C \neq \emptyset$, then $e \subseteq C \cup \chi(b')$; and
(4) for every $\chi(b')$-component C' of H, $C' \cap C = \emptyset$.

The tree $T_{b,C} = \langle\{b, b'\}, \{\langle b, b'\rangle\}\rangle$ clearly is a join tree of $\{b, b'\}$. We next show that $T_{b,C}$ satisfies DT1, DT2, and $\chi(\mathrm{br}_b(b')) = C$. We start with the last condition. First notice that $\chi(\mathrm{br}_b(b')) = \chi(b') - \chi(b)$. By condition 2 above, $\chi(\mathrm{br}_b(b')) \subseteq C$. Now assume there exists $v \in C - \chi(b')$, and let C' be the $\chi(b')$-component containing v. Then $C \cap C' \neq \emptyset$, contradicting condition 2 above. Thus, $C - \chi(b') = \emptyset$, whence $\chi(\mathrm{br}_b(b')) = C$.This also settles DT2. Finally, to show DT1, let e be an edge such that $e \cap \chi(\mathrm{br}_b(b')) = e \cap C \neq \emptyset$. By condition 2, $e \subseteq C \cup \chi(b') = \chi(b')$.

Now assume that the depth of recursion required to establish that β-DECOMPOSE(H, b, C) holds is greater than 0, and that the result holds in all cases where this depth is smaller. In this case we construct the rooted tree $T_{b,C}$ by linking all the trees $T_{b',C'}$ corresponding to recursive calls at their common root b', and then adding the node b as a parent of b'. It remains to show that $T_{b,C}$ is a join tree satisfying DT1 and DT2, that every arc $\langle n, n'\rangle$ of T satisfies β, and that $C = \chi(\mathrm{br}_b(b'))$.

To establish that $T_{b,C}$ is a join tree, note that, by the inductive hypothesis, each of the trees $T_{b',C'}$ used in the construction is a decomposition tree. Hence it only remains to show that

- Any vertex v occurring in the block of a node in two distinct subtrees T_{b',C'_1} and T_{b',C'_2} must also occur in $\chi(b')$. To see this, assume that v occurs in some node of T_{b',C'_1} but does not occur in $\chi(b')$. Then, for some child b'' of b', it must occur in $\chi(\mathrm{br}_{b'}(b''))$, which equals C'_1, by the inductive hypotheses. The same applies to T_{b',C'_2}. However, C'_1 and C'_2 are distinct $\chi(b')$-components chosen in the recursive step of the algorithm, so $C'_1 \cap C'_2 = \emptyset$, and the result follows.
- Any vertex v occurring in the block of a node in some subtree $T_{b',C'}$ and in $\chi(b)$ must also occur in $\chi(b')$. By the same argument as in the previous case, if v occurs in some node of $T_{b',C'}$ but does not occur in $\chi(b')$, then $v \in C'$. However, by the definition of β-DECOMPOSE, we have that $C' \cap \chi(b) = \emptyset$, and the result follows.

To establish conditions DT1 and DT2, note that, by the inductive hypothesis, these conditions hold for every subtree used in the construction. Therefore, it suffices to show that

(1) every edge e of H with $e \cap C \neq \emptyset$ is covered by some node of $\mathrm{br}_b(b')$;
(2) $\chi(\mathrm{br}_b(b')) = C$.

To show that every edge e of H with $e \cap C \neq \emptyset$ is covered by some node of $\mathrm{br}_b(b')$ we note that if $e \cap C \neq \emptyset$, then, by the definition of β-DECOMPOSE, we have two cases: either $e \subseteq \chi(b')$ or $e \cap C \cap C' \neq \emptyset$ for some $\chi(b')$-component C'. By the inductive hypothesis, if $e \cap C' \neq \emptyset$ for some $\chi(b')$-component C' with $C' \cap C \neq \emptyset$, then e is covered by some node of $T_{b',C'}$, and hence in some node of $\mathrm{br}_b(b')$, by the construction of $T_{b,C}$. Hence, in either case, we have the result.

To show that $\chi(\mathrm{br}_b(b')) = C$ we note that $\chi(\mathrm{br}_b(b')) \cup \chi(b)$ is the set of all vertices in the blocks of all trees $T_{b',C'}$ used in the construction of $T_{b,C}$, together with the vertices of $\chi(b)$. By the inductive hypothesis, this set consists of all vertices in all $\chi(b')$-components C' for which $C' \cap C \neq \emptyset$, together with the vertices of $\chi(b)$ and $\chi(b')$. However, every vertex of C must either lie in $\chi(b')$, or in some $\chi(b')$-component, because, together, these include all vertices of H. Hence $\chi(\mathrm{br}_b(b')) \cup \chi(b) \supseteq C \cup \chi(b)$, so $\chi(\mathrm{br}_b(b')) \supseteq C$.

For the reverse inclusion we first note that $\chi(b') \subseteq C \cup \chi(b)$, by the definition of β-DECOMPOSE. It only remains to show that, for any $\chi(b')$-components C' for which $C' \cap C \neq \emptyset$, we have that $C' \subseteq C$. Assume for contradiction that $C' \not\subseteq C$ for some $\chi(b')$-component C' with $C' \cap C \neq \emptyset$. By the definition of $\chi(b')$-connectedness, this implies that there exists some edge e of H such that $e \cap C' \cap C \neq \emptyset$ but $e \not\subseteq C$. In other words, there are vertices $v, w \in e$ such that $v \in C' \cap C$ and $w \notin C$. Since C is a $\chi(b)$-component, this implies that $w \in \chi(b)$, but then $w \in \chi(b')$, because we have already established that $T_{b,C}$ is a join-tree. However, by the inductive hypothesis, any such e is covered by some node of $\mathrm{br}_{b'}(b'')$, so $w \notin \chi(b')$, which yields the required contradiction.

Finally, to establish that every arc $\langle n, n'\rangle$ of T satisfies β, note that by the inductive hypothesis $\beta(n, n')$ holds for every arc $\langle n, n'\rangle$ in every subtree $T_{b',C}$ used in the construction. Hence it only remains to show that $\beta(b, b')$ holds, and this follows immediately from the definition of β-DECOMPOSE.

($\Leftarrow$) For the converse result, assume that H has a decomposition tree $T_{b,C}$ with root b such that $C = \chi(\mathrm{br}_b(b'))$ for the unique child b' of b in $T_{b,C}$ and every arc of T satisfies β. We will prove by induction on the depth of $T_{b,C}$ that β-DECOMPOSE(H, b, C) holds.

Since the root of $T_{b,C}$ has exactly one child, b', the depth of $T_{b,C}$ is at least 1.

If this depth is exactly 1, then $T_{b,C} = \langle\{b, b'\}, \{\langle b, b'\rangle\}\rangle$. By our assumptions about $T_{b,C}$, we know that $\beta(b, b')$ holds. From $C = \chi(\mathrm{br}_b(b')) = \chi(b') - \chi(b)$, it follows that $\chi(b') = \chi(b) \cup C$. By condition DT1, every edge of e which meets $C = \chi(\mathrm{br}_b(b'))$ is contained in $\chi(b') = C \cup \chi(b')$. Finally, since $C \subseteq \chi(b')$, there are no $\chi(b')$-components of H which meet C. Hence β-DECOMPOSE(H, b, C) holds.

Now assume that the depth of $T_{b,C}$ is greater than 1, and that the result holds in all cases where this depth is smaller. In this case, we consider the sub-trees of $T_{b,C}$ rooted at b' and containing a unique child of b' together with all its descendants (if any).

By the definition of a decomposition tree, each of these sub-trees is itself a decomposition tree $T_{b',C'}$ with root b' such that $\chi(\mathrm{br}_{b'}(b''))$ for the unique child b'' of b' is a distinct $\chi(b')$-component, C', of H. Also, every arc of each $T_{b',C'}$ satisfies β. Moreover, by condition DT2 applied to the branch $\mathrm{br}_b(b')$ of $T_{b,C}$, we have that the union of all these $\chi(b')$-components, together with $\chi(b')$ contains C. Hence, by the inductive hypothesis, β-DECOMPOSE(H, b', C') holds for each $\chi(b')$-component C' such that $C' \cap C \neq \emptyset$.

Now consider any $\chi(b')$-component C' such that $C' \cap C \neq \emptyset$. We will show that $C' \subseteq C$, and so $C' \cap \chi(b) = \emptyset$, because C is a $\chi(b)$-component. Assume for contradiction that $C' \not\subseteq C$ for some $\chi(b')$-component C' with $C' \cap C \neq \emptyset$. By the definition of $\chi(b')$-connectedness, this implies that there exists some edge e of H such that $e \cap C' \cap C \neq \emptyset$ but $e \not\subseteq C$. In other words, there are vertices $v, w \in e$ such that $v \in C' \cap C$ and $w \notin C$. Since C is a $\chi(b)$-component, this implies that $w \in \chi(b)$, but then $w \in \chi(b')$, because $T_{b,C}$ is a join-tree. However, by condition DT1 any such e is covered by some node of a branch out of b', so $w \notin \chi(b')$, which gives the required contradiction.

By condition DT1, every edge e of H which meets C is covered in some node of $\mathrm{br}_b(b')$, so either $e \subseteq \chi(b')$, or else e is covered by some node of $\mathrm{br}_{b'}(b'')$ for some child b'' of b'. In the latter case, by condition DT2, $e \subseteq \chi(b') \cup C'$ for some $\chi(b')$-component $C' = C_{\langle b', b''\rangle}$.

By condition DT2 applied to the branch $\mathrm{br}_b(b')$ of $T_{b,C}$, we have that $\chi(b') - \chi(b) \subseteq C$, so $\chi(b') \subseteq \chi(b) \cup C$.

Finally, by our assumptions about $T_{b,C}$, we know that $\beta(b, b')$ holds.

Putting all these observations together, we have established that β-DECOMPOSE(H, b, C) holds. □

In Theorem 4.13, we considered trees where the root b has a unique child, corresponding to a particular $\chi(b)$-component. The final construction step consists of considering all $\chi(b)$-components, and merging the corresponding trees into a single tree at their common root. This then yields the following corollary.

Corollary 4.14. *A hypergraph H has an acyclic guarded cover Ξ which has a decomposition tree in which every arc satisfies β if and only if there is some guarded block $b \in \Xi$ such that β-DECOMPOSE(H, b, C) holds for each $\chi(b)$-component C.*

Proof. ($\Rightarrow$) First assume that a hypergraph H has an acyclic guarded cover Ξ which has a decomposition tree T in which every arc satisfies β. Let b be the root of T, and let C be a $\chi(b)$-component. Then, by DT2, $C = \chi(\mathrm{br}_b(b'))$ for some child b' of b. Clearly, the tree $T_{b,C}$ obtained by deleting all other children of b from T satisfies the conditions of Theorem 4.13. Hence, β-DECOMPOSE(H, b, C) holds.

($\Leftarrow$) Now assume that there is some guarded block $b \in \Xi$ such that β-DECOMPOSE(H, b, C) holds for each $\chi(b)$-component C. If there are no such components, then the single block b is already an acyclic guarded cover for H. Otherwise, by Theorem 4.13, there exists a decomposition tree $T_{b,C}$ with root b such that $C = \chi(\mathrm{br}_b(b'))$ for the unique child b' of b in $T_{b,C}$ and every arc of $T_{b,C}$ satisfies β. We construct T by linking all trees $T_{b,C}$ at their common root b. By DT1, the nodes of T form a guarded cover. As in the proof of Theorem 4.13, it is straightforward to show that T is a join tree. It follows that T is a decomposition tree in which every arc satisfies β. □

The time required to establish that β-DECOMPOSE(H, b, C) holds, and hence compute a decomposition tree if one exists, depends on the time required to enumerate the elements of β, which will be denoted $|\beta|$.

Theorem 4.15. *Let $H = \langle V, E\rangle$ be a hypergraph, and let β be a binary predicate on pairs of guarded blocks of H. In $O(|\beta||E||V|^2)$ time it is possible to decide whether H has an acyclic guarded cover Ξ which has a decomposition tree where every arc satisfies β, and to construct such a cover if it exists.*

Proof. By Corollary 4.14, to decide whether H has an acyclic guarded cover with a decomposition tree where every arc satisfies β it is sufficient to determine whether the predicate β-DECOMPOSE(H, b, C) holds for some guarded block b of H, and every $\chi(b)$-component C of H. Moreover, by the proofs of Theorem 4.13 and Corollary 4.14, if all such β-DECOMPOSE(H, b, C) hold, then we can construct such a guarded cover by considering all the guarded blocks b' chosen in all the trees of recursive calls.

By storing the results, we can ensure that β-DECOMPOSE(H, b, C) only needs to be evaluated once for each choice of b and C. Furthermore, we only need to consider guarded blocks b for which $\beta(b, b')$ holds for some b', and in the evaluation of β-DECOMPOSE(H, b, C) we only need to consider b' for which $\beta(b, b')$ holds. For each such pair b, b', we can pre-compute all the $\chi(b)$-components and $\chi(b')$-components in $O(|E||V|\log|V|)$ time, and there are at most $O(|V|)$ such components. Taking into account that set operators and comparisons take $O(|V|)$ time, all of the necessary evaluations of β-DECOMPOSE can be carried out in $O(|\beta||E||V|^2)$ time. □

In view of this result, we shall focus in the rest of the paper on methods to find acyclic guarded covers with a decomposition tree where every arc satisfies β, for some binary predicate β such that the time required to enumerate the elements of β is polynomial in the size of H. We have shown that these conditions are sufficient to ensure that the decompositions we consider have the tractable-discovery property.

5. Existing decomposition methods

We will now show that many existing decomposition methods for the CSP can be defined in terms of finding acyclic guarded covers using specified sets of guarded blocks.

In the examples below we explicitly show, by finding a suitable predicate β in each case, how we can use the algorithm of the previous section to tractably discover (sometimes slight generalisations of) many of these known decompositions.

We first note that, for historical reasons, some existing decomposition methods for the CSP make use of *extended guards* that contain both vertices and hyperedges. To be able to present all of these methods in a uniform way we introduce the idea of transforming a CSP instance by adding a unary constraint c_v, for each variable v, where $c_v = \langle\{v\}, \{v\}^D\rangle$. This corresponds to extending the structure of the instance to ensure that it includes a hyperedge for each variable.

Definition 5.1. Let $H = \langle V, E\rangle$ be any hypergraph. A guarded cover for the hypergraph $\langle V, E \cup \{\{v\} \mid v \in V\}\rangle$ is called an *extended cover* for H.

The following result shows that the existence of an extended cover of width k implies the existence of a standard guarded cover of width at most k. This means that extended covers are simply a notational convenience and do not allow better decompositions.

Proposition 5.2. *If a hypergraph H has an acyclic extended cover of width k, then H also has an acyclic guarded cover of width at most k with at most the same number of guarded blocks.*

Proof. Let Ξ be an acyclic extended cover for a hypergraph H. Now replace every hyperedge in every guard with some hyperedge of H that includes it.

Since this process preserves all of the existing blocks, it is clear that the new set of guarded blocks obtained in this way is an acyclic guarded cover of the same or lower width, with the same number of guarded blocks (or possibly fewer). □

Also for historical reasons, the width of some decompositions has been defined to be the maximum number of *vertices* occurring in the block of any guarded block. However, this measure hides the fact that, in order to compute the generated constraints, we have to compute the relational join of the edges in the guard. In fact, whenever we have an acyclic guarded cover with vertex-width k we can easily construct an acyclic guarded cover with (hyperedge) width bounded by k, as the next result makes clear.

Proposition 5.3. *If a hypergraph H has an acyclic guarded cover with at most k vertices in any block, then H also has an acyclic guarded cover of width at most k with at most the same number of guarded blocks.*

Proof. Let Ξ be an acyclic guarded cover of H with at most k vertices in any block. For each guarded block $\langle\lambda, \chi\rangle$ of Ξ, and for each vertex v in χ, choose some hyperedge $e_v \in H$ such that $v \in e_v$. Now replace each guarded block $\langle\lambda, \chi\rangle$ with the guarded block $\langle\{e_v \mid v \in \chi\}, \chi\rangle$. It is clear that the width of this new guarded block is bounded by $|\chi|$.

Since we are keeping all of the existing blocks it is clear that the new set of guarded blocks is still an acyclic guarded cover. □

We also note that many existing decomposition methods are based on guarded covers in which every block is exactly the union of the edges of its guard.

Definition 5.4. A guarded block b is *edge-defined* if its block is exactly the set of vertices contained in the hyperedges of its guard, i.e., if $\chi(b) = (\bigcup \lambda(b))$. A set of guarded blocks is edge-defined if all of its guarded blocks are edge-defined.

Unfortunately, it has been shown that the class of edge-defined acyclic guarded covers[3] does not have tractable discovery.

Theorem 5.5. *(See [12].) For any $k > 3$ it is NP-hard to discover whether a hypertree has an edge-defined acyclic guarded cover of width at most k.*

However we can easily show that those edge-defined acyclic guarded covers *with tree decompositions* do give us a tractable decomposition method.

Proposition 5.6. *For any fixed k, the class of CSP instances whose structure has an edge-defined acyclic guarded cover of width k with a decomposition tree is a tractable structural class.*

Proof. The number of edge-defined guarded blocks of width k for a hypergraph $H = \langle V, E\rangle$ is simply the number of distinct subsets of k edges, $C_k^{|E|}$, which for any fixed k is polynomial in the size of H, and this collection of subsets can be enumerated in polynomial time.

Hence, if we define β to be the predicate that holds for all pairs of edge-defined guarded blocks of H of width at most k, then the result follows from Theorem 4.15. □

We now characterise many known structural decomposition methods in terms of a corresponding class of acyclic guarded covers where the elements are chosen from a specified set (see [10] for traditional definitions of the known decomposition methods).

For each of the examples except hypertrees we show that if H has a decomposition of width k, then it has a decomposition of width k with a decomposition tree. In order to do this we use the following simple proposition.

Proposition 5.7. *Let Ξ be any acyclic guarded cover of H. If, for any $p, p' \in \Xi$, the set of vertices $(\chi(p) - \chi(p'))$ is $\chi(p')$-connected, then Ξ has a decomposition tree.*

Proof. Let T be any rooted join tree of Ξ. Suppose that T has a node p with child s where $\chi(\mathrm{br}_p(s))$ is more than one $\chi(p)$ component, say $C_1, \ldots, C_r$. We modify T by replacing this branch with a collection of branches.

For $i = 1, \ldots, r$, the ith new branch T_i has nodes $\{t \in \mathrm{br}_p(s) \mid \chi(t) \cap C_i \neq \emptyset\}$.

The last new branch T_0 has nodes $\{t \in \mathrm{br}_p(s) \mid \chi(t) \subseteq \chi(p)\}$.

The parent of t in any T_i is the first node of T_i on the path from t to p in T, or p if this path includes no nodes of T_i.

Clearly we still have an acyclic cover as we have not changed the set of guarded blocks.

Since any block of Ξ is $\chi(p)$-connected, each guarded block of $\mathrm{br}_p(s)$ occurs in exactly one of the T_i. It follows that the blocks of any two nodes from distinct T_i must intersect inside $\chi(p)$ and so the new tree is a join tree.

We perform this process once for each non-leaf node of T and we obtain a decomposition tree for Ξ. □

[3] In the literature edge-defined acyclic guarded covers have been called *pure query decompositions* [12].

Example 5.8. A *biconnected-component* [9] decomposition of a hypergraph is an edge-defined acyclic complete guarded cover, Ξ, satisfying the following *articulation condition*:

$$\forall b_1, b_2 \in \Xi \quad (b_1 \neq b_2) \quad \Rightarrow \quad \big|\chi(b_1) \cap \chi(b_2)\big| \leqslant 1.$$

A guarded block b has an articulation point if there exists a vertex x for which $\chi(b) - \{x\}$ is not connected. The biconnected-component decomposition Ξ is *minimal* if none of its guarded blocks has an articulation point. It is easy to show that if H has any biconnected-component decomposition of width k then it has a minimal such decomposition.

It is trivial that any minimal biconnected-component decomposition of H satisfies the conditions for Proposition 5.7, and so has a decomposition tree.

It follows from this that minimal biconnected-component decompositions can be tractably discovered using the algorithm described in Section 4 by choosing the predicate β that allows all pairs b_1, b_2 of edge-defined guarded blocks of width at most k without articulation points such that $|\chi(b_1) \cap \chi(b_2)| \leqslant 1$.

Example 5.9. A *cycle-hypercutset* [10] decomposition of a hypergraph $H = \langle V, E\rangle$ is an edge-defined acyclic complete guarded cover, Ξ, satisfying the following *simplicity condition*:

$$\exists C \subseteq E, \quad \forall \langle \lambda, \chi\rangle \in \Xi, \quad (C \subseteq \lambda) \wedge \big(|\lambda - C| \leqslant 1\big).$$

We first show that the existence of a cycle-hypercutset decomposition of this kind is equivalent to the existence of a cycle-hypercutset, where a cycle-hypercutset of a hypergraph $H = \langle V, E\rangle$ is defined to be a set $C \subseteq E$ such that $H_C = \langle V - (\bigcup C), \{e - (\bigcup C) \mid e \in E\}\rangle$ is acyclic.

To see this equivalence, assume that C is a cycle-hypercutset of H and let T be a join tree of H_C. Label each node $e - (\bigcup C)$ of T with the edge-defined guarded block whose guard is $\{e\} \cup C$. This defines a rooted tree of guarded blocks which satisfies the join-tree condition and is a complete guarded cover of H.

Conversely, suppose that there is some C such that the set of edge-defined guarded blocks with guards $\{\{e\} \cup C \mid e \in E\}$ is an acyclic cover of H. Let T be a join tree for this set. It is clear that replacing the guarded block defined for e by the set of vertices $e - (\bigcup C)$ defines a join tree of H_C, so H_C is acyclic and C is a cycle-hypercutset.

It is trivial that any cycle-hypercutset decomposition satisfies the conditions for Proposition 5.7, and so has a decomposition tree.

However, it is not completely straightforward to discover cycle-hypercutset decompositions using the algorithm in Section 4, because the simplicity condition defined above becomes local only once we fix C. One possible approach is to run this algorithm for each $C \subseteq E$ with $|C| < k$ (which is at most polynomially often) and, in each run, define a predicate β_C which holds for all pairs of edge-defined guarded blocks, b, of width at most k such that $C \subseteq \lambda(b)$ and $|\lambda(b) - C| \leqslant 1$.

Another approach is to note that cycle-hypercutset decompositions can be extended to a more general tractable class of decompositions, by defining a single predicate β which holds for all pairs of edge-defined guarded blocks, b_1, b_2, of width at most k, such that $|\lambda(b_1) - \lambda(b_2)| \leqslant 1$.

Example 5.10. A *cycle-cutset* [7] decomposition of a hypergraph H is an edge-defined acyclic extended guarded cover, Ξ, satisfying the following *simplicity condition*:

$$\exists C \subseteq V, \quad \forall \langle \lambda, \chi\rangle \in \Xi, \quad \big(\big\{\{x\} \,\big|\, x \in C\big\} \subseteq \lambda\big) \wedge \big(\big|\lambda - \big\{\{x\} \,\big|\, x \in C\big\}\big| \leqslant 1\big).$$

An analogous proof to that for cycle-hypercutsets (Example 5.9) shows that the existence of a cycle-cutset decomposition of this kind is equivalent to the existence of a cycle-cutset, where a cycle-cutset of a hypergraph $H = \langle V, E\rangle$ is defined to be a set $C \subseteq V$ such that $H_C = \langle V - C, \{e - C \mid e \in E\}\rangle$ is acyclic.

It is trivial that any cycle-cutset decomposition satisfies the conditions for Proposition 5.7, and so has a decomposition tree.

As with cycle-hypercutsets (Example 5.9), we can obtain a cycle-cutset decomposition in polynomial time using the algorithm defined in Section 4 by running the algorithm for each $C \subseteq V$ with $|C| < k$, and in each run, defining a predicate β_C which holds for all pairs of extended edge-defined guarded blocks, b, of width at most k such that $\{\{x\} \mid x \in C\} \subseteq \lambda(b)$ and $|\lambda(b) - \{\{x\} \mid x \in C\}| \leqslant 1$.

We note that cycle-cutset decompositions can also be extended to a more general tractable class of decompositions, by defining a single predicate β which holds for all pairs of extended edge-defined guarded blocks, b_1, b_2, of width

at most k, such that all but at most one of the elements of $\lambda(b_1)$ and of $\lambda(b_2)$ consist of single vertices and $|\lambda(b_1) - \lambda(b_2)| \leqslant 1$.

Example 5.11. A *hinge-tree* [14] decomposition of a hypergraph H is an edge-defined acyclic complete guarded cover, Ξ, satisfying the following *separation condition*:

$$\forall b_1, b_2 \in \Xi \quad (b_1 \neq b_2) \quad \Rightarrow \quad \big(\exists e \in \lambda(b_1),\ \chi(b_1) \cap \chi(b_2) \subseteq e\big).$$

The standard definition of a hinge-tree [14] is an acyclic cover of edge-defined guarded blocks with a join tree T which satisfies, for each arc $\langle b_1, b_2\rangle$ of T,

$$\exists e \in \lambda(b_1) \cap \lambda(b_2), \quad \chi(b_1) \cap \chi(b_2) \subseteq e. \tag{5.1}$$

We will first establish that the existence of a hinge-tree T implies the existence of a hinge-tree decomposition as defined here. Since T is a join tree, it follows that, for every pair b_1 and b_2 of distinct nodes, we have that there exists $e \in \lambda(b_1)$ such that $\chi(b_1) \cap \chi(b_2) \subseteq e$, so the set of nodes of T satisfies the separation condition above and so forms a hinge-tree decomposition.

On the other hand, if we have a hinge-tree decomposition Ξ with join tree T, not all arcs need satisfy Eq. (5.1). However, we will now show that by adding certain additional guarded blocks to T, we can transform it to a new join tree T' with the same width[1] that does satisfy Eq. (5.1).

For any arc $\langle b_1, b_2\rangle$ of T which does not have Property 5.1 we use the separation condition twice to obtain edges $e_1 \in \lambda(b_1)$ and $e_2 \in \lambda(b_2)$ such that $\chi(b_1) \cap \chi(b_2) \subseteq e_1$ and $\chi(b_1) \cap \chi(b_2) \subseteq e_2$. Then we simply add the edge-defined guarded block with guard $\{e_1, e_2\}$ and insert it into the join tree T between b_1 and b_2.

Hence we have shown that for any $k \geqslant 2$ the existence of a hinge-tree decomposition of width k as defined here is equivalent to the existence of a standard hinge-tree whose largest guard contains k edges.

We now show that we can transform any hinge-tree decomposition of width k to a hinge-tree decomposition of width at most k with a tree decomposition.

It was shown [14] that we can transform the decomposition so that each hinge is minimal without increasing the width. This means that each hinge p of the transformed decomposition has no separating edge. We reformulate this as the fact that for every hyperedge $e \in \lambda(p)$, the set $\chi(p) - e$ is e-connected. Let p and p' be two hinges of the decomposition. From the separation condition we know that there exists a hyperedge $e \in \lambda(p)$ such that $\chi(p) \cap \chi(p') \subseteq e$. So $\chi(p) - \chi(p')$ is indeed connected with respect to $\chi(p')$.

So, if we have a hinge-tree decomposition of width k we also have a hinge-tree decomposition of width at most k which has a decomposition tree.

It follows from this that hinge-tree decompositions can be discovered using the algorithm described in Section 4 by choosing the predicate β on pairs of edge-defined guarded blocks which is defined as follows:

$$\beta(b_1, b_2) \stackrel{\text{def}}{=} \exists e_1 \in \lambda(b_1), e_2 \in \lambda(b_2), \quad \chi(b_1) \cap \chi(b_2) \subseteq e_1 \cap e_2.$$

We remark that the width of a hinge-tree composed of minimal hinges (with respect to inclusion) is an invariant of a hypergraph called the *degree of cyclicity* [14].

Example 5.12. A *query decomposition* [4] of a hypergraph $H = \langle V, E\rangle$ is a pair $\langle \Xi, T\rangle$ where Ξ is a complete edge-defined acyclic extended cover of H, and T is a join tree of Ξ that satisfies the following *connectedness condition* on the guards:

$$\forall e \in E, \quad \big\{\langle \lambda, \chi\rangle \in \Xi \;\big|\; e \in \lambda\big\} \quad \text{is connected in } T.$$

Unlike the decomposition methods discussed earlier, this is *not* a local condition that can be captured by choosing a suitable binary predicate β. Indeed, determining whether H has a query decomposition of width 4 is known to be NP-hard [12].

[1] Provided the width of the original decomposition is at least 2.

Example 5.13. A *hypertree decomposition* [11,12] of a hypergraph H is a pair $\langle \Xi, T \rangle$, where Ξ is an acyclic guarded cover and $T = \langle N, A \rangle$ is a rooted join tree of Ξ, which satisfies the following *descendant condition*:

$$\forall \langle b, b' \rangle \in A, \quad \left(\left(\bigcup \lambda(b) \right) \cap \chi\left(\mathrm{br}_b(b') \right) \right) = \emptyset.$$

Note that any edge-defined acyclic guarded cover satisfies this condition, but the guarded blocks of a hypertree decomposition are not required to be edge-defined.

It was shown in [12] that hypertree decompositions can be tractably discovered using a recursive algorithm similar to the one defined in Section 4 above. The minimum width of any hypertree decomposition of a hypergraph H is called the *hypertree width* of H.

6. Comparing decompositions

The relative strengths of different decomposition techniques derived from acyclic guarded covers can be compared using the measures developed by Gottlob et al. [10].

The class of hypergraphs having a guarded cover with width at most k in some fixed class Δ, will be denoted $C(\Delta, k)$.

Definition 6.1. Let Δ_1 and Δ_2 be any two classes of guarded covers. We say that Δ_1 *generalises* Δ_2 if there exists a constant $c \geqslant 0$ such that, for every k, $C(\Delta_2, k) \subseteq C(\Delta_1, k + c)$.

We say that Δ_1 *strongly generalises* Δ_2 if Δ_1 generalises Δ_2, and there exists k for which there does not exist l with $C(\Delta_1, k) \subseteq C(\Delta_2, l)$.

Example 6.2. In Section 5 we defined edge-defined acyclic guarded covers.

It is clear that the class of *all* edged-defined acyclic covers generalises any restricted edge-defined decomposition, defined with extra conditions.

Furthermore, from Proposition 5.2 it follows immediately that the class of *all* edge-defined acyclic covers generalises any restricted *extended* edge-defined decompositions.

In particular the class of all edge-defined decompositions generalises the cycle-hypercutset decomposition described in Example 5.9, the cycle-cutset decomposition described in Example 5.10, the hinge-tree decomposition described in Example 5.11, the biconnected component decomposition described in Example 5.8 and the query decomposition described in Example 5.12.

Since any edge-defined acyclic guarded cover has an associated hypertree, this yields a direct proof of the fact (see Theorem 23 of Gottlob et al. [10]) that hypertrees generalise each of these earlier decompositions.

It is perhaps surprising that hypertrees generalise query decompositions since hypertree decompositions are tractable to discover for any bound k on the width, whereas it is NP-complete to determine whether a hypergraph has a query decomposition of width at most 3, as has been shown by Gottlob et al. [12].

To the best of our knowledge, however, it is still an open problem whether hypertrees *strongly* generalise the entire class of edge-defined acyclic guarded covers.

In Section 7, we will introduce a new class of decompositions which we call spread-cut decompositions. As with hypertree decompositions, spread-cut decompositions are not edge-defined and they rely on the existence of a particular rooted tree. Spread-cut decompositions strongly generalise the biconnected-component, cycle-cutset, cycle-hypercutset and hinge-tree decompositions described in Section 5. They also generalise the class of edge-defined acyclic guarded covers which have tree decompositions.

It has recently been shown that the hypertree width of any hypergraph is at most three times the minimal possible width of any acyclic guarded cover [2]. It follows that hypertrees cannot be strongly generalised by any class of acyclic guarded covers. Notwithstanding this result, we will show in Section 7 that, for some families of hypergraphs, the minimal width of a spread-cut decomposition is smaller than the hypertree width by some constant factor. Recall that solution runtime is exponential in the decomposition width; it follows that there are families of hypergraphs for which we have reduced the exponent in the runtime by a constant factor. Such an improvement allows the corresponding instances to be solved significantly faster using spread-cut decompositions than by using hypertree decompositions.

7. Spread cuts

Definition 7.1. A guarded block b of a hypergraph H has *unbroken components* if each $\chi(b)$-component of H meets (has non-empty intersection with) at most one $(\bigcup\lambda(b))$-component of H.

Example 7.2. Consider again the hypergraph H_{AG} defined in Example 2.6 and illustrated in Fig. 1.

Now consider the guarded block $b = \langle\lambda, \chi\rangle$ with $\lambda = \{e_2, e_6\}$ and $\chi = \{3, 6, 7, 9, 10\}$. It was shown in Example 4.9 that H has two χ-components, $\{4, 5\}$ and $\{1, 2, 8\}$, and two $(\bigcup\lambda)$-components, $\{4, 5\}$ and $\{1, 8\}$. Since each χ-component meets (has non-empty intersection with) exactly one $(\bigcup\lambda)$-component, it follows that b has unbroken components.

This special property of b relies on χ being "large enough." If we instead set $\chi = \{3, 7, 10\}$, then it was shown in Example 4.9 that H has just one χ-component, $\{1, 2, 4, 5, 6, 8, 9\}$, and this meets both $(\bigcup\lambda)$-components, so in this case the guarded block does not have unbroken components.

In general, the number of guarded blocks of a given hypergraph which have unbroken components is not polynomially bounded. For example, any guarded block $\langle\lambda, \chi\rangle$, where λ contains all of the edges of the hypergraph $\langle V, E\rangle$, has no $(\bigcup\lambda)$-components, and so has unbroken components, whatever choice is made for χ (and in this case there are $2^{|V|}$ choices for χ).

Hence, if we are to use the algorithm β-DECOMPOSE, defined in Section 4, to discover decompositions into guarded blocks with unbroken components, then it will necessary to impose further conditions on the guarded blocks that we allow in these decompositions in order to ensure tractability. To describe these additional conditions we will now define the notion of a *label* for the vertices in a set of edges.

Definition 7.3. Let λ be any set of hyperedges of a hypergraph $H = \langle V, E\rangle$.

We define the *label*, $L_\lambda(v)$, of any vertex $v \in (\bigcup\lambda)$ to be a pair, where the first component is the set of $(\bigcup\lambda)$-components which meet (have non-empty intersection with) a hyperedge containing v, and the second component is the set of hyperedges of λ which meet v.

That is,

$$L_\lambda(v)[1] = \left\{ C \;\middle|\; C \text{ is a } \left(\bigcup\lambda\right)\text{-component, } \exists e \in E,\ e \cap C \neq \emptyset,\ v \in e \right\},$$
$$L_\lambda(v)[2] = \{e \in \lambda \mid v \in e\}.$$

Example 7.4. Consider again the hypergraph H_{AG} defined in Example 2.6 and illustrated in Fig. 1. If we set $\lambda = \{e_2, e_6\}$, then $(\bigcup\lambda) = \{2, 3, 6, 7, 9, 10\}$, and we have:

$$L_\lambda(2) = \langle\{\{1, 8\}\}, \{e_2\}\rangle;$$
$$L_\lambda(3) = \langle\{\{4, 5\}\}, \{e_2\}\rangle;$$
$$L_\lambda(6) = \langle\{\{4, 5\}\}, \{e_6\}\rangle;$$
$$L_\lambda(7) = \langle\{\{1, 8\}\}, \{e_6\}\rangle;$$
$$L_\lambda(9) = \langle\{\{1, 8\}, \{4, 5\}\}, \{e_2\}\rangle;$$
$$L_\lambda(10) = \langle\{\{1, 8\}, \{4, 5\}\}, \{e_6\}\rangle.$$

Definition 7.5. We say that a guarded block $\langle\lambda, \chi\rangle$ *respects labels* if

$$\forall v, w \in \left(\bigcup\lambda\right) \quad \big(v \in \chi \text{ and } L_\lambda(w) = L_\lambda(v)\big) \quad \Rightarrow \quad w \in \chi.$$

Proposition 7.6. *For any fixed k and any hypergraph $H = \langle V, E\rangle$, the set of guarded blocks of H with width k which have unbroken components and respect labels can be enumerated in polynomial time in the size of H.*

Types:

$H = \langle V, E\rangle$ is a hypergraph; k is a positive integer.

GBENUMERATE(H, k)

1. For each $\lambda \subseteq E$ with $|\lambda| \leqslant k$:
2. Calculate the set of $(\bigcup\lambda)$-components of H;
3. Label each vertex $v \in (\bigcup\lambda)$;
4. For each suitable set of labels $\mathcal{L}$:
5. Set $\chi := \{v \in (\bigcup\lambda) \mid L_\lambda(v) \notin \mathcal{L}\}$;
6. If $\langle\lambda, \chi\rangle$ has unbroken components, then output $\langle\lambda, \chi\rangle$.

Fig. 3. Definition of the procedure GBENUMERATE.

Proof. For any guarded block $\langle\lambda, \chi\rangle$, define $\mathcal{L}(\lambda, \chi) = \{L_\lambda(v) \mid v \in (\bigcup\lambda) - \chi\}$.

If $\langle\lambda, \chi\rangle$ respects labels, then no vertex in χ has the same label as a vertex in $(\bigcup\lambda) - \chi$. Hence, for any $v \in (\bigcup\lambda)$, $v \in \chi$ if and only if $L_\lambda(v) \notin \mathcal{L}(\lambda, \chi)$.

Moreover, if $\langle\lambda, \chi\rangle$ also has unbroken components, then $\mathcal{L}(\lambda, \chi)$ only contains labels whose first component contains at most one element. In fact, for each edge $e \in \lambda$ we can choose a $(\bigcup\lambda)$-component, C_e, such that

$$\mathcal{L}(\lambda, \chi) \subseteq \big\{\langle\emptyset, l_2\rangle \,\big|\, l_2 \subseteq \lambda\big\} \cup \big\{\langle C_e, l_2\rangle \,\big|\, l_2 \subseteq \lambda,\ e \in l_2\big\}.$$

There are at most $|V|$ choices for each C_e, so all possible choices for $\mathcal{L}(\lambda, \_)$ can be enumerated in $O(|V|^k 2^{(2^k + k2^k)})$ time. Furthermore, each possible $\mathcal{L}(\lambda, \_)$ has size at most 2^{k+1}.

Hence to enumerate all such guarded blocks of width k we can simply run through all the subsets λ of E with at most k edges, and all possible choices for $\mathcal{L} = \mathcal{L}(\lambda, \_)$, set $\chi = \{v \in (\bigcup\lambda) \mid L_\lambda(v) \notin \mathcal{L}\}$, and output all those pairs $\langle\lambda, \chi\rangle$ which have unbroken components. An algorithm to do this is shown in Fig. 3.

To analyse the time complexity of this algorithm we make the following observations:

- There are $(|E| + 1)^k$ choices for λ in the outer loop.
- Step 2 (calculating $(\bigcup\lambda)$-components) can be completed in $O(|V|^3)$ time.
- Step 3 (labelling the vertices) can be completed in $O(|E||V|^2)$ time.
- There are $O(|V|^k 2^{(2^k + k2^k)})$ choices for $\mathcal{L}$ in the inner loop.
- Step 5 (choosing vertices of χ) can be completed in $O(2^k|V|^2)$ time.
- Step 6 (checking for unbroken components) can be completed in $O(|V|^3)$ time.

Hence this algorithm can be completed in $O((|E| + 1)^k |V|^{k+2} 2^{(2^k + k2^k)}(2^k + |V|))$ time, which is polynomial in the size of H (for fixed k). □

Definition 7.7. A *spread-cut* decomposition of H is an acyclic guarded cover Ξ with a decomposition tree, where every guarded block in Ξ has unbroken components and respects labels.[4]

In the proof of Theorem 4.15, for any guarded block b we needed to consider those guarded blocks b' satisfying $\beta(b, b')$. In the case of spread-cuts the predicate β is the direct product of a unary predicate with itself. Hence, in this case, we can simply test each b' against this unary predicate, which reduces the time complexity to $O(\sqrt{|\beta|}|E||V|^2)$. Using this fact, and combining with Proposition 7.6, we obtain the following corollaries.

Corollary 7.8. *A naive bound on the time complexity of discovering a spread-cut decomposition of width at most k is given by* $O((2^k + |V|)|E|^{k+1}|V|^{k+4}2^{(2^k + k2^k)})$.

[4] Note that this definition is different from the definition originally given in [5]. In fact, the definition given in [5] may be too weak to ensure the tractable discovery property.

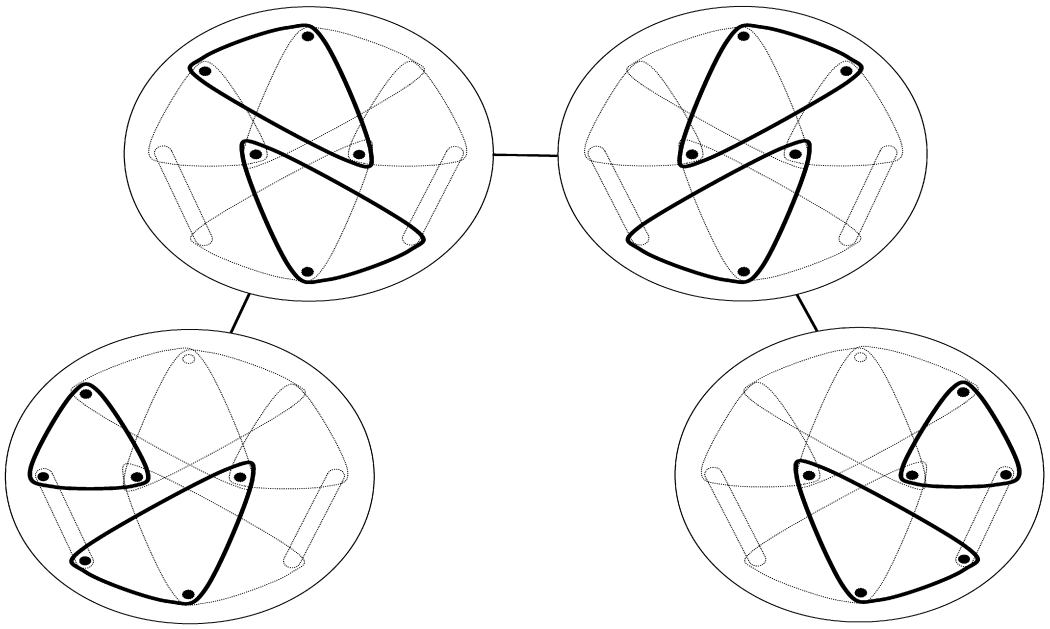

Fig. 4. The spread-cut defined in Example 7.10 for the hypergraph H_{AG} defined in Example 2.6.

Corollary 7.9. *For any fixed k, the class of CSP instances whose structure has a spread-cut decomposition of width at most k is a tractable structural class.*

Example 7.10. Consider again the hypergraph H_{AG} defined in Example 2.6 and illustrated in Fig. 1. It is straightforward to verify that the following set of guarded blocks is a spread-cut decomposition of H_{AG} of width two, as illustrated in Fig. 4:

$$\begin{aligned}\big\{&\big\langle\{e_3, e_5\}, \{3, 4, 5, 6, 9, 10\}\big\rangle,\\ &\big\langle\{e_2, e_6\}, \{3, 6, 7, 9, 10\}\big\rangle,\\ &\big\langle\{e_3, e_7\}, \{3, 7, 8, 9, 10\}\big\rangle,\\ &\big\langle\{e_2, e_8\}, \{1, 2, 3, 8, 9, 10\}\big\rangle\big\}.\end{aligned}$$

However, the minimal width of any hypertree decomposition of H_{AG} is three.[5]

Proposition 7.11. *Any edge-defined guarded block has unbroken components and respects labels.*

Proof. Let p be an edge-defined guarded block of some hypergraph H. Since $\chi = (\bigcup \lambda)$, the set of χ-components is equal to the set of $(\bigcup \lambda)$-components, and each of these components is disjoint, so each χ-component meets exactly one $(\bigcup \lambda)$-component, and p has unbroken components. Furthermore, there is no vertex in $(\bigcup \lambda) - \chi$, so p respects labels. □

Corollary 7.12. *Spread-cut decompositions generalise the biconnected component, cycle-cutset, cycle-hypercutset and hinge-tree decompositions described in Section* 5. *They also generalise the class of all edge-defined acyclic guarded covers which have tree decompositions.*

In fact, we have the stronger result that spread-cut decompositions *strongly* generalise the biconnected component, cycle-cutset, cycle-hypercutset and hinge decompositions described in Section 5. To establish this we simply refer

[5] This is most easily shown by using the "Robbers and Marshals" characterisation of hypertree width [13].

to the proofs of strong generalisation given by Gottlob et al. [10] and observe that every hypertree decomposition used to establish strong generalisation in that paper is also a spread-cut decomposition. Using the idea of a "switch graph" [1], we now give an example of a family of hypergraphs H_n for $n = 1, 2, \ldots$ where each H_n has a spread-cut decomposition of width $2n + 1$, but hypertree width $3n$.

Example 7.13. For each positive integer n we define a hypergraph H_n as follows. The set of vertices of H_n is the union of five disjoint sets:

$$K = \{k_i \mid i = 1, \ldots, 2n\},$$
$$K' = \{k'_i \mid i = 1, \ldots, 2n\},$$
$$S = \{s_i \mid i = 1, \ldots, n\},$$
$$S' = \{s'_i \mid i = 1, \ldots, n\},$$
$$B = \{b_{i,j} \mid i, j = 1, \ldots, 2n\}.$$

The set of hyperedges of H_n includes every binary edge from a vertex in K to a vertex in $K \cup S$ and every binary edge from a vertex in K' to a vertex in $K' \cup S'$:

$$\{\{k_i, k_j\} \mid i \neq j,\ i, j = 1, \ldots, 2n\},$$
$$\{\{k_i, s_j\} \mid i = 1, \ldots, 2n,\ j = 1, \ldots, n\},$$
$$\{\{k'_i, k'_j\} \mid i \neq j,\ i, j = 1, \ldots, 2n\},$$
$$\{\{k'_i, s'_j\} \mid i = 1, \ldots, 2n,\ j = 1, \ldots, n\}.$$

It also includes every binary edge[6] linking a vertex in B to a vertex in $K \cup K' \cup S \cup S'$:

$$\{\{b_{i,j}, k_l\} \mid i, j = 1, \ldots, 2n,\ l = 1, \ldots, 2n\},$$
$$\{\{b_{i,j}, k'_l\} \mid i, j = 1, \ldots, 2n,\ l = 1, \ldots, 2n\},$$
$$\{\{b_{i,j}, s_l\} \mid i, j = 1, \ldots, 2n,\ l = 1, \ldots, n\},$$
$$\{\{b_{i,j}, s'_l\} \mid i, j = 1, \ldots, 2n,\ l = 1, \ldots, n\},$$

The hypergraph H_n also contains the following (non-binary) hyperedges:

$$\{e_j = \{b_{i,j}, k_j, k_{n+j} \mid i = 1, 2, \ldots, 2n\} \mid j = 1, 2, \ldots, n\},$$
$$\{e'_i = \{b_{i,j}, k'_i, k'_{n+i} \mid j = 1, 2, \ldots, 2n\} \mid i = 1, 2, \ldots, n\},$$
$$\{f_i = \{b_{i+n,j}, s_i \mid j = 1, 2, \ldots, 2n\} \mid i = 1, 2, \ldots, n\},$$
$$\{f'_j = \{b_{i,j+n}, s'_j \mid i = 1, 2, \ldots, 2n\} \mid j = 1, 2, \ldots, n\}.$$

The hypergraph H_1 is illustrated in Fig. 5.

Now consider the following set of guarded blocks of H_n, which we will denote Ξ_n:

$$\{\langle\{e_1, \ldots, e_n, f'_1, \ldots, f'_n, \{k_i, s_j\}\}, B \cup K \cup \{s_j\}\rangle \mid i = 1, \ldots, 2n,\ j = 1, \ldots, n\}$$
$$\cup \{\langle\{e'_1, \ldots, e'_n, f_1, \ldots, f_n, \{k'_i, s'_j\}\}, B \cup K' \cup \{s'_j\}\rangle \mid i = 1, \ldots, 2n,\ j = 1, \ldots, n\}.$$

The set Ξ_1 is illustrated in Fig. 6.

The width of Ξ_n is clearly $2n + 1$, since each guard contains $2n + 1$ hyperedges. Furthermore, it is straightforward to verify that Ξ_n is an acyclic guarded cover for H_n which has a decomposition tree.

[6] Note that all of the edges of H_n defined so far are binary, which means that we have so far defined a graph G_n, which can be described in graph-theoretic terminology as follows. Let K_{2n} denote the complete graph on $2n$ vertices, S_n denote the empty graph on n vertices, B_{4n^2} denote the empty graph on $4n^2$ vertices, and let $G \bowtie G'$ denote the *graph join* of graphs G and G' (that is, the graph constructed by joining each vertex in G to each vertex in G'). Then we have that $G_n = ((K_{2n} \bowtie S_n) \uplus (K_{2n} \bowtie S_n)) \bowtie B_{4n^2}$.

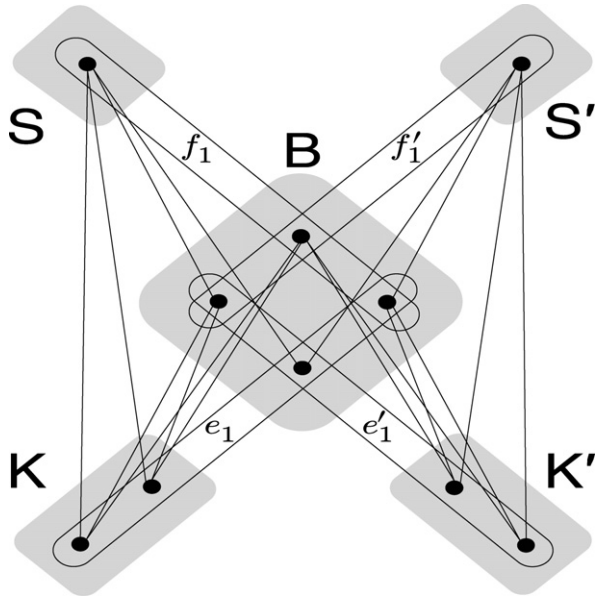

Fig. 5. The hypergraph H_1 defined in Example 7.13. (Edges totally contained in other edges are not shown.)

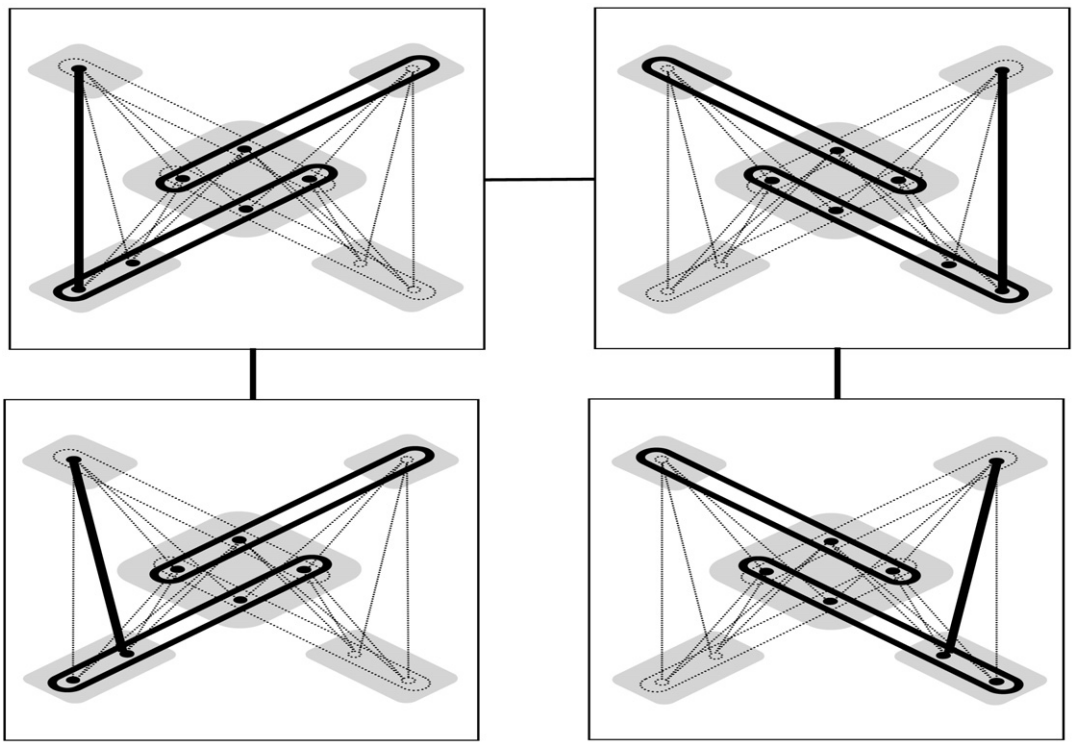

Fig. 6. The spread-cut decomposition Ξ_1 defined in Example 7.13 for the hypergraph H_1.

Now consider the χ-components and $(\bigcup \lambda)$-components of the guarded blocks $\langle \lambda, \chi \rangle \in \Xi$. By symmetry, we only need to do this for one guarded block, for example (with the choice $i = 1$ and $j = 1$):

$$\langle \lambda, \chi \rangle = \langle \{e_1, \ldots, e_n, f_1', \ldots, f_n', \{k_1, s_1\}\}, B \cup K \cup \{s_1\} \rangle.$$

With this choice of λ and χ, the $(\bigcup\lambda)$-components of H_n are $\{s_j\}$, $j = 2,\ldots,n$ and K'. The χ-components are $\{s_j\}$, $j = 2,\ldots,n$ and $K' \cup S'$. Hence each χ-component meets exactly one $(\bigcup\lambda)$-component, so this guarded block has unbroken components.

Now consider the labels of the elements of $(\bigcup\lambda)$:

$$
\begin{aligned}
&L_\lambda(s_1) = \langle \emptyset, \{e_i, \{k_1, s_1\}\}\rangle;\\
&L_\lambda(k_1) = \langle \{\{s_j\} \mid j = 2,\ldots,n\}, \{e_i, \{k_1, s_1\}\}\rangle;\\
&\forall v \in e_i \cap K - \{k_1\}, \quad L_\lambda(v) = \langle \{\{s_j\} \mid j = 2,\ldots,n\}, \{e_i\}\rangle;\\
&\forall v \in e_i \cap B, \quad L_\lambda(v) = \langle \{\{s_j\} \mid j = 2,\ldots,n\} \cup \{K'\}, \{e_i\}\rangle;\\
&\forall v \in f_i' \cap B, \quad L_\lambda(v) = \langle \{\{s_j\} \mid j = 2,\ldots,n\} \cup \{K'\}, \{f_i'\}\rangle;\\
&\forall v \in f_i' \cap S', \quad L_\lambda(v) = \langle \{K'\}, \{f_i'\}\rangle.
\end{aligned}
$$

For this example guarded block the vertices of $(\bigcup\lambda) - \chi$ are precisely the vertices of S'. We have shown that this guarded block respects labels.

Hence (by symmetry amongst the guarded blocks) we have shown that Ξ_n is a spread-cut decomposition of H_n of width $2n + 1$. However, we will show in Proposition 7.14 that the hypertree width of H_n is $3n$.

Proposition 7.14. *The hypertree width of the hypergraph H_n defined in Example* 7.13 *is exactly* $3n$.

Proof. We first define the following subsets of the edges of H_n:

$$
\begin{aligned}
F &\stackrel{\text{def}}{=} \{e_1,\ldots,e_n, f_1',\ldots,f_n'\},\\
F' &\stackrel{\text{def}}{=} \{e_1',\ldots,e_n', f_1,\ldots,f_n\}.
\end{aligned}
$$

The two edge-defined guarded blocks whose edge sets are $F \cup \{\{k_1, s_j\} \mid j = 1,\ldots,n\}$ and $F' \cup \{\{k_1', s_j'\} \mid j = 1,\ldots,n\}$ form an acyclic guarded cover of H_n of width $3n$. Since any edge-defined acyclic guarded cover has an associated hypertree, we have shown that the hypertree-width of H_n is at most $3n$.

We will now show that there are no width $3n - 1$ hypertree decompositions of H_n.

It has been shown (Proposition 18, p. 262 of Gottlob et al. [10]) that a hypergraph H has a hypertree decomposition of width k if and only if it has a *normal-form* hypertree decomposition of width k. The definition of a normal-form hypertree decomposition, given in Definition 17, p. 262 of Gottlob et al. [10], may be reformulated as follows: a hypertree decomposition $\langle \Xi, T\rangle$ is in *normal form* if T is a decomposition tree whose root is edge-defined, and whenever p is the parent of s in T, $\chi(s) = (\bigcup\lambda(s)) \cap (\chi(p) \cup \chi(\mathrm{br}_p(s)))$.

Let T be a decomposition tree of any width $3n - 1$ acyclic cover Ξ of H_n. It will be enough to prove that $\langle \Xi, T\rangle$ is *not* a normal-form hypertree decomposition of H_n. Hence in the rest of this proof we shall assume, for contradiction, that $\langle \Xi, T\rangle$ is a normal-form hypertree decomposition of H_n.

We first make two straightforward observations about the guarded blocks in Ξ:

- If $p \in \Xi$ and $B \not\subseteq \chi(p)$ then p is a leaf of T.
  This is because every vertex of H_n is directly connected to every vertex in $B - \chi(p)$, so H_n has at most one $\chi(p)$-component, and hence T has at most one branch out of p.
- If $p \in \Xi$ and $B \subseteq \chi(p)$, then either $F \subseteq \lambda(p)$ or $F' \subseteq \lambda(p)$.
  This is because if $F \not\subseteq \lambda(p)$, then there are at least $2n$ vertices of B not covered by edges of F. Each hyperedge of H_n not in F meets at most one of these vertices, so we must have at least $2n$ hyperedges in $\lambda(p) - F$, so $|\lambda(p) \cap F| \leqslant n - 1$. By symmetry, we also have that if $F' \not\subseteq \lambda(p)$, then $|\lambda(p) \cap F'| \leqslant n - 1$. It follows that there are at least $(n + 1)^2$ vertices of B not covered by $\lambda(p) \cap (F \cup F')$. Since $(n + 1)^2 \geqslant 3n$ for any $n \geqslant 1$, these vertices cannot be covered by the remaining edges of $\lambda(p)$, so $B \not\subseteq \chi(p)$.

Assume first that there is *no* guarded block of Ξ whose block includes B. In this case, each guarded block of Ξ is a leaf of T, by the observation above, so $|\Xi| \leqslant 2$. If we choose any $p \in \Xi$, then, by assumption, there is some $x \in B - \chi(p)$. All of the binary edges of H_n containing x must be covered in the only remaining guarded block, p',

of Ξ, so $\chi(p') \supseteq K \cup K' \cup S \cup S'$. However, $|K \cup K' \cup S \cup S'| = 6n$, no hyperedge of H_n contains more than two vertices of $K \cup K' \cup S \cup S'$, and $|\lambda(p)| \leqslant 3n - 1$, so it is impossible for any such Ξ to cover all edges.

Hence we may assume that some guarded block of Ξ includes all the vertices of B in its block. The nodes of Ξ with this property form a connected subtree of T, so we can choose a unique $p \in \Xi$ such that $B \subseteq \chi(p)$ and p has no ancestor with this property. By the observation above, and by symmetry, we can also assume without loss of generality that $F \subseteq \lambda(p)$.

We will now show that p is edge-defined. If p is the root of T then it is edge-defined by the definition of normal form. Otherwise the parent s of p is a leaf (by the choice of p and the fact that all non-leaves must include B in their block). So s must be the root of T, and p must be its unique descendant. In this case $\chi(p) = (\bigcup \lambda(p)) \cap (\chi(s) \cup \chi(\mathrm{br}_s(p)))$ because $\langle \Xi, T \rangle$ is in normal form. Since $\chi(s) \cup \chi(\mathrm{br}_s(p))$ includes all vertices of H_n, we again see that p is edge-defined.

The set $\lambda(p) - F$ contains at most $n - 1$ hyperedges, and so cannot cover all of K'. Let $y \in K' - \chi(p)$. None of the edges of H_n in the set $\{y\} \times (B \cup K' \cup S')$ are covered by p. Furthermore, these edges are $\chi(p)$-connected in H_n, and T is a join tree of an acyclic cover, so all of these edges are covered by guarded blocks in a single branch out of p. Let s be the neighbour of p in this branch.

If $B \not\subseteq \chi(s)$ then, as before, s is a leaf and this branch cannot cover all these edges, so T is not a cover. Hence we may assume that $B \subseteq \chi(s)$, which implies that either $F \subseteq \lambda(s)$ or $F' \subseteq \lambda(s)$, by the observation above.

Furthermore, since T is a join tree, there cannot be an edge from $\chi(\mathrm{br}_p(s)) - \chi(s)$ to $\chi(p) - \chi(s)$, so we know that $K' \subseteq \chi(s)$, and hence $F' \subseteq \chi(s)$.

The set $\lambda(s) - F'$ contains at most $n - 1$ hyperedges, and so cannot cover all of S'. Hence $\chi(s)$ cannot cover all hyperedges in $\{y\} \times S'$. Any such edge not covered joins $\chi(\mathrm{br}_p(s)) - \chi(s)$ to $\chi(p) - \chi(s)$, and so cannot be covered below s because T is a join-tree.

Hence the guarded blocks of Ξ cannot cover all edges of H_n, which gives the required contradiction. □

8. Conclusion and discussion

We have introduced the general notion of a guarded decomposition and shown how it can be used to describe many known tractable structural subproblems of the CSP. We have described a generic algorithm to calculate guarded decompositions and have given simple sufficient conditions on a class of guarded decompositions for the algorithm to be effective.

Using these results we were then able to define a new form of decomposition, the spread-cut decomposition, which sits in much the same place in the generalisation hierarchy as hypertrees. We have shown that spread-cut decompositions can be used to define tractable structural classes of the CSP and that for some classes of hypergraphs they allow us to obtain decompositions with a smaller width than any hypertree decomposition.

A number of open questions remain:

- Is there a family of hypergraphs for which the hypertree width is smaller than the width of the best spread-cut decomposition by some constant factor?
- Do hypertree decompositions *strongly* generalise the class of all edge-defined acyclic covers?
- Do spread-cut decompositions generalise the class of all edge-defined decompositions?
- Can the conditions on spread-cut decompositions be relaxed? In particular, does the existence of any acyclic guarded cover where every guarded block has unbroken components guarantee the existence of a spread-cut decomposition with the same width?

It is somewhat surprising that we can achieve nearly the same generalisation results as hypertrees with a decomposition that only allows polynomially many distinct guarded blocks for any hypergraph. More surprising still is that we have shown that these spread-cut decompositions can have arbitrarily smaller width than the best edge-defined decomposition. We believe that this is still an open question for hypertree decompositions.

References

[1] I. Adler, Marshals, monotone marshals, and hypertree-width, J. Graph Theory 47 (4) (2004) 275–296.

[2] I. Adler, G. Gottlob, M. Grohe, Hypertree-width and related hypergraph invariants, in: Proceedings of the 3rd European Conference on Combinatorics, Graph Theory and Applications, EUROCOMB '05, in: DMTCS Proceedings Series, vol. AE, 2005.
[3] C. Beeri, R. Fagin, D. Maier, M. Yannakakis, On the desirability of acyclic database schemes, J. ACM 30 (1983) 479–513.
[4] C. Chekuri, A. Rajaraman, Conjunctive query containment revisited, Theoret. Comput. Sci. 239 (2) (2000) 211–229.
[5] D. Cohen, M. Gyssens, P. Jeavons, A unified theory of structural tractability for constraint satisfaction and spread cut decomposition, in: Proceedings of IJCAI '05, 2005.
[6] R. Dechter, Constraint Processing, Morgan Kaufmann, 2003.
[7] R. Dechter, J. Pearl, Network-based heuristics for constraint satisfaction problems, Artificial Intelligence 34 (1) (1988) 1–38.
[8] R. Dechter, J. Pearl, Tree clustering for constraint networks, Artificial Intelligence 38 (1989) 353–366.
[9] E. Freuder, A sufficient condition for backtrack-bounded search, J. ACM 32 (1985) 755–761.
[10] G. Gottlob, N. Leone, F. Scarcello, A comparison of structural CSP decomposition methods, Artificial Intelligence 124 (2000) 243–282.
[11] G. Gottlob, N. Leone, F. Scarcello, Hypertree decompositions: A survey, in: Proceedings 26th International Symposium on Mathematical Foundations of Computer Science, MFCS' 01, in: Lecture Notes in Comput. Sci., vol. 2136, Springer-Verlag, 2001.
[12] G. Gottlob, N. Leone, F. Scarcello, Hypertree decomposition and tractable queries, J. Comput. System Sci. 64 (3) (2002) 579–627.
[13] G. Gottlob, N. Leone, F. Scarcello, Robbers, marshals, and guards: Game theoretic and logical characterizations of hypertree width, J. Comput. System Sci. 66 (2003) 775–808.
[14] M. Gyssens, P. Jeavons, D. Cohen, Decomposing constraint satisfaction problems using database techniques, Artificial Intelligence 66 (1) (1994) 57–89.
[15] P. Kolaitis, M. Vardi, Conjunctive-query containment and constraint satisfaction, J. Comput. System Sci. 61 (2000) 302–332.
[16] A. Mackworth, Consistency in networks of relations, Artificial Intelligence 8 (1977) 99–118.
[17] K. Marriott, P. Stuckey, Programming with Constraints, MIT Press, Cambridge, MA, 1998.
[18] F. Rossi, P. van Beek, T. Walsh (Eds.), The Handbook of Constraint Programming, Elsevier, 2006.