J. Borrett, & E.P.K.Tsang, A context for constraint satisfaction problems formulation selection, Constraints, Kluwer Academic Publishers, Vol.6, No.4, 2001, 299-327
Manuscript, to appear in *CONSTRAINTS* (Accepted for publication, November 1999)

To appear in CONSTRAINTS

A Context for Constraint Satisfaction Problem Formulation Selection

Abstract: Much research effort has been applied to finding effective ways for solving constraint satisfaction problems. However, the most fundamental aspect of constraint satisfaction problem solving, problem formulation, has received much less attention. This is important because the selection of an appropriate formulation can have dramatic effects on the efficiency of any constraint satisfaction problem solving algorithm.

In this paper, we address the issue of problem formulation. We identify the heuristic nature of generating a good formulation and we propose a context for this process. Our work presents the research community with a focus for the many elements which affect problem formulation and this is illustrated with the example adding redundant constraints. It also provides a significant step towards the goal of automatic selection of problem formulations.

Keywords: Constraint Satisfaction, Problem Formulation, Heuristics, Modelling

James E. Borrett (Contact Author)

Dept of Computer Science
University of Essex
Wivenhoe Park
Colchester CO4 3SQ
United Kingdom
Tel: +44 1206 872770
Fax: +44 1206 872788
email: jborrett@gol.co.uk

Edward P. K. Tsang

Dept of Computer Science
University of Essex
Wivenhoe Park
Colchester CO4 3SQ
United Kingdom
Tel: +44 1206 872774
Fax: +44 1206 872788
email: edward@essex.ac.uk

1. INTRODUCTION

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 and the use of constraint satisfaction techniques has been incorporated into many applications (Lever *et al* 1995; Puget 1995; Simonis 1995; Wallace 1996). 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.

The basic elements of any constraint satisfaction problem are its variables, domains and constraints are. It is defined as follows;

Definition 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. ■

From definition 1, 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 allow us to obtain the desired solutions to the problem. We refer to a formulation generated in this way as a *ZDC formulation*.

There are often many ways in which a problem can be formulated as a constraint satisfaction problem. Furthermore, decisions made at this stage in the problem solving process are very important and can have a dramatic effect on the eventual cost of solving the problem. (Amarel 1968) and (Korf 1980) discuss the effects of changes in representation at length in the context of search in general. (Nadel 1990) represents one of the few instances of work which analyses the effects of changes in formulation of CSPs, where he considers different *ZDC* formulations of the n-Queens problem. (Borrett 1998) also addresses the issue of *ZDC* formulation selection.

Problem formulation is an extremely important part of problem solving. 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. 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.

There are many different aspects of constraint satisfaction research which affect the process of *ZDC* formulation selection. An example of this is the area of problem transformation techniques such as the removal of redundant constraints (Dechter&Dechter 1987). Preprocessing using problem reduction techniques (Tsang 1993) represents another form of transformation. Other areas of research include the investigation of the properties of CSPs and estimating the complexity of searching for solutions to a particular problem (Nudel 1983a). These all have a role to play in the process of formulation selection.

In this paper we develop a context in which many aspects of CSP research can be related to *ZDC* formulation selection. By doing so, we provide a focus for research in this area which, we

believe, will result in improved techniques for achieving more reasoned *ZDC* formulation selection. In the next section we describe the space of possible *ZDC* formulations. In section 3 we describe the task of selection and in section 4 we identify the elements of our context which provide a framework for performing the selection task. In order to illustrate the mechanics of our context, in section 5 we provide an example application where redundant constraints are added to an original *ZDC* formulation. Finally, in section 6 we present our conclusions.

2. THE SPACE OF POSSIBLE *ZDC* FORMULATIONS

There are many different approaches to generating a set of candidate *ZDC* formulations of a problem. The starting point of this process is to translate the original problem description into an instantiation of *Z*, *D* and *C*. In fact we can generate a range of *ZDC* formulations in this way. Further formulations are then possible using transformation techniques[1].

Some transformations can result in very different *ZDC* formulations being created, while other transformations can result in *ZDC* formulations with only very subtle differences in their characteristics. Transformations, combined with manual generation can lead to a large spread of *ZDC* formulations, some having very similar properties and others having very different ones. The idea of there being such a spread is illustrated in figure 1.

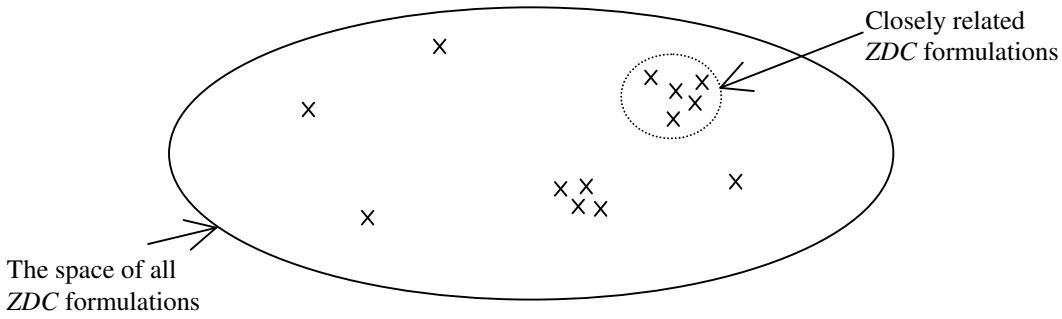

Figure 1 - The space of possible *ZDC* formulations for a given problem

One restriction which must hold within the *ZDC* formulation space for a given problem is that they must all be equivalent. By this we mean equivalent in terms of the definition used by (Rossi et al 1990) whereby the solution sets of different formulations are said to be *mutually reducible*.

Definition 2 (Rossi et al 1990) - Two CSPs P_1 and P_2 are *equivalent* if P_1 is reducible to P_2 and P_2 is reducible to P_1. This is written as $P_1 \equiv_e P_2$. ■

A CSP P_1 is reducible to CSP P_2 if there is a way to go from the solutions of P_2 to the solutions P_1 by mapping variables and values in P_2 into variables and values in P_1. When this mapping works in both directions the problems are said to be mutually reducible.

2.1 The SENDMORY Puzzle: an Example *ZDC* Formulation Space

As an example space of candidate *ZDC* formulations, we consider the crypto-arithmetic SENDMORY puzzle. This problem is shown in figure 2.

[1] Transformation techniques are discussed in more detail in section 4.

```
          S E N D
+         M O R E
=     M O N E Y
```

Figure 2 - The SENDMORY problem

The aim of the SENDMORY puzzle is to assign a unique digit to each of the letters used in the summation. One candidate *ZDC* formulation which we could have is *SENDMORY_1*;

SENDMORY_1:
Z: The eight letters of the problem
D: Each variable represents a digit having the domain {0,...,9}
C: C_1 all variables have different values
C_2 - 1000(S+M) + 100(E+O) + 10(N+R) + D + E =
10000M + 1000O + 100N + 10E + Y

A second candidate *ZDC* formulation is possible if we use carry variables in order to split up the large equality constraint. This gives us *SENDMORY_2*;

SENDMORY_2:
Z: The eight letters of the problem plus three carry variables *a, b* and *c*
D: Each letter variable represents a digit having the domain {0,...,9}
each carry variable has the domain {0,1}
C: C_1 - all letter variables have different values
C_2 - D + E = Y + 10*a*
C_3 - N + R + *a* = E + 10*b*
C_4 - E + O + *b* = N + 10*c*
C_5 - S + M + *c* = O + 10M

If we compare these two *ZDC* formulations we see that there are significant differences. For example, we can calculate the complexity of the search space, *S*, for each *ZDC* formulation. This is given by the product of the domain sizes of the variables and is equal to;

$$S = \prod_{i=1}^{i=n} \left| D_{x_i} \right| \qquad (1)$$

The value of *S* is 10^8 for *SENDMORY_1* and $10^{8.9}$ for *SENDMORY_2*. Another feature of *SENDMORY_2* is that it has more diversity in its topology when compared to *SENDMORY_1*. For example, the constraints C_2 - C_5 have relatively low arities, ranging from 4 to 5. In contrast, the arities of the constraints C_1 and C_2 in *SENDMORY_1* are both 8.

As a third candidate *ZDC* formulation we can make some modifications to *SENDMORY_1*. More specifically, we can add three redundant constraints, C_3, C_4 and C_5, which provide additional explicit constraint based information about the nature of the solution set. This gives us *SENDMORY_3*;

SENDMORY_3:
Z: The eight letters of the problem
D: Each variable represents a digit having the domain {0,...,9}
C: C_1 - all variables have different values
C_2 - 1000(S+M) + 100(E+O) + 10(N+R) + D + E =
10000M + 1000O + 100N + 10E + Y
C_3 - S+M ≤ 10M + O

C_4 - 10S + E + 10M + O ≤ 100M + 10O + N
C_5 - 100S + 10E + N + 100M + 10O + R ≤ 1000M + 100O + 10N + E

SENDMORY_3 is very similar to *SENDMORY_1* since the only difference is the three redundant constraints C_3, C_4 and C_5. This is an example of two *ZDC* formulations that might be considered to be "close" in terms of the space of all *ZDC* formulations of the problem. Of course, the notion of "closeness" is somewhat subjective in nature and depends on the criteria used.

Despite the relative simplicity of the SENDMORY problem and the restricted set of *ZDC* formulations presented in this section, it is not obvious which *ZDC* formulation we should use. With more complex problems, with no tools available to help make a decision, it is difficult to choose from among candidate *ZDC* formulations, especially in cases where the problem solver has no knowledge of constraint satisfaction techniques.

3. THE TASK OF *ZDC* FORMULATION SELECTION

In order to aid the problem solver in making reasonable decisions about how to generate or select a *ZDC* formulation, some very general guidelines have been suggested. Examples of these guidelines, or rules of thumb, include the use of redundant constraints (VanHentenryck 1989) and making constraints as tight as possible (Chamard et al 1995). However, in general, current approaches to selecting the best *ZDC* formulation of a problem are somewhat ad-hoc in nature.

This situation is undesirable since there are significant gains to be made from selecting the correct formulation and, conversely, there are significant losses that can be incurred by selecting a poor one. We should therefore like to take advantage of the potential gains, and avoid the potential losses. In this section we define the task of *ZDC* formulation selection and the goals of an ideal *ZDC* formulation selection method. We follow this by a discussion of what we believe to be practical and achievable goals.

3.1 The Task Defined

We have seen how there are many ways problems can be formulated as CSPs. In addition, we notice that different algorithms are affected in different ways by different CSPs, and hence by different *ZDC* formulations. This was demonstrated in (Tsang et al 1995) and (Kwan 1997). These facts lead us to the definition of the primary task of the *ZDC* selection process;

Task 1: Let *cost(a, f)* be the cost of solving an individual *ZDC* formulation *f* using algorithm *a.* Given a problem *p* and the set of all *ZDC* formulations *F*, the task of the selection process is to find $f \in F$ such that we minimise the value of *cost(a, f)*.

This task assumes that we have committed ourselves to the use of a particular algorithm for solving the problem *p*. An extension of task 1 is to find the *ZDC* formulation which gives the minimum value of *cost(a, f)* for a set of algorithms, *A*. This leads us to a second task;

Task 2: Given a problem *p*, a set of *ZDC* formulations *F* and a set of algorithms *A*, the task of the selection process is to find $f \in F$ and $a \in A$ such that we minimise the value *cost(a, f)*.

The two tasks we have outlined above can be considered the requirements for an ideal *ZDC* formulation selection method. There are also some further requirements which are implicit in these tasks. These are;

Generality - the method of selection should be general in that it can be applied to all classes of problem.

Accuracy - the method of selection should be highly accurate, ideally 100% accurate in choosing the most effective formulation.

Cost - the method of selection should have a low cost, ideally the cost should be nil.

Currently no such method exists and it seems unlikely that all three of these requirements can be satisfied simultaneously. However, a more reasonable proposition is that these ideal properties can act as a guide to the qualities of any practical selection method.

3.2 Expectations

Tasks 1 and 2 are extremely important tasks. However, we must acknowledge that they are both impossible to achieve with certainty. One reason for this is that they depend on the availability of all possible *ZDC* formulations of a problem. Furthermore, the expression *cost(a, f)* cannot be 100% accurate unless we actually solve the *ZDC* formulation in question.

For any practical *ZDC* formulation selection method, we only ever have a subset of all possible *ZDC* formulations. For example, with our SENDMORY example, we only considered three *ZDC* formulations of the problem. In addition, we also have to rely on approximations of *cost(a, f)*. For a practical method to be effective, we therefore rely on;

- a good pool of candidate *ZDC* formulations
- an effective method for approximating *cost(a, f)*

The higher the quality of these elements, the higher the quality of the selection and the closer we can get to fulfilling tasks 1 and 2.

The only selection method which has zero cost is that of arbitrary selection. For other approaches, the cost should be taken into account. If this were not the case then we would have a simple solution to the 100% accuracy criterion whereby we solve each *ZDC* formulation of the problem before making our selection. Such an approach is unacceptable since it will always give worse overall search complexity than simply choosing one of the *ZDC* formulations at random and solving that one. We do not propose to put a figure on the cost that should be allowed in the selection process. However, it is desirable that this cost should be minimised and remain a low proportion of the overall solving cost.

The fact that the ideal is not achievable leads us to an approximation. The use of heuristics can help produce an effective approximation. Our approach to *ZDC* formulation selection is to develop heuristics based on the properties of CSPs. This idea is presented in (Borrett 1998) where a set of heuristics was developed, based on theoretical complexity estimates. As we shall see in the next section, these heuristics have a significant role to play in our context for *ZDC* formulation selection.

4. A CONTEXT FOR SELECTION

The aim of this section is to define our context for the selection of *ZDC* formulations of a given problem. Such a context has not been defined previously and, as a result, many aspects of constraint satisfaction research which affect the *ZDC* formulation selection process remain unconnected. We believe our context will help to bring these together. It also acts as a focus for

our work, allowing us to target specific aspects of the selection process. Furthermore, our context provides a useful framework which can act as a base model for automatic *ZDC* formulation generation techniques.

A problem formulation generator $\mathcal{G}$ takes a problem specification and generates an instantiation of *Z*, *D* and *C*. If P is the set of problem specifications within the domain of $\mathcal{G}$ and F is the set of all possible *ZDC* formulations then we have the mapping;

$$\mathcal{G}: \mathsf{P} \rightarrow \mathsf{F} \tag{2}$$

As we discussed earlier, our view of *ZDC* formulation selection is one of a heuristic traversal of the space of all possible formulations, F. If we are given a particular *ZDC* formulation *f* as a starting point, we can always move to alternative points in that space using transformation algorithms, or *move operators*. We then need two types of heuristic in order to facilitate that movement. The first type of heuristic is one that suggests an appropriate transformation that we expect to produce an improvement on the current *ZDC* formulation. The second type is used to evaluate the actual result of any suggested transformation. These three elements of our context, move operators, suggestion heuristics and evaluation heuristics, are described in the following sections.

4.1 Move Operators and Transformation Algorithms

The role of move operators is to provide mobility through the space of possible *ZDC* formulations. There are two basic approaches to moving through this space. The first is to use the actual problem solver to generate alternative *ZDC* formulations. A second approach is to use *ZDC* formulation *transformation algorithms*;

Definition 2 - Given a problem *p* and a *ZDC* formulation *f*, a *ZDC* formulation *transformation algorithm* $\mathcal{T}(f)$ generates as its output an equivalent *ZDC* formulation for that problem. ■

This gives us;

$$\mathcal{T}: \mathsf{F} \rightarrow \mathsf{F} \tag{3}$$

An example *ZDC* transformation algorithm is the dual transformation (Dechter & Pearl 1989). The output *ZDC* formulation of this transformation, *f2*, is dual in the sense that for every constraint in the original, *f1*, we create a variable in *f2*. The domain of these variables is then determined to be the set of legal compound labels defined in the associated original constraint. This is seen in figure 3 where we have two variables in *f2*, corresponding to the two constraints C_{AB} and C_{AC} in *f1*. In order to ensure equivalence in both *ZDC* formulations, the final step in the transformation is to add constraints between variables in *f2* which originate from constraints in *f1* having common variables. In our example, x_{AB} and x_{AC} have a common *f1*-variable, namely x_A. The constraint $C_{AB,AC}$ simply ensures that the identical label is used for the common *f1* variable.
An interesting property of the dual transformation is that it always produces a binary CSP as its output.

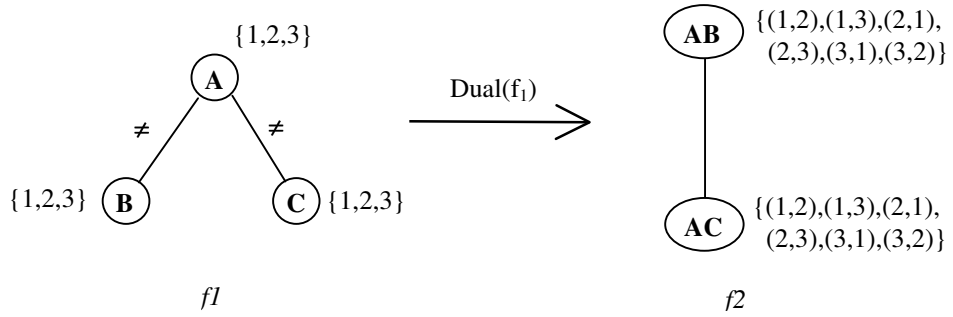

Figure 3 - The dual transformation

Many other transformations are possible. Examples include;

- problem reduction techniques (Tsang 1993) such as arc-consistency (Mackworth 1977)
- abstraction (Freuder et al 1995) (Freuder & Sabin 1997)
- the addition and removal of redundant constraints (VanHentenryck 1989) (Dincbas et al 1988) (Smith 1996) (Borrett 1998)

Having such a range of transformations available means that we have more freedom to move within the space of all possible *ZDC* formulations. We may also use combinations or sequences of transformations in order to generate yet more *ZDC* formulations. For example, we could use the dual transformation followed by the removal of certain redundant constraints.

4.2 Suggestion Heuristics

As the number of *ZDC* transformation algorithms available to us increases we gain an increasing level of mobility within the space of *ZDC* formulations. However, this gain does not come without cost and we only want to apply transformations that are likely to produce beneficial effects. Simply using all transformation algorithms available to us is not a sensible path to *ZDC* formulation generation. What we should like is to use heuristics which guide us to sensible moves through the *ZDC* formulation space. This is the role of *suggestion heuristics*.

Definition 3 - Given a *ZDC* formulation f and a suggestion criterion g, the role of a *suggestion heuristic*, $\mathcal{H}_s$, is to select one or more transformation algorithm, $\mathcal{T}$, which is expected to improve the suggestion criterion g. ■

This gives us;

$$\mathcal{H}_s: \mathsf{F} \rightarrow \mathsf{T} \qquad (4)$$

where T is the set of available *ZDC* formulation transformation functions.

The key to an effective suggestion heuristic is the choice of the suggestion criterion, g. A range of candidates are criteria based on the properties of *ZDC* formulations as discussed in (Borrett 1998). For example, the search space complexity, S, is a potentially useful candidate. A suggestion heuristic based on this criterion might point us towards the use of problem reduction algorithms or the merging of variables, both of which fulfil the criterion. It would not recommend transformation algorithms which are likely to increase S, which is sometimes the case with the

dual transformation. Other candidates include increasing the level of constraint based redundancy (VanHentenryck 1989) and tightening the constraints (Chamard et al 1995).

One view of suggestion heuristics is to compare them with variable ordering heuristics. These are usually chosen before search begins and can be extremely effective. However, no variable ordering heuristic has yet been found which is the universal champion, to be used for all problem classes. Results to this effect were seen in (Tsang et al 1995). In similar way, we believe *ZDC* formulation suggestion heuristics may have domains according to the particular *ZDC* formulation they are operating on.

4.3 Evaluation Heuristics

An important element of tasks 1 and 2 is the mechanism for evaluating the function *cost(a, f)*. Without such a function, we are left with making an arbitrary choice between *ZDC* formulations. Since we do not know of any ideal selection method, our objective is to obtain approximations to the cost function. In other words we need to find heuristics which allow us to make good decisions about their relative merits. We call such heuristics *evaluation heuristics*.

Definition 4 - Given two *ZDC* formulations *f1* and *f2*, an *evaluation heuristic*, $\mathcal{H}_e$, returns a tri-state value which gives an indication of the relative expected cost of solving them. A value of *1* denotes that *f1* is expected to have the cheapest solving cost, a value of *-1* denotes that *f2* is expected to have the cheapest solving cost and a value of *0* denotes that there is no expected difference. ■

This gives us;

$$\mathcal{H}_e: \mathsf{F} \times \mathsf{F} \rightarrow \subset \tag{5}$$

where

$$\subset = \{\ 1, 0, -1\ \}$$

In (Borrett 1998), a selection of properties of CSPs was described. Many of these can be used to give an indication of the expected problem solving cost. Since it is not practical to actually solve candidate *ZDC* formulations before making any selection, $\mathcal{H}_e$ must be based on a subset of such properties, or measures. When this subset includes more than one measure, $\mathcal{H}_e$ must resolve them to generate an indication of expected relative problem solving cost.

Since there is an almost unlimited source of potential measures that can be defined, the key to effective evaluation heuristic design is the use of properties which provide a good correlation between their value and the cost of solving the *ZDC* formulations in question. Furthermore, while increasing the number of properties used may provide a greater resolution of discrimination between *ZDC* formulations, there is a trade-off between the number of such properties and complexity of the method for resolving them.

A summary of the whole context and how evaluation heuristics fit into it is given in figure 4.

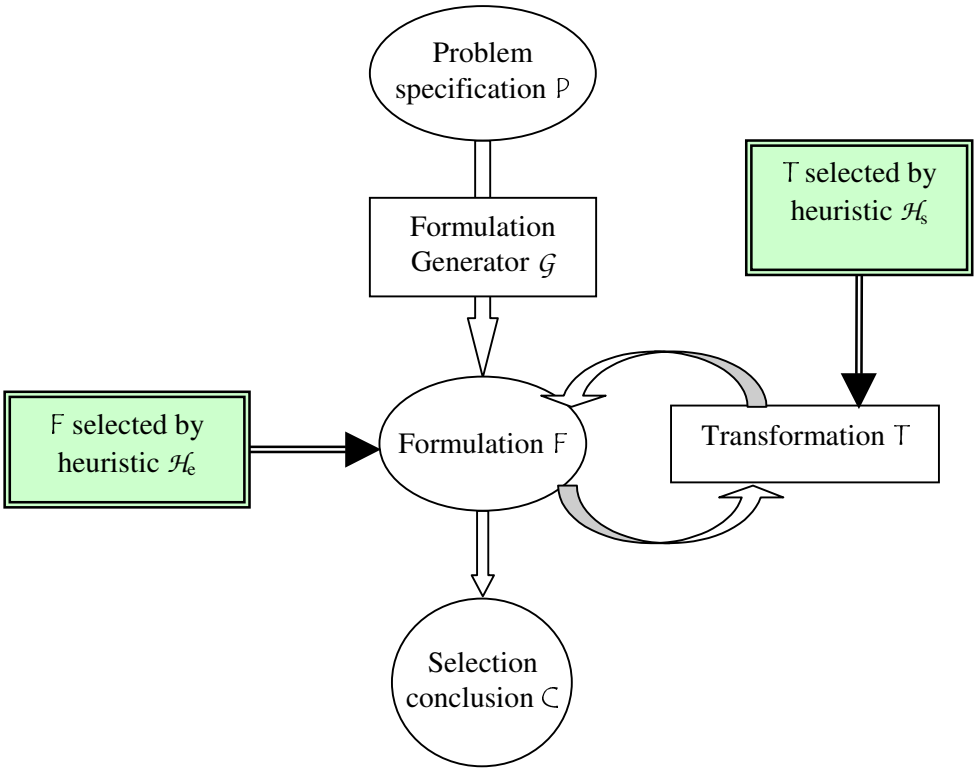

Figure 4. – The context for constraint satisfaction problem solving

5. ADDING REDUNDANT CONSTRAINTS - AN EXAMPLE

Redundant constraints provide a widely recognised technique for improving the efficiency of constraint satisfaction problem solving. A classic example of this is seen in (Dincbas et al 1988) and (van Hentenryck et al 1992) where redundant constraints are added to an initial *ZDC* formulation in order to improve the cost of solving car sequencing problems. The effects of their use was significant since it allowed for many problems to be solved which were previously believed to be beyond the capabilities of some computational techniques. (Smith 1996) also discusses this approach.

In some cases it can be advantageous to remove redundant constraints. This was seen in (Dechter & Dechter 1987) where the motivation for removing redundant constraints was to alter the topology of the constraint graph. In this way, they found that considerable savings in search cost could be achieved for certain problem classes.

The manipulation of redundant constraints can be an extremely powerful tool. At the same time, decisions about how to use them are not always straightforward. When redundant constraints are added, they offer the potential for eliminating futile sections of the search space. However, the addition redundant constraints introduce an overhead to search algorithms since the total number of constraints which actually need to be checked is increased. A trade off in these effects must therefore be achieved.

In this section we illustrate the main features of our context. We develop a *ZDC* transformation, a selection heuristic and an evaluation heuristic for the addition of redundant composition

constraints to binary CSPs. Our example shows how these elements can be combined to give significant savings in search cost.

5.1 Redundant Constraints

To illustrate the notion of redundant constraints, consider the problem in figure 5.

$$P < Q < R$$

Figure 5 - A simple arithmetic problem, where P, Q and R are digits in the range 1 to 10.

A straightforward and natural *ZDC* formulation of this problem is to have three variables in Z: p, q and r, corresponding to P, Q and R. Each of these variables is then given a domain of {1...10}. Having made these decisions, we now need to define the constraints in C. One obvious possibility would be to have two constraints corresponding to the inequalities;

$$C_{pq}: \quad p < q \tag{6}$$

and

$$C_{qr}: \quad q < r \tag{7}$$

By further analysing the original problem, we note that the sum of Q and R is also greater than P. We can incorporate this additional knowledge of the problem into our *ZDC* formulation by adding the constraint;

$$C_{pqr}: \quad p < q + r \tag{8}$$

The addition of constraint C_{pqr} is valid, though not necessarily useful, because its presence or absence in the *ZDC* formulation does not affect the number of possible solutions. Because of this we say that constraints such as C_{pqr} are *redundant*;

Definition 3 (Tsang 1993) - A k-constraint in a CSP is *redundant* if it does not restrict the k-compound labels of the subject variables further than the restrictions imposed by the other constraints in that problem. This means that the removal of it does not change (increase) the set of solution tuples in the problem. ■

As we have already discussed, the addition, or removal of redundant constraints can have a marked effect on the efficiency of search in a particular *ZDC* formulation.

5.2 Generating Redundant Constraints - a *ZDC* Formulation Transformation

Redundant constraints can sometimes be generated automatically. For example, consider an original *ZDC* formulation having a core constraint graph, such as the graph indicated by the solid edges in figure 6. Without any knowledge of the actual nature of the problem being solved, we know that there are potential redundant binary constraints for all of the dashed edges in the graph. If the content of these constraints can be determined, then we have potentially useful redundant constraints.

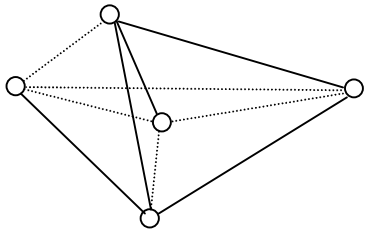

Figure 6 - An example constraint graph with some candidate redundant edges shown as dashed

If we revisit the problem described in figure 5, it can be viewed as two individual arithmetic expressions. This view of the problem resulted in the two constraints C_{pq} and C_{qr} for our original *ZDC* formulation. In addition, we can also derive a further expression from the problem which is the composition of these two base expressions. This composition expression is;

$$P<R \tag{9}$$

The above composition expression represents further implied knowledge about the problem which was determined by a simple rule of arithmetic. We can use the same approach to determine *redundant composition constraints*. These constraints are important because for any group of three variables, a redundant composition constraint can always be found, provided two of the three possible binary constraints between the variables exist. So in the example problem of figure 5, a redundant composition constraint C_{pr} exists.

It is these redundant composition constraints that we use in the remainder of this section. In terms of our context, we have *ZDC* formulation transformation function, T_{rc}, which takes an original *ZDC* formulation and adds redundant composition constraints to it.

5.3 Suggestion Heuristics for Redundant Composition Constraints

While redundant constraints clearly provide us with potential benefits, they are not an essential part of any given *ZDC* formulation in the sense that they have no effect on the solution set. The additional constraint-based information a redundant constraint provides gives explicit details of illegal states in the search space which are already implicit in the original *ZDC* formulation. For systematic search algorithms this means that the usefulness of any redundant constraint is dependent on it bringing forward the possibility of using the explicit constraint-based information it provides. If a redundant constraint brings forward knowledge of a no-good in the search, this can be useful to an algorithm. However, if it does not, then the additional constraint simply presents itself as a further constraint which needs to be checked without providing any benefit. As a result, one factor which affects the usefulness of a redundant constraint is the order in which variables are labelled, and hence the variable ordering heuristic used.

Both static and dynamic variable orderings can be used with search algorithms to give significant savings in search cost. For some classes of problem, a particular dynamic variable ordering and algorithm combination might prove the best choice, whilst for another class of problem, a static

variable ordering and algorithm can be the best choice.[2] For the purposes of our example, we shall base our suggestion heuristic on the scenario where a static variable ordering is used.

By analysing the effects of a redundant composition constraint with respect to a given algorithm, we can be more selective in how their addition is applied.

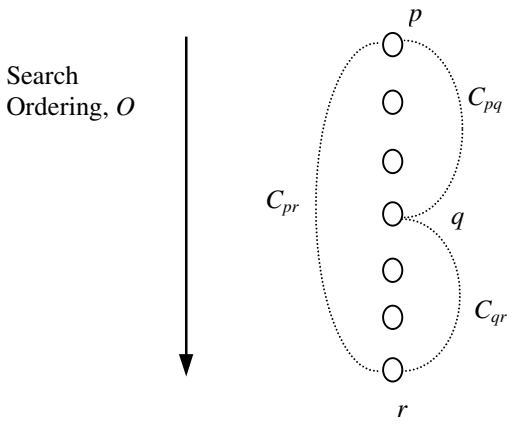

Figure 7 - Possible scenarios for redundant composition constraints under search ordering O

Considering figure 7, when constraints C_{pr} and C_{qr} exist, we can generate a redundant constraint C_{pq} by composition. For standard backtracking the introduction of constraint C_{pq} can only result in fewer, or at worst the same number of nodes being visited by the algorithm. This leads us to proposition 1;

Proposition 1: The addition of redundant composition constraint C_{pq} results in the *same or fewer* nodes being visited by standard backtracking, when a static variable ordering is used. C_{pq} never increases the total number of nodes visited.

Proof: The constraint C_{pq} is first checked by standard backtracking when the search reaches level q. For all previous search levels, the algorithm visits the same nodes as it would do in an original *ZDC* formulation which does not include C_{pq}. When C_{pq} is checked at level q, if no compatible values are found, a backtrack will occur and no further nodes are expanded in that particular sub-search space. For the case of the original *ZDC* formulation, a backtrack resulting from the

[2] Many believe that different algorithms are efficient for different problems. This observation applies to algorithms with dynamic and static ordering. Tsang et al (1995) and Kwan (1997) mapped different classes of constraint satisfaction problems to a variety of algorithms (covering algorithms that use dynamic ordering as well as algorithms that use static ordering) and demonstrated that such mapping can enhance search efficiency. Borrett et al (1996) used a set of algorithms, some using dynamic ordering and some using static ordering, in a sequence to solve a large class of constraint satisfaction problems. It was found that problems that were "exceptionally hard" (Gent & Prosser 1994; Hogg & Williams 1994; Smith & Grant 1995) to one algorithm could be solved easily by another. Dynamic variable ordering is currently believed to have wide applicability. However, the role of static ordering cannot be under-estimated. For example, building on Freuder's work (1982), Dechter & Pearl (1988) proposed the directional arc-consistency idea and the tree-search procedure (which requires static ordering) for solving constraint satisfaction problems whose constraint graph may form a tree. Zabih (1990) showed that the minimal bandwidth ordering can be an effective (static) ordering heuristic. Zabih further related the minimal bandwidth ordering to the induced width (Dechter 1988) and proved interesting results in complexity analysis.

composition of conflicts with variables p and r will not take place at this point and it is deferred, possibly until the search reaches q. As a result, the addition of C_{pq} can only result in the same or less nodes being visited. ■

The effect of C_{pq} on the number of constraint checks performed by standard backtracking is less easily identified. If fewer nodes are visited then this should result in a reduction in the number of checks performed. However, countering this gain, there is the overhead of actually performing the check of the new constraint, as we have previously indicated. As a result, we can say that the addition of C_{pq} to an original *ZDC* formulation can have both beneficial and detrimental effects on the number of constraint checks performed.

When constraints C_{pr} and C_{pq} exist, we can generate a redundant constraint C_{qr} by composition. For standard backtracking, the introduction of C_{qr} has no effect on the number of nodes visited. This gives us proposition 2;

Proposition 2: The addition of redundant constraint C_{qr} has *no effect* on the number of nodes visited by standard backtracking, when a static variable ordering is used.

Proof: The constraint C_{qr} is first checked by standard backtracking when the search reaches level r. At this level the algorithm would have also checked C_{pq}. Since C_{qr} is the composition of the constraints C_{pq} and C_{pr}, it rules out no further compound labels at or after level r because it can provide no effect different to that of checking C_{pr}. As a result, we can therefore say that the same number of nodes are expanded in *ZDC* formulations with or without constraint C_{qr}. ■

It follows from proposition 2 that we cannot gain in terms of the number of constraint checks when C_{qr} is added.

Proposition 3: The addition of redundant constraint C_{qr} can only *increase* the number of constraint checks performed by standard backtracking, when a static variable ordering is used.

Proof: The number of nodes is unaffected by the addition of C_{qr}. Since its addition increases the number of constraints checkable at level r, provided it is checked at least once, C_{qr} must result in the total number of constraint checks being the same or greater than for the original *ZDC* formulation. ■

When constraints C_{pq} and C_{qr} exist, we can generate a redundant constraint C_{pr} by composition. For standard backtracking, the introduction of C_{pr} has no effect on the number of nodes visited.

Proposition 4: The addition of redundant constraint C_{pr} has *no effect* on the number of nodes visited by standard backtracking, when a static variable ordering is used.

Proof: The constraint C_{pr} is first checked by standard backtracking when the search reaches level r. At this level the algorithm would have already checked C_{pq}. Since C_{pr} is the composition of the constraints C_{pq} and C_{qr}, it rules out no further compound at or after r because its effect can provide no effect different to that of checking C_{qr}. We can therefore say that the same number of nodes are expanded in *ZDC* formulations with or without constraint C_{pr}. ■

It follows from proposition 4 that we cannot gain in terms of the number of constraint checks when C_{pr} is added.

Proposition 5: The addition of redundant constraint C_{pr} can only *increase* the number of constraint checks performed by standard backtracking, when a static variable ordering is used.

Proof: The number of nodes is unaffected by the addition of C_{pr}. Since its addition increases the number of constraint checkable at level r, provided it is checked at least once, C_{pr} must result in the total number of constraint checks being the same or greater than for the original *ZDC* formulation. ■

The above analysis is important because it identifies occasions when redundant composition constraints are never likely to be useful. It also identifies scenarios where they may prove useful. In terms of the context, we can regard propositions 1-5 as the basis of a *ZDC* formulation suggestion heuristic, $\mathcal{H}_s$, for the standard backtracking algorithm. They suggest to us when we might gain from using *ZDC* transformation T_{rc}. In (Borrett 1998) similar analyses are given for the backjumping (Gaschnig 1979) and forward checking (Haralick&Elliott 1980) algorithms.

5.4 Evaluation Heuristics for Redundant Composition Constraints

We have seen how some redundant composition constraints can be useful for certain algorithms. In order to take advantage of this important opportunity for reducing search costs, we need to develop a heuristic for evaluating the actual expected impact of these constraints. If this can be achieved, it will allow us to selectively modify an original *ZDC* formulation by only accepting moves by T_{rc} which show promise.

5.4.1 Expected Search Cost

In section 4.3 we concluded that an effective evaluation heuristic should provide us with a good correlation between their value and the actual search cost of the transformed *ZDC* formulation. There are many possible ways of addressing this issue. One approach, which we shall use for the purposes of our example, is to generate estimates of the expected search cost. Haralick and Elliott devised a simple probabilistic model for constraint satisfaction problems in (Haralick & Elliott 1980). Nadel extended this work to allow estimates of complexity to be generated for classes of problem defined to a greater level of detail (Nudel 1983a).

A comprehensive assessment of Nadel's work, with respect to evaluation heuristics, was performed in (Borrett 1998). The aim of that assessment was to determine whether or not the equations given by Nadel could form the basis of reliable heuristics over a wide range of problem classes. The principle findings were:

- for some classes of CSP, estimates of complexity were reasonably accurate
- estimates of complexity can provide good heuristic properties when used to compare the relative search costs of alternative *ZDC* formulations
- complexity estimates can show useful heuristic properties for the case where only a single solution was searched for – this was important since the complexity estimates given by Nadel were for finding all solutions to a problem

These findings showed that there was a significant potential for the use of theoretical complexity estimates for evaluation heuristics. Further details are given in appendix A.

5.4.2 Evaluation Heuristics Based on Expected Search Cost

We demonstrated in section 5.3 that the redundant composition constraint C_{pq} could have a beneficial effect on the search cost. Our evaluation heuristic therefore applies to this constraint.

Taking the findings of the previous section, we now demonstrate how the theoretical complexity estimates of Nadel can be modified and used to develop an evaluation heuristic for redundant composition constraints. We call our evaluation heuristic for standard backtracking ρ_{bt}:

$\rho_{bt}(C_{pq}) < 1.0 \Rightarrow$ the addition of redundant composition constraint C_{pq} is likely to be useful

and

$\rho_{bt}(C_{pq}) \quad 1.0 \Rightarrow$ the addition of redundant composition constraint C_{pq} is not likely to be useful

From (Borrett 1998) we have;

$$\rho_{bt}(C_{pq}) = c\_red(bt) / c(bt) \tag{10}$$

where *c(bt)* is the expected complexity of search, in terms of constraint checks, in the original problem and *c_red(bt)* is the expected complexity of search in the transformed problem, which includes the redundant constraint.

c(bt) is given in (Nudel 1983a) and is equal to;

$$c(bt) = \sum_{k=1}^{n} c(bt,k) \tag{11}$$

where

$$c(bt,k) = \left(\sum_{i=1}^{|G_k|-1} \prod_{j=1}^{i-1} p_{g_{jk}k} \right) \times \left(\prod_{i \in A_k} |D_{xi}| \right) \left(\prod_{i<j \in A_{k-1}} p_{ij} \right) \tag{12}$$

and the notation used is;

k	- a search level, ranging from 1 to n, the number of variables in the CSP
A_k	- the set of assigned variables at level k
G_k	- the set of previous variables constrained by the variable at level k; these are in a fixed order
g_{jk}	- the *jth* variable in the set G_k
p_{ij}	- the *looseness* or *satisfiability* of the constraint between variables i and j. It is the opposite of tightness and is equal to 1-*p2*

c_red(bt), the expected number of constraint checks performed by standard backtracking with the transformed formulation is equal to;

$$c\_red(bt) = \sum_{k=1}^{n} c\_red(bt,k) \tag{13}$$

where *c_red(bt, k)*, the expected number of constraint checks performed at search level *k*, is given by;

$$c\_red(bt,k) = \left(\sum_{i=1}^{|G_k|-1} \prod_{j=1}^{i-1} p_{g_{jk}{}^k} \right) \times \left(\prod_{i \in A_k} |D_{xi}| \right) \left(\prod_{\substack{\neg(i=p \wedge j=q) \wedge \\ i<j \in A_{k-1}}} p_{ij} \right) \qquad \forall k : k \quad r \qquad (14)$$

c_red(bt, k) effectively ignores the effect of the redundant constraint on the expected number of nodes expanded for search levels where it provides the algorithm with no additional constraint based information.

Using (10), when the expected cost after adding C_{pq} is less than the expected cost of the original *ZDC* formulation, we have $\rho_{bt} < 1.0$ which fits the definition of ρ_{bt} given at the start of this section.

5.5 Experiments

To assess the effectiveness of the heuristics we have outlined above, we conducted a series of experiments. Our approach was to generate several sets of random binary CSPs, each having different characteristics. These CSPs are defined using the 4-tuple *<n, m, p1, p2>* where *n* is the number of variables in the problem, *m* is the uniform domain size of the variables, *p1* is the density of the constraint graph and *p2* is the tightness of the individual constraints. This model was used in (Smith 1994).

For each instance generated, we call that *ZDC* formulation *R1*. We then applied the redundant composition constraint addition algorithm, given in figure 8, to that *ZDC* formulation in order to generate a second *ZDC* formulation, *R2*. The transformation algorithm looks at candidate redundant constraints according to propositions 1-5 and adds them according to evaluation heuristic $\mathcal{H}_e$. Two instantiations of, $\mathcal{H}_e$, at line 9 of the algorithm were used;

i. $\mathcal{H}_e = 1.0$ - uninformed addition for standard backtracking (control)
ii. $\mathcal{H}_e = \rho_{bt}$

The reason for including case *i* was to test our "informed" evaluation heuristic against this uninformed baseline. We call the *ZDC* formulation resulting from this transformation *R3*.

For each problem class considered, we generated 100 instances and each was solved using standard backtracking. In addition, the above *R2* and *R3* formulations were created for each of those instances. We then solved each of these using the standard backtracking algorithm.

```
1  Given a CSP(Z, D, C), algorithm alg and a variable v in ordering O:
2  BEGIN
3     FOR i = 1 to i = |Z|
4        FOR j = i+1 to j = |Z|
5           FOR k = j+1 to |Z|
6              IF (CONNECTED(v_i, v_k) ∧
7                  CONNECTED(vj, v_k) ∧
8                  ¬CONNECTED(v_i, v_j))
9                 IF ℋ_e < 1.0
10                   ADD_COMPOSITION(v_i, v_j)
11          k++
12       j++
13    i++
14 END
```

Figure 8 - The redundant composition constraint addition algorithm

Our expectation was that the use of the uninformed heuristic should show good performance with some problem classes and bad in others. The reason for this is that for problem classes where the redundant composition constraints are relatively tight, these constraints have a high chance of being useful. However, such a naive approach should break down when the redundant composition constraints become looser. If our ρ_{bt} evaluation heuristic is effective, then would should expect them to perform well over a wide range of problem classes, thus enabling them to take advantage of useful redundant composition constraints while rejecting those which are not. Our results are presented in the next section.

5.5.1 Results

For each problem instance we had a cost measure;

cc(R1) - the cost of solving the original *ZDC* formulation
cc(R2) - the cost of solving the output of figure 8 using case *ii*
cc(R3) - the cost of solving the output of figure 8 using the uninformed cases *i*

Our results, over a range of problem classes, were processed with a view to observing both qualitative and quantitative aspects of performance. In order to assess the qualitative performance of our ρ_{bt} heuristic, for each problem class tested, we divided the results into three categories;

cat. 1 - Instances where *cc(R2)* was less than *cc(R1)* by a margin of significance - i.e. a benefit was seen from the using the transformation. These results are given in columns 3 and 5 in table 1

cat. 2 - Instances where cc(R2) was greater than cc(R1) by a margin of significance - i.e. using the transformation resulted in a degradation in performance. These results are given in columns 4 and 6 in table 1

cat. 3 - Instances where the difference between *cc(R2)* and *cc(R1)* is within the margin of significance - i.e. using the transformation resulted in no significant change in performance.

By partitioning the results in this way we are able compare the frequency of improvement and the frequency of degradation when using the transformation. For example, if there are more category 1 instances than category 2 instances, then we can say we are seeing a benefit from using the transformation in that we are gaining more often than we are losing. In fact, if the number of category 1 instances is greater than category 2 then we have a good result. Conversely, if the number of category 2 instances is greater than the number of category 1 instances then the result is considered bad. For the ideal case, we should like to see as many category one instances as possible and as few category 2 as possible.

Processing the results as we have outlined above we can assess the performance of the redundant composition constraint transformation when combined with the ρ_{bt} heuristic. However, we should also like to be able to assess how much improvement our ρ_{bt} heuristic provides over the uninformed addition of redundant composition constraints. To see this effect we need to compare the number of instances in each category for *ZDC* formulations *R2* and *R3*. The results for *R3* are given as the figures in brackets in table 1. If the ρ_{bt} heuristic is performing well and providing an improvement on the uninformed approach, then we should see more instances in columns 3 and 5 and fewer in columns 4 and 6.

Some of the cells in table 1 are shaded grey. This is to highlight the cases where the uninformed addition of redundant constraints resulted in *cc(R3)* being higher than *cc(R1)* more often than not. At the same time, the results for our ρ heuristics in these cells show much better performance, easily satisfying that criterion.

		Accuracy with 5% margin (%instances)		Accuracy with 15% margin (%instances)	
Class	Algorithm+ Heuristic	$\frac{cc(R2)}{cc(R1)} \leq 0.95$	$\frac{cc(R2)}{cc(R1)} \quad 1.05$	$\frac{cc(R2)}{cc(R1)} \leq 0.85$	$\frac{cc(R2)}{cc(R1)} \quad 1.15$
<20, 5, 0.10, 0.84>	bt+nat	99 (100)	0 (0)	98 (100)	0 (0)
<20, 5, 0.30, 0.32>	bt+nat	75 (73)	1 (14)	62 (68)	0 (7)
<20, 5, 0.30, 0.36>	bt+nat	91 (95)	0 (1)	81 (84)	0 (0)
<20, 5, 0.30, 0.40>	bt+nat	98 (98)	0 (1)	95 (94)	0 (1)
<20, 5, 0.30, 0.44>	bt+nat	100 (100)	0 (0)	97 (98)	0 (0)
<20, 10, 0.10, 0.81>	bt+nat	94 (99)	0 (0)	94 (99)	0 (0)
<20, 10, 0.10, 0.86>	bt+nat	92 (96)	0 (0)	92 (96)	0 (0)
<20, 10, 0.10, 0.91>	bt+nat	98 (99)	0 (0)	98 (99)	0 (0)
<20, 10, 0.30, 0.50>	bt+mwo	3 (7)	0 (84)	1 (3)	0 (71)
<20, 10, 0.30, 0.55>	bt+mwo	11 (7)	0 (78)	4 (1)	0 (51)
<20, 10, 0.30, 0.60>	bt+mwo	12 (12)	0 (53)	2 (3)	0 (14)
<40, 5, 0.10, 0.44>	bt+mwo	37 (91)	0 (3)	24 (87)	0 (1)
<40, 5, 0.10, 0.48>	bt+mwo	47 (98)	0 (1)	34 (93)	0 (0)
<40, 5, 0.10, 0.52>	bt+mwo	48 (92)	0 (1)	36 (86)	0 (0)
<40, 5, 0.30, 0.20>	bt+mwo	8 (59)	0 (12)	5 (39)	0 (0)
<40, 5, 0.30, 0.24>	bt+mwo	12 (36)	0 (41)	5 (23)	0 (19)

Table 1 - Results of adding redundant composition constraints using ρ_{bt} - uninformed figures, *cc(R3)*/*cc(R1)*, are in brackets

The qualitative performance of our heuristic was very good, with high levels of accuracy seen. In order to see the quantitative performance, we show results in figure 9 for the performance of ρ_{bt} on problem class <40, 5, 0.10, 0.48>. These results show a scatter plot of the ratio of *cc(R2)* and

cc(R1) for a sample of 500 instances. This gives us a useful indication of the actual savings that can be expected through the use of our new *ZDC* formulation evaluation heuristics.

These results demonstrate how our approach allows us to obtain significant savings in search cost from the addition of redundant composition constraints. There are several instances where order of magnitude gains are seen. In addition, we see that, while there are a few instances where degradation in performance is seen, and the ratio *cc(R2)/cc(R1)* is greater than 1, the magnitude of that degradation is very small.

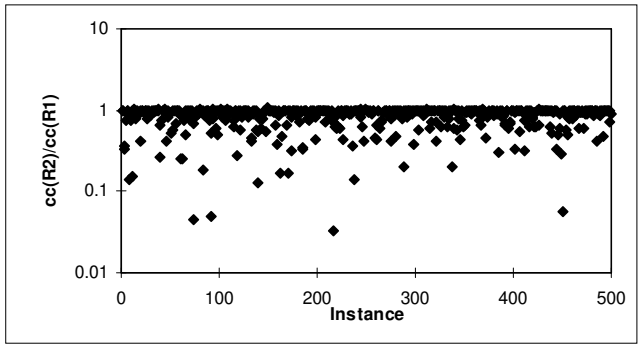

Figure 9 - Search costs on *ZDC* formulations *R1* and *R2* using ρ_{bt} - sample size of 500

On inspection of the results in table 1, we see that for some cases, the use of uninformed addition of redundant composition constraints performs well. An example of this is for the problem class <20, 10, 0.10, 0.81>. However, there are also many cases where arbitrary addition leads to the generation of *ZDC* formulations where the search performance is degraded more often that it is improved as seen for the class <20, 10, 0.30, 0.55>. Further examples of these classes are indicated as the grey cells in the tables. This contrasts greatly with gains we see from using the ρ_{bt} heuristic which provide excellent all round performance.

6. DISCUSSION

In this paper we have described a context for the heuristic selection of *ZDC* formulations which allows us to focus on areas of constraint satisfaction research which affect the *ZDC* formulation selection process, both directly and indirectly. These three main elements of our context are;

- move operators
- suggestion heuristics
- evaluation heuristics

We have presented an example incorporating all of these three elements of the context where redundant composition constraints are added to *ZDC* formulations. Our approach has demonstrated how individual instances can show improved search by systematic manipulation of their ZDC formulation. Another example of the context being used effectively is presented in (Borrett 1998) where pairs of variables are merged selectively.

A significant amount of work which has already been done in relation to *ZDC* formulation transformation algorithms, which act as move operators. Relating such algorithms directly to our context will allow us to view them more directly in association with the *ZDC* formulation

selection process. This contrasts with the traditional view of simply being a pre-processing step carried out before search, as is often the case with problem reduction techniques such as arc-consistency.

Much work needs to be done in the area of suggestion and evaluation heuristics ($\mathcal{H}_s$ and $\mathcal{H}_e$). We have used a method makes use of expected complexity estimates. There are many other possible approaches which could be adopted to make use of the many different properties of *ZDC* formulations. The important challenge is to identify specific properties or combinations which provide us with useful heuristic information.

The example we have presented represents a small step in a vast area of research. However, we believe that our results are promising and that our work represents a significant step towards the goal of more reasoned and systematic *ZDC* formulation selection.

ACKNOWLEDGEMENTS
The authors would like to thank Jim Doran, Rachel Cardel Oliver, Dave Cohen, Peter van Beek and Gene Freuder for their useful discussions on this work. We should also like to thank the members of the constraint satisfaction group at the University of Essex for their comments.

REFERENCES

Amarel, S., 1968, On Representations of Problems of Reasoning about Actions, *Machine Intelligence, vol 3,* 131-171

Borrett, J. E., 1998, Formulation Selection for Constraint Satisfaction Problems: A Heuristic Approach, *PhD thesis*, *Dept. of Computer Science, University of Essex, UK*

Borrett, J. E., Tsang E. P. K. & Walsh N. R., 1996, Adaptive Constraint Satisfaction: the Quickest First Principle, *Proc. 12th European Conference on Artificial Intelligence*, 160-164

Chamard A., Fischler A., Guinaudeau, D-B. & Guillard A., 1995, CHIC Lessons on CLP Methodology, *ECRC report*

Dechter A. & Dechter R., 1987, Removing Redundancies in Constraint Networks, *Proc. National Conference on Artificial Intelligence*, 105-109

Dechter, R. & Pearl, J., 1988, Network-based Heuristics for constraint-satisfaction problems, *Artificial Intelligence*, Vol.34, 1-38

Dechter R. & Pearl J., 1989, Tree Clustering for Constraint Networks, *Artificial Intelligence, vol 38*, 353-366

Dincbas M., Van Hentenryck P., Simonis H., Aggoun A., Graf T. & Berthier F., 1988, The Constraint Logic Programming Language CHIP, *Proc. the International Conference on Fifth Generation Computer Systems*, 693-702

Freuder E., 1982, A Sufficient Condition for Backtrack-Free Search, *Journal of ACM, vol 29(1),* 24-32

Freuder E. C. & Sabin D., 1997, Interchangeability Supports Abstraction and Reformulation for Multi-Dimensional Constraint Satisfaction, *Proc. 14th National Conference on Artificial Intellilgence*

Freuder E. C., Wallace J. & Sabin D., 1995, Generalisation and Abstraction for Constraint Satisfaction, *Proc. Constraint-95, an International Workshop on Constraint-Based Reasoning*, 16-24

Freuder, E.C., 1999, Modeling: the final frontier, *The First International Conference on The Practical Application of Constraint Technologies and Logic Programming* (PACLP), London, 15-21

Gaschnig, J., 1979, Performance Measurement and Analysis of Certain Search Algorithms, CMU-CS-79-124 Technical Report, Carnegie-Mellon University, Pittsburg

Gent, I.P. & Prosser, P., 1994, Easy problems are sometimes hard, Research Note, Artificial Intelligence, Vol.70, 335-345

Haralick, R. M., and Elliott, G. L., 1980, Increasing Tree Search Efficiency for Constraint Satisfaction Problems, *Artificial Intelligence, vol 14,* 263-313

Hogg, T. & Williams, C.P., 1994, The hardest constraint problems: a double phase transition, Research Note, *Artificial Intelligence*, Vol.69, 359-377

Korf R. E., 1980, Toward a Model of Representation Changes, *Artificial Intelligence, vol 14,* 41-78

Kwan A. C. M., 1997, A Framework for Mapping Constraint Satisfaction Problems to Solution Methods, *PhD thesis*, *Dept. of Computer Science, University of Essex, UK*

Lever, J., Wallace, M. & Richards, B., 1995, *Constraint logic programming for scheduling and planning*, British Telecom Technology Journal, Vol.13, No.1., Martlesham Heath, Ipswich, UK, 73-80

Mackworth A. K., 1977, Consistency in Networks of Relations, *Artificial Intelligence, vol 8,* 99-118

Nudel B. A., 1982, Consistent-Labelling Problems and Their Algorithms, *Proc. National Conference of Artificial Intelligence*, 128-132.

Nudel B. A., 1983a, Consistent-Labelling Problems and Their Algorithms: Expected-Complexities and Theory-Based Heuristics, *Artificial Intelligence, vol 21(1-2),* 135-178

Nudel B. A., 1983b, Solving the General Consistent-Labelling(or Constraint Satisfaction) Problem: Two Algorithms and their Expected Complexities, *Proc. National Conference on Artificial Intelligence,* 292-296

Nadel B. A., 1990, Representation Selection for Constraint Satisfaction: A Case Study Using n-Queens, *IEEE Expert, vol. 5*, 16-23.

Puget, J-F., 1995, *Applications of constraint programming*, in Montanari, U. & Rossi, F. (ed.), Proceedings, Principles and Practice of Constraint Programming (CP'95), Lecture Notes in Computer Science, Springer Verlag, Berlin, Heidelberg & New York, 647-650

Rossi F., Petrie C. & Dhar V., 1990, On the Equivalence of Constraint Satisfaction Problems, *Proc. 9th European conference on Artificial Intelligence, 550-557*

Simonis, H., 1995, *The CHIP system and its applications*, in Montanari, U. & Rossi, F. (ed.), Proceedings, Principles and Practice of Constraint Programming (CP'95), Lecture Notes in Computer Science, Springer Verlag, Berlin, Heidelberg & New York, 643-646.

Smith B. M., 1994, Phase Transition and the Mushy Region in Constraint Satisfaction Problems, *Proc. 11th European Conference on Artificial Intelligence,* 100-104

Smith, B.M. & Grant, S.A., 1995, Where the exceptionally hard problems are, Proceedings, Workshop on Studying and Solving Really Hard Problems, *First International Conference on Principles and Practice of Constraint Programming*, September, 172-182

Smith B. M., 1996, Succeed-first or Fail-first: A Case Study in Variable and Value Ordering, *Research Report 96.26, School of Computer Studies, University of Leeds, UK*

Tsang E. P. K., 1993, Foundations of Constraint Satisfaction*, Academic Press, London*

Tsang E. P. K., Borrett J. E. & Kwan A. C. M., 1995, An Attempt to Map a Range of Constraint Satisfaction Algorithms and Heuristics*, Proc. 10th Biennial Conference on AI and Cognitive Science, Society for the Study of Artificial Intelligence and Simulation of Behaviour,* 203-216

Van Hentenryck, P., 1989, Constraint Satisfaction in Logic Programming, *MIT Press, Cambridge MA.*

Van Hentenryck P., Simonis H. & Dincbas M., 1992, Constraint Satisfaction using Constraint Logic Programming, *Artificial Intelligence, vol 58,* 113-159

Wallace, M., 1996, Practical Applications of Constraint Programming, *Constraints*, *An International Journal*, *vol 1*, 139-168

Zabih, R., Some applications of graph bandwidth to constraint satisfaction problems, Proc., *National Conference for Artificial Intelligence* (AAAI), 1990, 46-51

APPENDIX A - THEORETICAL ESTIMATES FOR EVALUATION HEURISTICS

Probabilistic models of constraint satisfaction problems provide a useful mechanism for estimating search costs. If such estimates can be shown to be sufficiently accurate, then they provide us with a tool which can form the basis of some *ZDC* formulation evaluation heuristics, $\mathcal{H}_e$. Detailed work on the use of theoretical complexity estimates is presented in (Borrett 1998). To give the readers some flavour of this work, we provide in this appendix a summary of some aspects of that work, with respect to the standard backtracking algorithm. For complexity estimation in look ahead and back jumping algorithms, readers are referred to (Borrett 1998).

A.1 INTRODUCTION

One of the first attempts to develop a probabilistic model for constraint satisfaction problems was presented in (Haralick & Elliott 1980). Haralick and Elliott devised a very simple model for problem classes defined by n the number of variables and m the uniform domain size of all n variables. Nadel's early research (Nudel 1982, 1983a, 1983b) looked at improving the ideas presented by Haralick and Elliott in terms of the resolution of problem class being considered.

Nadel observed that the more detail one chooses to ignore about CSPs, the coarser the corresponding partition and the easier it is to carry out a (worst case or expected) complexity analysis over a class of problems - *but* the less relevant it becomes for an individual problem. The result of Nadel's work was a set of equations for estimating the expected complexity of CSPs for a given n, m and set of constraints. The equations could be applied to the standard backtracking and forward checking algorithms.

The following sections are organised as follows:

- A.2 summarises Nadel's work on complexity analysis for standard backtracking;
- A.3 produces evidence to show that Nadel's estimations of the search costs are reasonably accurate;
- A.4 produces evidence to show that even if the estimates are not very accurate for certain problem instances, they can be used to reflect the relative costs on transformed problems. This suggests that Nadel's complexity analysis can be a useful evaluation heuristic ($\mathcal{H}_e$).

A.2 THEORETICAL COMPLEXITY EQUATIONS FOR STANDARD BACKTRACKING

In this section we summarise the equations developed by Nadel for the standard backtracking algorithm. For more details of the derivation of these equations, we refer the reader to (Nudel 1983a). The following symbols are used;

k	- a search level, ranging from 1 to n, the number of variables in the CSP
A_k	- the set of assigned variables at level k
G_k	- the set of previous variables constrained by the variable at level k; these are in a fixed order
g_{jk}	- the jth variable in the set G_k
$c(alg)$	- the expected number of constraint checks for algorithm alg
$c(alg, k)$	- the expected number of constraint checks for algorithm alg at level k
$n(alg)$	- the expected number of nodes expanded for algorithm alg
$n(alg, k)$	- the expected number of nodes expanded for algorithm alg at level k

$|D_{x_i}|$ - the domain size of the variable at level i

p_{ij} - the *looseness* or *satisfiability* of the constraint between variables i and j. It is the opposite of tightness (*p2* described in section 5.5) and is equal to $1 - p2$

In general, for a given search algorithm, *alg*, the total number of nodes visited during search when searching for all solutions is equal to the sum of nodes expanded at each level, k, in the search;

$$n(alg) = \sum_{k=1}^{n} n(alg,k) \tag{A-1}$$

The total number of constraint checks carried out during the search is equal to the product of the number of nodes at each level in the search and the expected number of constraint checks performed for each value of the variable at that level. This is given by:

$$c(alg) = \sum_{k=1}^{n} \left(c(alg,k) \times n(alg,k) \right) \tag{A-2}$$

These expressions give us the top level calculation required for estimating nodes and constraint checks for specific algorithms, when finding all solutions. For the standard backtracking algorithm in particular, the expected number of nodes at level k is effectively the number of values that the algorithm attempts to label at that level. This is determined by multiplying the product of the domains of all variables up to the search level k, by the probability that all constraints connected between variables at search levels preceding k were satisfied. That probability is given in (A-3).

$$\prod_{i<j \in A_{k-1}} p_{ij} = \prod_{i<A_{k-1}} \prod_{\substack{j<i \\ \wedge \\ j \in A_{k-1}}} p_{ij} \tag{A-3}$$

Using (A-4), we obtain (Nudel 1983a);

$$n(bt,k) = \left(\prod_{i \in A_k} |D_{x_i}| \right) \left(\prod_{i<j \in A_{k-1}} p_{ij} \right) \tag{A-4}$$

The expected number of checks per node is dependent on how many constraints are connected to the variable at the current search level, k. Furthermore, we only continue checking the constraints if all previous checks against the current assignment are successful. The total expected number of constraint checks at level k is therefore equal to the sum of the probabilities of each constraint between x_k and previously assigned variables is checked, multiplied by the number of nodes expanded at that level;

$$c(bt,k) = \left(\sum_{i=1}^{|G_k|-1} \prod_{j=1}^{i-1} p_{g_{jk}k} \right) \times n(bt,k) \tag{A-5}$$

Combining (A-2), (A-4) and (A-5), we obtain an estimate for the total number of constraint checks performed by standard backtracking (Nudel 1983a);

$$c(bt) = \sum_{k=1}^{n} c(bt,k) \tag{A-6}$$

$$c(bt) = \sum_{k=1}^{n} \left(\sum_{i=1}^{|G_k|-1} \prod_{j=1}^{i-1} p_{g_{jk}k} \right) \times \left(\prod_{i \in A_k} |D_{xi}| \right) \left(\prod_{i<j \in A_{k-1}} p_{ij} \right) \tag{A-7}$$

A.3 THE ACCURACY OF THEORETICAL ESTIMATES FOR SEARCH COST PREDICTION

Since equation (A-7) is derived from a probabilistic model, randomly generated CSPs are a good candidate for assessing their underlying accuracy - if they are not accurate for random problems, then it is unlikely that they will be accurate for non-random ones. For our experiments we used a range of binary random CSPs defined by the tuple *<n, m, p1, p2>*, where *n* is the number of variables, *m* is the uniform domain size of the variables, *p1* is the density of constraints within the constraint graph and *p2* is the tightness of each constraint. For this particular experiment we used a different constraint graph for each problem instance.

The sample size for each set of parameters used was 100 and for each instance the standard backtracking algorithm was run using the minimum width variable ordering heuristic (Freuder 1982). The heuristic was used in order to reduce the amount of time taken to solve the CSPs. The observed cost for solving each instance was recorded and then compared with the estimated number of constraint checks.

In order to give a concise overview of our results we analysed each set with respect to four parameters;

i. minimum ratio of expected and measured search cost
ii. maximum ratio of expected and measured search cost
iii. mean ratio of expected and measured search cost
iv. standard deviation of ratio of expected and measured search cost

The above parameters give a good indication of the worst case accuracy of our estimate, together with a typical accuracy. The complete set of results are given in table A.1.

		Expected Cost / Measured Cost			
Algorithm	Problem Class	Minimum	Maximum	Mean	Standard Deviation
bt	<10,10,0.2,0.88>	0.31	3.59	1.42	0.77
bt	<10,10,0.5,0.63>	0.49	1.48	1.02	0.20
bt	<10,10,1.0,0.40>	0.70	1.52	1.02	0.15
bt	<20,10,0.2,0.63>	0.24	17.00	2.1	2.28
bt	<20,10,0.5,0.38>	0.62	1.90	1.06	0.24
bt	<20,10,1.0,0.21>	0.57	1.50	1.03	0.18
bt	<30,10,0.2,0.52>	0.42	5.67	1.27	0.67
bt	<30,10,0.5,0.26>	0.49	1.72	1.12	0.28

Table A.1 - General accuracy of estimated complexities for bt with randomly generated binary CSPs. A sample size of 100 problems per class was used.

Our results are significant in that they show a reasonable degree of accuracy for all of the problem classes, except for those with the lower density problems. However, only the set of results which is shaded grey showed an average worse than a factor 2 error. Most estimates of the worst case search costs are well within a factor 2 of the observed cost. The average ratio of estimated to measured is much closer to 1.0 in most cases.

These results suggest that we can use the complexity estimates reliably for some randomly generated binary CSPs with high-density constraint graphs. These findings show promise, indicating that the use of theoretical complexity estimates is possible for the purposes of estimating the search cost of algorithms for some problem classes. As a result they have the potential for playing a significant role in the design of *ZDC* formulation evaluation heuristics. In the next section we investigate the direct usefulness of this approach when comparing different *ZDC* formulations of a given problem.

A.4 THEORETICAL ESTIMATES AS *ZDC* FORMULATION EVALUATION HEURISTICS

In the previous section we considered the use of theoretical complexity estimates for predicting the search costs of solving CSP instances, with respect to their solving by a particular algorithm. One important application of such estimates is with the comparison of different *ZDC* formulations of a problem, as suggested in (Nudel 1983b) (Nadel 1990). If the theoretical complexity estimates give a good indication of the actual cost of solving a given CSP, then they provide us with a candidate for use as an evaluation heuristic.

Furthermore, the purpose of an evaluation heuristic is to identify the relative expected search cost of a given pair of formulations. This means that even if the estimates are not particularly accurate in terms of predicting the actual cost of particular instances, they may still be useful, provided they accurately reflect the relative cost. We should remember that what we are looking for is evaluation *heuristics* which give us this relationship. The important point is the qualitative relationship and not necessarily the exact quantitative nature of it. This is analogous to Nadel's use of estimates to obtain "optimal" search orderings (Nudel 1983a). His results there showed that complexity estimates could accurately reflect the qualitative nature of varying the search ordering.

The problem type that we used for the purposes of assessing the usefulness of equation (A-7), as the basis for an evaluation heuristic, was that of randomly generated binary CSPs. We required two different formulations of each problem instance considered before any comparison could be made. As the first *ZDC* formulation we used CSPs generated in the same way as we did in section A.3, using the 4-tuple $\langle n, m, p1, p2 \rangle$. We denote this formulation to be *R1*;

R1 - *Z*: The n variables
D: $\{1..m\}$ for each variable.
C: $p1 \times n(n\text{-}1)/2$ random binary matrix constraints each having tightness $p2$.

Our second formulation was based on the idea of taking an *R1* instance and transforming it by replacing pairs of constrained variables with a new variable which is generated by merging the original pair. For example, given a pair of variables x_a and x_b which are constrained by C_{ab}, the domain of the new merged variable, x_{ab}, is given as the set of legal tuples in the constraint C_{ab}. An example of this process is given in figure A.1.

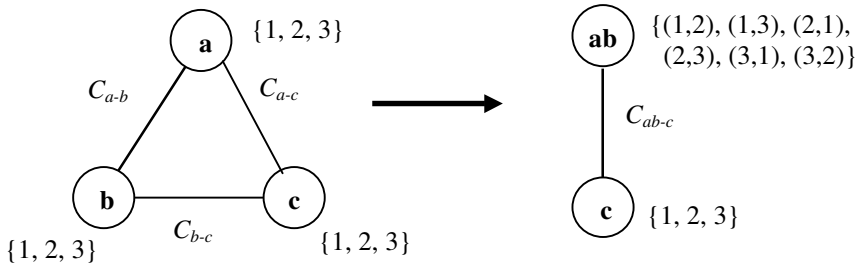

Figure A.1 - An example of variable merging for the case where the constraint $C_{a\text{-}b}$ is "not equal".

In order to create an alternative *ZDC* formulation we arrange the variables of *R1* in the search ordering we propose to use for *R1*. We then systematically take a single pass through that ordering and merge consecutive pairs of variables which are constrained by each other. The result of the transformation process is a new *ZDC* formulation of the problem which we call *R2*;

R2:	*Z*:	A mixture of merged and non-merged variables. If variables v_i and v_j from *R1* are merged, then they form a single variable $v_{i,j}$ in *R2*, as indicated in figure A.1
	D:	Either {1...*m*} for non-aggregated variables or the set of legal tuples given by an aggregation constraint
	C:	A combination of original constraints and constraint which have been modified to accommodate the aggregated variables

For our experiments we used samples of 100 problems for each *n*, *m*, *p1*, *p2* class. We looked at the relative merits of the two candidate *ZDC* formulations, *R1* and *R2*. We also used the minimum width variable ordering for the larger problems. For these cases, as we described above, we pass through the variables in the search ordering used when solving *R1* when we decide which pairs of variable are to be merged for *ZDC* formulation *R2*. This ensures that we effectively visit the variables in the same order for both formulations. The important point is that we have two *ZDC* formulations which we can expect to have differing search costs.

The results of our experiments are shown in table A.2. For each instance in a given problem class we compare the expected complexity values of *R1* and *R2* which we call *ec1* and *ec2* respectively. If we find these expected costs to be within a specified percentage of each other, called *margin*, then we regard the two formulations as having the same expected cost in solving and we make no prediction as to which is likely to be the best *ZDC* formulation of the two. Such instances contribute to the "*% no prediction*" column in table A.2. For cases where the difference is predicted to be greater than the specified percentage, we make a prediction of the actual instance search costs based on the estimates *ec1* and *ec2*. If the actual measured values show the same qualitative relationship as the predictions, we regard it as being a correct prediction. Otherwise, it is regarded as being incorrect.

		Heuristic Accuracy - margin=5%			Heuristic Accuracy - margin=15%		
Algorithm + Heuristic	Problem Class	%correct	%incorrect	% no prediction	%correct	%incorrect	% no prediction
bt+nat	<10,10,0.2,0.88>	73	10	17	60	6	34
bt+nat	<10,10,0.5,0.63>	77	7	16	62	2	36
bt+nat	<10,10,1.0,0.40>	100	0	0	100	0	99
bt+mwo	<20,10,0.1,0.88>	64	29	5	60	27	13
bt+mwo	<10,10,0.2,0.63>	62	23	15	58	19	23
bt+mwo	<10,10,0.5,0.38>	100	0	0	100	0	0
bt+mwo	<30,10,0.07,0.85>	98	2	0	95	2	3
bt+mwo	<30,10,0.2,0.52>	72	22	6	66	19	15

Table A.2- summary of formulation comparison for randomly generated binary CSPs. 100 problem instances generated per problem class.

As we can see from the data in table A.2, equations (A-7) were seen to be effective in selecting between *ZDC* formulations *R1* and *R2*. We can say this since the number of times the selections were correct or did not make a prediction was greater than 50% for all problem sets. In other words selections based on our theoretical complexity estimates give better results than simply picking one of the two formulations at random. In fact our heuristics performed considerably better than this.

A further, important observation that we can make from table A.2 is that our heuristics are also effective on problems with low-density constraint graphs. This is important because we have previously noted that our complexity equations are less accurate in estimating the actual search cost of algorithms for lower density problems. Our results suggest that when we use different formulations of a problem, the qualitative nature of the relative complexity values is reflected more reliably.

A.5 DISCUSSION

In this appendix, we have summarised some of the work presented in (Borrett 1998). We have introduced Nadel's complexity analysis framework and provided evidence to demonstrate its usefulness as an evaluation heuristic ($\mathcal{H}_e$) for the standard backtracking algorithm. Similar results have been obtained for the forward checking and backjumping algorithms, see (Borrett 1998). More work needs to be done to further assess the properties of theoretical estimates for different types of CSP.

It is important to note that the aim of any evaluation heuristic is to give us some heuristic indication of the relative merits of different *ZDC* formulations. We should not lose sight of the fact that it is heuristics that we seek. Using theoretical complexity estimates provides us with just one approach which has shown promise with certain classes of CSP. Other approaches are possible and much more work remains to be done in this area.