A Survey of Tractable Constraint Satisfaction Problems

Justin Pearson & Peter Jeavons

July 14, 1997

Abstract

In this report we discuss constraint satisfaction problems. These are problems in which values must be assigned to a collection of variables, subject to specified constraints. We focus specifically on problems in which the domain of possible values for each variable is finite.

The report surveys the various conditions that have been shown to be sufficient to ensure tractability in these problems. These are broken down into three categories:

- Conditions on the overall structure;
- Conditions on the nature of the constraints;
- Conditions on bounded pieces of the problem.

1 Introduction

A constraint satisfaction problem is a way of expressing simultaneous requirements for values of variables.

The study of constraint satisfaction problems was initiated by Montanari in 1974 [34], when he used them as a way of describing certain combinatorial problems arising in image-processing. It was quickly realised that the same general framework was applicable to a much wider class of problems, and the general problem has since been intensively studied, both theoretically and experimentally (for a general introduction see [32]).

The following examples indicate the wide variety of problems which can be viewed as constraint satisfaction problems:

- A classic example of a problem which is often formulated as a constraint satisfaction problem is the problem of placing eight queens on a chess board so that no queen can capture any other queen [40].

- A more practical example is the problem of *scheduling* a collection of tasks or activities. In this problem a list of tasks is given, together with specified *constraints* on which tasks can be carried out at the same time, which tasks must precede which others, and so on. To solve the problem, it is required to find an assignment of times to each task which satisfies all of these constraints simultaneously. (A good introduction to scheduling as a constraint satisfaction problem can be found in [43].)

- Another example of considerable current interest is the *frequency assignment problem.* In this problem an arrangement of radio transmitters and receivers is given, together with a description of how the signals from each transmitter propagate to each receiver. To solve the problem, it is required to find an assignment of one (or more) available frequencies to each of the transmitters such that, when each transmitter broadcasts at its assigned frequency, the desired signals can be received at each receiver without excessive interference from other, unwanted, signals. Typically, this means that transmitters which are geographically close must be assigned frequencies which are widely separated. The frequency assignment problem can be modelled as a constraint satisfaction problem in a number of different ways, see [14].

- Many classic combinatorial problems, such as the SATISFIABILITY problem from propositional logic [36], the COLORABILITY problem and the GRAPH ISOMORPHISM problem from graph theory, and the BANDWIDTH problem from operational research, can be formulated very naturally as constraint satisfaction problems. For a unifying approach to problems of this type, within the framework of constraint satisfaction, see [25].

- The final example we mention is the solution of crossword puzzles. This simple application will be used to illustrate the basic framework and terminology defined in the next section, so we now introduce a particular crossword puzzle that will be used as a running example:

  Example 1.1 A typical crossword puzzle is specified by two things: a grid, as shown in Figure 1, and a set of clues.

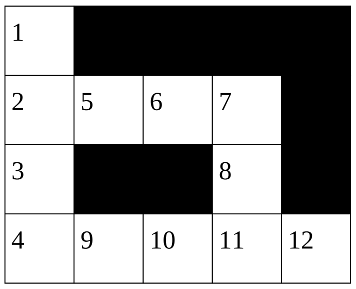

Figure 1: A crossword grid

For our purposes, we shall think of each clue as a *constraint*, which specifies the allowed words for a particular set of empty spaces in the grid. (To make the description of the constraints easier we have numbered each empty square in the grid.) For example, the clue for '1 Down' might allow as possible words "ELIA", "PAUL", and "SAFE", while the clue for '2 Across' might allow as possible words "APES" and "FAIR". □

A *solution* to a constraint satisfaction problem is an assignment of values to all of the variables in the problem which does not violate any of the constraints.

Finding a solution to a constraint satisfaction problem by a simple-minded search, which goes through all possible assignments and checks each one to see if it satisfies the constraints, is generally impractical. The maximum time taken to complete this procedure grows exponentially with the number of variables.

In this report we shall describe a number of special cases where it can be shown that there is a much more efficient algorithm for finding a solution. The structure of the report is as follows.

- In Section 2, we give a formal definition of a constraint satisfaction problem, illustrate this definition with a number of examples, and describe the close connections between constraint satisfaction problems and relational databases.
- In Section 3. we show how certain restrictions on the overall structure of a problem can be used to obtain efficient solution algorithms of various kinds.
- In Section 4. we show how certain restrictions on the form of constraints used in a problem can also be used to obtain efficient solution

algorithms of various kinds.

- In Section 5. we show how certain restrictions on bounded sub-parts of a problem can ensure that the complete problem is easy to solve.
- Finally, in Section 6. we summarise the results presented and identify some directions for future research.

2 Constraint Satisfaction Problems

2.1 Basic definitions

This section defines the framework we shall use for expressing constraint satisfaction problems. A formal framework is necessary in order to allow a precise analysis of the efficiency of algorithms for finding solutions, which is the main purpose of this report.

We shall only consider constraint satisfaction problems in which there are a finite number of variables, and each variable has a finite number of possible values. These are defined as follows.

Definition 2.1 *A* constraint satisfaction problem, *$\mathcal{P}$, is specified by a tuple,*

$$\mathcal{P} = (V, D, R_1(S_1), \ldots, R_n(S_n))$$

where

- *V is a finite set of* variables*;*
- *D is a finite set of* values *(this set is called the* domain *of $\mathcal{P}$);*
- *Each pair $R_i(S_i)$ is a* constraint.

  In each constraint, $R_i(S_i)$,

  - *S_i is an ordered list of k_i variables, called the constraint* scope*;*
  - *R_i is a relation*[1] *over D of arity k_i, called the constraint* relation.

Definition 2.2 *A* solution *to $\mathcal{P} = (V, D, R_1(S_1), \ldots, R_n(S_n))$ is an assignment of values from D to each of the variables in V, which satisfies all of the constraints simultaneously.*

Formally, a solution is a map $h : V \to D$ such that $h(S_i) \in R_i$, for all i, where the expression $h(S_i)$ denotes the result of applying h to the tuple S_i, coordinate-wise (in other words, if $S_i = \langle v_1, v_2, \ldots, v_k\rangle$, then $h(S_i) = \langle h(v_1), h(v_2), \ldots h(v_k)\rangle$).

The *set of all solutions* to a problem $\mathcal{P}$ will be denoted Sol($\mathcal{P}$). Two problems with the same set of solutions will be said to be *equivalent*.

[1]A *relation* is simply a set of tuples of some fixed length. The length of the tuples is called the *arity* of the relation.

Example 2.3 We will now construct a simple constraint satisfaction problem with variables $V = \{x, y, z\}$ and domain $D = \{1, 2, \ldots, 6\}$ (i.e., the natural numbers from 1 to 6).

Suppose we want to express the requirements that the sum of x and y must be 6, and that the product of y and z must be at least 20. This can be done with the constraints $R_1(\langle x, y\rangle)$ and $R_2(\langle y, z\rangle)$ where:

- $R_1 = \{\langle 1,5\rangle, \langle 2,4\rangle, \langle 3,3\rangle, \langle 4,2\rangle, \langle 5,1\rangle\}$
- $R_2 = \{\langle 4,5\rangle, \langle 4,6\rangle, \langle 5,4\rangle, \langle 5,5\rangle, \langle 5,6\rangle, \langle 6,4\rangle, \langle 6,5\rangle, \langle 6,6\rangle\}$

The solutions to this problem can be calculated by hand. For example, the map

$$\begin{aligned} f(x) &= 1 \\ f(y) &= 5 \\ f(z) &= 4 \end{aligned}$$

is a solution, because $f(\langle x, y\rangle) = \langle 1, 5\rangle$, which is in R_1, and $f(\langle y, z\rangle) = \langle 5, 4\rangle$, which is in R_2.

The complete set of solutions is

$$\{\langle 1,5,4\rangle, \langle 1,5,5\rangle, \langle 1,5,6\rangle, \langle 2,4,5\rangle, \langle 2,4,6\rangle\},$$

where a triple in the solution defines the values assigned to x, y and z, respectively. □

Example 2.4 One way to formalise crossword puzzles is to define a variable for each empty square in the grid, and set the domain D to be the set of all alphabetic letters. We can then associate with each clue in the crossword a constraint, giving allowed words for the corresponding squares.

The crossword puzzle described in Example 1.1 has 12 empty squares, so it would be represented by a constraint satisfaction problem with 12 variables. It contains 4 words, and hence has 4 clues, so we would define 4 constraints $R_1(S_1), \ldots, R_4(S_4)$ where, for example, we might have:

- $S_1 = \langle 1, 2, 3, 4\rangle$,
- $S_2 = \langle 2, 5, 6, 7\rangle$,
- $S_3 = \langle 7, 8, 11\rangle$,
- $S_4 = \langle 4, 9, 10, 11, 12\rangle$,

with

- $R_1 = \{\langle \text{E,L,I,A} \rangle, \langle \text{P,A,U,L} \rangle, \langle \text{S,A,F,E} \rangle\}$,
- $R_2 = \{\langle \text{A,P,E,S} \rangle, \langle \text{F,A,I,R} \rangle\}$, and so on.

Note, it is perfectly possible to represent the same constraint with a different ordering of the variables. For example, the constraint $R_2(S_2)$ defined above could be represented as the constraint $R_2'(S_2')$ where:

- $S_2' = \langle 5, 6, 2, 7 \rangle$
- $R_2' = \{\langle \text{P,E,A,S} \rangle, \langle \text{A,I,F,R} \rangle\}$

without changing the solutions to the constraint, or to the overall problem. □

We will occasionally make use of the notion of a *partial solution.* This may be defined in a number of different ways, depending on the stringency of the requirements which we wish to impose. For consistency with the majority of the literature, we shall use the following definition.

Definition 2.5 *A* partial solution *to a constraint satisfaction problem $\mathcal{P} = (V, D, R_1(S_1), \ldots, R_n(S_n))$ is a mapping h from some subset, say W, of V to D, such that for each S_i contained in W, $h(S_i) \in R_i$.*

Example 2.6 A partial solution to the crossword puzzle described in Example 1.1 is shown in Figure 2. This partial solution satisfies all the constraints on complete words where all letters have been assigned. Notice, however, that it is unlikely to be extendible to a complete solution because of the letters assigned to squares 8 and 12. □

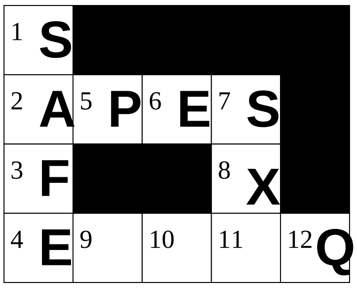

Figure 2: A partial solution to the crossword in Example 1.1

2.2 Links with Relational Database theory

To further illustrate the definitions we have given, we now say a little about the close connections between constraint satisfaction problems and relational databases.

It is very valuable to be aware of these links, because relational database theory provides a rich body of concepts and techniques which can be applied to constraint satisfaction problems. In particular, the use of *relational algebra* [7], which is a well-established tool in database theory, allows many properties and algorithms used in the study of constraint satisfaction to be expressed in a concise and elegant way.

Definition 2.7 ([7]) *A relational database is a finite collection of* tables[2]*.*
A table consists of a scheme *and an* instance*:*

- *A scheme is a finite set of* attributes*, where each attribute has an associated set of possible values, referred to as a* domain*.*
- *An instance is a finite set of* rows*, where each row is a mapping that associates with each attribute of the scheme a value in its domain.*

Relational database theory [41] has at least two central concerns: the efficient storage of tables, and the expressibility and efficiency of queries. A *query* is a request for information from some collection of tables. For example, if one table stores names and addresses, and another table stores names and salaries, then a query might ask for the salaries of people who live in a certain area.

Various standard operations have been defined on tables, which allow many queries to be expressed, and these operations are collectively known as the relational algebra [7]. We will only make use of two of these operations: *projection* and *join*, which are defined as follows.

Definition 2.8 *Given a table T with set of attributes I, and a subset J of I, the* projection *of T onto J, denoted $\pi_J T$, is the table with set of attributes J and the following set of rows:*

$$\{f_{|J} \mid f \in T\}$$

where $f_{|J}$ denotes the function f restricted to the arguments in J. That is, each row of $\pi_J T$ is a restriction of some row in T, containing values for attributes in J only.

[2]Tables are often referred to as relations. We will call them tables to avoid clashing with the set-theoretic definition of a relation used above.

Definition 2.9 *Given a table T with set of attributes I, and a table S with set of attributes J, the* join *of T and S, denoted $T \bowtie S$, is defined to be the table with set of attributes $I \cup J$ and the following set of rows:*

$$\{f \mid f_{|I} \in T \text{ and } f_{|J} \in S\}$$

Example 2.10 Here is an example of a table, which we shall call **PayRoll**:

Name	Salary	Age
Fred	30,000	32
Susan	35,000	37
Jim	25,000	27
Sheila	35,000	37

The scheme of this table has three attributes, Name, Age and Salary, (each with an appropriate domain of possible values), and the instance has four rows.

The query $\pi_{\text{Age,Salary}}$ **PayRoll** gives the table:

Age	Salary
32	30,000
37	35,000
27	25,000

(Note that a table is a set, which means that it cannot have duplicate rows, so any duplicates rows arising from the projection are eliminated.)

Now assume that our database also contains a second table, which we shall call **Addresses**:

Name	Address
Dylan	Cwmdonkin Drive
Jim	Eton Terrace
Sheila	Seaview Gardens

The query which asks for the join of these two tables, written as

$$\textbf{PayRoll} \bowtie \textbf{Addresses},$$

gives the following table:

Name	Age	Salary	Address
Jim	27	25,000	Eton Terrace
Sheila	37	35,000	Seaview Gardens

□

The very close connection between constraint satisfaction problems and databases is indicated in the following table:

Constraint Terminology		Database Terminology
constraint satisfaction problem	$\equiv$	database
variable	$\equiv$	attribute
domain	$\equiv$	union of all attribute domains
constraint	$\equiv$	table
constraint scope	$\equiv$	scheme
constraint relation	$\equiv$	instance
set of solutions	$\equiv$	join of all tables

In summary, a constraint satisfaction problem $\mathcal{P} = (V, D, R_1(S_1), \ldots, R_n, (S_n))$ can be seen as a relational database with n tables, having schemes $S_1, \ldots, S_n$ and instances $R_1, \ldots, R_n$. The set $\mathrm{Sol}(\mathcal{P})$ is equal to the table

$$R_1 \bowtie R_2 \bowtie \cdots \bowtie R_n.$$

For further discussion of the important and fruitful connection between these two fields, see [23].

2.3 The complexity of finding solutions

Many techniques have been developed over the past 20 years to find solutions to constraint satisfaction problems (for a general introduction to solution methods for constraint satisfaction problems, see [40, 32]).

One obvious approach is to employ some form of *backtrack search* algorithm. This simple form of search algorithm may be specified as follows:

Algorithm 2.11

1. Pick some ordering of the variables, say, $v_1, v_2, \ldots, v_{|V|}$;
2. Pick some ordering of the domain, say $d_1, \ldots, d_{|D|}$;
3. Call BACKTRACK(1).

```
BACKTRACK($i$)
If $i > |V|$, then output the current assignment;
          else for $j = 1, 2, \ldots, |D|$
                 Assign the value $d_j$ to variable $v_i$.
                 If the current assignment of $v_1, v_2, \ldots, v_i$ is a partial solution,
                   then call BACKTRACK($i + 1$);
```

Many improvements to the standard backtrack search have been described in the literature (for a recent survey, see [31]). These all attempt to speed up the basic algorithm by using extra information about the problem to guide the search more effectively, hence making fewer unnecessary assignments, and backtracking less often.

The maximum time taken to find a solution by any form of backtrack search (or to establish that no solution exists) grows exponentially with the number of variables, in general. However, for any particular problem instance the time required to find a single solution depends on

- the details of the given problem;
- the chosen variable ordering;
- the chosen domain ordering.

In some cases, a search procedure can find a solution without backtracking at all, and hence the time taken is only proportional to the number of variables.

In order to analyse more precisely the computational difficulty of finding solutions to constraint satisfaction problems, we shall make use of some of the techniques and terminology of computational complexity theory.

In particular, we shall attempt to determine the *time complexity*[3] of various restricted classes of constraint satisfaction problems. (For a general introduction to complexity theory, see [22] or [36].)

The main results we shall describe show that, for certain special types of problems, it is possible to design algorithms which will *always* find a solution efficiently (or discover that there are no solutions). A class of problems will be called *tractable* if there is an algorithm which finds a solution to all problems in that class, or reports that there are no solutions, and whose time complexity is *polynomial* in the size of the problem to be solved. The rest of this report lists a wide variety of conditions which are sufficient to ensure tractability, in this sense.

On the other side of the coin, it is sometimes possible to show that a class of problems is very unlikely to be tractable. In several cases, we shall establish that a particular class of constraint satisfaction problems is *NP-complete* [22]. To do this we show that any algorithm which could solve all the problems in this class in polynomial time would also allow us to solve some well-established difficult problems, such as GRAPH COLORABILITY [36], in polynomial-time. If a class of problems is NP-complete, then this provides

[3]The *time complexity* of any collection of problem instances is a function which gives the maximum time taken by some fixed algorithm that solves any member of that class, for each possible instance size.

very good evidence that any algorithm for solving such a class is likely to require exponential time to complete (for at least some cases). We are therefore unlikely to be able to solve all large instances of problems in that class within a reasonable length of time.

3 Tractability due to restricted structure

3.1 Defining problem structure

In the following subsections we shall review some results concerning the tractability of problems with restricted structure.

First, we need to define some terminology for describing the structure of a constraint satisfaction problem. With any constraint satisfaction problem $\mathcal{P}$, we will associate a mathematical structure, known as a *hypergraph*, which captures how the variables of the problem are related. (A hypergraph is a generalisation of the more familiar concept of a graph, as described below.).

Definition 3.1 ([5]) *A* hypergraph *is a pair (V, E), where V is a set of* vertices, *and E is a set of* edges. *Each edge is a (non-empty) subset of V.*

In the special case where each edge contains exactly two vertices, we normally refer to the hypergraph as a *graph.* A constraint satisfaction problem where all the constraints are binary can be naturally associated with a graph, where the vertices of the graph are the variables of the problem, and there is an edge in the graph linking vertices v_1 and v_2 exactly when there is some constraint $R_i(S_i)$ with scope $S_i = \langle v_1, v_2 \rangle$. This graph is often referred to as the *constraint graph* of the problem [13].

More generally, an arbitrary constraint satisfaction problem, with constraints of any arity, can be associated with a hypergraph, where the vertices of the hypergraph are the variables of the problem, and there is an edge containing $v_1, v_2, \ldots, v_k$ exactly when there is some constraint $R_i(S_i)$ with scope $S_i = \langle v_1, v_2, \ldots, v_k \rangle$.

Example 3.2 The hypergraph associated with the constraint satisfaction problem described in Example 2.3 is

$$(\{x, y, z\}, \{\{x, y\}, \{y, z\}\}),$$

which may be represented pictorially as in Figure 3. $\Box$

Example 3.3 The hypergraph associated with the constraint satisfaction problem described in Example 2.4 is

$$(\{1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12\}, \{\{1, 2, 3, 4\}, \{4, 9, 10, 11, 12\}, \{2, 5, 6, 7\}, \{7, 8, 11\}\}),$$

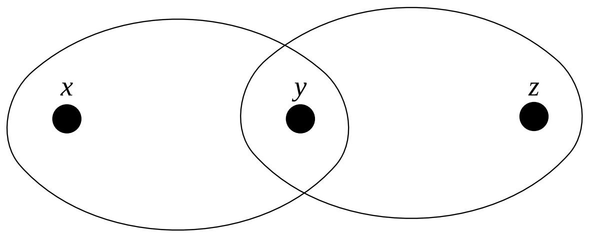

Figure 3: The hypergraph associated with Example 2.3

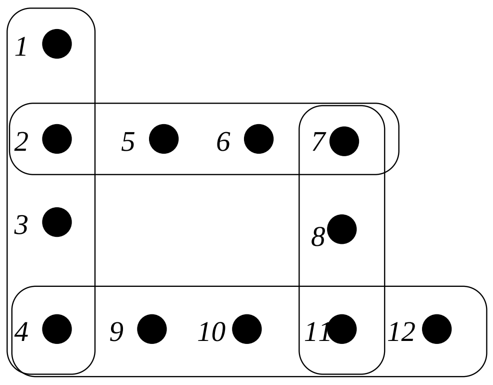

Figure 4: The hypergraph associated with Example 2.4

which may be represented pictorially as in Figure 4 □

It is important to remember that the existence of an edge in an associated graph or hypergraph only records the fact that the values assigned to certain variables are required to be related by some constraint relation. The hypergraph gives no information about which particular relation must be satisfied by the values assigned to those variables.

However, we shall now show that, if the associated graph or hypergraph is restricted in certain ways, then the corresponding problems are tractable, whatever constraint relations may be specified.

3.2 Trees

The first restricted class of hypergraphs that we shall consider is the class of (generalised) *trees*.

We give a definition of trees which can be applied to arbitrary hypergraphs as well as to graphs. This definition makes use of the the notion of a *chain* [5], which is simply a list of distinct vertices with connecting edges.

Definition 3.4 ([5]) *A* chain *of length q in a hypergraph (V, E) is defined to be a sequence $(x_1, E_1, x_2, E_2, \ldots, E_q, x_{q+1})$ such that:*

- *$x_1, x_2, \ldots, x_q$ are all distinct vertices from V;*
- *$E_1, \ldots, E_q$ are distinct edges from E;*
- *$x_k, x_{k+1} \in E_k$ for $k = 1, 2, \ldots, q$.*

A chain of length greater than 1 is said to be cyclic *if $x_1 = x_{q+1}$.*

A hypergraph is said to be a tree *if it contains no cyclic chains.*

The following result was established by Montanari, in the very first paper to deal explicitly with constraint satisfaction problems [34]. It was later obtained by Freuder [19], as a special case of a much more general result, to be discussed below (Section 3.4).

Theorem 3.5 ([34, 19]) *Let $\mathcal{C}_{tree}$ be the class of all binary constraint satisfaction problems for which the associated constraint graph is a tree.*

$\mathcal{C}_{tree}$ is tractable.

In other words, there is a polynomial-time algorithm which solves any binary constraint problem for which the associated graph is a tree, regardless of the constraint relations.

To prove Theorem 3.5 we need to establish that there is an efficient algorithm which can solve any tree-structured problem.

The algorithm we shall describe deals with both binary and non-binary problems. It has three stages. In the first stage it chooses a particular ordering of the edges, in the second stage it tightens the constraints, and in the third stage it constructs an assignment, by assigning values to the variables of each edge in turn.

Algorithm 3.6

Input: *A constraint satisfaction problem $\mathcal{P}$ whose associated hypergraph G is a tree;*

Output: *A solution to $\mathcal{P}$ (or a signal that there are no solutions).*

Stage 1: *While G contains any edges, do the following:*

1. *Remove all vertices in G which belong to only one edge (such a vertex exists because all chains in G must terminate, since G is a tree).*
2. *Remove all edges which are now empty, or completely contained in another edge, and add these edges to the ordering (in any order).*

Stage 2: *Assume that the list of edges in the chosen edge ordering is $e_1, \ldots, e_n$, and let $R_1(S_1), R_2(S_2), \ldots, R_n(S_n)$ be the corresponding constraints of $\mathcal{P}$.*

For each i in the range $1, 2, \ldots, n$, and each $j > i$, replace the constraint $R_j(S_j)$ with the constraint $R'_j(S_j)$, where

$$R'_j = \pi_{S_j}(R_i \bowtie R_j)$$

If any of the resulting constraints are empty, then terminate and signal that $\mathcal{P}$ has no solutions;

Stage 3: *For $i = n, n-1, \ldots, 1$, assign values to the variables whose associated vertices lie in $e_n, e_{n-1}, \ldots, e_i$, such that the assignment obtained at each step extends the assignment at the previous step, and is a partial solution.*

At no point in Stage 3 does the algorithm need to backtrack and undo any previous assignment. This is because, at each step in the assignment of values to the variables, all the possible choices of assignment extend to some possible choice of values for the variables introduced at the next step, otherwise that choice of values would be removed in Stage 2.

In fact, it is easy to see that Algorithm 3.6 works for a wider class of hypergraphs than trees. Any hypergraph where the set of edges can be totally ordered, by Stage 1 of Algorithm 3.6, will be solved correctly by the rest of the Algorithm, without backtracking. Any hypergraph for which Stage 1 of Algorithm 3.6 can successfully order the complete set of edges is referred to as *acyclic* [16]. We can therefore generalise Theorem 3.5, as follows.

Theorem 3.7 *Let $\boldsymbol{C}_{acyclic}$ be the class of all constraint satisfaction problems for which the associated hypergraph is acyclic.*

$\boldsymbol{C}_{acyclic}$ is tractable.

(This generalisation was pointed out in [13].)

The technique of successively removing edges in the way we have described, in order to determine whether or not a hypergraph is acyclic, is referred to as GYO reduction [41]. Many other characterisations of acyclic hypergraphs have been identified [16], and we shall give another useful characterisation in the next section. The desirable properties of such hypergraphs are well-known in relational database theory [4].

Acyclic hypergraphs include the class of generalised trees defined above, but for non-binary hypergraphs they represent a significant generalisation of this class, as the next example illustrates.

Example 3.8 The hypergraph illustrated in Figure 5 is acyclic, but it is *not* a tree (removing the edge represented by the heavy line leaves a cyclic chain). □

In order to obtain even larger tractable classes we need to further generalise the ideas described in this section. This has been done in two essentially different ways, which will be described in the next two sections.

3.3 Decomposing problems

If we have a constraint satisfaction problem with an associated hypergraph that breaks up into two separate disconnected components, as illustrated in Figure 6, then it is clear that each of these components can be solved independently. In fact, even if the two parts of the problem share an edge in common, as shown in Figure 7, then after solving one part, information can be carried forth into the second part which can be used to solve that part of the problem in a compatible way.

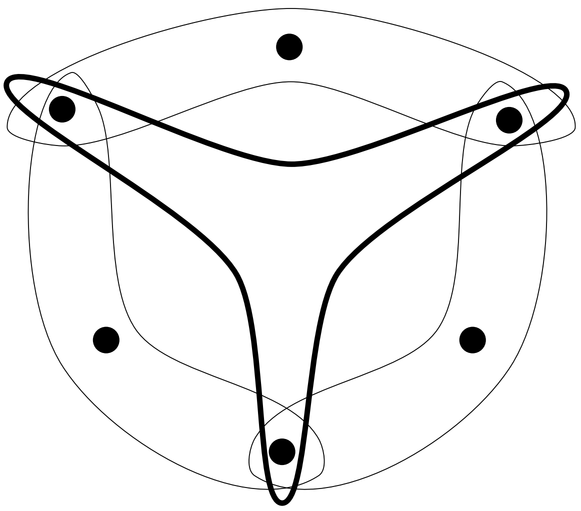
Figure 5: An acyclic hypergraph

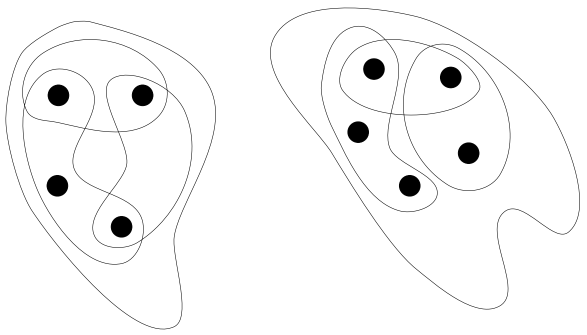
Figure 6: A disconnected hypergraph

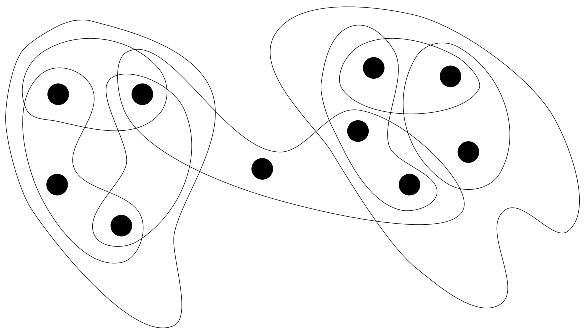
Figure 7: A hypergraph with limited connectivity

The idea of decomposing a problem into smaller pieces with limited interconnections, which can be solved separately, was first explored by Freuder in [20]. Freuder showed that binary constraint satisfaction problems can be decomposed into smaller problems corresponding to the *biconnected components* [5] of the associated graph.

The idea was extended, and generalised to hypergraphs, by Gyssens et al. [23], who introduced the notion of *hinges* as the fundamental building blocks of any graph or hypergraph. Using this idea allows us to identify a much wider class of tractable hypergraphs than the acyclic hypergraphs discussed in Section 3.2.

To describe these ideas, we first define precisely what it means for a set of edges in a hypergraph to be connected.

Definition 3.9 *For any hypergraph (V, E), and any subset of edges $F \subseteq E$, we say that F is* connected *if for any two edges, $e, f \in F$, there exists a sequence of edges $e_1, \ldots, e_n$, with*

- *$e_1 = e$;*
- *$e_n = f$;*
- *for $i = 1, 2, \ldots, n$, $e_i \cap e_{i+1} \neq \emptyset$.*

We refer to a subset $F \subseteq E$ as a *maximal connected component* if it is a connected subset, and there is no larger connected subset containing it. The hypergraph shown in Figure 6 has two maximal connected components, and the hypergraph shown in Figure 7 has only one maximal connected component.

We now refine the notion of connectedness, to allow us to identify collections of edges which separate others.

Definition 3.10 *For any hypergraph (V, E), any subset of edges $H \subseteq E$, and any subset of edges $F \subseteq E$, we say that F is* connected with respect to H *if for any two edges, $e, f \in F$, there exists a sequence of edges $e_1, \ldots, e_n$, with*

- *$e_1 = e$;*
- *$e_n = f$;*
- *for $i = 1, 2, \ldots, n$, $e_I \cap e_{i+1} \not\subseteq \bigcup H$.*

(Note that $\bigcup H$ is the set of all vertices which occur within the edges in the set H.)

Now we are in a position to define a *hinge* of a hypergraph. Informally, a hinge is a set of at least two edges which cuts the hypergraph into separate connected components such that each connected component intersects with the hinge within only one edge. The precise definition is as follows.

Definition 3.11 ([24, 23]) *Let (V, E) be a hypergraph, $H \subseteq E$ be a set of at least two edges, and $H_1, \ldots, H_n$ be the connected components of (V, E) with respect to H. We shall say that H is a* hinge *if, for $i = 1, \ldots, n$, there exists an edge $h_i \in H$ such that:*

$$(\bigcup H_i) \cap (\bigcup H) \subseteq h_i$$

Example 3.12

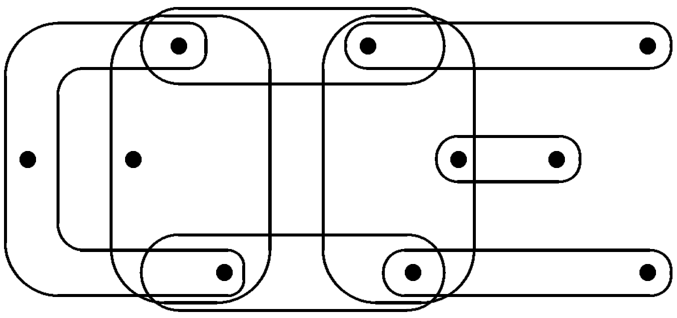

Figure 8: A hypergraph

Consider the hypergraph illustrated in Figure 8. Figures 9 and 10 show two of the hinges contained in this hypergraph. □

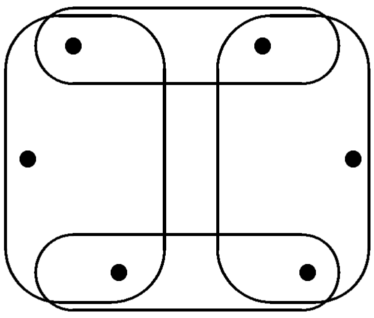

Figure 9: A hinge of Figure 8

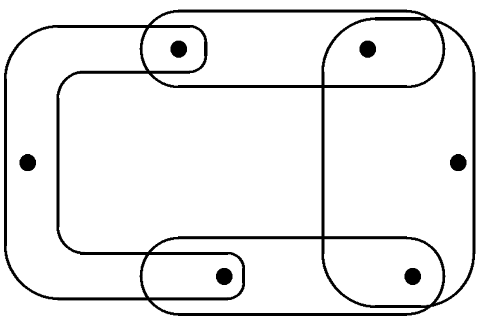

Figure 10: Another hinge of Figure 8

A *minimal hinge* is a hinge that does not contain any other hinges. It is shown in [23] that that the minimal hinges of a hypergraph are fundamental structural components. In particular, *any* hypergraph can be decomposed into a collection of minimal hinges, which overlap each other in a tree structure. This structure is referred to as a *hinge-tree*, and is defined as follows.

Definition 3.13 ([23]) *A* hinge-tree *of a hypergraph* (V, E) *is a tree* (N, A) *with the following properties:*

- *each tree node,* $n \in N$*, is a minimal hinge of* (V, E)*;*
- *each edge of the hypergraph is contained in at least one tree node (i.e.* $\bigcup N = E$*);*
- *adjacent tree nodes share exactly one edge of the hypergraph;*
- *the vertices shared by any two tree nodes are entirely contained within each tree node on their connecting path in the tree.*

Example 3.14 Figure 11 shows one possible hinge-tree for the hypergraph described in Example 3.12, and illustrated in Figure 8. □

It is possible to calculate a hinge-tree for any given hypergraph in a time which is polynomial in the size of that hypergraph [23].

For any given hypergraph there may be more than one hinge-tree, and they may contain different minimal hinges, but it is shown in [23, 26] that they all have an important feature in common.

Theorem 3.15 ([23, 26]) *For any hypergraph* (V, E)*, there is a number,* Δ*, called the* degree of cyclicity*, such that, in all hinge-trees of* (V, E)*, the largest node has exactly* Δ *edges.*

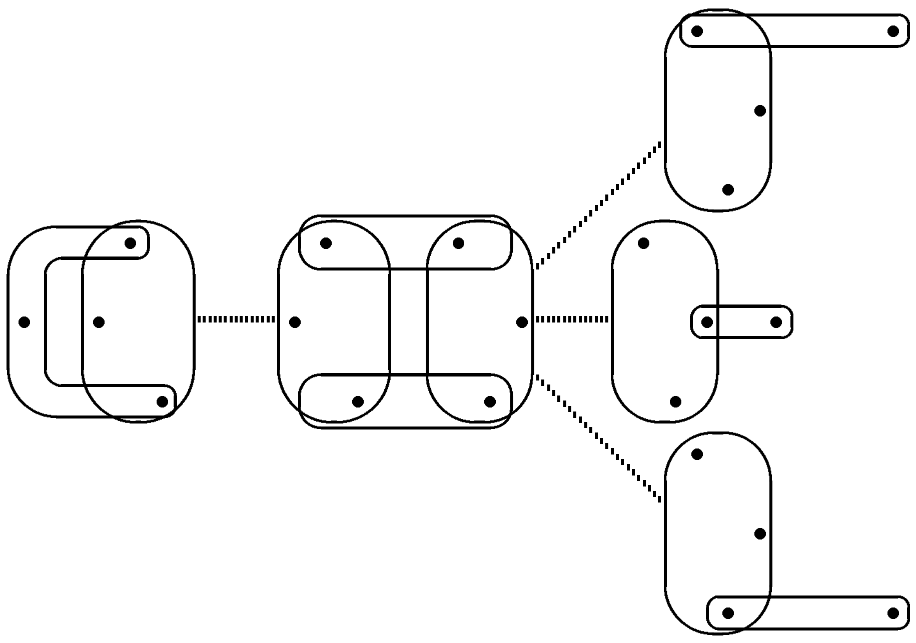

Figure 11: A possible hinge tree of Figure 8

If the degree of cyclicity of a hypergraph is 2, then the hypergraph is acyclic [26][4].

For any constraint satisfaction problem, any corresponding hinge-tree can be used to obtain a new constraint satisfaction problem, which has the same solutions as the original problem, but whose associated hypergraph is acyclic [23]. To construct this equivalent problem one simply solves each of the constraint satisfaction problems associated with each of the hinges in the hinge-tree, and replaces that hinge with a single constraint relation, consisting of the set of solutions[5]. The resulting acyclic constraint satisfaction problem can then be solved in polynomial time, as described in Section 3.2.

If the degree of cyclicity is small, then each minimal hinge in the hinge-tree will be small (i.e., will contain a small number of edges), so each of the corresponding constraint satisfaction problems will be small, and hence the cost of solving each of them separately and constructing the equivalent tree-structured constraint satisfaction problem will be small. In general, the time complexity of constructing a hinge-tree, then constructing the corresponding tree-structured problem, and then solving that, is:

$$O(|V|n^2) + O(nl^{\Delta}\Delta logl)$$

[4]This definition of acyclic hypergraphs has been shown to be equivalent [24] to the more standard definitions, given in, for example, [4, 16]

[5]This procedure is equivalent to the 'perfect relaxation' strategy described in [35]

where V is the set of variables of the problem,n is the number of constraints, l is the size of the largest constraint (number of tuples), and Δ is the degree of cyclicity [23].

It follows from this that, if the degree of cyclicity is fixed, then we have a polynomial-time algorithm for the corresponding constraint satisfaction problems.

Theorem 3.16 ([23]) *For any fixed value of Δ, the class of constraint satisfaction problems whose associated hypergraphs have degree of cyclicity at most Δ is tractable.*

The class of hypergraphs with degree of cyclicity at most Δ, for some fixed Δ, is much larger than the class of acyclic hypergraphs, but problems associated with these hypergraphs still remain tractable. Therefore the class of tractable constraint satisfaction problems is much larger than would be expected from the results in Section 3.2.

Unfortunately, for many hypergraphs (for example, Figure 4) the degree of cyclicity is the same as (or close to) the number of edges in the hypergraph. When this is the case it can be shown that the hypergraph cannot, in general, be decomposed into smaller units which can be solved separately, regardless of the constraint relations [23].

However, as we remarked earlier, the degree of cyclicity of a hypergraph can be determined in polynomial time [23], so the hinge-tree decomposition technique can always be used as a first step on a given problem without sacrificing efficiency[6].

Another approach to decomposing constraint satisfaction problems is described in [13]. This approach involves forming subproblems from clusters of variables and then solving these subproblems separately. For many problems this clustering approach results in a finer decomposition than the hinge-tree method described here, and hence this approach can be very useful in practice. On the other hand, it is not clear how to obtain a tight bound on the size of the clusters which are formed in this technique. Hence this clustering approach does not lead to the specification of tractable problem classes whose members can be efficiently identified. (For a comparison between the two approaches, and suggestions on how to combine them, see [23].)

[6]A direct comparison between the notion of degree of cyclicity and the notion of *width* (described in Section 3.4) is problematic, because the first is defined in terms of the number of *edges* and the second is defined in terms of the number of *vertices*. For arbitrary hypergraphs there may be arbitrary numbers of vertices in an edge. We therefore claim that *both* measures may be useful in different contexts

3.4 Consistency and backtrack-free search

One of the key concepts which has been applied to the study of constraint satisfaction problems is the concept of *consistency.* In this section, we shall define this concept, and then examine how it may be used to identify tractable problem classes.

The basic insight behind the notion of consistency is that much of the information in a constraint satisfaction problem is present only *implicitly.* This information may be discovered during the course of a search, for example, when certain combinations of values are found to be disallowed by some collection of constraints, and the search is forced to backtrack. It may be possible to guide a search procedure more effectively, and hence find a solution more quickly, if this implicit information is made explicit. This can be done by adding additional constraints to the problem, as described below[7].

We now define various forms of consistency which have been widely-used in the literature.

Definition 3.17 ([18]) *A problem is said to be k-*consistent *if every partial solution on any set of $k-1$ variables can be extended to a partial solution on any superset containing k variables.*

A problem is said to be strong *k-consistent if it is i-consistent for $i = 1 \ldots k$.*

A problem is said to be globally *consistent if any partial solution can be extended to a full solution.*

The maximal value of k for which k-consistency holds is referred to as the *level* of consistency present in a problem.

Any constraint satisfaction problem can be made k-consistent for any fixed k in polynomial time, by the addition of extra constraints. For each variable v, simply add new constraints on each subset of $k-1$ variables to ensure that all of the allowed combinations of values for these variables are consistent with some assignment of v. The required constraint relations can be constructed from the constraint relations in the original problem using a combination of join and projection operators. (For a more detailed analysis of efficient algorithms which can modify a problem to ensure k-consistency see [10, 11].)

The earliest results concerning consistency and tractability were obtained by Freuder [19]. These results give conditions on a constraint graph that

[7]There is a useful analogy between the process of adding or modifying constraints based on implicit information and the *presolve* process used in most commercial linear programming systems to pre-process the problem formulation [6]

guarantee efficient backtrack search when the problem has a certain level of strong consistency.

Freuder's central result relies on the notion of *width*, which is well-established in graph theory (although less commonly applied to hypergraphs). Intuitively, in any hypergraph where the vertices are arranged in some order, the width of a vertex is the number of earlier vertices to which it is connected.

Definition 3.18 *Given a hypergraph (V, E), and an ordering $\sqsubseteq$ on V, the* width of a vertex *v is the size of the set:*

$$\{w \mid w \sqsubseteq v \text{ and } \exists e \in E, \{v, w\} \subseteq e\}$$

The width of the ordering *is the maximum width over all the vertices.*

The width of the hypergraph *(V, E) is the minimum width over all possible orderings of V.*

Example 3.19 Consider the hypergraph illustrated in Figure 4, with the vertex ordering

$$1 \sqsubseteq 2 \sqsubseteq 3 \sqsubseteq 4 \sqsubseteq 5 \sqsubseteq 6 \sqsubseteq 7 \sqsubseteq 8 \sqsubseteq 9 \sqsubseteq 10 \sqsubseteq 11 \sqsubseteq 12.$$

A straightforward check shows that:

- the width of vertex 11 is 5;
- there is no other vertex which has width greater than 5 in this ordering, so the width of this ordering is 5.

Now consider the same hypergraph, with the vertex ordering

$$2 \sqsubseteq 7 \sqsubseteq 4 \sqsubseteq 11 \sqsubseteq 1 \sqsubseteq 3 \sqsubseteq 5 \sqsubseteq 6 \sqsubseteq 8 \sqsubseteq 9 \sqsubseteq 10 \sqsubseteq 12.$$

A straightforward check shows that:

- the width of vertex 12 is 4;
- there is no other vertex which has width greater than 4 in this ordering, so the width of this ordering is 4.
- there is no other ordering which has width less than 4, so the width of this hypergraph is 4.

□

Theorem 3.20 ([19]) *Let $\mathcal{P}$ be a constraint satisfaction problem, let (V, E) be its associated hypergraph, and let $\sqsubseteq$ be an ordering of V with width w.*

If $\mathcal{P}$ is strong $(w+1)$-consistent, then a solution to $\mathcal{P}$ can be obtained by performing a backtrack-free search using this variable ordering.

A proof of this theorem, for the special case of binary constraint satisfaction problems, is given in [19]. The generalisation to problems of arbitrary arity is straightforward. Intuitively, the result is obtained because at each step which assigns a value to a variable the number of previously assigned values which need to be taken into account is at most the width of the ordering. Hence, if the level of consistency is greater than the width of the ordering, then the partial assignments can be extended at each step, and the search will proceed without backtracking.

Theorem 3.20 can be generalised to allow a weaker form of consistency, known as *directional consistency* [12]. In fact, many of the results concerning consistency which appear in the literature can be generalised in this way, but this idea will not be explored further in this report.

One difficulty with applying Theorem 3.20, is that, in general, finding a minimal width ordering of a graph or hypergraph is an NP-complete problem [2, 22]. However, there are heuristic approaches which obtain an ordering whose width is a good approximation to the hypergraph width in many cases [13].

It has been shown [1] that the width of any graph can be characterised in terms of a generalised form of tree-structure, known as a *k-tree*, which is defined as follows. First note that a *k-clique* in a graph is a set of k vertices, such that there is an edge containing each pair.

Definition 3.21 ([1]) *A k-*tree *is defined recursively as follows:*

- *A k-clique is a k-tree.*
- *Any graph obtained by adding an extra vertex to an existing k-tree, and edges from this vertex to all the vertices of some existing k-clique, is a k-tree.*

A graph has width less than or equal to k if and only if it is a subgraph of a k-tree [1]. In particular, a graph has width 1 if and only if it is a tree, as defined in Section 3.2. Hence, the results in this section can be seen as a generalisation of the results for trees, above (see [21]). In fact, for binary problems whose associated graph is a tree, Theorem 3.20 shows that all that is required to ensure backtrack-free search is strong 2-consistency. (For binary problems, the algorithm described in Section 3.2 essentially finds a minimal width ordering (in reverse!) and then enforces a weak form of 2-consistency.)

4 Tractability due to restricted constraints

4.1 Closure, clones and complexity

The characterisations of tractable constraint satisfaction problems that we have discussed so far have been in terms of the structure of the associated hypergraph. We now investigate properties of the constraint *relations* which are sufficient to ensure tractability, regardless of the associated hypergraph.

It turns out that the relevant properties of relations are algebraic *closure* properties, which are defined as follows.

Definition 4.1 *Given a k-ary relation R and a function $\phi : D^n \to D$, we say that R is* closed under *ϕ, if for all collections of tuples,*

$$\begin{array}{rcl} \langle d_1^1, d_2^1, \ldots, d_k^1 \rangle & \in & R \\ \langle d_1^2, d_2^2, \ldots, d_k^2 \rangle & \in & R \\ & \vdots & \\ \langle d_1^n, d_2^n, \ldots, d_k^n \rangle & \in & R \end{array}$$

the tuple

$$\langle \phi(d_1^1, d_1^2, \ldots d_1^n), \phi(d_2^1, \ldots, d_2^n), \ldots, \phi(d_k^1, \ldots d_k^n) \rangle$$

also belongs to R.

Example 4.2 The relation

$$R_2 = \{\langle 4,5\rangle, \langle 4,6\rangle, \langle 5,4\rangle, \langle 5,5\rangle, \langle 5,6\rangle, \langle 6,4\rangle, \langle 6,5\rangle, \langle 6,6\rangle\}$$

which was introduced in Example 2.3 is closed under the binary operation max, which returns the maximum value of its two arguments. For example,

$$\max(\langle 4,5\rangle, \langle 6,4\rangle) = \langle \max(4,6), \max(5,4)\rangle = \langle 6,5\rangle \in R_2.$$

On the other hand, the relation

$$R_1 = \{\langle 1,5\rangle, \langle 2,4\rangle, \langle 3,3\rangle, \langle 4,2\rangle, \langle 5,1\rangle\},$$

also introduced in Example 2.3, is *not* closed under this operation. For example,

$$\max(\langle 1,5\rangle, \langle 5,1\rangle) = \langle \max(1,5), \max(5,1)\rangle = \langle 5,5\rangle \notin R_2.$$

□

We note that when a relation is closed under an operation, then any projection of that relation is also closed under that operation. Furthermore, the join of any two relations closed under an operation is also closed under that operation [27].

Throughout this section we shall assume that Γ is a set of relations over a finite set D with at least two elements.

Notation 4.3

- *The class of all constraint satisfaction problems in which the constraint relations are members of Γ will be denoted $\mathcal{C}_\Gamma$.*
- *The set of all functions $\phi : D^n \to D$, for arbitrary values of k, under which every member of Γ is closed, will be denoted Fun(Γ).*

The following result was recently established by Jeavons [25].

Theorem 4.4 **([25])** *The complexity of $\mathcal{C}_\Gamma$ is determined by Fun(Γ).*

For any set of relations Γ, the set of functions Fun(Γ) has certain algebraic properties which mean that this set is a *clone* [9, 39]. There are very general algebraic results about clones [39, 37] which show that the possibilities for Fun(Γ) are therefore limited in certain ways, as the next result indicates.

Theorem 4.5 **([28])** *For any set of relations, Γ, over a finite set D, the set Fun(Γ) must contain at least one of the following six types of functions:*

1. *A constant function;*
2. *An* idempotent *binary function, that is, a function ϕ, of arity 2 such that $\phi(d, d) = d$, for all $d \in D$;*
3. *A* majority *function, that is, a function ϕ, of arity 3 such that $\phi(d, d, d') = \phi(d, d', d) = \phi(d', d, d) = d$, for all $d, d' \in D$;*
4. *An* affine *function, that is, a function ϕ ,of arity 3 such that $\phi(d_1, d_2, d_3) = d_1 - d_2 + d_3$, for all $d_1, d_2, d_3 \in D$, where $+$ is a binary operation on D that gives D an Abelian group structure;*
5. *A* semiprojection, *that is, a function ϕ of arity $n > 3$, such that $\phi(d_1, \ldots, d_n) = d_i$ for some i, for all $d_1, \ldots, d_n \in D$ with $|\{d_1, d_2, \ldots, d_n\}| < n$;*
6. *An* essentially unary *function, that is, a function ϕ of arity n such that $\phi(d_1, \ldots, d_n) = f(d_i)$ for some i and some non-constant unary function f, for all $d_1, \ldots, d_n \in D$.*

By examining each of these possibilities in turn, it is possible to obtain an almost complete classification of the complexity of $\mathbf{C}_\Gamma$, for any given set of relations Γ.

In the following sections we shall examine each part of Theorem 4.5 in a little more detail, to illustrate the various kinds of tractable problems which have been identified using these techniques, and the algorithms which solve these problems efficiently.

4.2 Constant functions

The first case we examine is rather trivial, but introduces the flavour of the rest of the results.

Theorem 4.6 ([28]) *If the set Fun(Γ) contains a constant operation, then $\mathbf{C}_\Gamma$ is tractable, and can be solved in constant space.*

Proof: Suppose Fun(Γ) contains the constant function ϕ, which always returns the value d. Let $\mathcal{P}$ be any constraint satisfaction problem in $\mathbf{C}_\Gamma$. If any of the constraint relations used in $\mathcal{P}$ are empty, then the problem has no solution. Otherwise, because each of these relations is closed under ϕ, each constraint relation contains a tuple $\langle d, d, \ldots, d\rangle$. Assigning d to each variable will therefore be a solution. ■

4.3 Majority functions and near-unanimity functions

It was first shown in [27] that if a set of relations, Γ, is closed under a majority function, then $\mathbf{C}_\Gamma$ is tractable. We shall here prove a slightly more general result, which concerns the class of functions known as *near-unanimity functions.*

Definition 4.7 ([3]) *A* near-unanimity function *of arity n is a function ν such that, for all $x_1, \ldots x_n$ where at least $n-1$ of the x_i are equal to x*

$$\nu(x_1, \ldots x_n) = x$$

(Note that a majority function is a near-unanimity function of arity 3.)

There is a remarkably close connection between near-unanimity functions and consistency properties, as the next result indicates.

Theorem 4.8 *For any set of relations Γ, the following are equivalent:*

- *Every relation in Γ is closed under a near-unanimity function of arity n,*

- *Every constraint satisfaction problem in $\mathbf{C}_\Gamma$ that is strong n-consistent is globally consistent.*

Proof: This result follows from a classical result of universal algebra concerning near-unanimity functions, obtained by Baker and Pixley [3]. This result states that every algebra in a variety contains a near-unanimity function of arity n amongst its term operations if and only if all subalgebras of product algebras in that variety are uniquely determined by their $(n-1)$-ary projections [3].

Now let Γ be a fixed set of relations over D, and let $\mathcal{P}$ be any constraint satisfaction problem in $\mathbf{C}_\Gamma$. To apply the above result, we note that the set $\mathrm{Sol}(\mathcal{P})$ may be seen as a subalgebra of a direct product of algebras of the form $(D, \mathrm{Fun}(\Gamma))$. In fact, by Theorem 10 of [8], *every* possible subalgebra can be obtained as the projection of $\mathrm{Sol}(\mathcal{P})$, for some $\mathcal{P}$.

Hence, by the result of Baker and Pixley, $\mathrm{Sol}(\mathcal{P})$ is determined by its $(n-1)$-ary projections, which means that if $\mathcal{P}$ is strong n-consistent, then it is globally consistent. Conversely, if this property holds for every $\mathcal{P}$ in $\mathbf{C}_\Gamma$, then $\mathrm{Fun}(\Gamma)$ must contain a near-unanimity function. ■

We can use this result to show that when Γ is closed under a near-unanimity function, then the whole of $\mathbf{C}_\Gamma$ is tractable.

Corollary 4.9 *If Γ is closed under a near-unanimity function, then the class of problems $\mathbf{C}_\Gamma$ is tractable.*

Proof: As we remarked earlier, any problem can be made strong n-consistent, for any fixed n, in polynomial time [10]. The new constraints introduced by this process can be obtained from the original constraints by some sequence of join and projection operations, so they are all closed under the near-unanimity function. Now applying Theorem 4.8 to the set of constraint relations in the strong n-consistent problem gives the result. ■

We also remark that Theorem 4.8 indicates that if the constraint relations in any problem are all closed under some near-unanimity function of arity n, then any level of consistency can be achieved without increasing the arity of the constraints beyond $n-1$.

A similar result to Theorem 4.8 is given in [17], although the connection with consistency is not made explicit. (The full version of [17] also gives an interesting characterisation of closure under near-unanimity functions in terms of the language Datalog, which is widely-used to specify relations.)

4.4 Affine functions

If the relations in Γ are closed under an affine function, then $\mathbf{C}_\Gamma$ is tractable [28, 17]. To establish this result in the general case requires sophisticated group-theoretic techniques, which are beyond the scope of this report. (The proof relies on showing that any relation closed under an affine function is a coset of a product group of an Abelian group with universe D.)

In the special case when the domain contains a prime number of elements the situation is much simpler. In this case, any relation which is closed under an affine function must be of the following form [27]:

$$\{(x_1, x_2, \ldots, x_r) \in \mathbf{Z}_p^r \mid \sum_{i=1}^{r} a_i x_i \cong a\} \text{ for some } a, a_1, a_2, \ldots, a_r \in Z_p.$$

Thus, any constraint satisfaction problem over a prime domain size d, with constraint relations that are closed under an affine operation, corresponds to a set of simultaneous linear equations over the integers modulo d. Such a set of equations can be solved in polynomial time using a standard technique of linear algebra, such as Gaussian elimination.

4.5 Idempotent binary functions

When Γ is closed under a binary idempotent function, ϕ, it is possible for $\mathbf{C}_\Gamma$ to be either tractable, or NP-complete, depending on the precise nature of the function ϕ.

At present, no criterion is known which will distinguish between these two possibilities in the general case. However, in two special cases it is possible to be more specific about the complexity of $\mathbf{C}_\Gamma$.

4.5.1 ACI functions

Binary functions may be any combination of the following:

- idempotent (i.e. $\phi(d, d) = d$, for all $d \in D$);
- commutative (i.e. $\phi(d_1, d_2) = \phi(d_2, d_1)$, for all $d_1, d_2 \in D$);
- associative (i.e. $\phi(d_1, \phi(d_2, d_3)) = \phi(\phi(d_1, d_2), d_3)$, for all $d_1, d_2, d_3 \in D$).

A binary idempotent function which is both associative and commutative is known as an *ACI function*. The complexity of constraint satisfaction problems in which the constraint relations are closed under ACI functions was first investigated in [30].

Theorem 4.10 ([27, 28]) *If Γ is closed under a binary idempotent function which is associative and commutative, then $\mathcal{C}_\Gamma$ is tractable.*

Example 4.11 The constraint programming language CHIP[45] incorporates constraint solving techniques for certain arithmetical constraints. In particular, it includes efficient algorithms for constraints over the natural numbers of the following forms:

- *domain constraints*, which are unary constraints which restrict the values of individual variables to some finite set ; and
- *arithmetic constraints*, which have one of the following 4 forms:
  - $aX \neq b$,
  - $aX = bY + c$,
  - $aX \leq bY + c$,
  - $aX \geq bY + c$,

  where upper case letters represent variables, lower case letters represent positive constants, and a is non-zero.

All of these constraints are closed under the ACI function max, which yields the arithmetic maximum of its two arguments, and hence are tractable by the result above. Further, the following constraints (listed in[30]) are also closed under this ACI function, and could therefore be added to the CHIP system without compromising the efficiency of the system.

- $a_1X_1 + a_2X_2 + \ldots a_rX_r \geq bY + c$,
- $aX_1X_2 \ldots X_r \geq bY + c$,
- $(a_1X_1 \geq b_1) \vee (a_2X_2 \geq b_2) \ldots \vee (a_rX_r \geq b_r) \vee (aY \leq b)$,

$\square$

To clarify the structure of relations which are closed under an ACI function, we first describe the close connection between ACI functions and orderings of the domain. Given an ACI function, ϕ, on a set D, we can define a partial order, $\sqsubseteq$, on D by setting:

$$d_1 \sqsubseteq d_2 \longleftrightarrow \sqcup(d_1, d_2) = d_2$$

In this partial ordering the least upper bound of d_1 and d_2 is given by $\phi(d_1, d_2)$.

Because any ACI function is ϕ calculates a least upper bound, or maximum, of its arguments, relative to this ordering, constraints in which the constraint relation is closed under an ACI function were called *max-closed* constraints in [30][8]. One possible algorithm for solving problems in which the constraint relations are closed under an ACI function works as follows [30]. First, establish what is called *pairwise consistency*, by repeatedly forming the join of every pair of constraints and projecting the result onto the original scopes, until there are no further changes in the constraints. Now, for each variable v, set

$$D(v) = \bigcap\{\pi_{\{v\}}R_i(S_i) \mid v \in S_i\}$$

Each variable v now has an associated domain of values $D(v)$, such that $d \in D(v)$ if and only if the projection of every constraint onto v contains d. These sets $D(v)$ are still closed under the same ACI function (because they are obtained from the original constraints by a sequence of join and projection operations). If any set $D(v)$ is empty, then the problem has no solution. If all $D(v)$ are non-empty, then assigning the least upper bound of the set $D(v)$ to the variable v gives a solution to the problem.

4.5.2 Rectangular band functions

A *rectangular band* function [33] is an associative, idempotent, binary function ϕ, such that $\phi(d_1, \phi(d_2, d_3)) = \phi(d_1, d_3)$ for all $d_1, d_2, d_3 \in D$.

We will now show that closure under a rectangular band function is not a sufficient condition for tractability. We do this by giving an example[9] of an NP-complete problem class in which the constraint relations are all closed under a rectangular band function.

Example 4.12 Let S and T be sets, with $S = \{s_1, s_2, \ldots, s_k\}$, let $D = S \times T$, and consider the following binary relation over D:

$$R = \{\langle\langle s,t\rangle, \langle s',t'\rangle\rangle \in (S \times T)^2 \mid (s \neq s')\}$$

Note that this relation is closed under the rectangular band function, ρ_0, on $S \times T$, defined by $\rho_0(\langle s,t\rangle, \langle s',t'\rangle) = \langle s,t'\rangle$.

We will now show that $\mathbf{C}_{\{R\}}$ is NP-complete, by showing that the k-COLORABILITY problem [22], can be reduced to $\mathbf{C}_{\{R\}}$ in polynomial time.

Let $\mathcal{P}$ be an instance of the k-COLORABILITY problem, specified by a graph (V, E), where $E = \{e_1, e_2, \ldots, e_n\}$. Now consider the problem $\mathcal{P}' =$

[8]Actually, in [30] the order $\sqsubseteq$ was required to be a total order, but the results are still valid when the order is a partial order, as shown in [27]

[9]This example was suggested by Marc Gyssens.

$(V, D, R(e_1), R(e_2), \ldots, R(e_n))$, in which the constraint scopes correspond to the edges of E, and the constraint relation in each constraint is the relation R defined above. Any solution to $\mathcal{P}'$ can be transformed into a solution to $\mathcal{P}$ by mapping values of the form $\langle s_i, t_j \rangle$ to i. Conversely, any solution to $\mathcal{P}$ can be transformed into a solution of $\mathcal{P}'$ by simply assigning $\langle s_i, t \rangle$ to v if $v = i$, for some $t \in T$. These transforms can be carried out in polynomial time, so the result follows. $\Box$

It is known from general algebraic results [33] that for *any* rectangular band function $\rho : D^2 \to D$, the algebra (D, ρ) is isomorphic to the algebra $(S \times T, \rho_0)$ for some sets S and T, where $\rho_0(\langle s,t \rangle, \langle s', t' \rangle) = \langle s, t' \rangle$, as in the example above. Hence, by Theorem 4.4, $\mathbf{C}_\Gamma$ is NP-complete for *any* Γ which is only closed under a rectangular band function, but a proof of this requires more theoretical machinery than is presented here (see [25]).

4.6 Semiprojections

We will now show that closure under a semiprojection is not a sufficient condition for tractability. To do this we give an example of an NP-complete problem class in which the constraint relations are closed under all semiprojections.

Example 4.13 Consider the set, Δ_3, of relations over $\{0, 1, 2\}$, defined as follows.

$$\Delta_3 = \{\{0,1\}^3 - \{t\} \mid t \in \{0,1\}^3\}$$

The set Δ_3 contains 2^3 relations, where each relation contains all 3-tuples over $\{0, 1\}$ except for one.

Now consider any constraint satisfaction problem $\mathcal{P}$ in $\mathbf{C}_{\Delta_3}$. We shall show that each constraint in $\mathcal{P}$ can be seen as expressing a Boolean disjunction. For example, a constraint with scope $\langle v_1, v_2, v_3 \rangle$, and relation $\{\{0,1\} \setminus \langle 1, 0, 1 \rangle\}$ allows any combination of the values 0 and 1 for v_1, v_2, v_3 except for $v_1 = 1, v_2 = 0, v_3 = 1$. This can be expressed by the Boolean formula $\neg(v_1 \wedge \neg v_2 \wedge v_3) = v_1 \vee v_2 \vee \neg v_3$. Conversely, any Boolean disjunction involving 3 distinct variables is satisfied by any combination of Boolean values for those variables except one, so it can be expressed with a relation from Δ_3. Hence, there is a polynomial time reduction from the 3-SATISFIABILITY problem [36], which is NP-complete, to $\mathbf{C}_{\Delta_3}$. This establishes that $\mathbf{C}_{\Delta_3}$ is NP-complete.

However, every relation in Δ_3 only involves 2 distinct values, 0 and 1, so it is easy to show that it is closed under *every* semiprojection on the domain of $\mathcal{P}$, which was actually set to be $\{0, 1, 2\}$. $\Box$

Theorem 4.4 can be used to show that $\mathbf{C}_\Gamma$ is NP-complete for *any* Γ which is only closed under semiprojections, but a proof of this requires more theoretical machinery than is presented here (see [25]).

4.7 Essentially unary functions

We will now show that closure under an essentially unary function is not a sufficient condition for tractability. To do this we give an example of an NP-complete problem class in which the constraint relations are closed under all essentially unary functions.

Example 4.14 Consider the relation, N over $\{0,1\}$, defined as follows:

$$N = \{\langle 0,0,1\rangle, \langle 0,1,0\rangle, \langle 1,0,0\rangle, \langle 1,1,0\rangle, \langle 1,0,1\rangle, \langle 0,1,1\rangle\}.$$

The class of constraint satisfaction problems $\mathbf{C}_{\{N\}}$ is equivalent to the NOT-ALL-EQUAL SATISFIABILITY problem [38, 22], which is NP-complete.

However, N is closed under *every* essentially unary function on $\{0,1\}$ (the only non-constant unary functions on this set are the identity function, and the function which exchanges the two values, and both of these leave N unchanged). $\square$

Theorem 4.4 can be used to show that $\mathbf{C}_\Gamma$ is NP-complete for *any* Γ which is only closed under essentially unary functions, but a proof of this requires more theoretical machinery than is presented here (see [25]).

4.8 Summary of results for closure functions

The results about closure functions and tractability presented above are summarised in the following theorem.

Theorem 4.15 ([28])

- *If Fun*(Γ) *contains a constant function, then* $\mathbf{C}_\Gamma$ *is tractable.*
- *If Fun*(Γ) *contains a function,* ϕ*, of arity* 2 *that is associative, commutative, and idempotent, then* $\mathbf{C}_\Gamma$ *is tractable.*
- *If Fun*(Γ) *contains a majority function, then* $\mathbf{C}_\Gamma$ *is tractable.*
- *If Fun*(Γ) *contains an affine function, then* $\mathbf{C}_\Gamma$ *is tractable.*
- *If Fun*(Γ) *contains only semiprojections, then* $\mathbf{C}_\Gamma$ *is NP-complete.*

- *If Fun*(Γ) *contains only essentially unary operations, then* $\mathcal{C}_\Gamma$ *is NP-complete.*

This powerful result means that in order to determine whether $\mathbf{C}_\Gamma$ is a tractable class of problems, we simply need to calculate the closure functions of Γ. These closure functions are themselves precisely the solutions to certain constraint satisfaction problems in $\mathbf{C}_\Gamma$, which are called *indicator problems* [29, 28]. (There is one indicator problem for each arity of closure function.) For more discussion and examples of the use of indicator problems to establish tractability, see [29].

4.9 Other restricted constraint types

In the preceding sections, we have examined each of the possible forms of closure function identified in Theorem 4.5. It follows from Theorem 4.4 that we have therefore covered every possible way in which placing restrictions on the constraint relations alone can ensure tractability.

On the other hand, if we impose other conditions on the problems as well, then it is possible to obtain tractable classes of problems which do not fall into any of the categories discussed earlier. One example of a result of this type was obtained by van Beek [42], and later extended by van Beek and Dechter [44]. It concerns a class of relations known as *row-convex* relations.

Definition 4.16 *A binary relation, R, over an ordered set D is* row-convex *if, for all $d, d_1, d_2, d_3 \in D$ such that $d_1 < d_2 < d_3$ the following implication holds:*

$$\langle d, d_1 \rangle \in R \text{ and } \langle d, d_3 \rangle \in R \Rightarrow \langle d, d_2 \rangle \in R$$

(Note that if D only contains 2 elements, then the condition is satisfied vacuously, so *any* constraint over a 2-valued domain is row-convex.)

Theorem 4.17 (**[42, 44]**) *If Γ is a set of binary relations such that every relation in Γ is row-convex, then any strong 3-consistent problem in $\mathcal{C}_\Gamma$ is globally consistent.*

(Extensions of this result to non-binary constraints are given in [44].)

If $\mathcal{P}$ is a constraint satisfaction problem in $\mathbf{C}_\Gamma$ which is *not* strong 3-consistent, then it can be modified to make it strong 3-consistent, as described earlier, but doing so may introduce new constraints which are not row-convex. (In fact, Theorem 4.8 indicates that a collection of row-convex constraint relations will always give rise to relations which are not row-convex, on some problems, unless they are all closed under some majority function.)

Even if the constraints in a problem are not row-convex for one particular domain ordering, it is sometimes possible to find another ordering such that they are. It is shown in [44] that if such a re-ordering exists, then it can be calculated in polynomial time.

5 Tractability due to local properties

In this section, we consider results which relate local properties of a constraint satisfaction problem to global properties, such as the existence of a solution, or the possibility of backtrack-free search.

First, we examine the result given by Dechter in [11], which relates the arity of the constraints, the size of the domain, and the level of consistency that is required to ensure global consistency.

Theorem 5.1 ([**11**]) *Let $\mathcal{P}$ be a constraint satisfaction problem with domain size d, and let r be the length of the largest scope in $\mathcal{P}$.*

If $\mathcal{P}$ is strong $d(r-1)+1$ consistent, then it is globally consistent.

It is clear that any problem which is globally consistent may be solved efficiently by a backtrack-free search. Furthermore, it was stated above that any problem can be made k-consistent for any fixed value of k in polynomial time. In view of Theorem 5.1, it might therefore seem that it would be sufficient to achieve $d(r-1)+1$ consistency, and then solve the resulting problem in a backtrack-free way. Hence, at first sight, this result appears to imply that all constraint satisfaction problems can be solved efficiently!

Unfortunately, this is not the case, because achieving $(d(r-1)+1)$-consistency will, in general, introduce higher arity constraints into the problem, which increases the value of r, and hence requires an even higher level of consistency. However, in certain special cases it is possible to achieve the required level of consistency without increasing the value of r (see, for example, Section 4.3) . In these cases, Theorem 5.1 is sufficient to establish tractability.

In general, Theorem 5.1 provides a surprising link between a *local* property and a global property. It says, in effect, that if all subproblems up to a certain size are easily solved, then the whole problem is easily solved.

Another result of this kind may be obtained by an application of a well-known result in combinatorial theory, which is usually referred to as the Lovasz Local Lemma [15].

Theorem 5.2 *Let $\mathcal{P}$ be a constraint satisfaction problem in which each variable occurs in at most t constraint scopes, the length of the largest constraint* *scope is r, and the proportion of assignments allowed by each constraint relation is at least p.*

If

$$p > 1 - \frac{1}{e(r(t-1)+1)},$$

then $\mathcal{P}$ has a solution. (The constant e in the inequality is the base of natural logarithms, $2.718\ldots$.)

Proof: The Lovasz Local Lemma [15] states that for any collection of events $E_1, E_2, \ldots, E_n$, which each have probability at most p_0, if each E_i is independent of all but at most s of the others, and $p_0(s+1) < 1/e$, then with positive probability none of the E_i occurs.

Let $R_1(S_1), R_2(S_2), \ldots, R_n(S_n)$ be the constraints of $\mathcal{P}$, and choose some random assignment of values to all the variables. Let E_i be the event that constraint $R_i(S_i)$ is not satisfied by this assignment, so E_i has probability at most $1-p$. Since this constraint overlaps at most $r(t-1)$ others, E_i is independent of all but at most $r(t-1)$ other events. Applying the Lovasz Local Lemma, we conclude that, if $(1-p)(r(t-1)+1) < 1/e$, then with positive probability none of the E_i occurs, and hence $\mathcal{P}$ has at least one solution. Rearranging this inequality gives the result. ∎

This result guarantees the existence of a solution when certain local conditions are satisfied. However, it is a non-constructive result, which gives no information on how a solution may be found.

6 Conclusions and Future Directions

In this report we have reviewed the current state of knowledge about the computational complexity of constraint satisfaction problems.

We have shown that tractability can arise in a wide variety of ways:

- from the overall structure of a problem;
- from properties of the constraint relations; or
- from properties of subproblems of bounded size.

Strong theoretical results are known for each of these aspects, as described above.

At present, however, little is known about how these different problem features interact to affect the complexity of constraint satisfaction problems. For example, it is possible to construct classes of problems in which the problem structure alone does not ensure tractability, and nor does the nature of the constraint relations alone, but the *combination* of these properties *does* ensure tractability, as the following example indicates.

Example 6.1 Consider the class of binary constraint satisfaction problems which have n variables and m values, and which require that each variable is assigned a different value from each other variable. (These problems are sometimes referred to as 'pigeon-hole' problems)[10].

This class of problems is tractable because a solution can be found, or discovered to be impossible, by simply assigning a new value to each variable in turn, for as long as there are new values available.

However, we note two interesting features of this class of problems:

- Since there is a constraint between each pair of variables, the graph associated with each problem in this class is a complete graph on n variables.

  This class of graphs is not sufficiently restricted to ensure tractability regardless of the constraints, since any binary constraint problem is equivalent to a problem which is associated with a complete graph. We can construct this equivalent problem simply by adding a constraint on each unconstrained pair of variables which allows any pair of values.

- Since the constraints require variables to take different values, the only constraint relation used is the binary disequality relation on m values.

  This relation does not, in general, ensure tractability, since it can be used to construct arbitrary instances of the GRAPH COLORABILITY problem, which is well-known to be NP-complete [36].

$\square$

In many cases of practical interest, such as the frequency assignment problem, it seems from experimental evidence that many of the problems arising in practice can be solved very efficiently, using simple heuristic algorithms [14]. However, it is currently difficult to identify properties of these problems which ensure tractability, so each case must be investigated experimentally, without any guarantee of success. By further developing the theory of tractability described in this report, it may be possible to identify some combination of properties concerning the structure of such problems,

[10]This class of problems is not completely trivial. When n is greater than m there are clearly no solutions, but many constraint programming languages are unable to discover this fact (without additional guidance) in a reasonable amount of time, even for moderate values of n and m (say, 15-20).

the nature of the constraints, and other special features, which is sufficient to guarantee tractability. Identifying such properties might also lead to the design of optimal algorithms for problems with these features. This question is currently being investigated.

Acknowledgements

This research was supported by EPSRC research grant GR/L09936.

References

[1] S. Arnborg. Efficient algorithms for combinatorial problems on graphs with bounded decomposability. *BIT*, 25:2–23, 1985.

[2] S. Arnborg, D.G. Corneil, and A. Proskurowski. Complexity of finding an embedding in k-trees. *SIAM journal of Algebraic and Discrete Methods*, 8:277–284, 1987.

[3] K.A. Baker and A.F. Pixley. Polynomial interpolation and the chinese remainder theorem. *Mathematische Zeitschrift*, 143:165–174, 1975.

[4] C. Beeri, R. Fagin, D. Maier, and M. Yannakakis. On the desirability of acyclic database schemes. *Journal of the ACM*, 30:479–513, 1983.

[5] Claude Berge. *Graphs and Hypergraphs.* North Holland, 1973.

[6] A.L. Brearley, G. Mitra, and H.P. Wiliams. Analysis of mathematical programming problems prior to applying the simplex method. *Mathematical Programming*, 8:54–83, 1975.

[7] E.F. Codd. A relational model of data for large shared databanks. *Communications of the ACM*, 13(6):377–387, 1970.

[8] D.A. Cohen, M. Gyssens, and P.G. Jeavons. Derivation of constraints and database relations. In *Proceedings 2nd International Conference on Constraint Programming—CP'96 (Boston, August 1996)*, volume 1118 of *Lecture Notes in Computer Science*, pages 134–148. Springer-Verlag, 1996.

[9] P.M. Cohn. *Universal Algebra.* Harper & Row, 1965.

[10] M.C. Cooper. An optimal k-consistency algorithm. *Artificial Intelligence*, 41:89–95, 1989.

[11] R. Dechter. From local to global consistency. *Artificial Intelligence*, 55(1):87–107, 1992.

[12] R. Dechter and J. Pearl. Network-based heuristics for constraint satisfaction problems. *Artificial Intelligence*, 34(1):1–38, 1988.

[13] R. Dechter and J. Pearl. Tree clustering for constraint networks. *Artificial Intelligence*, 38:353–366, 1989.

[14] Nick Dunkin and Stuart Allen. Frequency assignment problems: Representations and solutions. Technical Report CSD-TR-97-14, Department of Computer Science, Royal Holloway, University of London, Egham, Surrey, UK, 1997.

[15] P. Erdös and L. Lovasz. Problems and results on 3-chromatic hypergraphs and some related questions. In A. Hajnal et al., editors, *Infinite and Finite Sets*, volume 11 of *Colloq. Math. Soc. Janos Bolyai*, pages 609–627. North-Holland, 1975.

[16] R. Fagin. Degrees of acyclicity for hypergraphs and relational database schemes. *Journal of the ACM*, 30:514–550, 1983.

[17] T. Feder and M.Y. Vardi. Monotone monadic SNP and constraint satisfaction. In *Proceedings of 25th ACM Symposium on the Theory of Computing (STOC)*, pages 612–622, 1993.

[18] E.C. Freuder. Synthesizing constraint expressions. *Communications of the ACM*, 21:958–966, 1978.

[19] E.C. Freuder. A sufficient condition for backtrack-free search. *Journal of the ACM*, 29(1):24–32, 1982.

[20] E.C. Freuder. A sufficient condition for backtrack-bounded search. *Journal of the ACM*, 32:755–761, 1985.

[21] E.C. Freuder. Exploiting structure in constraint satisfaction problems. In M. Mayoh, E. Tyugum, and J. Penjam, editors, *Constraint Programming*, volume 131 of *NATO ASI Series*. Springer-Verlag, 1993.

[22] M. Garey and D.S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness.* Freeman, San Francisco, CA., 1979.

[23] M. Gyssens, P.G. Jeavons, and D.A. Cohen. Decomposing constraint satisfaction problems using database techniques. *Artificial Intelligence*, 66(1):57–89, 1994.

[24] M. Gysssens and J. Paradaens. A decomposition methodology for cyclic databases. In *Advances in Database Theory*, volume 2, pages 85–122. Plenum Press, New York, NY, 1984.

[25] P.G. Jeavons. On the algebraic structure of combinatorial problems. Technical Report CSD-TR-95-15, Computer Science Department, Royal Holloway, University of London, Egham , Surrey , UK, 1995. to appear in Theoretical Computer Science.

[26] P.G. Jeavons, D.A. Cohen, and M. Gyssens. A structural decomposition for hypergraphs. *Contemporary Mathematics*, 178:161–177, 1994.

[27] P.G. Jeavons, D.A. Cohen, and M. Gyssens. A unifying framework for tractable constraints. In *Proceedings 1st International Conference on Constraint Programming—CP'95 (Cassis, France, September 1995)*, volume 976 of *Lecture Notes in Computer Science*, pages 276–291. Springer-Verlag, 1995.

[28] P.G. Jeavons, D.A. Cohen, and M. Gyssens. Closure properties of constraints. Technical Report CSD-TR-96-15, Computer Science Department, Royal Holloway, University of London, Egham, Surrey, UK, 1996. to appear in Journal of the ACM.

[29] P.G. Jeavons, D.A. Cohen, and M. Gyssens. A test for tractability. In *Proceedings 2nd International Conference on Constraint Programming—CP'96 (Boston, August 1996)*, volume 1118 of *Lecture Notes in Computer Science*, pages 267–281. Springer-Verlag, 1996.

[30] P.G. Jeavons and M.C. Cooper. Tractable constraints on ordered domains. *Artificial Intelligence*, 79(2):327–339, 1995.

[31] G. Kondrak and P. van Beek. A theoretical evaluation of selected backtracking algorithms. *Artificial Intelligence*, 89:365–387, 1997.

[32] A.K. Mackworth. Constraint satisfaction. In S.C. Shapiro, editor, *Encyclopedia of Artificial Intelligence*, volume 1, pages 285–293. Wiley Interscience, 1992.

[33] R.N. McKenzie, G.F. McNulty, and W.F. Taylor. *Algebras, Lattices and Varieties*, volume I. Wadsworth and Brooks, California, 1987.

[34] U. Montanari. Networks of constraints: fundamental properties and applications to picture processing. *Information Sciences*, 7:95–132, 1974.

[35] U. Montanari and F. Rossi. Constraint relaxation may be perfect. *Artificial Intelligence*, 48:143–170, 1991.

[36] C.H. Papadimitriou. *Computational Complexity.* Addison-Wesley, 1994.

[37] I.G. Rosenberg. Minimal clones I: the five types. In *Lectures in Universal Algebra (Proc. Conf. Szeged 1983)*, volume 43 of *Colloq. Math. Soc. Janos Bolyai*, pages 405–427. North-Holland, 1986.

[38] T.J. Schaefer. The complexity of satisfiability problems. In *Proceedings 10th ACM Symposium on Theory of Computing (STOC)*, pages 216–226, 1978.

[39] A. Szendrei. *Clones in Universal Algebra*, volume 99 of *Seminaires de Mathematiques Superieures.* University of Montreal, 1986.

[40] E. Tsang. *Foundations of Constraint Satisfaction.* Academic Press, London, 1993.

[41] Jeffrey D. Ullman. *Database and Knowledge-Base Systems*, volume 1 & 2. Computer Science Press, 1988.

[42] P. van Beek. On the minimality and decomposability of row-convex constraint networks. In *Proceedings AAAI-92 (San Jose, CA)*, pages 447–452, 1992.

[43] P. van Beek. Reasoning about qualitative temporal information. *Artificial Intelligence*, 58:297–326, 1992.

[44] P. van Beek and R. Dechter. On the minimality and decomposability of row-convex constraint networks. *Journal of the ACM*, 42:543–561, 1995.

[45] P. van Hentenryck, Y. Deville, and C-M. Teng. A generic arc-consistency algorithm and its specializations. *Artificial Intelligence*, 57:291–321, 1992.