CHAPTER 2

On the Properties of Constraint Satisfaction Problems

The diverse range of problems which can be solved using constraint satisfaction techniques means that there is a correspondingly diverse range of constraint satisfaction problems and *ZDC* formulations. Different problems will result in different instantiations of *Z*, *D* and *C*. We can use these differences to identify a range of characteristics, or properties, of *ZDC* formulations.

In this thesis we are interested in the ability to distinguish between different *ZDC* formulations of a given problem, with respect to our expectations of the cost of solving them. The identification of salient properties of constraint satisfaction problems is an essential part of that ability. In particular, we should like to be able to identify properties which give us indications of the likely search costs of a given *ZDC* formulation.

In this chapter we describe a selection of important properties of CSPs. Our aim is to analyse the effects of variations in the instantiations of *Z*, *D* and *C* on these properties. We do not claim to have produced a definitive list since it would be impossible to say when such a list were complete. However, we believe that the properties we describe give us a clearer understanding of how the nature of a *ZDC* formulation is likely to affect the eventual problem solving cost. Furthermore, they provide a useful basis for aspects of our work in later chapters.

2.1 Properties Relating to Variables and Domains

There are many different properties of *ZDC* formulations relating to the variables in *Z* and the domains in *D*, which can be observed. These range from the very simple and obvious, to the more complex. This can be seen if we consider the example constraint satisfaction problem given in figure 2.1.

In our example, we can see that the total number of variables, *n*, is equal to four, namely *A*, *B*, *C* and *D*. We can also see that the CSP has a maximum domain size of 4, a minimum domain size of 2 and an average domain size of 3. These are denoted as m_{max}, m_{min} and $\overline{m}$ respectively. They are all observable properties of a CSP, and although they appear to be trivial, they do provide us with some detail of the topology of the search space. For example, if we have only a few variables, each having a large domain size, then the search space can be thought of as being shallow and broad. Conversely, if we have many variables with few values, then the search space can be viewed as being deep and narrow. This can affect the efficiency of some algorithms.

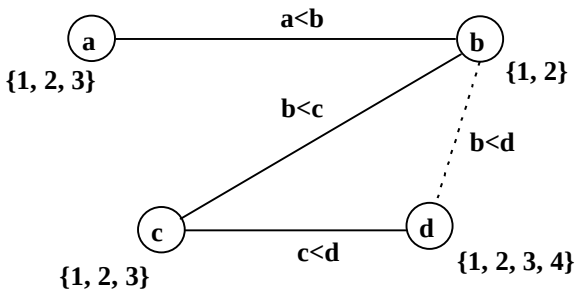

Figure 2.1 - An example CSP

One very important property of constraint satisfaction problems, which is defined by the variables and domains in a *ZDC* formulation, is that of the size of its state space. The finite nature of this state space could be considered as one of the fundamental characteristic features of a CSP and its size is a key indicator of the complexity of searching for solutions to the problem.

2.1.1 Search Space Complexity

For a given CSP with *n* variables, the complexity of its search space, *S*, is the number of possible *n*-compound labels. This is equal to;

$$S = \prod_{i=1}^{i=n} |D_{x_i}| \qquad (2\text{-}1)$$

S is clearly an algorithm independent measure which effectively gives us an indication of the worst case complexity of search by an algorithm in terms of the number of complete candidate assignments of full compound labels. For a systematic search algorithm such as standard backtracking, *S* gives us the number of candidate leaf nodes in the search tree. It also represents the worst case size of a generate and test search. For example, for the CSP in figure 2.1, the value of is *S* equal to $|D_A| \times |D_B| \times |D_C| \times |D_D| = 72$.

The simplicity of *S* contributes to its importance since it is cheap and easy to calculate accurately, and hence it is always available for use when analysing the properties of a *ZDC* formulation. It is often regarded as being a key target for minimisation when it comes to formulating a problem. However, it should be noted that it is not always the case that *ZDC* formulations with the smallest *S* are necessarily the best choice.

Observation 2.1: Suppose we are given two *ZDC* formulations of a problem, f_1 and f_2, each having a search space complexity of S_1 and S_2 respectively. If S_1 is greater than S_2, it is still possible for *ZDC* formulation f_1 to be solved at a lower cost than f_2.

An example, demonstrating observation 2.1, is seen with the *8*-Queens problem. This well known puzzle problem involves placing eight queens on a chess board such that no queen can attack any other queen. One natural *ZDC* formulation of this problem is given by *8Q_1*;

8Q_1: *Z*: 8 variables, v_1, v_2 ... v_8, representing the eight rows on the chess board

D: {A, B, C, D, E, F, G, H} for each variable, representing the position of the queen in the row

C: no queen may attack any other

A second *ZDC* formulation of the *8*-Queens problem, *8Q_2*, is possible if we consider the pairs of adjacent rows. These pairs of rows then form the basis of variables in our new *ZDC* formulation. This is illustrated in figure 2.2.

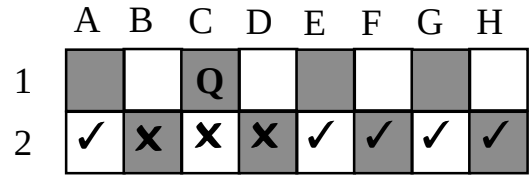

Figure 2.2 - Effects of using two rows as a single variable

Effectively, our variables in *8Q_2* can be seen as representing the sets of legal positions of queens in pairs of rows. In figure 2.2, the ticks indicate the legal positions of the queen in row two, with respect to the queen in row 1, when the queen in row 1 is placed in column C. If we denote the position of two queens in a given pairs of rows by their column position, then figure 2.2 shows five legal positions as {(C,A),(C,E),(C,F),(C,G),(C,H)}. In total, there are 42 legal positions for a pair of queens placed on a pair of adjacent rows if we consider all placements of the two queens in the two rows. These 42 positions represent the domains of the variables in *ZDC* formulation *8Q_2*.

The constraints in our second *ZDC* formulation simply state that no queen on the board may attack any other queen. The complete description of *8Q_2* is summarised as follows;

8Q_2: *Z*: 4 variables, $v_{1,2}$ representing the adjacent pair of rows 1 and 2, $v_{3,4}$ representing rows 3 and 4, $v_{5,6}$ representing rows 5 and 6 and $v_{7,8}$ rows 7 and 8.

D: for each variable, the 42 legal positions of two queens on adjacent rows.
e.g.{(A,C), (A,D) .... (H,F)}

C: no queen may attack any other

If we calculate the value of *S* for these two *ZDC* formulations we obtain values of $10^{7.2}$ for *8Q_1*, and $10^{6.5}$ for formulation *8Q_2*. The figures presented in table 2.1 show the actual cost to solve the two formulations of the problem using the standard backtracking with the natural lexical search ordering (bt+nat), and forward checking with the fail-first variable ordering (fc+ff) (Haralick&Elliott 1980).

	Constraint Checks to First Solution	
	8Q_1	*8Q_2*
bt+nat	2438	2766
fc+ff	792	888

Table 2.1 - Search costs for solving the 8-Queens problem

As we can see from the results in table 2.1, observation 2.1 is clearly demonstrated. The ramification of this is that we cannot rely on the use of *S* as a unique measure for selecting *ZDC* formulations.

2.2 Properties Relating to Constraints

As we discussed in chapter 1, the role of constraints is to restrict the sets of legal compound labels between variables. This is the main mechanism through which a problem solver can incorporate knowledge about the problem being solved, into the *ZDC* formulation. This constraint based knowledge effectively defines the set of solutions to the problem and it helps to guide algorithms towards those solutions. It is achieved through the inclusion of information about legal and illegal states in the search space, as defined by the constraints. These states are often referred to as *goods* and *no-goods*.

Definition 2.1: Given a set of *k* variables, a *good* is a legal *k*-compound label over those variables. ■

Definition 2.2: Given a set of *k* variables, a *no-good* is an illegal *k*-compound label over those variables.■

A useful property which describes the level of no-goods in a constraint is that of *constraint tightness*. Constraint tightness is often referred to in the literature as *p2* (Smith 94)(Prosser 94);

$$p2 = \frac{N(no\_goods)}{N(compound\_labels)} \qquad (2\text{-}2)$$

where, for a constraint with arity *k*, *N(no_goods)* is the number of no-goods length *k* and *N(compound_labels)* is the number of possible compound labels length *k*.

In general, tight constraints are a desirable feature of a *ZDC* formulation because they provide algorithms with more, usable information at a relatively low cost.

2.2.1 Redundancy and the Level of Constraint Based Information

For a *ZDC* formulation to be meaningful it must define the problem solution set. The solution set can be defined for a CSP using a minimal amount of constraint-based information, which can be viewed as the *most general* specification of the solution set. In other words, it is achieved using as little constraint based information as possible. However, there is also scope for introducing more information about intermediate states within the search space, resulting in a *more specific* definition. This idea relates to the notion of minimal graphs and minimal problems (Tsang 1993) (Montanari 1974). Before defining these terms, we need to understand the concept of redundancy.

2.2.1.1 Redundancy

It is possible that some values in the domains of variables never appear in any of the solutions to the problem. These values are said to be redundant;

Definition 2.3: If a value *v* from the domain of variable *x* appears in none of the solutions to that CSP, we say that *v* is *redundant*. ■

If we refer to figure 2.1, there are redundant values in the domain of variable *a*. This is so because *a* must be strictly less than *b*, according to C_{ab}. This means that the values 2 and 3 can never appear in solutions, hence they are redundant.

We can also have redundant compound labels in constraints;

Definition 2.4: If a legal compound label, *cl*, in a constraint is removed and there is no decrease in the number of solutions to the CSP we say that *cl* is *redundant*. ■

Again referring to figure 2.1, the constraint C_{bc} is equal to {<1, 2>, <1,3>, <2, 3>}. However, the only solution to the problem is {(*a*,1), (*b*,2), (*c*,3), (*d*,4)}. This means the compound labels <1, 2> and <1,3> are redundant in C_{bc} because their removal from does not affect the number of solutions to the problem.

Furthermore, we sometimes find the situation where an entire constraint is redundant and its removal results in no increase in the number of solutions to the problem. Constraint C_{bd} in figure 2.1 is one such constraint. These constraints are discussed in detail in chapter 6 and they are defined in definition 6.1.

2.2.1.2 Minimal and Maximal Problems

Using the concept of redundancy, we can define the notion of a *minimal problem*;

Definition 2.5: A CSP is called a *minimal problem* if no domain contains any redundant values and no constraint contains any redundant compound labels ■

Definition 2.6: A CSP is called a *complete minimal problem* when a constraint exists for every possible subset of variables, such that no domain contains any redundant values and no constraint contains any redundant compound labels ■

We could make the CSP in figure 2.1 a minimal problem by removing all of the redundant values from the domains of the variables. This would result in the domains of *a*, *b*, *c* and *d* comprising the single values 1, 2, 3 and 4 respectively. Further manipulation would make it a complete minimal problem if we added all other possible redundant constraints such as C_{ac}, C_{ad} and C_{abd}.

The effect of a complete minimal problem is to have a maximally specific *ZDC* formulation of the problem since every legal intermediate search state is explicitly stated in the constraints. In other words, we have produced a *ZDC* formulation with the maximal amount of information regarding the solution set. The attraction of producing such a *ZDC* formulation is that we can then find a solution with backtrack-free search. However, the drawback is that, in general, even finding the minimal problem of a CSP is NP-Complete (Mackworth 1977) (Montanari 1974).

At the other end of the spectrum we have what we shall term the *maximal problem*. This is defined as;

Definition 2.7: A CSP is called a *maximal problem* when each constraint includes the maximum number of redundant compound labels such that no new solutions are added to the problem. In addition, a maximal problem contains no redundant constraints. ■

Referring again to our example in figure 2.1, we can remove the constraint C_{bd} since it is redundant, due to the presence of constraints C_{bc} and C_{cd}. The result is a maximal problem because no other redundant constraints exist and the remaining constraints all contain redundant compound labels.

A maximal problem can be seen as the most general way of formulating a problem, in terms of the amount of constraint based information included in it. Effectively, the maximal problem has as little explicit constraint based information as possible incorporated into the *ZDC* formulation. This can be an undesirable quality of *ZDC* formulations since it means that more work may be required by search algorithms in order to eliminate certain states in the search space.

In reality, a *ZDC* formulation is typically somewhere in between these two extremes of minimal and maximal problems. However, when formulating a problem we should like to have as much information included into the constraints of a *ZDC* formulation as possible, but without incurring the cost of deriving the minimal problem. There is also a cost associated with increasing the level of constraint based information in some cases, as with the addition of extra constraints, since the information provided by the redundant constraints must still be checked. Clearly there is a trade off in this respect - a redundant constraint must contain enough new information to justify the cost of checking it.

An example of the effects of changing the level of explicit constraint based information in a *ZDC* formulation is seen with the magic series problem.

2.2.1.2.1 The Magic Series Problem

The magic series problem is a simple puzzle which consists of assigning values to a series of numbers. More specifically, for a series of numbers length $n+1$, the task is to assign a number, ranging from 0 to n, to each series position such that the number at each position is equal to the number of times it appears in the series. For example, if n=3, a legal solution to the problem is (2, 0, 2, 0). This is a solution because the number 0 appears twice, at series positions 1 and 3, and the number 2 also appears twice, at series positions 0 and 2.

Two simple *ZDC* formulations are possible, for a magic series problem length n, as shown in (VanHentenryck 1989). The first is *MS_1*;

MS_1: *Z*: One variable, x_i, for each of the n positions in the series

D: $\{0, ..., n\}$ for each variable, representing the value at the series position

C: C_1 - n+1 n-ary constraints, one for each series position stating that;

$$\forall i,\ 0 \le i \le n:\ x_i = occurences(\ i,\ \{x_0, \ldots x_n\})$$

A second formulation, *MS_2*, is possible if we add a further two constraints, C_2 and C_3, to *MS_1* as follows;

$$C_2 - \sum_{i=0}^{n} x_i = n+1$$

$$C_3 - \sum_{i=0}^{n} i \times x_i = n+1$$

These constraints are known to be a feature of the solution set. As a result they provide additional, redundant, constraint based information about the problem.

As an illustration of how these changes in the amount of constraint based information can affect the problem solving cost, we solved the two *ZDC* formulations *MS_1* and *MS_2* using the ILOG Solver (ILOG 94) constraint library. We solved each formulation for the first solution, for a range of values of n. Our results are plotted in figure 2.3.

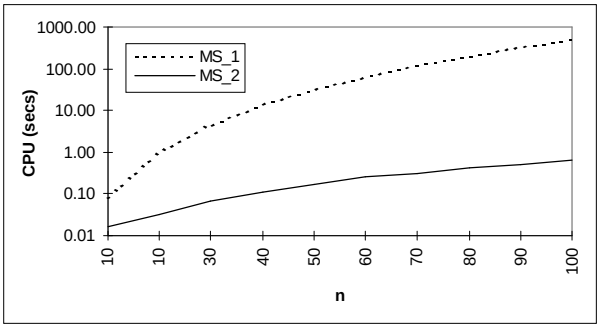

Figure 2.3 - The effect of adding redundant constraints to *MS_1*

As we can see, the addition of new constraint based information has resulted in a dramatic effect on the problem solving cost. For the series length 100, the difference is in excess of three orders of magnitude.

2.2.1.3 Other Properties Reflecting the Level of Constraint Based Information

Realistically, obtaining an accurate picture of how close we are to a complete minimal problem or a maximal problem is not a simple task. We have already stated that finding a minimal problem is NP-complete in general. However, some properties can give us an indication of the amount of constraint based information in a *ZDC* formulation. For example, the average tightness value of constraints, $\overline{p2}$, gives us an idea of the proportion of compound labels that are constrained. Another obvious candidate is the total number of constraints, |C|.

A further measure of the level of constraint based information in a *ZDC* formulation is the *constraint density*. This property gives us the proportion of all possible edges in the constraint graph which are present in the *ZDC* formulation. For a binary CSP, the constraint density, or *p1* as it is often referred to in the literature, is given by;

$$p1 = Edges\_present \div Total\_possible\_edges = |C| \div \frac{n(n-1)}{2} \qquad (2\text{-}3)$$

Assuming that only one constraint exists per edge. This property has been used by many researchers, in conjunction with $\overline{p2}$ as a means of classifying classes of randomly generated binary CSPs (Frost & Dechter 1994) (Freuder & Sabin 1994) (Smith 1994) (Prosser 1994).

2.2.2 Topological Properties

So far we have discussed properties relating to the actual content of constraints. However, the topological arrangement of constraints can also have a marked impact on the effectiveness of problem solving techniques. One group of techniques particularly affected by the constraint graph topology is that of graph based variable ordering heuristics. These include the minimum width ordering (Freuder 1982), the maximum cardinality ordering (Dechter & Meiri 1989) and the Brélaz heuristic (Brélaz 1979). An important property used by these heuristics is the degree of nodes in the constraint graph;

Definition 2.8: The *degree* of a node, or variable, is equal to the number of constraints connected to it. We denote the degree of a node to be *d*.

Two other important properties are the *width* of a constraint graph and the *bandwidth* of a constraint graph;

Definition 2.9: For a given variable in a search ordering, the number variables preceding it in the ordering and to which it is constrained, gives us the *width* of that variable. The width of an ordering is the maximum width of all the variables in the ordering. The width of a constraint graph, *w*, is the minimum width of all possible orderings of the variables.

Definition 2.10: For a given variable in a search ordering, the *bandwidth* of that variable is equal to the maximum distance from that variable to any other variable by which it is constrained. The bandwidth of an ordering is the maximum bandwidth of all the variables in the ordering. The bandwidth of a constraint graph, *b*, is the minimum bandwidth of all possible orderings of the variables.

The notion of width is important as it puts an upper bound on the complexity of solving a CSP (Freuder 1982). The motivation for using the bandwidth as the basis of variable ordering heuristics is that it tends to group mutually constrained variables closely.

The topology of constraint graphs can also affect the effectiveness of different algorithms. For example, when the density of the constraint graph is low and few constraints exist between

variables, there is sometimes more scope for intelligent backjumping algorithms to take advantage of this. As a result, they often perform well on such problem classes. This was seen in (Tsang et al 95) and (Borrett et al 96).

2.3 Properties Relating to the Number of Solutions

The task of a CSP solving algorithm is to find one, or all solutions to that CSP if solutions exist. If no solutions exist then the task becomes that of proving that fact. In some problems there may be many solutions, in others just a few. In insoluble problems there may be many ways of proving that insolubility, or there may be just a few. The number of solutions in a problem can have a marked effect on chances of an algorithm finding one efficiently. In general if there are very few solutions, we can expect an algorithm to spend more effort in trying to find one.

The level of solutions in CSPs relates closely to work on the solubility phase transition (Cheeseman et al 1991) (Williams & Hogg 1994) (Smith 1994) (Gent et al 1996). If we consider a class of constraint satisfaction problem which can be classified by a given measure, *p*, there is a spectrum of solubility which varies as we vary that measure *p*. At one end of this spectrum is a region in which problem instances are under constrained, having many possible solutions. At the other end there is a region where problems are over constrained and very easy to prove insoluble. In between these two regions is a *phase transition* from predominantly soluble instances to predominantly insoluble instances. The point at which the probability of a problem being soluble is 0.5 is referred to as the *phase transition point* or *crossover point* for that class of problem. This is illustrated in figure 2.4.

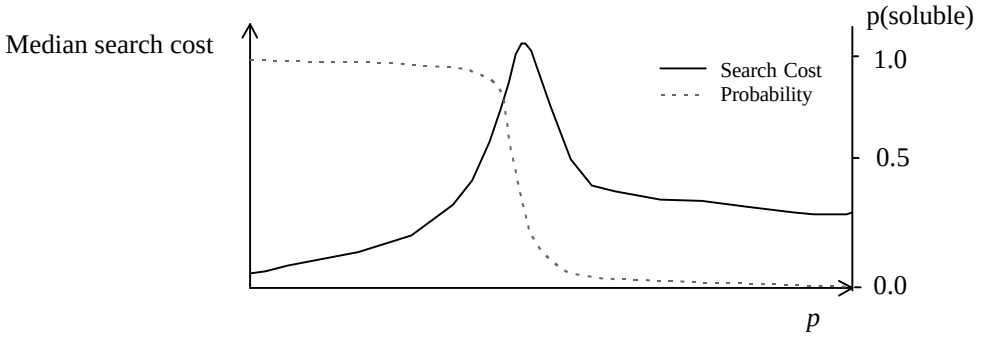

Figure 2.4 - The solubility phase transition

Phase transition is an important phenomenon since it is at the solubility crossover point that we expect to see the peak in median search cost for a particular class of problem. Furthermore the peak is algorithm independent, although the exact shape of the search cost curve is likely to vary across algorithms. There has been extensive work in this area in recent years.

Work on the solubility phase transition is clear evidence for the number of solutions being an important property in CSPs. It gives important insight to the expected behaviour of search on classes of CSP. It is also relevant to the properties of different *ZDC* formulations of a problem. However, there is a complication with this since different *ZDC* formulations of a problem are likely to lead to different classes of CSP. This makes it more difficult to interpret the relative merits of candidate formulations in terms of solutions since other factors, which define the difference in the problem classes, may play a role in the effectiveness of solving them.

There have been several important properties proposed which relate the numbers of solutions to the size of the search space as an indication of problem difficulty. These are described in the following sections

2.3.1 Tightness

The first attempt to quantify the difficulty of a problem in terms of solution density was given in (Tsang 93). The term used is the *tightness, T*, of a CSP and it is defined as;

$$T = \frac{Solutions}{S} \qquad (2\text{-}4)$$

where *Solutions* is the number of solution tuples in the CSP.

The arguments presented in (Tsang 1993) suggest that tighter problems are likely to incur a greater search cost, when a single solution to the problem is to be found. Tightness is also an algorithm independent property.

2.3.2 T-Factor

Formulation tightness is only defined for problems that are soluble. In (Borrett&Tsang 95) the idea is extended to cover the complete range of both soluble and insoluble problems. This is achieved by using a qualitative expression which incorporates the degree of insolubility, for insoluble problems. The result is a property known as *T-Factor* and it is defined as;

$$T\text{-}Factor = \begin{cases} \dfrac{E}{S} & (\text{for } E \geq 1) \\ \dfrac{1}{E \times S} & (\text{for } E < 1) \end{cases} \qquad (2\text{-}5)$$

where E is the expected number of solutions to the problem.

In (Borrett&Tsang 95) estimates were used for E based on the model for binary CSPs devised in (Smith 94). In addition, for cases where the number of solutions is actually known, this can be substituted for E. A further possibility is to use relative numbers of solutions when the relationship between solution tuples is known. In this way we would obtain a relative *T-Factor*.

T-Factor is algorithm independent and covers the whole range of CSPs. For a class of CSPs, it has a minimum when there is only one solution. This correlates to the point of phase transition, or crossover.

2.3.3 Kappa

Work on the phase transition by Gent et al (Gent et al 1996) has resulted in the development of another property based on the number of solutions. They call it the *constrainedness* of search , or *kappa*, κ. κ is defined to be;

$$\kappa = 1 - \frac{\log_2(Solutions)}{\log_2(S)} \qquad (2\text{-}6)$$

Where *Solutions* is the number of solutions in the search space and S is the size of the search space as defined in equation (2-1).

Some interesting properties arise from kappa. Its value lies in the range [0,∞) and for a given *ZDC* formulation, if $\kappa << 1$ then it is likely to be soluble. Correspondingly, if $\kappa >> 1$, it is likely to be insoluble. The point of critical constrainedness, at the phase transition point, occurs when κ has a value of 1.

As with *T-Factor*, constrainedness covers the whole range of problem solubilities and is algorithm independent

2.3.4 A Note on the Limitations of Solution-Based Properties

While properties based on the number of solutions give us an indication of how constrained a search space is likely to be, they are not necessarily a reliable indicator of *ZDC* formulation efficiency. For example, in some cases, the solution density of different *ZDC* formulations can be identical while the cost of solving them can be very different. This was the case for the magic series problem in section 2.2.1.2.

One scenario where reduction in solution density can actually result in an improvement in solving cost is seen with the removal of symmetry in CSPs. Symmetry exists in CSPs when variables can be interchanged without affecting the solution set. The existence of these symmetries can result in futile search being repeated many times.

In (Puget 1993) it was shown that some symmetries in constraint satisfaction problem can be removed by adding certain constraints to a *ZDC* formulation that remove permutations of interchangeable variables. The result of this is to remove some of the solutions to the problem which involve permutations of a subset, *s*, of the variables in the problem. These removed solutions can be obtain from a solution in the new *ZDC* formulation by permuting the values of the variable in *s*.

To illustrate the effectiveness of symmetry removal, Puget used the Schur problem. The Schur problem involves placing a fixed number of balls into three separate containers. The balls are numbered from 1 to *n*, where *n* is the number of balls, and certain restrictions are applied on the legal combinations of balls in the containers. These restrictions are that ball *i* may not appear in

the same container as ball *2i* and that balls *i*, *j* and *i+j* may not appear in the same container. One possible *ZDC* formulation for this problem is *Schur_1*;

Schur_1: Z: one variable, x_i for each of the *n* balls

D: {1, 2, 3} for each variable, representing the three legal containers

C: $\forall i$: $x_i \neq x_{2i}$

$\forall i,j$: $x_i \neq x_j \vee x_i \neq x_{i+j} \vee x_j \neq x_{i+j}$

However, there is a high level of symmetry in this problem since for any solution, we can interchange the box values between variables to obtain an equally valid solution. This knowledge allows us to add two unary constraints to *Schur_1* which remove some of this symmetry. These constraints are;

C_{s1}: $x_1=1$

C_{s2}: $x_2=2$

By adding these constraints to *Schur_1* we obtain *Schur_2*. We solved both of these *ZDC* formulations of the Schur problem using the ILOG Solver constraint library for varying values of *n*. Our results are in figure 2.5.

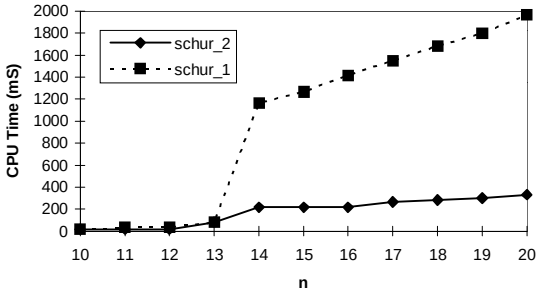

Figure 2.5 - Results for solving different *ZDC* formulations of Schur problem.

As we can see from figure 2.5, as *n* increases, the difference in search cost differs significantly. In fact for *n*=20, the use of symmetry constraints gives us a gain of about one order of magnitude.

2.4 Discussion

The aim of this chapter has been to identify a selection of potentially useful properties of constraint satisfaction problems. These range from the most basic and obvious, such as the number of variables, to more complex properties such as κ. One application of these properties is as a means for classifying different types of CSP. This can be used, for example, for the purposes of mapping problem instances to algorithms best suited to solving them was proposed in (Tsang & Kwan 1993). These ideas were further developed in (Tsang et al 1995) and (Kwan 1997).

A common technique adopted by many researchers is to use randomly generated binary CSPs in order to evaluate the performance of particular algorithms (Frost & Dechter 1994) (Freuder & Sabin 1994) (Smith 1994) (Prosser 1994). One of the attractions of this approach is that the CSPs are easily generated in well defined classes, defined by four basic properties. Typically these four properties are the number of variables in the problem, *n*, the uniform domain size, *m*, the density of the constraint graph, *p1*, and the tightness of the individual constraints, *p2*.

For the purposes of our work, we are interested in comparing *ZDC* formulations such that we can select the one which is likely to have the lowest solving cost. We have demonstrated how important these decisions can be. Our results for different *ZDC* formulations of the magic series problem and the Schur problem resulted in search cost savings measuring orders of magnitude. However, making use of these many different properties in order to achieve an accurate selection of the best *ZDC* formulation is not a straightforward process,

One researcher who has considered this problem is Nadel (Nadel 1990a). Nadel investigated the task of ranking a selection of different *ZDC* formulations of the *n*-Queens problem, in terms of their expected cost for solving. He initially considered the use of the properties *n*, $\overline{m}$, $\overline{p2}$ and |C| for this purpose. However, Nadel concluded that this approach was not feasible stating that;

Observation 2.2 (Nadel 1990a): "*Since all these factors may strongly affect problem-solving complexity, and yet can contradict each other in their recommendations, we see that even a semiformal comparison* [as presented in the paper] *is quite inadequate for ranking representations*"

In fact, such an observation is to be expected in many respects, since Nadel only considered these particular properties individually. If this is done, it becomes extremely difficult to observe their effects on problem solving complexity without other properties also being affected. In other words, it is more likely that useful relationships between the values of properties and their effects on complexity will be seen if groups of properties are considered together.

As an alternative, Nadel proposes the use of theoretical estimates of complexity for solving *ZDC* formulations. This approach showed some promise, but the work was preliminary in nature. We believe that the work by Nadel has a significant role to play in *ZDC* formulation selection and we have extended his work, as will be seen in chapters 4, 5 and 6. In addition we also believe that other properties should not be discarded, and that they can have a role to play in the *ZDC* formulation selection process. This, too, will become clear in the later chapters.

2.5 Summary

Our contribution in this chapter has been to identify a selection of properties of constraint satisfaction problems. These properties range from the very simple, to the more complex, and they can be used to classify different *ZDC* formulations. They also provide us with insight into what makes a *ZDC* formulation more effective, with respect to the cost of solving them. In addition, we have also given further evidence of the impact of selecting good *ZDC* formulation, as was shown with our different versions of the magic series problem and the Schur problem.