CHAPTER 1

Introduction

Constraint satisfaction is a paradigm naturally suited to solving many types of complex combinatorial problem. One of the great attractions to research in this area has been the high rate of transfer of constraint based technology to commercially viable products. This has resulted in many successful applications ranging from oil exploration (Hasle et al 1995) to circuit design (Mezhoud & Dufourd 1994) (VanHentenryck et al 1992), scheduling (Weil et al 1993) (Lever et al 1995) (Heslop & Pegman 1996) (Azarmi & Abdul-Hameed 1995) and automotive design (Nadel et al 1993), to mention but a few.

The success of constraint based technology has resulted in a considerable amount of research effort aimed at producing ever more effective and efficient ways of tackling constraint satisfaction problems. It has now become a vast field. However, one facet of constraint satisfaction, which has remained largely neglected to date, is perhaps the most fundamental of all - how to effectively formulate a given problem as a constraint satisfaction problem.

Problem formulation is an extremely important part of problem solving. As we shall see, the choice of a good formulation can result in order of magnitude savings in search cost. Conversely, if a bad formulation is adopted, we can experience order of magnitude increases in search cost. In this thesis, our aim is to address the issue of problem formulation and how to traverse the space of possible alternatives in a systematic and effective manner. The result of our work, as will become clear, is a significant step towards that goal.

We begin with this introductory chapter where our aim is to familiarise the reader with constraint satisfaction and to introduce the concept of problem formulation. In the next section we give a formal definition of the constraint satisfaction problem and we present some important terminology which will be used throughout the thesis. This is followed by an overview of the many approaches which exist for solving such problems. Finally, in section 1.3 we outline the key aspects of the problem formulation process.

1.1 The Constraint Satisfaction Problem

There are numerous problem solving tasks which can be expressed using constraints. A commonly encountered example in the field of artificial intelligence is that of colouring different regions of a map such that no two neighbouring regions have the same colour. In the case of colouring a map of the United Kingdom, we may wish to colour the counties such that no two adjacent counties are the same. This problem can be viewed in terms of the constraint that joining regions, or counties, must have different colours. Such a view also implies a set of variables representing the individual regions on the map and a range, or domain, of candidate colours for each of these variables.

Variables, domains and constraints are the basic elements of any *constraint satisfaction problem*. The constraint satisfaction problem is defined as follows;

Definition 1.1 (Tsang 1993) - A *constraint satisfaction problem*, or CSP, is a triple (Z, D, C). Z is a finite set of variables $\{x_1, x_2, \ldots x_n\}$. D is a function which maps each variable in Z to its domain of possible values, of any type, and D_{x_i} is used to denote the set of objects mapped from x_i by D. C is a finite, possibly empty, set of constraints on an arbitrary subset of variables in Z. ■

When values are assigned to variables in a CSP, we refer to such assignments as *labels*;

Definition 1.2 - A *label* is a variable-value pair that represents the assignment of the value to the variable. We use $<x, v>$ to denote the label of assigning the value v to variable x. ■

The notion of *compound label* is also useful when we want to refer to the labelling of a group of variables.

Definition 1.3 - A *compound label* is the simultaneous assignment of a set of variables. We use ($<x_1, v_1>,<x_2, v_2>$) to denote the compound label of assigning value v_1 to variable x_1 and value v_2 to variable x_2. ■

The role of a constraint is to restrict the legal set of compound labels between of the variables it constrains. For example, if we have two variables x_1 and x_2 each having the domain {1, 0}, we might have a constraint $x_1 \neq x_2$. In this case the "≠" constraint restricts the possible set of compound labels to be {($<x_1,0>,<x_2,1>$), ($<x_1,1>,<x_2,0>$)}. A constraint such as this, where two variables are involved, has an *arity* of two and is known as a *binary* constraint.

Definition 1.4 - The *arity* of a constraint is the number of variables that it constraints. When k variables are constrained, the constraint is said to be k-ary. If k is greater than two, the constraint is also often referred to as a general constraint. ■

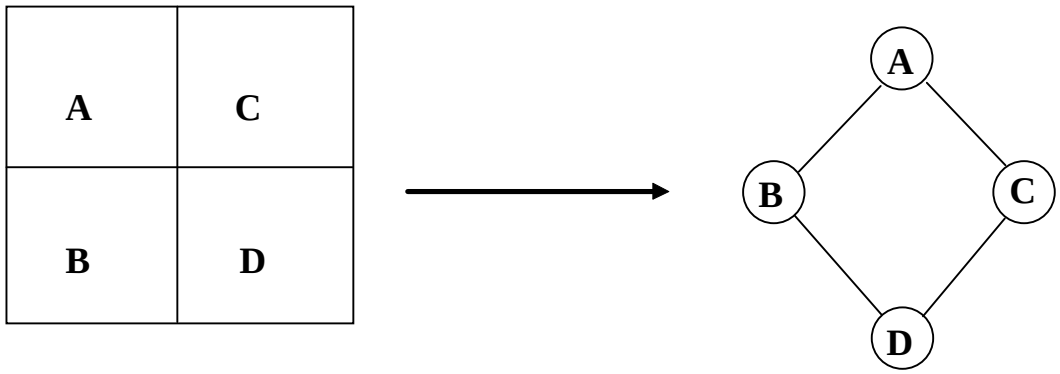
Constraint satisfaction problems where the maximum arity of any constraint is two are known as *binary* CSPs. When constraints having arities greater than two exist in the CSP we refer to these as *general* CSPs. Interestingly, a general CSP can always be represented as a binary equivalent (Rossi et al 1990). This is an important observation for certain problem solving techniques, for example the tree-clustering algorithm (Dechter & Pearl 1989).

A solution to a CSP is an assignment of values to all of the variables in Z such that all of the constraints in C are satisfied. If no solution exists then the problem is said to be *insoluble*. There may be one, many or no solutions to any given problem and the task in solving a CSP is to find a solution or to prove that none exists. This is known to be an NP-Complete problem in general (Mackworth 1977) (Garey & Johnson 1979) although certain specific types or *classes* of CSP can be shown to be tractable. Recent research has seen the development of methods for the systematic detection of some tractable classes of problem (Jeavons et al 1996).

Constraint satisfaction problems share some common ground with ideas found in graph theory (Swamy & Thulasiraman 1981). A CSP can be viewed as an undirected graph if we map the variables in Z to the nodes and the constraints in C to edges[1].

Definition 1.5 - An undirected graph is a tuple (V, E) where V is a set of nodes and E is a set of edges, each of which is a collection of exactly two elements in V. ■

An example of a graph representation is shown in figure 1.1 where we view a small colouring problem.



Colour the four regions with one of red, green, blue or yellow such that no neighbouring region has the same colour.

The CSP formulation shown as a graph, where each edge represents a "not equal" constraint

Figure 1.1 - representing a CSP as a graph.

The graph-based view of a CSP has been utilised by several problem solving techniques as we shall see in the next section.

1.2 Solving Constraint Satisfaction Problems

The generality of the constraint satisfaction paradigm has led to the development of many and varied approaches to solving CSPs. Some specialised algorithms have been developed for tackling specific classes of CSP. An example of this is the cycle-cutset algorithm (Dechter & Pearl 1987). However, many CSP algorithms adopt a more general approach to problem solving.

[1] In the case of general constraints, these are represented as hyperedges, which join more than two nodes in the graph.

One significant group of CSP solving algorithms is based on systematic search using a backtracking mechanism. These algorithms are said to be *complete* since they will find every solution to the problem if required to do so. Examples range from chronological backtracking algorithms which include standard backtracking, forward checking (Haralick & Elliott 1980) and arc-consistency lookahead (Haralick & Elliott 1980), to more sophisticated *backjumping* mechanisms like standard backjumping (Gaschnig 1979) and conflict directed backjumping (Prosser 1993) (Kondrak & Van Beek 1995). Combinations of different techniques have also been developed, giving rise to so-called *hybrid* algorithms such as forward checking with backjumping (Prosser 1993) and maintaining arc-consistency with conflict-directed backjumping (Prosser 1995).

A further method which can be used in conjunction with many complete algorithms is that of problem reduction (Tsang 1993), or consistency techniques (Mackworth 1977) (Mohr & Henderson 1986) (Bessière 1994). These can be used either as an initial pre-processing step or as a part of the actual search process, as seen with lookahead algorithms.

Another important aspect of CSP solving, which is often used to improve efficiency, is variable ordering heuristics. They can be applied to the order in which variables are visited by systematic search algorithms and their effect on the search cost can be dramatic (Tsang et al 1995). An example of such a variable ordering heuristic is the minimum width ordering (Freuder 82) which uses information about the graph of a CSP. The fail-first heuristic (Haralick & Elliott 1980) is a dynamic variable ordering based on choosing the next variable with the smallest domain size, often used with lookahead algorithms. Heuristics can also apply to the choice of value from a variable's domain as with the min-conflicts heuristic (Minton & Philips 1990).

In contrast to complete, systematic algorithms, another group is known as stochastic, incomplete algorithms, where we are not guaranteed that solutions are always found. For example the heuristic repair method (Minton et al 1992), GENET (Davenport et al 1995) and GSAT (Selman et al 1992) have all been demonstrated to be effective with certain classes of problem. The trade-off with these techniques is that while they do not guarantee finding all possible solutions, their non-systematic approach can result in finding solutions much faster.

The diversity of algorithms available for solving CSPs has led to the realisation that different techniques are more suited to different classes of CSP (Tsang et al 1995), (Kwan 1997). One result of this is has been the development of *adaptive* techniques, an interesting new area of research (Borrett et al 1996b). The REBA algorithm is an example of such strategy where chains of different algorithms were used in an adaptive fashion (Borrett et al 1996a). Sakkout et al (Sakkout et al 1996) investigate the potential of adaptive consistency techniques.

Overviews of many of the techniques available for tackling CSPs are given in (Tsang 1993) (Kumar 1992). For a discussion of the issues relating to the trade off between systematic and non-systematic search (IJCAI95 Panel 1995) provides some interesting insights.

1.3 Constraint Satisfaction Problem Formulations

We have discussed some of the many approaches to solving constraint satisfaction problems. However, before a CSP can be solved it must be formulated and there are many different options available to problem solvers when it comes to generating formulations. The impact of these options on problem solving cost can be immense. This means that decisions made when formulating a constraint satisfaction problem are extremely important, as we shall illustrate later in this section.

1.3.1 Problem Formulation

In definition 1.1 we stated that a constraint satisfaction problem is defined by the triple (*Z, D, C*). Given a problem, the process of formulating it as a CSP must at some stage involve mapping a problem description to an instantiation of *Z, D* and *C*. Put simply, problem formulation involves defining the variables in *Z*, the domains in *D* and the constraints in *C* such that the solutions to the resulting CSP provide us with the desired solutions to the problem. We refer to a formulation generated in this way as a *ZDC formulation*.

A *ZDC* formulation was generated for the example in figure 1.1. There we chose to have four variables in *Z*, one for each of the regions. The domains were all the same - the four legal colours. Finally we had a set of four constraints, $\{C_{ab}, C_{ac}, C_{bd}, C_{cd}\}$, each constraint stating that the connected variables have different colours. An alternative to this *ZDC* formulation is to have four

different variables in *Z* which represent the four colours allowed for the regions. We would then have domains for these variables which represent sets of regions having that particular colour. These sets include any combination ranging from no regions to all four. Finally the constraint set, *C*, would consist of a single, global constraint[2] stating that all four regions occur in exactly one of the colour variables, as well as unary constraints on each of the variables stating that adjacent regions must have different colour.

The *ZDC* formulations we have proposed for the colouring problem could be described as being quite natural and simple in that they come very easily to mind. However, with much larger and more complex problems, decisions about the members of *Z*, *D* and *C* can become less straightforward.

1.3.1.1 The Impact of Problem Formulation

If we consider the problem of colouring the map of English counties, four colours can be used to achieve a colouring. We can formulate the problem of generating such a colouring in the same two ways that we used for the problem in figure 1.1. This gives us;

MAP_1:

Z: 45 variables representing the counties of England

D: {red, yellow, blue, green} for all variables, representing the possible colours

C: - For all adjacent counties, a binary constraint stating that their colours are different e.g. Suffolk is coloured differently to Essex

and

MAP_2:

Z: 4 variables representing the colours to be used

D: {Suffolk, Essex, Cambs....} for each colour variable, representing the set of 45 counties which could have that colour

C: - A global constraint on the four variables stating that each county is represented exactly once in the union of the assignments

- A set of unary constraints on each of the variables stating that adjacent counties are not allowed in the same colour variable

[2] A global constraint is one which constrains all of the variables

These two *ZDC* formulations were used to solve our map colouring problem using the ILOG Solver system (ILOG 1994)[3]. The times taken to find a solution to the problem were 16mS for *MAP_1* and 217mS for *MAP_2*. This illustrates how very different *ZDC* formulations can result in very different solving costs - in this case in excess of an order of magnitude.

1.3.2 Approaches to Generating *ZDC* Formulations

There are often many different possible approaches to formulating a given problem. In the past, the quality of a *ZDC* formulation has largely depended on a problem solver's experience or on trial and error. In order to aid the problem solver in making reasonable decisions about the nature of the formulation, some very general guidelines have been suggested. Examples of these guidelines, or rules of thumb, include the use of redundant constraints[4] (VanHentenryck 1989), making constraints as tight as possible (Chamard et al 1995) and keeping the number of variables as low as possible (ILOG 1994).

In order to facilitate the expression of constraint satisfaction problems several languages have been developed. Examples of these range from Alice (Lauriere 1978), one of the earliest, to modern day languages like ILOG Solver (ILOG 1994), CHIP (Dincbas et al 1988), ECLiPSe (ECRC 1995) and PROLOG III (Colmerauer 1990).

Another important step towards improving the reliability of the *ZDC* formulation generation process is the introduction of a degree of methodology to the overall problem solving process. One researcher who has been working in this area is Paltrinieri (Paltrinieri 1994) (Paltrinieri 1995). His work considers the use of object-oriented techniques to give an object-oriented CSP or OOCSP. Paltrinieri also considers the development of a graphical user interface in order to help users input the problem. The technique is demonstrated using a construction problem. Note, however, that this work is primarily concerned with the generation of "a" formulation, not necessarily a "good" formulation.

As we have previously mentioned, we can often formulate a problem in more than one way. In section 1.3.1 we described two different candidate *ZDC* formulations for our colouring problem.

---

[3] Using a DEC Alpha 3000 AXP machine running at 175MHz.
[4] Also often referred to as *surrogate* constraints

However, once candidates have been created, there are always further formulations that can be automatically generated. This is made possible through the use of *problem transformation* algorithms.

An example of a problem transformation is the dual transformation (Dechter & Pearl 1989) (Rossi et al 1990). Others include abstraction (Freuder et al 1995) (Freuder & Sabin 1997) (Schrag & Miranker 1996) and the removal or addition of redundant constraints (Dechter & Dechter 1987) (Borrett 1998). A further group of transformations involves the reduction in the size of the search space by techniques such as interchangeability (Freuder 1991) (Haselböck 1993) and consistency pre-processing (Mackworth 1977). The variables in a *ZDC* formulation can also be manipulated as is the case with variable joining or grouping (Dechter & Pearl 1987) (Nadel 1990) and by the addition or removal of hidden variables (Dechter 1990) (Rossi 1994). Jégou (Jégou 1993) considers transforming *ZDC* formulations based on analysis of what he describes the "micro-structure" of the CSP.

Once we have a set of transformation algorithms, the possibility of generating even more *ZDC* formulations arises if we apply some of these transformations repeatedly. We can also apply them in sequences.

The diversity of transformation techniques means that there is potential for generating a huge range of *ZDC* formulations for any given problem. However, whilst some of these may be improvements on the original, it is clear that some can be much worse. The challenge lies in being able to make reasoned comparisons of the alternatives in order to make a *good* selection.

1.3.3 Comparing *ZDC* Formulations

The ability to generate a range of different *ZDC* formulations can result in a selection of CSPs with very different properties. This in turn gives rise to the significant and open question - which is the best *ZDC* formulation? How do we compare the formulations such that we can select the most appropriate one for solving?

Should we always choose the formulation which has had redundant constraints added to it? Dechter and Dechter argue that there are cases for removing redundant constraints (Dechter &

Dechter 1987). At the same time, some redundant constraints can be useful as shown in (Van Hentenryck 1989). Clearly there are cases where each of these approaches is valid. Similarly, should we always choose the formulation with the smallest search space dimensions? Once again this is by no means a straightforward decision and Nadel's results, using the *n*-Queens problem (Nadel 1990), show that while this is often the best choice it is not always the case.

The idea of comparing different formulations of problems in the context of AI search in general was considered as early as 1968 (Amarel 1968). In that work, Amarel looks at different formulations of the Missionaries and Cannibals problem and briefly discusses the idea of choosing a formulation based on size of what he calls "N-State space". Amarel states that;

Observation 1.1 (Amarel 1968): "*the choice of appropriate representations is capable of having spectacular effects on problem solving efficiency*"

Many examples of these effects can be seen with CSPs. Our map colouring example is one such example. Another is given in (Puget 1993). Puget demonstrates the potential for significant gains in problem solving cost through formulation manipulation using the addition of constraints in the *ZDC* formulations of a range of small problems to remove symmetry in their search spaces. Further examples will be presented in this thesis.

A study by Nadel (Nadel 1990) looks at the feasibility of comparing different formulations of the *n*-Queens problem. Nadel's basis for comparison is a theoretical estimate of the expected cost of search for a particular algorithm and search ordering. The results, for values of *n* equal to 3, 4 and 5, suggest a possible way forward and should perhaps be credited as the first serious attempt to resolve the issue of comparing the relative merits of different *ZDC* formulations of a problem.

In a similar way to Amarel, Korf (Korf 1980) looks at different formulations of several search problems. He advocates the idea of using "information content" as a basis for comparing problem formulations. No detail is given of how the comparison might be achieved, but Korf identifies one of the major attractions of being able to make a reasoned *ZDC* formulation comparison;

Observation 1.2 (Korf 1980): "*to completely automate the process of finding good representations for problems will require a method for evaluating the efficiency of a representation relative to a particular problem solver*"

Korf's observation provides us with further motivation for our work, although the complete automation of problem formulation is not the objective of this thesis.

1.3.4 An Opportunity

The arguments of Amarel and Korf together with the other issues discussed in this chapter give weight to the notion that the comparison of constraint satisfaction problem formulations is useful and worth investigating. The ability to make informed and effective decisions about the nature of a *ZDC* formulation provides us with the potential to improve the overall problem solving process.

As will become clear in this thesis, there are many ways in which *ZDC* formulation comparison and selection can be effective. Furthermore, even if it can only be applied to a selection of problems it can still be useful - it is still an opportunity for improving problem solving efficiency and this opportunity should be exploited as far as possible.

1.4 Overview of this Thesis

In the next chapter, we consider a wide range of key properties of constraint satisfaction problems which impact on their effectiveness for problem solving. These properties are important since they provide us with a deeper understanding of what makes a *ZDC* formulation effective.

The expectation that any method for comparing and selecting appropriate *ZDC* formulations of a problem should be totally accurate is clearly not realistic. Such a situation would virtually require us to solve each candidate before making a choice and hence we would not stand to gain anything. A more realistic view is that of a *heuristic comparison*. In chapter 3 we explore the view of *ZDC* formulation selection as a heuristic search of the space of all possible formulations. We also develop a context in which work related to such a comparison can be discussed.

In chapter 4 we investigate the issue of evaluating the expected effects of differences between formulations. We give an appraisal of previous work by Nadel (Nudel 1983a) (Nadel 1990a) and

we go on to develop an extension to his work by applying it to an intelligent backjumping algorithm. This is important because Nadel's work has only previously applied to chronological backtracking algorithms. The result of our work is an increase in the scope of the use of theoretical complexity for *ZDC* formulation comparison.

While comparisons often need to be made between two different *ZDC* formulations which have no known relationship, other than the fact that they are equivalent *ZDC* formulations (Rossi et al 1990), we also have the scenario where incremental changes are made to an initial formulation. In such cases, the question of whether or not the changes are likely to be useful must be answered. Examples of these types of comparisons are given in chapters 5 and 6, where variable aggregation and the addition of redundant constraints are investigated.

In the final chapter, chapter 7, we present a set of conclusions from our work and summarise our main contributions. We also suggest possible extensions which could be pursued in further work.