Decomposing Constraint Satisfaction Problems Using Database Techniques

Marc Gyssens

Department WNI
University of Limburg
B-3590 Diepenbeek, Belgium

Peter G. Jeavons
David A. Cohen

Department of Computer Science
Royal Holloway and Bedford New College
University of London
Egham, Surrey TW20 0EX, UK

March 22, 1994

Abbreviated Title

Decomposing CSPs Using Database Techniques

Correspondence Address

Peter Jeavons at the address above

Abstract

There is a very close relationship between constraint satisfaction problems and the satisfaction of join-dependencies in a relational database which is due to a common underlying structure, namely a hypergraph. By making that relationship explicit we are able to adapt techniques previously developed for the study of relational databases to obtain new results for constraint satisfaction problems. In particular, we prove that a constraint satisfaction problem may be decomposed into a number of subproblems precisely when the corresponding hypergraph satisfies a simple condition. We show that combining this decomposition approach with existing algorithms can lead to a significant improvement in efficiency.

1 Introduction

A number of important problems in artificial intelligence, graph theory and operational research may be formulated very naturally as *constraint satisfaction problems* [12, 20, 22, 26]: they involve putting together pieces of information (constraints) to obtain a global solution which simultaneously satisfies all of them. Recent examples include object recognition problems [15, 16], generalized graph coloring [21] and stock cutting problems [9]. On the other hand, one of the major problems of storing information in a database may be seen as the problem of breaking down global information into pieces of manageable size. The relational database model [5, 23, 31] has tackled this problem by storing a database as a number of projections from which the original relation may be reconstructed. The connection between constraint satisfaction problems and relational databases has been pointed out by a number of authors [4, 8, 33], and results obtained in the field of constraint satisfaction problems have been used to obtain new results on database relations [8].

Here, we intend to investigate the interaction between the two fields in the opposite direction, which we believe to be very promising. In order to do this we show precisely how a constraint satisfaction problem may be represented as a relational database. Establishing this connection immediately allows all the terminology and techniques of relational database theory to be applied to the description and analysis of constraint satisfaction problems. The principal aim of this paper is to demonstrate that applying a number of powerful database techniques not previously used in this area leads to a significant improvement in our understanding of constraint satisfaction problems.

More concretely, we show that techniques for the decomposition of join-dependencies in a relational database [17, 18, 19] may be adapted to constraint satisfaction problems to provide a decomposition strategy. We exhibit a method of decomposition derived from these database techniques which involves clustering together constraints of the original problem, to form subproblems which are simply related. The solutions to the subproblems may then be efficiently combined to solve the original problem. We also show that the decomposition obtained by this method is fundamental, in the sense that if the subproblems are decomposed further then the new subproblems obtained will no longer always be simply related.

The tractability of a constraint satisfaction problem is known to be related to the structure of the associated constraint hypergraph [6, 14, 29], and there have been a number of attempts to describe conditions on this associated hypergraph which ensure that a constraint satisfaction problem can be solved efficiently. For example, Seidel demonstrated that in some cases small numbers of variables may be used to separate a problem into solved and unsolved pieces during the solution process [29], and Freuder has shown that some simple classes of binary constraint satisfaction problems may be solved using backtrack-free [13] or backtrack-bounded [14] tree search. All of this work has been used by Dechter and Pearl as the basis for a heuristic scheme which attempts to improve the efficiency of finding a solution by decomposing a constraint satisfaction problem into a number of simply-related subproblems [7]. However, although this solution strategy is efficient in many cases, it remains a heuristic scheme, and in some cases it leads to very inefficient decompositions. We show in this paper that combining this previous work with the fundamental decomposition algorithm derived here results in a new algorithm with a significantly improved worst-case complexity bound.

The paper is organized as follows. In Section 2 we review the necessary terminology for constraint satisfaction problems, relational database theory and hypergraphs, and establish the precise connections between these notions. Then, in Section 3, we show how the structure of the underlying hypergraph may be used to obtain a fundamental decomposition of a constraint satisfaction problem. Finally, in Section 4, we show how this decomposition is related to existing heuristic solution strategies and may be combined with one of these to obtain a solution strategy with a better worst-case complexity bound.

2 Constraint satisfaction problems and databases

2.1 Constraint satisfaction problems

A *constraint satisfaction problem* [12, 20, 22, 26] consists of a number of *variables* which must be assigned values from associated *domains*, subject to a number of *constraints*. Each constraint specifies allowed combinations of values for some subset of the variables, referred to as the *scope* of the constraint. We now give a formal definition:

Definition 2.1 *A* constraint satisfaction problem *is a sextuple* $\mathcal{P} = (X, \Delta, \delta, C, \Sigma, \sigma)$ *where*

- δ *is a mapping from the finite set of variables* X *onto the set of sets* Δ*; for each* $x \in X$, $\delta(x)$ *is called the* domain *of* x*;*
- σ *is a one-to-one mapping from the finite set of constraints* C *onto the set of sets* Σ *satisfying* $\bigcup \Sigma = X$*; for each* $c \in C$, $\sigma(c)$ *is called the* scope *of* c*.*

A mapping t *from* $Y \subseteq X$ *into* $\bigcup \Delta$ *such that* $t(x) \in \delta(x)$*, for all* $x \in Y$*, is called a* labeling *of* Y*. Each* constraint $c \in C$ *is a set of labelings of* $\sigma(c)$*.*

The mappings δ and σ are extended in the natural way to subsets of X and C respectively.

Definition 2.2 *Let* $\mathcal{P} = (X, \Delta, \delta, C, \Sigma, \sigma)$ *be a constraint satisfaction problem.*

- *Given any constraint* $c \in C$ *a labeling* t *of* $\sigma(c)$ *is said to* satisfy c *if* $t \in c$*.*
- *A labeling* t *of* X *is said to be a* solution *to* $\mathcal{P}$ *if, for every* $c \in C$*, the restriction of* t *to* $\sigma(c)$ *satisfies* c*. The set of all solutions to* $\mathcal{P}$ *is denoted* $Sol(\mathcal{P})$*.*

To illustrate these definitions, we now give an example of a specific constraint satisfaction problem, which will be used as a running example.

Example 2.3 Let $\mathcal{P} = (X, \Delta, \delta, C, \Sigma, \sigma)$ be the constraint satisfaction problem in which

- $X = \{x_0, \ldots, x_9\}$;
- $\Delta = \{\{0, 1, 2, \ldots\}\}$;
- $\delta(x_i) = \{0, 1, 2, \ldots\},\ i = 0, 1, \ldots, 9$;
- $C = \{c_1, \ldots, c_8\}$;
- $\Sigma = \{s_1, \ldots, s_8\}$ where the values of each s_i are as shown in Table 2.1;
- $\sigma(c_i) = s_i,\ i = 1, 2, \ldots, 8$.

Table 2.1: The scopes of the constraints in C (Example 2.3).

s_1	=	$\{x_0, x_1, x_3\}$	s_5	=	$\{x_4, x_5, x_6\}$
s_2	=	$\{x_1, x_2, x_3\}$	s_6	=	$\{x_4, x_7\}$
s_3	=	$\{x_1, x_4\}$	s_7	=	$\{x_5, x_8\}$
s_4	=	$\{x_3, x_6\}$	s_8	=	$\{x_6, x_9\}$

Table 2.2: The constraints in C (Example 2.3).

$c_1 = \{(0,0,0), (0,1,0), (1,0,1), (1,1,1), (0,1,2)\}$	$c_5 = \{(0,0,0), (0,0,1), (1,1,1), (1,0,2)\}$
$c_2 = \{(0,0,0), (0,0,1), (1,1,0), (1,0,1), (0,1,2)\}$	$c_6 = \{(0,1), (1,0)\}$
$c_3 = \{(0,0), (1,1)\}$	$c_7 = \{(0,1), (1,0), (1,1)\}$
$c_4 = \{(0,0), (1,1), (1,0), (2,0)\}$	$c_8 = \{(0,0), (1,1)\}$

Using the natural subscript ordering for the variables in each scope, we define the constraints of $\mathcal{P}$ to be as shown in Table 2.2. A simple exhaustive search shows that $\mathrm{Sol}(\mathcal{P})$ is composed of the five solutions shown in Table 2.3. □

Table 2.3: The solutions to $\mathcal{P}$ (Example 2.3).

$(0,0,0,0,0,0,0,1,1,0)$
$(1,0,0,1,0,0,0,1,1,0)$
$(1,0,0,1,0,0,1,1,1,1)$
$(1,1,0,1,1,1,1,0,0,1)$
$(1,1,0,1,1,1,1,0,1,1)$

We will often want to deal with *subproblems* of a given constraint satisfaction problem which arise from considering subsets of the set of constraints:

Definition 2.4 *Let $\mathcal{P} = (X, \Delta, \delta, C, \Sigma, \sigma)$ be a constraint satisfaction problem, and let D be any subset of C. The* subproblem of $\mathcal{P}$ generated by D *is the constraint satisfaction problem $\mathcal{P}|_D = (X|_D, \delta(X|_D), \delta|_{X|_D}, D, \sigma(D), \sigma|_D)$, where $X|_D = \bigcup_{c \in D} \sigma(c)$.*

If t is a solution to $\mathcal{P}$, then, obviously, the restriction of t to $X|_D$ is a solution to $\mathcal{P}|_D$. Conversely, we say that a solution t' to $\mathcal{P}|_D$ can be *extended* to a solution to $\mathcal{P}$ if there exists a solution t to $\mathcal{P}$ such that t' is the restriction of t to $X|_D$.

Example 2.5 Reconsider the constraint satisfaction problem $\mathcal{P}$ of Example 2.3 and choose $D_1 = \{c_2, c_3, c_4, c_5\}$ as a subset of the constraints. Then $X|_{D_1} = \{x_1, x_2, x_3, x_4, x_5, x_6\}$. The set of solutions $\mathrm{Sol}(\mathcal{P}|_{D_1})$ is given in Table 2.4. Note that the last solution shown in Table 2.4 cannot be extended to a solution to $\mathcal{P}$. □

Finally, if $\mathcal{P} = (X, \Delta, \delta, C, \Sigma, \sigma)$ and $\tilde{\mathcal{P}} = (X, \Delta, \delta, \tilde{C}, \tilde{\Sigma}, \tilde{\sigma})$ are two constraint satisfaction problems over the same set of variables, then $\mathcal{P}$ and $\tilde{\mathcal{P}}$ are said to be *equivalent* if $\mathrm{Sol}(\mathcal{P}) = \mathrm{Sol}(\tilde{\mathcal{P}})$.

Table 2.4: The solutions to $\mathcal{P}|_{D_1}$ (Example 2.5).

$$(0,0,0,0,0,0)$$
$$(0,0,1,0,0,0)$$
$$(0,0,1,0,0,1)$$
$$(1,0,1,1,1,1)$$
$$(0,1,2,0,0,0)$$

2.2 Relations and databases

We now turn our attention to the relational database model, initially described by Codd [5]. Good introductions to relational database theory can be found in [23, 27, 31]. Within this model a database is a finite set of *relations*:

Definition 2.6 *A* relation *consists of a* relation scheme *and a* relation instance*:*

- *A* relation scheme *is a finite set of* attributes*. Each attribute is associated with a (possibly infinite) set of values, called its* domain*;*
- *A* tuple *over a relation scheme is a mapping that associates with each attribute of the relation scheme a value from its corresponding domain;*
- *A* relation instance *over a relation scheme is a finite set of tuples over that relation scheme.*

Intuitively, a relation scheme represents the structure of a relation and a relation instance its contents.

There exists a close relationship between constraint satisfaction problems and databases. For any constraint satisfaction problem $\mathcal{P} = (X, \Delta, \delta, C, \Sigma, \sigma)$, the following connections can be established:

- The variables in X can be interpreted as attributes;
- The domains in Δ are the domains of these attributes;
- A labeling of a subset $Y \subseteq X$ of variables is a tuple over the relation scheme with set of attributes Y.

Two alternative representations of $\mathcal{P}$ as a database now seem natural:

- the database consisting of a single relation with scheme X and instance $\mathrm{Sol}(\mathcal{P})$;
- the database $\{R_c \mid c \in C\}$ where R_c has scheme $\sigma(c)$ and instance c.

In order to describe the relationship between both representations of a constraint satisfaction problem as a database, we need two operators from Codd's relational algebra [5].

Definition 2.7 *Let Y, Z be sets of attributes with $Z \subseteq Y$. Let t be a tuple over Y, and let r be a relation instance over Y. The* projection *onto Z of t, denoted $t[Z]$, is the restriction of t to Z. The* projection *onto Z of r, denoted $\pi_Z(r)$, is the set $\{t[Z] \mid t \in r\}$.*

Definition 2.8 *Let $Y_1, \ldots, Y_k$ be sets of attributes, and let r_i be a relation instance over Y_i, for $i = 1, \ldots, k$. Let $Y = \bigcup_{i=1}^{k} Y_i$. The* join *of $r_1, \ldots, r_k$, denoted $r_1 \bowtie \cdots \bowtie r_k$, is the set $\{t \mid t$ is a tuple over Y & $t[Y_i] \in r_i$ for $i = 1, \ldots, k\}$.*

It follows from these definitions that the set of all solutions to a constraint satisfaction problem is equal to the join of the relation instances corresponding to the constraints. In other words, the join operation corresponds precisely to the notion of composition of constraints in a constraint satisfaction problem. This fact has an important consequence which can be expressed using the notion of a *join-dependency* in database theory.

Definition 2.9 *A* join-dependency [28] *over Y is an expression $Y_1 \bowtie \cdots \bowtie Y_k$ where $Y_1, \ldots, Y_k$ are sets of attributes with $Y = \bigcup_{i=1}^{k} Y_i$. The sets $Y_1, \ldots, Y_k$ are called the* edges *of the join-dependency. A relation instance r over Y* satisfies *the join-dependency if $r = \pi_{Y_1}(r) \bowtie \cdots \bowtie \pi_{Y_k}(r)$.*

Satisfying a join-dependency $Y_1 \bowtie \cdots \bowtie Y_k$ is a necessary and sufficient condition for the contents of a relation to be storable as its projections onto $Y_1, \ldots, Y_k$, and to be recoverable by performing the corresponding join. Join-dependencies play a major role in relational database theory, since, in general, it is much more efficient to store a database as a set of relations, rather than as a single relation [11]. It is clear from Definition 2.9 that the set of all solutions to a constraint satisfaction problem satisfies the join-dependency whose edges are the scopes of the constraints.

Obviously, the constraints of a constraint satisfaction problem are supersets of the appropriate projections of the set of all solutions. If all constraints *equal* the respective projections of the set of all solutions, then the set of constraints is said to be a set of *minimal* constraints [26]. This is equivalent to saying that, for each constraint, each member of that constraint can be extended to a solution to the full problem.

Example 2.10 Reconsider the constraint satisfaction problem $\mathcal{P}$ of Example 2.3. The projection of $\mathrm{Sol}(\mathcal{P})$ onto s_1, the scope of c_1, is shown in Table 2.5.

Table 2.5: The projection of $\mathrm{Sol}(\mathcal{P})$ onto s_1 (Example 2.10).

$R_{\tilde{c}_1}$		
x_0	x_1	x_3
0	0	0
1	0	1
1	1	1

As this projection does not equal c_1 we have shown that C is not a set of minimal constraints.

We now construct a constraint satisfaction problem $\tilde{\mathcal{P}} = (X, \Delta, \delta, \tilde{C}, \Sigma, \tilde{\sigma})$ that is equivalent to $\mathcal{P}$, but with $\tilde{C} = \{\tilde{c}_1, \ldots, \tilde{c}_8\}$ a set of minimal constraints. For each $i = 1, \ldots, 8$, the scope of $\tilde{c}_i$ is equal to the scope of c_i; the value of $\tilde{c}_i$, is equal to the corresponding projection of $\mathrm{Sol}(\mathcal{P})$. The constraints obtained in this way are shown in Table 2.6 as a relational database. This construction can always be used to find a problem with a set of minimal constraints, which is equivalent to a given problem. Unfortunately, it involves calculating all the solutions to the given problem and so is generally very time-consuming. □

Table 2.6: The constraints of $\tilde{\mathcal{P}}$ as a relational database (Example 2.10).

$R_{\tilde{c}_1}$			$R_{\tilde{c}_2}$			$R_{\tilde{c}_3}$		$R_{\tilde{c}_4}$		$R_{\tilde{c}_5}$			$R_{\tilde{c}_6}$		$R_{\tilde{c}_7}$		$R_{\tilde{c}_8}$	
x_0	x_1	x_3	x_1	x_2	x_3	x_1	x_4	x_3	x_6	x_4	x_5	x_6	x_4	x_7	x_5	x_8	x_6	x_9
0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	0	0
1	1	1	0	0	1	1	1	1	0	0	0	1	1	0	1	0	1	1
1	0	1	1	0	1			1	1	1	1	1			1	1		

Minimality of a set of constraints is a crucial notion in the theory of constraint satisfaction problems. In fact, Montanari [26] designated the problem of finding a set of minimal constraints as the key issue in dealing with a constraint satisfaction problem.

The notion of minimality of the constraints of a constraint satisfaction problem can be translated to the comparably important notion of *consistency* of a relational database:

Definition 2.11 *Let $Y_1, \ldots, Y_k$ be sets of attributes, and let $r_1, \ldots, r_k$ be relation instances over $Y_1, \ldots, Y_k$ respectively. Then $\{r_1, \ldots, r_k\}$ is said to be* consistent *if, for $i = 1, \ldots, k$, $\pi_{Y_i}(r_1 \bowtie \cdots \bowtie r_k) = r_i$.*

Since the set of all solutions to a constraint satisfaction problem is precisely the join of the relation instances corresponding to the constraints, the constraints are minimal if and only if the corresponding set of relation instances is consistent.

2.3 Hypergraphs

It turns out that constraint satisfaction problems and databases with join-dependencies have a common underlying structure, namely a hypergraph:

Definition 2.12 [3] *A* hypergraph *is an ordered pair (V, E) where V is a finite set of vertices and E is a set of edges, each of which is a subset of V.*

Undirected graphs can be seen as special cases of hypergraphs, where each edge contains exactly two vertices.

The underlying structure of a constraint satisfaction problem $\mathcal{P} = (X, \Delta, \delta, C, \Sigma, \sigma)$ is the hypergraph (X, Σ), the edges of which are the scopes of the constraints.

Example 2.13 Reconsider the constraint satisfaction problem $\mathcal{P}$ of Example 2.3. The hypergraph (X, Σ) associated with $\mathcal{P}$ is shown in Figure 2.1. □

Similarly, the underlying structure of a database with a join-dependency specified on it can also be described by a hypergraph the edges of which are the edges of the join-dependency.

Now, let (V, E) be a hypergraph, let $H \subseteq E$, and let $F \subseteq E - H$. F is called *connected with respect to H* if, for any two edges $e, f \in F$, there exists a sequence $e_1, \ldots, e_n$ of edges in F such that *(i)* $e_1 = e$; *(ii)* for $i = 1, \ldots, n-1$, $e_i \cap e_{i+1}$ is not contained in $\bigcup H$; and *(iii)* $e_n = f$. The maximal connected subsets of $E - H$ with respect to H are called the *connected components of $E - H$ with respect to H*. They obviously form a partition of $E - H$. In the case that H is the empty set of edges, we usually drop the phrase "with

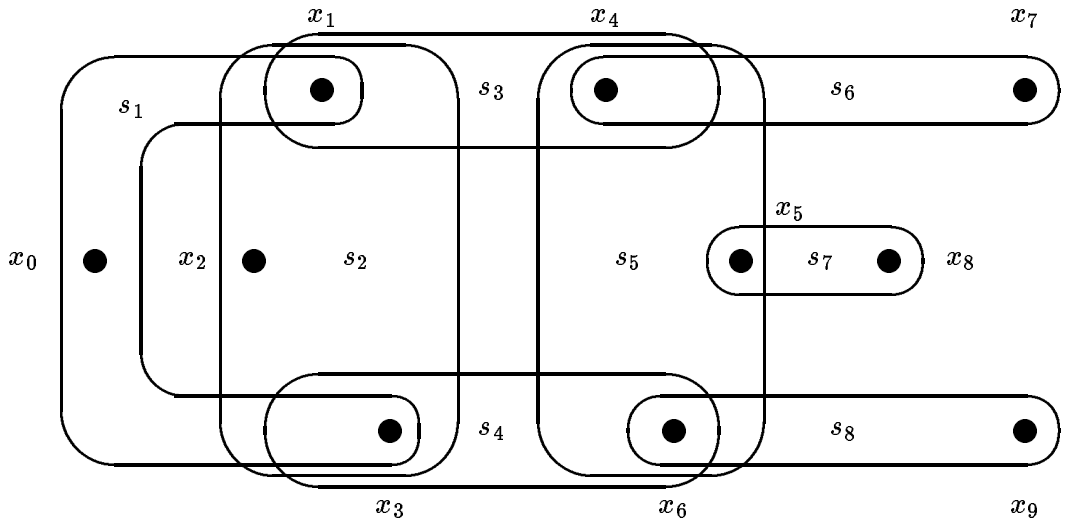

Figure 2.1: The hypergraph associated with $\mathcal{P}$ (Example 2.13).

respect to H" and speak about *connected* subsets and *connected components* of E. If E itself is connected, then (V, E) is said to be a *connected* hypergraph.

The *reduction* of a hypergraph is obtained by removing each edge that is properly contained in another edge. A hypergraph is called *reduced* if it equals its reduction, i.e., if none of its edges is properly contained in any other.

For the remainder of this paper, we shall only consider constraint satisfaction problems whose underlying hypergraph is connected and reduced. This is not a very heavy restriction. Indeed, if the hypergraph associated with a constraint satisfaction problem were not connected, then the subproblems generated by the connected components have disjoint sets of variables and may be solved independently. Also, if the hypergraph of a constraint satisfaction problem is not reduced, it is straightforward to construct reasonably efficiently an equivalent constraint satisfaction problem whose associated hypergraph is the reduction of the original one.

Since hypergraphs are a generalization of graphs and since acyclic graphs are an important subclass of the latter, it is natural to define *acyclic hypergraphs*. Thereto, we need to introduce two preliminary notions. Let (V, E) be a connected and reduced hypergraph, and let $F \subseteq E$ be a connected set of edges. A pair of edges $\{f, g\} \subseteq F$ is called an *articulation pair* of F, and $W = f \cap g$ is called an *articulation set* of F, if the set of modified edges $\{e - W \mid e \in F\}$ is not connected. (Clearly, an articulation set of a hypergraph is a generalization of an articulation point of a graph.) F is called *closed* if for every edge $e \in E$ there exists an edge $e' \in F$ such that $e \cap (\bigcup F) \subseteq e'$. A connected and reduced hypergraph is acyclic if every closed connected set of edges consisting of at least two elements has an articulation set.

Using the definition, it may be easily verified that the hypergraph in Figure 2.1, Example 2.13, is *not* acyclic. Hypergraphs that are not acyclic are called *cyclic*.

Acyclic hypergraphs, acyclic join-dependencies (join-dependencies of which the corresponding hypergraph is acyclic), and acyclic databases (databases described by acyclic join-dependencies) have received wide attention [1, 2, 10, 11]. The definition given above is only one of many equivalent characterizations of acyclicity, which demonstrate the desirability of the notion.

3 Structure of constraint satisfaction problems

In [17, 18, 19], Gyssens and Paredaens studied the structure of hypergraphs in the context of join-dependencies in relational databases. Many problems connected with join-dependencies are computationally hard. Standard algorithms for checking satisfaction of a join-dependency (in the case that all information is stored in one relation) and consistency checking (in the case that the information is distributed over several relations) take exponential time in the number of edges of the join-dependency. In order to overcome these problems, an algorithm was developed to decompose join-dependencies as far as possible into smaller ones (i.e., into join-dependencies with fewer edges). Recently, some of these results were generalized by Miller et al. [24, 25] to the case in which functional dependencies are also present.

Having established the close connection between relational database theory and constraint satisfaction problems, we may now adapt the decomposition technique developed for join-dependencies in order to obtain a decomposition technique for constraint satisfaction problems. This decomposition will then provide a method for establishing whether the constraints in a given problem are minimal, calculating a set of minimal constraints which is equivalent, or finding the solutions.

The central notion in the decomposition algorithm is that of a *hinge* in a hypergraph:

Definition 3.1 [17] *Let (V, E) be a reduced and connected hypergraph, and let H be either E or a proper subset of E containing at least two edges. Let $H_1, \ldots, H_m$ be the connected components of $E - H$ with respect to H. Then H is called a* hinge *if, for $i = 1, \ldots, m$, there exists an edge h_i in H such that*

$$\left(\bigcup H_i\right) \cap \left(\bigcup H\right) \subseteq h_i.$$

The edge h_i is called a separating edge *for H_i.*

We shall call a proper subset (of a given set) containing at least two elements a *non-trivial subset*. Using this terminology, a hinge of a hypergraph is either the entire hypergraph or a non-trivial subset of its edges such that each connected component with respect to that subset intersects it within one of its edges.

Example 3.2 Reconsider the hypergraph in Figure 2.1, Example 2.13. In Table 3.7, we list all the hinges of this hypergraph. □

From [17], we mention that a hinge of a reduced and connected hypergraph is always a connected node-generated set of edges.

We now establish two theorems which demonstrate that hinges correspond to natural subproblems of a constraint satisfaction problem. Although these theorems deal with sets of minimal constraints, we will show that they can be used to develop a decomposition strategy that can be applied to arbitrary constraint satisfaction problems.

First, if a set of constraints is minimal then every solution to the subproblem generated by a hinge can be extended to the entire hypergraph.

Theorem 3.3 *Let $\mathcal{P} = (X, \Delta, \delta, C, \Sigma, \sigma)$ be a constraint satisfaction problem with C a set of minimal constraints, and let D be a subset of C.*

If $\sigma(D)$ is a hinge in the hypergraph (X, Σ), then any solution to $\mathcal{P}|_D$ can be extended to a solution to $\mathcal{P}$.

Table 3.7: All hinges of the hypergraph in Figure 2.1.

$\{s_1, s_2\}$	$\{s_2, s_3, s_4, s_5\}$
$\{s_1, s_2, s_3, s_4, s_5\}$	$\{s_2, s_3, s_4, s_5, s_6\}$
$\{s_1, s_2, s_3, s_4, s_5, s_6\}$	$\{s_2, s_3, s_4, s_5, s_7\}$
$\{s_1, s_2, s_3, s_4, s_5, s_7\}$	$\{s_2, s_3, s_4, s_5, s_8\}$
$\{s_1, s_2, s_3, s_4, s_5, s_8\}$	$\{s_2, s_3, s_4, s_5, s_6, s_7\}$
$\{s_1, s_2, s_3, s_4, s_5, s_6, s_7\}$	$\{s_2, s_3, s_4, s_5, s_6, s_8\}$
$\{s_1, s_2, s_3, s_4, s_5, s_6, s_8\}$	$\{s_2, s_3, s_4, s_5, s_7, s_8\}$
$\{s_1, s_2, s_3, s_4, s_5, s_7, s_8\}$	$\{s_2, s_3, s_4, s_5, s_6, s_7, s_8\}$
$\{s_1, s_2, s_3, s_4, s_5, s_6, s_7, s_8\}$	$\{s_5, s_6\}$
$\{s_1, s_3, s_4, s_5\}$	$\{s_5, s_7\}$
$\{s_1, s_3, s_4, s_5, s_6\}$	$\{s_5, s_8\}$
$\{s_1, s_3, s_4, s_5, s_7\}$	$\{s_5, s_6, s_7\}$
$\{s_1, s_3, s_4, s_5, s_8\}$	$\{s_5, s_6, s_8\}$
$\{s_1, s_3, s_4, s_5, s_6, s_7\}$	$\{s_5, s_7, s_8\}$
$\{s_1, s_3, s_4, s_5, s_6, s_8\}$	$\{s_5, s_6, s_7, s_8\}$
$\{s_1, s_3, s_4, s_5, s_7, s_8\}$	
$\{s_1, s_3, s_4, s_5, s_6, s_7, s_8\}$	

Proof: Let $\sigma(D)$ be a hinge in (X, Σ). If $D = C$ then the result holds trivially. Otherwise, let $H_1, \ldots, H_m$ be the connected components of $\Sigma - \sigma(D)$ with respect to $\sigma(D)$, and let $h_1, \ldots, h_m$ be corresponding separating edges in $\sigma(D)$, as in Definition 3.1. Now let t' be a solution to $\mathcal{P}|_D$. For all $i = 1, \ldots, m$, the restriction of t' to h_i can be extended to a solution t_i to $\mathcal{P}$, because of the minimality of the constraints in C. We now define a labeling t of X as follows: for all $x \in X$,

$$t(x) = \begin{cases} t'(x) & \text{if } x \text{ occurs in } \sigma(D) \\ t_i(x) & \text{if } x \text{ occurs in } H_i \text{ but not in } \sigma(D) \end{cases}$$

In order to prove the theorem, it suffices to show that t is a solution to $\mathcal{P}$, i.e., that t satisfies all constraints in C. For constraints in D, this is trivially the case. Therefore, let $c \notin D$, and let H_i be the unique connected component of $\Sigma - \sigma(D)$ with respect to $\sigma(D)$ to which $\sigma(c)$ belongs. By construction, $t[\sigma(c) - X|_D] = t_i[\sigma(c) - X|_D]$ and $t[h_i] = t_i[h_i]$. Since $\sigma(D)$ is a hinge, $\sigma(c) \cap X|_D \subseteq h_i$, whence also $t[\sigma(c)] = t_i[\sigma(c)]$, implying that c is satisfied. ∎

Example 3.4 Consider the constraint satisfaction problem, $\widetilde{\mathcal{P}}$ described in Example 2.10, which has a set of minimal constraints $\widetilde{C}$. We invite the reader to verify that any solution to the subproblem generated by any of the hinges listed in Example 3.2 can be extended to a solution to $\widetilde{\mathcal{P}}$.

On the other hand, consider the equivalent constraint satisfaction problem $\mathcal{P}$ of Example 2.3 and let $D_1 = \{c_2, c_3, c_4, c_5\}$. By Example 3.2, $\sigma(D_1)$ is a hinge of the hypergraph (X, Σ) associated with $\mathcal{P}$. However, not every solution to $\mathcal{P}|_{D_1}$ can be extended to a solution to $\mathcal{P}$, as was shown in Example 2.5. Hence, by Theorem 3.3 the constraints in the set C of $\mathcal{P}$ are not all minimal. □

Conversely, if every solution on a non-trivial subset of edges can be extended to the entire hypergraph for all appropriate minimal sets of constraints, then that subset is a hinge.

Theorem 3.5 *Let X be a set of variables, and let Δ be a set of domains each of which contains at least as many values as there are variables in X. Let Σ be a set such that $\bigcup \Sigma = X$, and let Σ' be a non-trivial subset of Σ.*

If for each constraint satisfaction problem $\mathcal{P} = (X, \Delta, \delta, C, \Sigma, \sigma)$, with C a set of minimal constraints, each solution to $\mathcal{P}|_{\sigma^{-1}(\Sigma')}$ can be extended to a solution of $\mathcal{P}$, then Σ' is a hinge of the hypergraph (X, Σ).

Proof: Assume on the contrary that Σ' is *not* a hinge of (X, Σ). We shall exhibit a constraint satisfaction problem $\mathcal{P} = (X, \Delta, \delta, C, \Sigma, \sigma)$ with C a set of minimal constraints, together with a solution t' to $\mathcal{P}|_{\sigma^{-1}(\Sigma')}$ that cannot be extended to a solution to $\mathcal{P}$.

Thereto, let $H_1, \ldots, H_m$ be the connected components of $\Sigma - \Sigma'$ with respect to Σ'. Since Σ' is not a hinge, at least one of these connected components does not intersect Σ' within a single edge of Σ'. Without loss of generality, let H_1 be such a connected component. Suppose H_1 and Σ' have k vertices in common, say $x_0, \ldots, x_{k-1}$. We select k values in each domain of Δ (this is possible by assumption), which, without loss of generality, we will identify as $0, \ldots, k-1$.

A labeling t of X will be called *legal* if, for some i with $0 \leq i \leq k-1$,

1. for all $j = 0, \ldots, i-1, i+1, \ldots, k-1$, $t(x_j) = j$;
2. $t(x_i) = t(x_{(i-1) \bmod k})$;
3. for all $x \in (\bigcup H_1) - (\bigcup \Sigma')$, $t(x) = t(x_i)$.

The value i will be called the *type* of the legal labeling.

Now let $\mathcal{P} = (X, \Delta, \delta, C, \Sigma, \sigma)$ be any constraint satisfaction problem in which C is the set of projections onto the scopes of Σ of the set of all legal labelings of X.

We first show that $\mathrm{Sol}(\mathcal{P})$ is precisely the set of legal labelings of X. By construction, each legal labeling of X is a solution to $\mathcal{P}$. Conversely, let t be a solution to $\mathcal{P}$. Then, for each $s \in H_1$, the projection, $t[s]$, of t onto s is in $\sigma^{-1}(s)$, whence there exists a legal labeling t_s of X such that $t_s[s] = t[s]$. We now distinguish two cases:

1. *For some $s \in H_1$, $s - (\bigcup \Sigma') = \emptyset$.* Since H_1 is connected with respect to Σ', we then must have that $H_1 = \{s\}$ with $s = \{x_0, \ldots, x_{k-1}\}$. Since $t_s[s] = t[s]$, t is a legal labeling of X, of the same type as t_s.
2. *For all $s \in H_1$, $s - (\bigcup \Sigma') \neq \emptyset$.* Then, for each $s \in H_1$, the type of t_s is fully determined by the common value of the variables in $s - (\bigcup \Sigma')$. Since, moreover, H_1 is connected with respect to Σ', this common value is the same for all $s \in H_1$. Consequently, all t_s, $s \in H_1$, have the same type, say i. Then, obviously, t is also a legal labeling of X of type i.

Hence, in all cases $C = \{\pi_s(\mathrm{Sol}(\mathcal{P})) \mid s \in \Sigma\}$, so C is a set of *minimal* constraints.

Now let t' be any labeling of $\bigcup \Sigma'$ satisfying $t'(x_i) = i$ for all $i = 0, \ldots, k-1$. Obviously, t' *cannot* be extended to a legal labeling of X. The proof will now be completed by showing that t' is a solution to $\mathcal{P}|_{\sigma^{-1}(\Sigma')}$.

Thereto, let s be an arbitrary scope of Σ'. Since Σ' is not a hinge, at least one of the variables $x_0, \ldots, x_{k-1}$, say x_i, is *not* in s. Hence $t'[s]$ can be extended to a legal labeling of X of type i, whence $t'[s] \in \sigma^{-1}(s)$. ∎

In general, hinges can in turn contain other hinges. We are most interested in hinges that do *not* contain other hinges; these are called *minimal hinges.*

Example 3.6 Reconsider the hypergraph in Figure 2.1, Example 2.13. Of the 32 hinges listed in Table 3.7, Example 3.2, only the six listed in Table 3.8 are minimal. □

Table 3.8: All minimal hinges of the hypergraph in Figure 2.1.

$$
\begin{aligned}
M_1 &= \{s_1, s_2\} \\
M_4 &= \{s_5, s_6\} \\
M_2 &= \{s_1, s_3, s_4, s_5\} \\
M_5 &= \{s_5, s_7\} \\
M_3 &= \{s_2, s_3, s_4, s_5\} \\
M_6 &= \{s_5, s_8\}
\end{aligned}
$$

It turns out that a hypergraph can be covered by a "tree" of minimal hinges, as defined below.

Definition 3.7 *Let (V, E) be any hypergraph. A* hinge-tree *of (V, E) is a tree*[1]*, (N, A), with nodes N and labeled arcs*[2] *A, such that:*

1. *The tree nodes are minimal hinges of (V, E).*
2. *Each edge in E is contained in at least one tree node.*
3. *Two adjacent tree nodes share precisely one edge of E which is also the label of their connecting tree-arc; moreover, their shared vertices are precisely the members of this edge.*
4. *The vertices of V shared by two tree nodes are entirely contained within each tree node on their connecting path.*

Example 3.8 Reconsider the hypergraph (X, Σ) in Figure 2.1, Example 2.13. Let $\{M_1, \ldots, M_6\}$ be the set of all its minimal hinges, as shown in Table 3.8, Example 3.6. The reader is invited to check that the tree shown in Figure 3.2 is a hinge-tree of (X, Σ).

It should be noted that this hinge-tree does not contain all of the minimal hinges of (X, Σ), nor is it the only possible hinge-tree for (X, Σ). A different hinge-tree may be obtained by replacing the node M_3 with M_2 and relabeling the edge joining it to M_1. □

We shall now exhibit an algorithm that, given a hypergraph (V, E), computes a hinge-tree T for that hypergraph.

[1] By "tree", we understand an unrooted tree, i.e., an undirected acyclic graph.

[2] A labeled arc is formally defined as an ordered pair $(\{v, w\}, a)$ with v and w different tree nodes and a the arc-label.

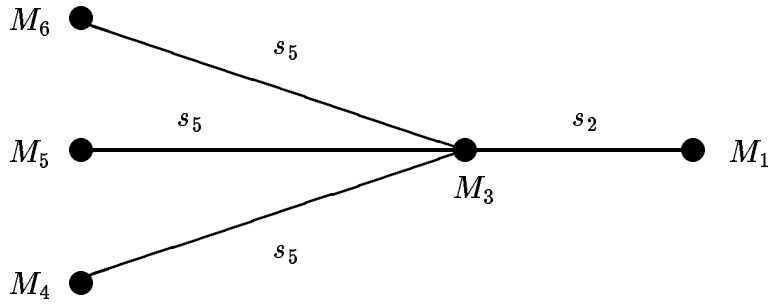

Figure 3.2: One possible hinge-tree for the hypergraph in Figure 2.1.

Algorithm 3.9 *Computing a hinge-tree.*

Input: A hypergraph (V, E).

Output: A hinge-tree T for (V, E).

Method:

1. Mark each edge in E as *unused*. Set $i = 0, N_0 = \{E\}$ and $A_0 = \emptyset$, and mark the node E in N_0 as *non-minimal*.
2. If all nodes of N_i are marked *minimal*, then set $T = (N_i, A_i)$ and stop. Else, choose a *non-minimal* node F in N_i.
3. If all edges in F are marked *used*, then mark F as *minimal* and return to 2. Else, choose an *unused* edge $e \in F$ and mark e as *used*.
4. Let $\Gamma = \{G \cup \{e\} \mid G \text{ is a connected component of } F - \{e\} \text{ with respect to } e\}$, and let $\gamma : F \rightarrow \Gamma$ be any function such that for all $f \in F$, $f \in \gamma(f)$. If $|\Gamma| = 1$, then return to 3.
5. Set

$$\begin{array}{rcl} N_{i+1} & = & (N_i - \{F\}) \cup \Gamma \\ A_{i+1} & = & (A_i - \{(\{F, F'\}, f) \mid (\{F, F'\}, f) \in A_i\}) \\ & & \cup \{(\{\gamma(f), F'\}, f) \mid (\{F, F'\}, f) \in A_i\} \\ & & \cup \{(\{\gamma(f), \gamma(e)\}, e) \mid f \in F,\ \gamma(f) \neq \gamma(e)\} \end{array}$$

   and mark all the new nodes added to N_{i+1} as *non-minimal*.
6. Increment i and return to 2.

■

In order to prove the correctness of Algorithm 3.9, we first need a technical lemma.

Lemma 3.10 [18] *Let (V, E) be a hypergraph, and let H be a hinge of that hypergraph. $H' \subseteq H$ is a hinge of $(\bigcup H, H)$ if and only if H' is a hinge of (V, E).*

We can now prove the following proposition.

Proposition 3.11 *Algorithm 3.9 correctly computes a hinge-tree T for a given hypergraph (V, E).*

Proof: Algorithm 3.9 clearly terminates, since each edge $e \in E$ may be chosen only once at Step 3, and is then marked as used.

Let k be the number of iterations of Steps 5 and 6 of Algorithm 3.9 on the given input (V, E), and let $T_i = (N_i, A_i)$ be the graph calculated at the ith iteration, for $i = 1, \ldots, k$. We show that all these graphs are trees satisfying

1′. *the tree nodes are hinges of* (V, E)

as well as conditions 2, 3 and 4 of Definition 3.7.

This assertion is trivially true for T_0. So assume, inductively, that for some value of i, $0 \leq i < k$, T_i is a tree satisfying conditions 1′, 2, 3 and 4. Let F be the node in T_i that is replaced in order to obtain T_{i+1}. Let e be the disconnecting edge of F chosen in Step 3 of Algorithm 3.9, and let Γ and γ be as defined in that step.

Each new arc added to T_i in order to obtain T_{i+1} is either an arc connecting two new nodes or an arc connecting a new node to one that already existed in T_i. By construction, the new nodes together with their interconnecting arcs form a tree. Moreover, each arc connecting F to another node of T_i is replaced by exactly one new arc connecting one new node to the same node of T_i; no other arcs are added. Hence T_{i+1} is also a tree.

For all $f \in F$, $\gamma(f) - \{e\}$ is a connected component of $F - \{e\}$ with respect to $\{e\}$, so $(\bigcup(\gamma(f) - \{e\})) \cap (\bigcup(F - \gamma(f))) \subseteq e$, whence also $(\bigcup \gamma(f)) \cap (\bigcup(F - \gamma(f))) \subseteq e$. Hence, $\gamma(f)$ (which contains e) is a hinge of $(\bigcup F, F)$. By the induction hypothesis, F is a hinge of (V, E). By Lemma 3.10, it then follows that $\gamma(f)$ is also a hinge of (V, E). Hence the nodes added to T_i in order to obtain T_{i+1} are hinges of (V, E), whence T_{i+1} also satisfies condition 1′.

Since $F = \bigcup \Gamma$ and T_i satisfies condition 2, it follows that $V = \bigcup N_{i+1}$, so T_{i+1} also satisfies condition 2.

We now consider the arcs added to A_i in order to obtain A_{i+1}. If $f \in F$ and $\gamma(f) \neq \gamma(e)$, then $\gamma(f) \cap \gamma(e) = \{e\}$ and $(\bigcup \gamma(f)) \cap (\bigcup \gamma(e)) \subseteq e$, by construction of γ. Also, if there is any arc $\{F, F'\} \in A_i$ with label f, then $F \cap F' = \{f\}$ and $(\bigcup F) \cap (\bigcup F') \subseteq f$ by the induction hypothesis, so $\gamma(f) \cap F' = \{f\}$ and $(\bigcup \gamma(f)) \cap (\bigcup F') \subseteq f$. Hence, T_{i+1} satisfies condition 3.

Finally, we turn to condition 4. Let L and M be distinct nodes of N_{i+1}. We consider three cases.

1. *Both L and M are not in N_i.* If either L or M equals $\gamma(e)$, then they are connected by an arc of A_{i+1}. Else, $\gamma(e)$ is the only interior node on the path connecting L and M in T_{i+1}. Since $L - \{e\}$ and $M - \{e\}$ are different connected components of $F - \{e\}$ with respect to $\{e\}$, it follows that $(\bigcup L) \cap (\bigcup M) \subseteq e \subseteq \bigcup \gamma(e)$.

2. *L is not in N_i but M is, or vice-versa.* Without loss of generality, we assume that $L \notin N_i$ and $M \in N_i$. Let $L, \ldots, L', M', \ldots, M$ be the path in T_{i+1} connecting L and M, where M' is the node in T_i nearest to L. (Note that L may equal L' and that M may equal M'.) By construction, $F, M', \ldots, M$ is the path in T_i connecting F and M. Since $L \subseteq F$, all the tree nodes on the path connecting M' to M contain all the vertices shared by L and M. Let $f \in F$ be the label of the arc in T_{i+1} connecting L' and M'. Then f is also the label of the arc in T_i connecting F and M'. Hence,

by the induction hypothesis, $(\bigcup L) \cap (\bigcup M) \subseteq (\bigcup F) \cap (\bigcup M') \subseteq f \subseteq \bigcup L'$. Since L' also contains all the vertices shared by L and M, each tree node on the path in T_{i+1} connecting L and L' also contains all the vertices shared by L and M, by Case 1.

3. *Both L and M are in N_i.* If all nodes on the path connecting L to M in T_{i+1} are in T_i, then, by the induction hypothesis, each of these nodes contains all vertices shared by L and M. Thus suppose the opposite. Let $L, \ldots, L', L'', \ldots, M'', M', \ldots, M$ be the path in T_{i+1} connecting L and M, where L'' is the node of T_{i+1} not in T_i that is nearest to L and M'' is the node of T_{i+1} not in T_i that is nearest to M. (Note that L may equal L', that L'' may equal M'' and that M may equal M'.) By construction, $L, \ldots, L', F, M', \ldots, M$ is the path in T_i connecting L and M. Let $f \in F$ be the label of the arc in T_{i+1} connecting L' and L''. Then f is also the label of the arc in T_i connecting L' and F. Hence, by the induction hypothesis, $(\bigcup L) \cap (\bigcup M) \subseteq (\bigcup L') \cap (\bigcup F) \subseteq f \subseteq \bigcup L''$. Similarly, M'' contains all the vertices shared by L and M. Hence each tree node on the subpath in T_{i+1} connecting L'' and M'' contains all the vertices shared by L and M by Case 1, and each tree node on the subpaths in T_{i+1} connecting L and L'', respectively M'' and M, contains all the vertices shared by L and M by Case 2.

Hence T_{i+1} also satisfies condition 4, concluding the inductive argument.

Finally, Lemma 3.10 guarantees that a node marked *minimal* is a minimal hinge of (V, E), whence the final value T_k of T in Algorithm 3.9 also satisfies condition 1 of Definition 3.7. ∎

Proposition 3.12 establishes the time complexity of Algorithm 3.9.

Proposition 3.12 *Algorithm 3.9 computes a hinge-tree T for a given hypergraph (V, E) in $O(|V||E|^2)$ time.*

Proof: The most costly step in Algorithm 3.9 is computing the connected components with respect to an edge in Step 4. Since each edge e for which Step 4 is called is marked as *used*, Step 4 can be executed at most $|E|$ times.

By representing sets of vertices as boolean arrays of arity $|V|$, it can be easily seen that one execution of Step 4 can be performed in $O(|V||E|)$ time, yielding the complexity bound in the proposition. ∎

The purpose of the remainder of this paper is to show that a hinge-tree of the associated hypergraph is a key construct for describing the structure of a constraint satisfaction problem, and, more concretely, finding a set of minimal constraints or solving a constraint satisfaction problem.

First, we note that although a hypergraph may, in general, have many different hinge-trees, containing different sets of minimal hinges, the size of the largest minimal hinge in any hinge-tree is a fixed parameter of a hypergraph and is independent of the particular hinge-tree chosen [19]. This result was first presented in the context of decomposition of join-dependencies and is reproduced here without proof.

Proposition 3.13 [19] *Let (V, E) be a hypergraph, and let T be any hinge-tree of (V, E). The size of the largest node in T is equal to the size of the largest minimal hinge of (V, E).*

The size of the largest minimal hinge of a hypergraph will be referred to as the *degree of cyclicity*. A hypergraph will be called *n-cyclic* if its degree of cyclicity is less than or equal to n.

Example 3.14 Reconsider the hypergraph in Figure 2.1, Example 2.13. Using the list of all its minimal hinges exhibited in Table 3.8, Example 3.6, it is readily seen that the hypergraph has degree of cyclicity 4. Hence, this hypergraph is 4-cyclic (as well as n-cyclic for all $n > 4$). □

Acyclicity is equivalent to 2-cyclicity [19], which corresponds to the fact that, for a constraint satisfaction problem associated with an acyclic hypergraph, imposing arc-consistency suffices to ensure minimality. The notion of n-cyclicity, however, leads to a hierarchy of classes of hypergraphs which is much finer than merely making the distinction between acyclic and cyclic hypergraphs, and many desirable properties of acyclic hypergraphs generalize to n-cyclic hypergraphs.

The next result shows how a hinge-tree of the associated hypergraph of a constraint satisfaction problem can be used to verify the minimality of the constraints.

Theorem 3.15 *Let $\mathcal{P} = (X, \Delta, \delta, C, \Sigma, \sigma)$ be a constraint satisfaction problem, and let $T = (N, A)$ be any hinge-tree for the hypergraph (X, Σ). If for every minimal hinge $M \in N$ and for every $s \in M$, every member of the constraint $\sigma^{-1}(s)$ can be extended to a solution of $\mathcal{P}|_{\sigma^{-1}(M)}$, then C is a set of minimal constraints.*

Proof: Let $s \in \Sigma$ be an arbitrary constraint scope, and let t_s be a member of the constraint $\sigma^{-1}(s)$.

Let $T' = (N', A')$ be any subtree of T with $s \in \bigcup N'$. We show by induction on $|N'|$ that t_s can be extended to a solution of $\mathcal{P}|_{\sigma^{-1}(\bigcup N')}$. This result holds, by assumption, when N' is a singleton. Now assume that for some $i \geq 1$ the result holds for each N' with $|N'| = i$ and choose T' such that $|N'| = i + 1$.

Let $M \in N'$ be a leaf node of T' such that $s \in \bigcup(N' - \{M\})$. (Such a node can always be found since $|N'| > 1$.) By the inductive hypothesis, t_s can be extended to a solution t' to $\mathcal{P}|_{\sigma^{-1}(\bigcup(N' - \{M\}))}$. Let s' be the label of the arc connecting M to $(\bigcup(N' - \{M\}))$ in T'. Then, by Definition 3.7,

$$\left(\bigcup\left(\bigcup(N' - \{M\})\right)\right) \cap \left(\bigcup M\right) \subseteq s'.$$

Since, by assumption, any member of the constraint $\sigma^{-1}(s')$ can be extended to a solution to $\mathcal{P}|_{\sigma^{-1}(M)}$, t' can be extended to a solution to $\mathcal{P}|_{\sigma^{-1}(\bigcup N')}$, whence the Theorem. ■

Theorem 3.15 also implies that minimality can be achieved by ensuring that each member of each constraint can be extended to each minimal hinge of the hinge-tree which contains the scope of that constraint. Moreover, if the nodes of a hinge-tree are further decomposed into smaller sets of edges, then these will not be hinges. Hence, by Theorem 3.5, we know that achieving "local" minimality within these smaller entities will *not* in general suffice to achieve global minimality. In this sense, a hinge-tree decomposition may be said to be fundamental, and the degree of cyclicity provides a natural upper bound on the complexity of the problem.

The next theorem indicates how a hinge-tree may be used to obtain a tree-structured problem which is equivalent to a given problem:

Theorem 3.16 *Let $\mathcal{P} = (X, \Delta, \delta, C, \Sigma, \sigma)$ be a constraint satisfaction problem, and let $T = (N, A)$ be any hinge-tree for the hypergraph (X, Σ). Let $\mathcal{P}' = (X, \Delta, \delta, C', \Sigma', \sigma')$ be the constraint satisfaction problem where $\Sigma' = \{\bigcup M \mid M \in N\}$ and, for each $M \in N$, $\sigma'^{-1}(\bigcup M) = Sol(\mathcal{P}|_{\sigma^{-1}(M)})$. Then*

1. *$\mathcal{P}'$ is equivalent to $\mathcal{P}$;*
2. *the hypergraph (X, Σ') is acyclic;*
3. *if C is a minimal set of constraints for $\mathcal{P}$, then C' is a minimal set of constraints for $\mathcal{P}'$.*

Proof: Since calculating $\mathrm{Sol}(\mathcal{P})$ corresponds to performing a join, and since the join operator is commutative and associative, the constraint satisfaction problem $\mathcal{P}'$ is clearly equivalent to $\mathcal{P}$.

In order to show that (X, Σ') is acyclic, let Σ'' be any closed connected subset of Σ' containing at least two edges. Choose $s'_1 \neq s'_2 \in \Sigma''$ such that their corresponding tree nodes M_1 and M_2 in T are of minimal distance. (Then there are no intermediate tree nodes M on the path connecting M_1 and M_2 in T such that $\bigcup M \in \Sigma''$.) We show that $\{s'_1, s'_2\}$ is an articulation pair of Σ''. Thereto, let $s'_1 = e_1, \ldots, e_n = s'_2$ be any path in the hypergraph $(\bigcup \Sigma'', \Sigma'')$. Let $M_1 = L_1, \ldots, L_n = M_2$ be the corresponding tree nodes. Since none of these tree nodes is an intermediate node on the path connecting M_1 and M_2 in T, they can be partioned into two categories: tree nodes nearer to M_1 and tree nodes nearer to M_2. Since, trivially, $L_1 = M_1$ is nearer to M_1 and $L_n = M_2$ is nearer to M_2, there must exist $1 \leq i < n$ such that L_i is nearer to M_1 and L_{i+1} is nearer to M_2. But then, by condition 4 of Definition 3.7, $(\bigcup L_i) \cap (\bigcup L_{i+1}) \subseteq \bigcup M_1 = s'_1$ and $(\bigcup L_i) \cap (\bigcup L_{i+1}) \subseteq \bigcup M_2 = s'_2$, whence $(\bigcup L_i) \cap (\bigcup L_{i+1}) \subseteq s'_1 \cap s'_2$. Hence, $s'_1 - s'_2$ and $s'_2 - s'_1$ are in different connected components of $\{s' - (s'_1 \cap s'_2) \mid s' \in \Sigma''\}$, so $\{s'_1, s'_2\}$ is an articulation pair of Σ''.

Finally, if C is a set of minimal constraints for $\mathcal{P}$, and M is a node of the hinge-tree T, then, by Theorem 3.3, any solution to $\mathcal{P}|_{\sigma^{-1}(M)}$ can be extended to a solution to $\mathcal{P}$. Hence C' is a set of minimal constraints for $\mathcal{P}'$. ∎

Example 3.17 Reconsider the constraint satisfaction problem $\mathcal{P}$ of Example 2.3 and recall the hinge-tree T for the hypergraph (X, Σ) that was shown in Figure 3.2, Example 3.8. The acyclic hypergraph associated with the equivalent constraint satisfaction problem $\mathcal{P}'$ defined in Theorem 3.16 is as shown in Figure 3.3. The corresponding set of constraints C' is shown in Table 3.9.

Clearly, C' is not a set of minimal constraints. If, however, we start with the the equivalent constraint satisfaction problem $\tilde{\mathcal{P}}$ from Example 2.10, then we obtain a problem $\tilde{\mathcal{P}}'$ with constraints $\tilde{C}'$ as shown in Table 3.10. The constraints in the set $\tilde{C}'$ are minimal, in accordance with Theorem 3.16. □

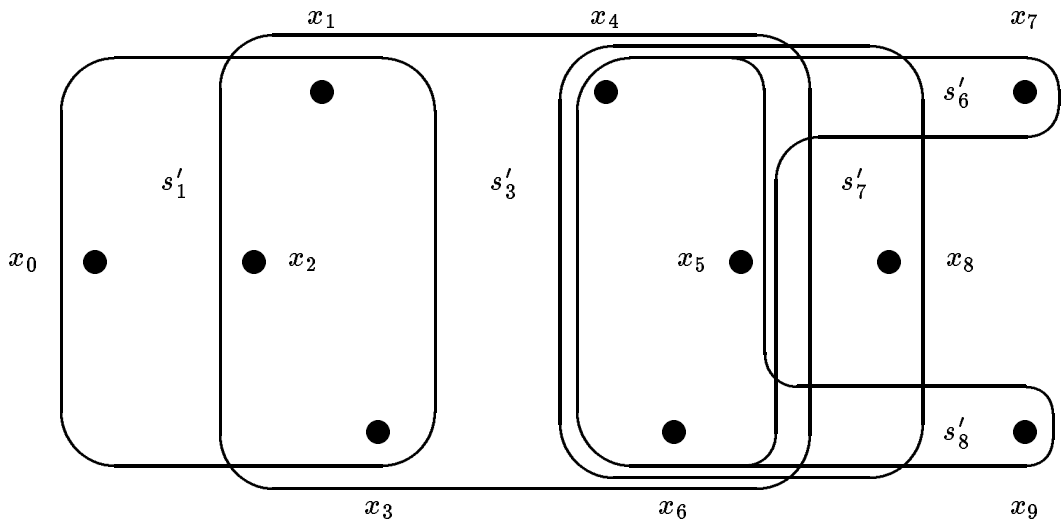

Figure 3.3: The hypergraph associated with $\mathcal{P}'$ (Example 3.17).

Table 3.9: The constraints of $\mathcal{P}'$ as a relational database (Example 3.17).

$R_{c'_1}$				$R_{c'_3}$						$R_{c'_6}$				$R_{c'_7}$				$R_{c'_8}$			
x_0	x_1	x_2	x_3	x_1	x_2	x_3	x_4	x_5	x_6	x_4	x_5	x_6	x_7	x_4	x_5	x_6	x_8	x_4	x_5	x_6	x_9
0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	1	0	0	0	0
0	1	1	0	0	0	1	0	0	0	0	0	1	1	0	0	1	1	0	0	1	1
1	0	0	1	0	0	1	0	0	1	1	1	1	0	1	1	1	0	1	1	1	1
1	1	0	1	1	0	1	1	1	1	1	0	2	0	1	1	1	1				
				0	1	2	0	0	0					1	0	2	1				

Table 3.10: The constraints of $\widetilde{\mathcal{P}}'$ as a relational database (Example 3.17).

$R_{\widetilde{c}'_1}$				$R_{\widetilde{c}'_3}$						$R_{\widetilde{c}'_6}$				$R_{\widetilde{c}'_7}$				$R_{\widetilde{c}'_8}$			
x_0	x_1	x_2	x_3	x_1	x_2	x_3	x_4	x_5	x_6	x_4	x_5	x_6	x_7	x_4	x_5	x_6	x_8	x_4	x_5	x_6	x_9
0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	1	0	0	0	0
1	0	0	1	0	0	1	0	0	0	0	0	1	1	0	0	1	1	0	0	1	1
1	1	0	1	0	0	1	0	0	1	1	1	1	0	1	1	1	0	1	1	1	1
				1	0	1	1	1	1					1	1	1	1				

4 A new decomposition strategy

4.1 Hinge decomposition

The results of the previous section have demonstrated that hinges form natural building blocks of constraint satisfaction problems. We now examine how this information may be used to solve such problems in an efficient way. Theorem 3.16 suggests the following decomposition algorithm to solve a constraint satisfaction problem:

Algorithm 4.1 *Solving a constraint satisfaction problem by hinge decomposition.*

Input: A constraint satisfaction problem $\mathcal{P} = (X, \Delta, \delta, C, \Sigma, \sigma)$.

Output: A solution to $\mathcal{P}$.

Method:

1. Compute a hinge tree T for (X, Σ), using Algorithm 3.9.
2. Using T, compute the equivalent constraint satisfaction problem, $\mathcal{P}'$, defined in Theorem 3.16 by solving the subproblems corresponding to the nodes of T.
3. Find a solution to the tree-structured problem $\mathcal{P}'$ in a backtrack-free manner, as described in [7].

■

The complexity of Algorithm 4.1 is determined by the degree of cyclicity of (X, Σ), as the following result demonstrates:

Theorem 4.2 *Algorithm 4.1 computes a solution to given constraint satisfaction problem* $\mathcal{P} = (X, \Delta, \delta, C, \Sigma, \sigma)$ *in*

$$O(|X||\Sigma|^2) + O(|\Sigma| l^d d \log l)$$

time, where l *is the maximal size of a constraint in* C*, and* d *is the degree of cyclicity of the hypergraph* (X, Σ)*.*

Proof: As shown in Proposition 3.12, Step 1 of Algorithm 3.12 takes $O(|X||\Sigma|^2)$. Finding all solutions to the subproblem corresponding to a node of T takes $O(l^d)$ time (the complexity of computing the join of all the constraints involved). By a straightforward inductive argument, it can be seen that the hinge-tree T produced by Algorithm 3.9 contains at most $|\Sigma| - 1$ nodes. Hence Step 2 takes $O(|\Sigma| l^d)$ time. Finding a solution to a constraint satisfaction problem $\mathcal{P}' = (X, \Delta, \delta, C', \Sigma', \sigma')$ with (X, Σ') an acyclic hypergraph can be achieved in $O(|C'| l' \log l')$, where l' is the maximal size of a constraint in C', by ordering the constraints lexicographically (cfr. [7]). Hence Step 3 takes $O(|\Sigma| l^d \log l^d) = O(|\Sigma| l^d \log l)$, yielding the desired complexity bound. ■

Hence, Algorithm 4.1 provides a very efficient method to solve problems with a small degree of cyclicity. When the degree of cyclicity is large, the subproblems obtained using Algorithm 4.1 will be large, and it may be useful to combine Algorithm 4.1 with other established techniques for solving these subproblems.

We now examine two previously proposed decomposition strategies: Freuder's use of biconnected components [14] and Dechter and Pearl's tree-clustering technique [7]. In Section 4.2 we show that the method of biconnected components is not as powerful as the hinge decomposition approach and cannot be usefully combined with it. On the other hand, by combining hinge decomposition with tree-clustering it is often possible to further decompose the minimal hinges. In Section 4.3, we prove that the combined approach has a better worst-case complexity bound than both tree-clustering and hinge decomposition on their own.

4.2 Decomposition using biconnected components

Freuder [14] considered binary constraint satisfaction problems in which each constraint scope contains exactly two variables, i.e., in which the associated hypergraph is a graph. He established that a binary constraint satisfaction problem may be decomposed into a tree-structured problem in which the nodes of the tree are the biconnected components of the associated graph [14]. (The use of biconnected components as subproblems is also discussed by Dechter and Pearl [6], where they are referred to as "nonseparable components.")

Now, it is clear from the definition of a hinge that every biconnected component containing two or more edges is a hinge. Hence, for any cyclic graph, the size of the largest node in the tree of biconnected components is greater than or equal to the size of the largest node in a hinge-tree. On the other hand, not every hinge is a biconnected component, as shown in the following example. Example 4.3 demonstrates, moreover, that the size of the largest biconnected component may be much larger than the degree of cyclicity, and therefore much larger than the size of the largest node in any hinge-tree.

Example 4.3 Consider the binary constraint satisfaction problem $\mathcal{Q}$ whose associated graph, G, is in the form of a row of overlapping squares, as shown in Figure 4.4. The only biconnected component of G is the whole graph, so no useful decomposition can be obtained in this way. However, the minimal hinges of G are the individual squares, so the degree of cyclicity of G is 4 and in *any* hinge-tree decomposition the nodes of the tree will contain at most four edges. □

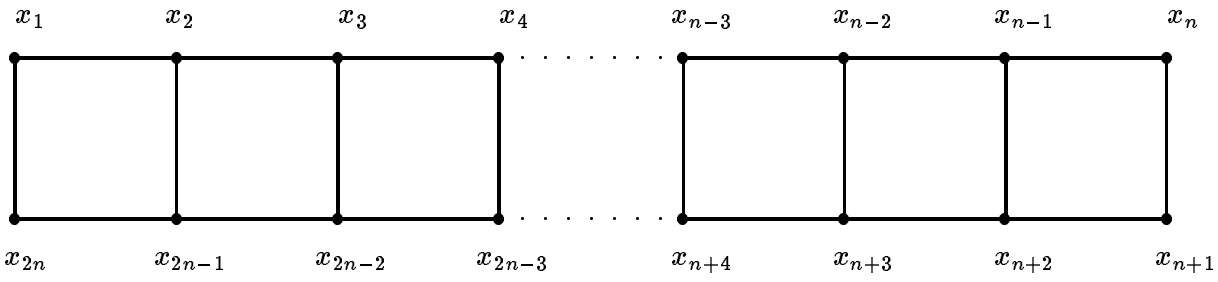

Figure 4.4: The graph G associated with $\mathcal{Q}$ (Example 4.3).

Hence, the hinge decomposition algorithm always obtains a decomposition which is at least as fine as Freuder's tree decomposition using biconnected components, and in several cases is much finer.

One way to extend the use of this biconnected components decomposition technique to an arbitrary hypergraph H is to consider its *dual graph* [7, 23], in which the edges of H are represented by vertices, and two vertices are joined if the corresponding edges of H overlap. The edges of the dual graph are labeled by the intersection of the two ends.

By using the dual graph it is possible to transform an arbitrary constraint satisfaction problem $\mathcal{P}$ into a binary constraint satisfaction problem $\mathcal{P}^d$. In this *dual* constraint satisfaction problem,

- the variables of $\mathcal{P}^d$ correspond to the scopes of the constraints of $\mathcal{P}$;
- the domain of each variable of $\mathcal{P}^d$ is the set of labelings allowed by the corresponding constraint of $\mathcal{P}$;
- the constraints of $\mathcal{P}^d$ require the values of each pair of variables corresponding to overlapping constraint scopes in $\mathcal{P}$ to agree on their intersection.

Obviously, the graph associated with the binary constraint satisfaction problem P^d is the dual graph of the hypergraph associated with $\mathcal{P}$.

Because of the restricted nature of the constraints in the dual problem it is often possible to identify redundant constraints, which may be removed without changing the set of solutions. In fact, if the two ends of an edge e in the associated dual graph are joined by a separate path, each of whose edges has a label containing the label of e, then the constraint corresponding to e may be removed [23]. After removing as many redundant constraints as possible in this way, the accordingly reduced dual graph may be decomposed into a tree of biconnected components, each of which may be solved separately, as proposed by Freuder [14].

The biconnected components obtained in this way will, in fact, correspond to hinges of the original hypergraph, but not necessarily to *minimal* hinges. Unlike the size of the largest minimal hinge of a hinge-tree, the size of the largest biconnected component of a reduced dual graph is *not* an invariant of the original hypergraph, and will in general depend on the order in which constraints are removed. There is no obvious way to ensure that the biconnected components obtained in this way are as small as possible, and the following example demonstrates that they may be arbitrarily large, even when the degree of cyclicity of the original problem is bounded.

Example 4.4 Reconsider the constraint satisfaction problem $\mathcal{Q}$ of Example 4.3 which has degree of cyclicity 4. The dual graph of $\mathcal{Q}$ is shown in Figure 4.5. One possible result of removing redundant edges as much as possible is shown in Figure 4.6, where the only biconnected component is the whole graph. □

Since the biconnected components obtained depend on the order in which the edges are removed, this decomposition scheme must be considered as a heuristic technique for arbitrary hypergraphs, whose performance cannot be guaranteed. On the other hand, by working with the original hypergraph, and focusing on hinges as well-defined, identifiable subunits of that hypergraph, the hinge-tree decomposition algorithm guarantees to obtain the finest possible decomposition in all cases (Proposition 3.13).

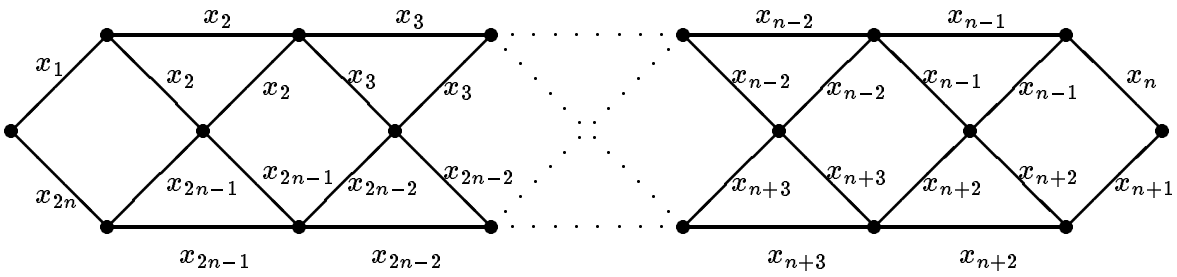

Figure 4.5: The dual graph associated with $\mathcal{Q}$ (Example 4.4).

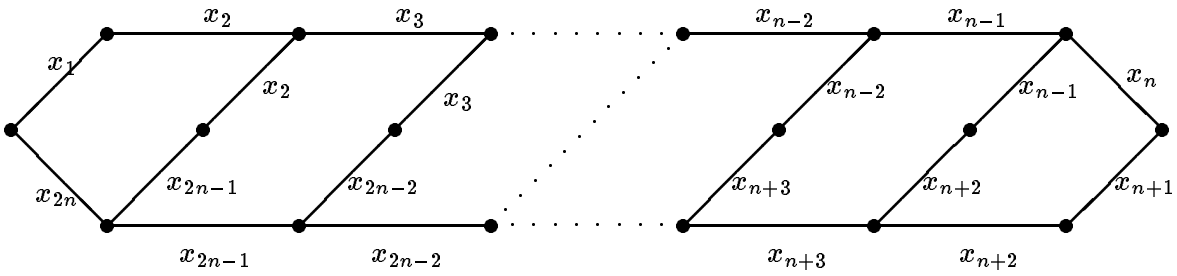

Figure 4.6: A reduced dual graph associated with $\mathcal{Q}$ (Example 4.4).

4.3 Decomposition using clustering

An alternative decomposition method for arbitrary constraint satisfaction problems has been proposed by Dechter and Pearl, which involves forming clusters of variables in the original problem to obtain a collection of subproblems which is tree-structured. Full details of this method may be found in [7], but a brief description is given here in order to demonstrate the advantages of combining this algorithm with the hinge-tree decomposition method.

The method proposed by Dechter and Pearl does not deal directly with the underlying hypergraph, but rather with an associated graph defined on the same set of vertices, called the *primal graph*. The primal graph of a hypergraph is defined as follows: two vertices in the primal graph are connected if and only if they are both members of a common edge in the hypergraph. In order to decompose a given constraint satisfaction problem, their method adds edges to the primal graph of the corresponding hypergraph in order to make it chordal. (A graph is *chordal* if every cycle of length at least 4 has a chord, i.e., an edge joining two nonconsecutive vertices along the cycle.) This process is known as *triangulating* the graph. The maximal cliques of the triangulated primal graph are then identified and used to form the constraint scopes of an equivalent constraint satisfaction problem. The constraint values of this equivalent problem are obtained by solving the subproblem corresponding to each maximal clique.

Using the fact that a hypergraph is acyclic if its primal graph is both chordal and conformal [2] they show that the equivalent constraint satisfaction problem obtained by this method is acyclic. (A primal graph is *conformal* if each of its maximal cliques corresponds to an edge in the original hypergraph.)

Dechter and Pearl report that the overall time complexity of finding a solution to a constraint satisfaction problem $\mathcal{P} = (X, \Delta, \delta, C, \Sigma, \sigma)$ using this tree-clustering decomposition method, is

$$O(|X|^2) + O(|X|k^r r \log k)$$

where k is the maximal size of a domain in Δ and r is the number of vertices in the largest maximal clique of the triangulated primal graph [7]. The complexity of this solution technique is therefore exponential in r.

In fact, the value of r is one greater than the *induced width* of the triangulated primal graph [7]. Unfortunately, calculating the minimal value of this induced width, and hence finding an optimal triangulation with respect to the above complexity bound, is NP-complete [32]. Therefore, Dechter and Pearl propose using a heuristic triangulation algorithm based on ordering the vertices in a minimal width ordering or a maximal cardinality ordering [7].

A *maximum cardinality ordering* numbers the vertices in increasing order, always assigning the next number to a vertex having the largest set of previously numbered neighbors (breaking ties arbitrarily). To achieve the triangulation, edges are then filled in (recursively) between any two nonadjacent vertices that are connected by a path only containing nodes higher up in the ordering. This heuristic triangulation algorithm was proposed by Tarjan and Yannakakis [30], who showed that it will transform any graph to a chordal graph.

When the triangulation is computed in this way the value of r is not an invariant of the problem but depends on the particular choice of maximum cardinality ordering. The next example provides a concrete illustration of how an unfortunate choice of this ordering may lead to a large value for r.

Example 4.5 Reconsider the constraint satisfaction problem $\mathcal{Q}$ of Example 4.3 which has degree of cyclicity 4. The primal graph of $\mathcal{Q}$ is the same as the associated hypergraph, which is shown in Figure 4.4. The ordering of the vertices in Figure 4.4 is one possible maximum cardinality ordering, and it is also a minimal width ordering. However, adding edges in accordance with the rule described above results in a clique consisting of the vertices $x_1, x_2, \ldots, x_n, x_{n+1}$. □

The problem with using any heuristic triangulation scheme is that its performance in some cases may be very poor, and therefore it is not possible to provide a tight upper bound on the complexity of the tree-clustering decomposition algorithm. Example 4.5 demonstrates that the triangulation algorithm just described may indeed result in arbitrarily large maximal cliques in the triangulated primal graph—and hence in arbitrarily large subproblems—even when the degree of cyclicity is bounded.

On the other hand, the following theorem shows that it is always possible to triangulate the primal graph by considering individually the hinges of a hinge-tree.

Theorem 4.6 *Let (V, E) be a hypergraph with a corresponding hinge-tree T, and let (V, F) be a graph containing the primal graph of (V, E). If for each edge $\{v, w\}$ of (V, F), $\{v, w\} \subseteq \bigcup M$, for some node M in T, then also, for each chord-free cycle $v_1, v_2, \ldots, v_k$ of (V, F), $\{v_1, v_2, \ldots, v_k\} \subseteq \bigcup M$, for some node M in T.*

Proof: Let $v_1, \ldots, v_k$ be a chord-free cycle of (V, F). Let S be a minimal subtree of T such that each pair of adjacent vertices in $v_1, \ldots, v_k$ is contained in $\bigcup M$ for some node M in S. For all s, $1 \leq s < k$, let M_s be any node of S such that $\{v_s, v_{s+1}\} \subseteq \bigcup M_s$, and let M_k be any node of S such that $\{v_k, v_1\} \subseteq \bigcup M_k$. (Obviously, some of the M_s, $1 \leq s \leq k$, may be equal.)

Now assume that $\{v_1, v_2, \ldots, v_k\} \not\subseteq M_k$. Then let i, $1 < i < k$, be the smallest number such that $v_i \notin \bigcup M_k$, and let j, $i \leq j < k$, be the smallest number greater than or equal

to i such that $v_{j+1} \in \bigcup M_k$. Note that, by the choice of i and j, none of the vertices $v_i, \ldots, v_j$ are contained in $\bigcup M_k$, so none of the nodes $M_{i-1}, \ldots, M_j$ equals M_k.

Let B be the branch of T with respect to M_k containing M_{i-1}. We claim that the nodes $M_{i-1}, M_i, \ldots, M_j$ all lie on B. To prove this, we use a simple inductive argument. By the choice of B, the claim is true for M_{i-1}. Suppose it is also true for M_s, $i-1 \leq s < j$. If M_{s+1} is not on B, then M_s and M_{s+1} are on different branches of T with respect to M_k. Consequently, $v_{s+1} \in (\bigcup M_s) \cap (\bigcup M_{s+1}) \subseteq \bigcup M_k$ by condition 4 of Definition 3.7, a contradiction.

Now let N be the node adjacent to M_k in B, and let $\{e\} = N \cap M_k$. By the connectedness of S, N is in S. By construction, $v_{i-1} \in (\bigcup M_k) \cap (\bigcup M_{i-1})$. By condition 4 of Definition 3.7, $(\bigcup M_k) \cap (\bigcup M_{i-1}) \subseteq (\bigcup M_k) \cap (\bigcup N) = e$. Hence, $v_{i-1} \in e$. By a similar argument, $v_{j+1} \in e$.

Since (V, F) contains the primal graph of (V, E), it follows that v_{i-1} and v_{j+1} are adjacent in (V, F). However, the cycle $v_1, v_2, \ldots, v_k$ is chord-free, so $v_{i-1} = v_1$ and $v_{j+1} = v_k$. Since moreover $v_1, v_k \in e \subseteq N$, it follows that each pair of adjacent vertices of $v_1, \ldots, v_k$ is contained in $\bigcup M$ for some node M of B, contradicting the minimality of S.

Consequently, our initial assumption is false, whence $\{v_1, \ldots, v_k\} \subseteq M_k$, proving the Proposition. ∎

With regard to triangulations of primal graphs, Theorem 4.6 has two important corollaries:

Corollary 4.7 *Let H be a hypergraph and let T be any hinge-tree for H. The primal graph of H may be fully triangulated by triangulating individually the subgraphs corresponding to the nodes of T.*

Proof: By Theorem 4.6, a complete triangulation of the primal graph can be achieved by only adding edges joining pairs of vertices contained in the same node of a hinge-tree.

Since the nodes of T overlap within edges of H, the vertices shared by two different nodes of T are all joined by edges of the primal graph of H, so edges which are added to the primal graph of one node of T cannot create chord-free cycles within another node of T. Hence each node may be triangulated individually. ∎

Corollary 4.8 *Let H be a hypergraph with degree of cyclicity d, and let m be the maximum number of vertices in any edge. There exists a triangulation for the primal graph of H such that*

$$r \leq (m-1)d + 1$$

where r is the number of vertices in the largest maximal clique of the triangulated primal graph.

Proof: Let T be any hinge-tree for H and triangulate individually the subgraph of the primal graph corresponding to each node of T. The cliques in the resulting triangulated primal graph must be contained within the nodes of T, and each node of T is a connected set of at most d edges, so it contains at most $(m-1)d + 1$ vertices. ∎

One special case of Corollary 4.8 is that the minimal induced width of any *graph* is bounded by its degree of cyclicity. We have therefore obtained an upper bound for the minimal induced width which may be calculated in polynomial time. On the other hand, because the induced width is defined in terms of the number of vertices, the minimal induced width of a hypergraph with a fixed degree of cyclicity may be made arbitrarily large simply by adding vertices to a single edge, and thereby increasing the value of m.

Corollaries 4.7 and 4.8 establish that the hinge-tree decomposition algorithm may be combined with Dechter and Pearl's decomposition algorithm in order to obtain a decomposition algorithm with a better complexity bound than either of these two algorithms alone. In this combined decomposition algorithm we first form a hinge-tree for the given problem, and then apply Dechter and Pearl's algorithm to each node individually.

Example 4.9 Reconsider the constraint satisfaction problem $\mathcal{Q}$ of Example 4.3, of which the associated hypergraph—which in this case coincides with the primal graph—is shown in Figure 4.4. The minimal hinges of this hypergraph are the squares, so the unique hinge-tree for this hypergraph is the linear tree with a node corresponding to each square. Triangulating each of these squares separately according to the method of the Dechter and Pearl results in a graph such as the one shown in Figure 4.7 in which the maximal cliques are all triangles containing two edges of the original graph.

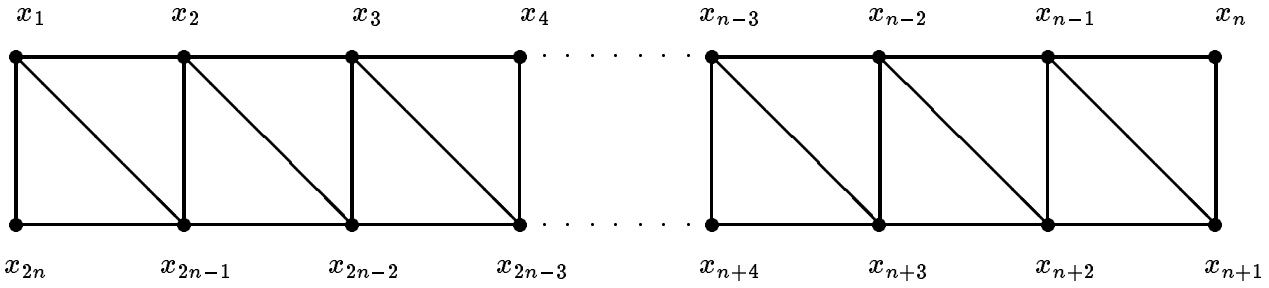

Figure 4.7: The triangulated primal graph G corresponding to $\mathcal{Q}$ (Example 4.9).

Hence the maximal number of constraints to be joined in order to obtain the constraints of the equivalent problem is 2, whereas the original Dechter and Pearl method may result in a subproblem with n constraints, as was observed in Example 4.5. □

Using this combined decomposition method, the number of constraints in the largest subproblem is bounded by the degree of cyclicity. The complexity bound for the combined decomposition method is therefore

$$O(|X||\Sigma|^2) + O(|X|^2|\Sigma|) + O(|X||\Sigma|l^d d \log l)$$

where l is the maximal size of a constraint in C and d is the degree of cyclicity of the hypergraph (X, Σ). By Proposition 3.13, the degree of cyclicity, appearing in the exponent of the complexity bound for our combined approach, *is* an invariant of the problem, which can be easily determined while constructing the hinge-tree.

In the worst case, the combined decomposition algorithm does not improve on the performance of the hinge decomposition algorithm (Algorithm 4.1). This case occurs when the induced width of each of the subproblems is maximal. Similarly, the combined approach does not improve Dechter and Pearl's original tree-clustering scheme when the

degree of cyclicity of the given problem is maximal. For example, a binary constraint satisfaction problem for which the associated graph has n edges arranged in the form of a ring has degree of cyclicity n, the largest possible value, while its minimum induced width is only 2 [7]. In general, however, the the maximum number of constraints that have to be joined in the combined approach may be significantly smaller than in *both* the hinge decomposition algorithm and Dechter and Pearl's algorithm, as was shown in Example 4.9. In fact, the combined algorithm we are proposing here makes maximal use of any simplifications which may be derived from *both* the degree of cyclicity and the induced width.

To conclude this section, we apply the combined decomposition method to the running example used throughout this paper.

Example 4.10 Reconsider the constraint satisfaction problem $\mathcal{P}$ of Example 2.3. The associated hypergraph shown in Figure 2.13 is covered by the hinge-tree shown in Figure 3.8. (For the meaning of the nodes, the reader is referred to Table 3.8, Example 3.6.) In the primal graph of the hypergraph associated with $\mathcal{P}$, only the subgraph corresponding to M_3 contains a chord-free cycle. After triangulation, this subgraph contains four maximal cliques, all of size 3. Consequently, the hypergraph of one possible equivalent problem, $\mathcal{P}^*$, obtained by applying the modified Dechter and Pearl algorithm is as shown in Figure 4.8.

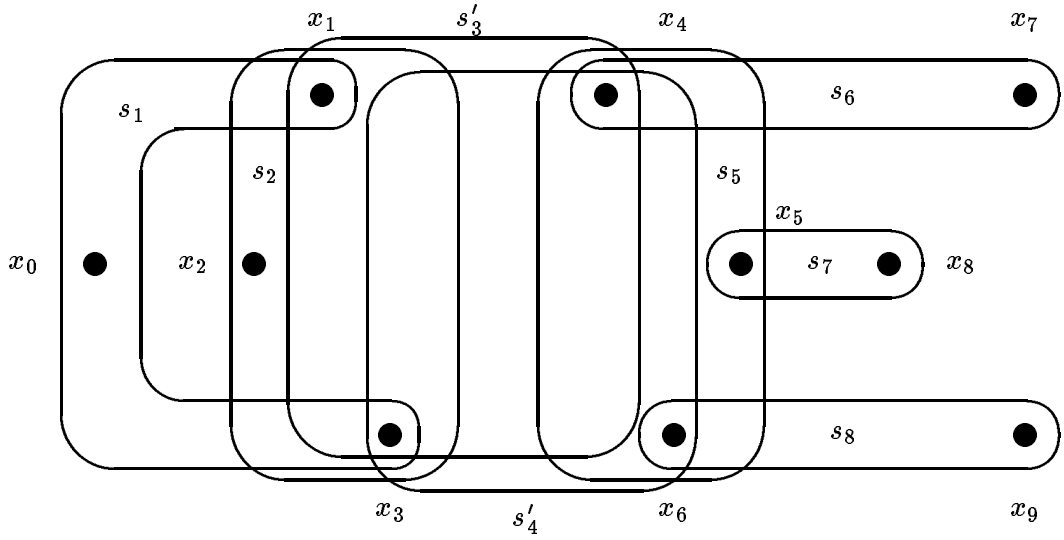

Figure 4.8: The hypergraph associated with $\mathcal{P}^*$ (Example 4.10).

Note that in this example the maximal number of constraints that must be joined to obtain the constraints of the equivalent problem, $\mathcal{P}^*$, (which is 2) is again smaller than the degree of cyclicity of the hypergraph associated with the original problem (which is 4). Finally, the constraints of $\mathcal{P}^*$ are obtained by replacing the constraints c_3 and c_4 in Table 2.2 by the constraints c'_3 and c'_4 shown in Table 4.11. □

Table 4.11: The new constraints of $\mathcal{P}^*$ (Example 4.10).

$R_{c_3'}$			$R_{c_4'}$		
x_1	x_3	x_4	x_3	x_4	x_6
0	0	0	0	0	0
0	1	0	1	0	0
0	2	0	1	0	1
1	0	1	1	1	1
1	1	1	2	0	0

5 Conclusions

In this paper, we have used notions and results from relational database theory to derive techniques for dealing with constraint satisfaction problems. In particular, we have shown that a constraint satisfaction problem can be decomposed according to a "covering" of minimal hinges of the underlying hypergraph. The size of the largest minimal hinge in any such covering is an invariant of the hypergraph, called the degree of cyclicity. The size of the largest subproblem which must be solved in order to check or establish minimality is therefore equal to the degree of cyclicity. Moreover, the degree of cyclicity can also provide a bound for the size of the subproblems obtained by other proposed decomposition strategies. In the light of these results, it is clear that the degree of cyclicity of the underlying hypergraph is an important measure for the complexity of a constraint satisfaction problem.

In conclusion, we believe that the unity we have established between the analysis of constraint satisfaction problems on the one hand, and the work on join-dependencies in relational database theory on the other hand, will allow further results in both of these fields to be applied in the other.

Acknowledgments

The authors wish to acknowledge the support of the Belgian National Fund for Scientific Research and the British Council which enabled them to make mutual visits during which this research was carried out.

References

[1] Beeri, C., Fagin, R. Maier, D., Mendelzon A. O., Ullman, J. D., and Yannakakis, M., "Properties of Acyclic Database Schemes," *Proceedings of the 13th Annual ACM Symposium on the Theory of Computing*, (1981), pp. 355–362.

[2] Beeri, C., Fagin, R., Maier, D., and Yannakakis, M. "On the Desirability of Acyclic Database Schemes," *Journal of the ACM 30* (1983), pp. 479-513.

[3] Bergé, C., *Graphs and Hypergraphs*, North-Holland, New York, (1976).

[4] Bibel, W., "Constraint Satisfaction From a Deductive Viewpoint," *Artificial Intelligence 35* (1988), pp. 401–413.

[5] Codd, E. F., "A Relational Model of Data for Large Shared Databanks," *Communications of the ACM 13* (1970), pp. 377–387.

[6] Dechter, R., and Pearl, J., "Network-Based Heuristics for Constraint-Satisfaction Problems," *Artificial Intelligence 34* (1988), pp. 1–38.

[7] Dechter, R., and Pearl, J., "Tree Clustering for Constraint Networks," *Artificial Intelligence 38* (1989), pp. 353–366.

[8] Dechter, R., "Decomposing a Relation into a Tree of Binary Relations," *Journal of Computer and System Sciences 41* (1990), pp. 2–24.

[9] Dincbas, M., Simonis, H., Van Hentenryck, P., "Solving a Stock Cutting Problem in Constraint Logic Programming," *Logic Programming: Proceeding of the 5th International Conference and Symposium*, MIT Press, (1988) pp. 42–58.

[10] Fagin, R., "Degrees of Acyclicity for Hypergraphs and Relational Database Schemes," *Journal of the ACM 30* (1983), pp. 514–550.

[11] Fagin, R., Mendelzon, A. O., and Ullman, J. D., "A Simplified Universal Relation Assumption and its Properties," *ACM Transactions on Database Systems 7* (1982), pp. 343-360.

[12] Freuder, E. C. , "Synthesising Constraint Expressions," *Communications of the ACM 21* (1978), pp. 958–966.

[13] Freuder, E. C., "A Sufficient Condition for Backtrack-Free Search," *Journal of the ACM 29* (1982), pp. 24–32.

[14] Freuder, E. C., "A Sufficient Condition for Backtrack-Bounded Search," *Journal of the ACM 32* (1985), pp. 755–761.

[15] Grimson, W. E. L., "The Combinatorics of Local Constraints in Model-Based Recognition and Localization from Sparse Data," *Journal of the ACM 33* (1986), pp. 658–686.

[16] Grimson, W. E. L., "The Combinatorics of Object Recognition in Cluttered Environments Using Constrained Search," *Artificial Intelligence 44* (1990), pp. 121–165.

[17] Gyssens, M., and Paredaens, J., "A Decomposition Methodology for Cyclic Databases," in *Advances in Database Theory 2*, Plenum Press, New York (1984), pp. 85–122.

[18] Gyssens, M., and Paredaens, J., "On the Decomposition of Join Dependencies," *Proceedings 3rd PODS*, Waterloo, Ontario, (1984), pp. 143–152.

[19] Gyssens, M., "On the Complexity of Join Dependencies," *ACM Transactions on Database Systems 11* (1986), pp. 81-108.

[20] Haralick, R. M., and Shapiro, L. G., "The Consistent Labeling Problem: Part I," *IEEE Transactions on Pattern Analysis and Machine Intelligence 1* (1979), pp. 173–184.

[21] Jeavons, P. G., "Counting Representable Sets on Simple Graphs," *Technical Report CSD-TR-642*, Department of Computer Science, RHBNC, University of London (1990), to appear in *Discrete Applied Mathematics.*

[22] Mackworth, A. K., "Consistency in Networks of Relations," *Artificial Intelligence 8* (1977), pp. 99–118.

[23] Maier, D., *The Theory of Relational Databases,* Computer Science Press, Rockville, Maryland, (1983).

[24] Miller, L. L., Leuchner, J. H., Kothari, S., and Liu K. C., "Testing Arbitrary Subhypergraphs for the Lossless Join Property," *Information Sciences 51* (1990), pp. 95–110.

[25] Miller, L. L., "Generating Hinges from Arbitrary Subhypergraphs," *Information Processing Letters 41* (1992), pp. 307–312.

[26] Montanari, U., "Networks of Constraints: Fundamental Properties and Applications to Picture Processing," *Information Sciences 7* (1974), pp. 95–132.

[27] Paredaens, J., De Bra, P., Gyssens, M., and Van Gucht, D., *The Relational Database Model*, EATCS Monographs on Computer Science 17, Springer-Verlag, (1989).

[28] Rissanen, J., "Theory of Joins for Relational Databases - A Tutorial Survey," *Proceedings of the 7th Symposium on Mathematical Foundations of Computer Science, Lecture Notes in Computer Science 64*, Springer-Verlag, (1978), pp. 537–551.

[29] Seidel, R., "A New method for Solving Constraint Satisfaction Problems," *Proceedings of the 7th International Joint Conference on Artificial Intelligence*, Vancouver, Canada, (1981), pp. 338–342.

[30] Tarjan, R. E., and Yannakakis, M., "Simple linear-time algorithms to test chordality of graphs, test acyclicity of hypergraphs and selectively reduce acyclic hypergraphs," *SIAM Journal of Computing 13* (1984), pp. 566–579.

[31] Ullman, J. D., *Principles of Database and Knowledge Base Systems*, vol. 1 and 2, Computer Science Press, Rockville, Maryland, (1988-89).

[32] Yannakakis, M., "Computing the minimum fill-in is NP-complete," *SIAM Journal of Algebraic and Discrete Methods 2* (1981), pp. 77–79.

[33] Zhang, Y., Mackworth, A. K., "Parallel and Distributed Algorithms for Finite Constraint Satisfaction Problems," *Proceedings 3rd IEEE Symposium on Parallel and Distributed Computing* (1991), pp. 394–397.