CHAPTER 6

Evaluation Heuristics for Redundant Constraints

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 effect 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 introduces 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 chapter we investigate the idea of selectively adding redundant constraints to *ZDC* formulations of binary CSPs. Our approach follows on from the work in chapters 4 and 5, by further extending the use of theoretical complexity measures of search cost. We develop a set of new *ZDC* formulation evaluation heuristics for use with the standard backtracking, backjumping and forward checking algorithms. These heuristics are important because the idea of selective addition of redundant constraints to *ZDC* formulations has not previously been investigated. As we shall show, our approach represents a major step forward in the this area.

6.1 Redundant Constraints

In chapter 1 we described the process of problem formulation. We said that once the *Z* and *D* structure of a *ZDC* formulation has been fixed the role of the constraint set, *C*, is to restrict the set of legal, fully assigned, compound-labels to be the solution set. This can often be achieved with one of a selection of different possible constraint sets. For example, consider the problem in figure 6.1.

$$P < Q < R$$

Figure 6.1 - 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;

$$C_{pq}: \quad p < q \tag{6-1}$$

and

$$C_{qr}: \quad q < r \tag{6-2}$$

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;

$$C_{pqr}: \quad p < q + r \tag{6-3}$$

The addition of constraint C_{pqr} moves the *ZDC* formulation closer being a complete maximal problem, as described in chapter 2. This 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 6.1 (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 described, the addition, or removal of redundant constraints can have a marked effect on the efficiency of search in a particular *ZDC* formulation. This was also seen in chapter 2 where the search costs of solving different *ZDC* formulations of the magic series problem varied by several orders of magnitude as a result of the addition of redundant constraints. As a further illustration, we show the effects of adding redundant constraints to the *ZDC* formulations of the *edge-numbering problem*.

6.1.1 Solving the Edge-Numbering Problem

The task in the edge numbering problem is to label each edge of a cube with a different number from 1 to 13 in such a way that the following conditions are satisfied;

i. the sum of the three edges meeting at each vertex is a constant, *x*
ii. the sum of the four edges round each face is constant, *y*

A possible solution to this problem is given in figure 6.2, where *x* is 21 and *y* is 28.

One possible *ZDC* formulation of the edge-numbering problem is to have a variable for each of the edges and two additional variables representing the "constants" *x* and *y*. Constraints are then added to ensure that all the edges are different and that the face and vertex conditions are satisfied. This gives us *ZDC* formulation *ENP_1*;

ENP_1:

Z:	12 variables to representing the individual edges and two additional variables to represent the face and vertex sums *x* and *y*.
D:	{1..13} for the edge variables. D_x = {1...39} and D_y {1...42}[1]
C:	One constraint stating that all the edges are different. Eight constraints stating the vertex conditions. Six constraints stating the face conditions.

On inspection of the edge-numbering problem, we also notice that the sum of the eight vertex constants is equal to the sum of the six face constants, since each edge is represented twice in each total. This gives us a redundant constraint;

C_{xy}: $8 \times x = 6 \times y$

We use this redundant constraint by adding it to *ENP_1*, forming a modified version which we call *ENP_2*.

Given these two *ZDC* formulations of the edge numbering problem, we implemented and solved both using the ILOG Solver programming language (ILOG 1994). The results for finding the first solution, which is depicted in figure 6.2, were 147mS for *ENP_1* and 80 mS for *ENP_2*. Since the formulations, and hence the implementations, are identical except for the one redundant constraint, we can conclude that the redundant constraint, C_{xy}, is responsible for this 83% gain in solving efficiency[2].

[1] The maximum domain size of *x* and *y* is based on the maximum total of the variables they constrain
[2] Using a DEC Alpha 3000 AXP machine running at 175MHz.

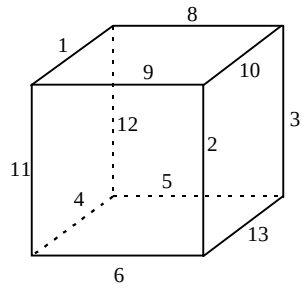

Figure 6.2 - A solution to the edge-numbering problem with x=21 and y=28

6.2 Generating Redundant Constraints

If we have an initial *ZDC* formulation, there are two basic ways in which redundant constraints can be generated and hence added. The first, and probably the most common, is through knowledge of the problem being solved. For example, in the problem described in figure 6.1, we used our knowledge of the arithmetic expression in order to derive the redundant constraint C_{pqr}. Another, more practical, example of this approach is seen in (Dincbas et al 1988). There, sets of redundant constraints are generated using knowledge of the nature of the car sequencing problem.

The second way in which redundant constraints can be obtained is using knowledge free, automatic, generation. For example, consider an original *ZDC* formulation having a core constraint graph, such as the graph indicated by the solid edges in figure 6.3. 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.

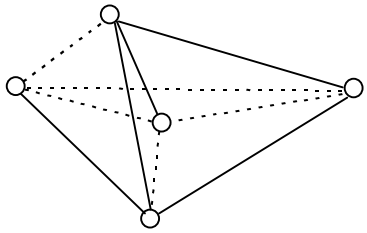

Figure 6.3 - An example constraint graph with some candidate redundant edges shown as dashed

One particular method which we can use to automatically generate redundant constraints is to identify *composition* constraints. These are described in detail in the following section and it is the intelligent, selective addition of these constraints to *ZDC* formulations that we investigate in the remainder of this chapter.

6.3 Redundant Composition Constraints

The problem described in figure 6.1 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;

$$P<R \qquad (6\text{-}4)$$

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 6.1, a redundant composition constraint C_{pr} exists.

In this section we investigate the properties of redundant composition constraints. We first look at how their usefulness is affected by the imposition of variable orderings, when used with systematic search algorithms. We then present a detailed analysis of how the effectiveness of search is affected by their inclusions for three systematic search algorithms - standard backtracking, backjumping and forward checking.

6.3.1 Redundant Constraints Under a Search Ordering

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.

The complication when using redundant constraints is that we do not always know when they are bringing forward explicit knowledge of new no-goods. This is because the content of a given redundant constraint can be the result of effects of many different combinations of other constraints in the *ZDC* formulation. However, with redundant composition constraints, we have the benefit of knowing where their content comes from - it results from the composition of two other known constraints. This is illustrated in figure 6.4.

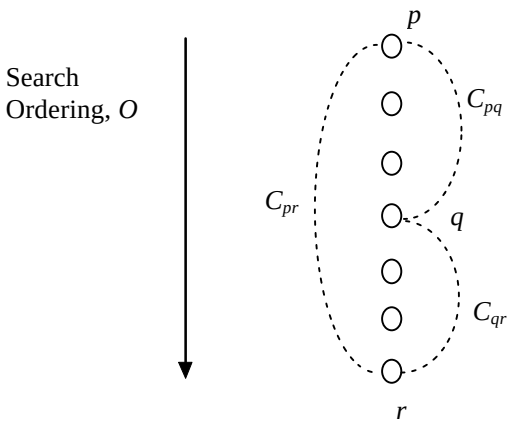

Figure 6.4 - Possible scenarios for redundant composition constraints under search ordering *O*

If any two of the constraints C_{pq}, C_{pr} and C_{qr} in figure 6.4 exist, then we have the opportunity to generate the third as a composition constraint. As we shall show in the remainder of this section, which of the three possible redundant composition constraints is useful, under a given search ordering, *O*, is dependent on the particular algorithm being used to solve the *ZDC* formulation.

6.3.2 Characteristics of Redundant Composition Constraints

As well as the algorithm and variable ordering, other factors affect the usefulness of a given redundant composition constraint. One of these factors is the tightness of the constraint. Clearly if the tightness, or *p2* value, is 0, which means it disallows no compound labels, then it will restrict the search space no further. Such a constraint serves no useful purpose, while it introduces the overhead of extra constraint checking. In contrast, if the constraint has a very high tightness value, we have more chance that a check of that constraint will result in elimination of futile search space, giving a reduction in the overall search cost.

Another factor affecting the usefulness of a constraint is the amount of search effort it is likely to save in the event that it causes a backtrack. This effort can be expressed in several ways. For example, we may consider the number of nodes in the search space saved, or the number of constraint checks saved.

From our analysis we see that there are some positive and some negative aspects to adding a redundant constraint. On the positive side, there are potential gains in avoiding portions of the search space. On the negative side, whenever the redundant constraint is checked there is an extra cost incurred. We now detail the effects of redundant composition constraints in terms of their effects on the number of nodes visited and the number of constraint checks performed during search.

6.3.2.1 Effects of Composition Constraint C_{pq}

Referring to figure 6.4, when constraints C_{pr} and C_{qr} exist, we can generate a redundant constraint C_{pq} by composition. In this section we consider the effect of this redundant constraint on standard backtracking, backjumping and forward checking.

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 6.1;

Proposition 6.1: 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 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. ■

For backjumping the effect of introducing redundant composition constraint C_{pq} is similar, but it is complicated by possible interactions with the effectiveness of the jumping mechanism. This leads to proposition 6.2;

Proposition 6.2: The addition of redundant composition constraint C_{pq} results in *the same*, *fewer or more* nodes being visited by backjumping, when a static variable ordering is used.

Proof: The effect of C_{pq} on the number of nodes visited by backjumping depends on whether or not there are constraints which exist between any variable prior to p and any variable between q and r. We denote case 1 to be the case where no such constraints exist and case 2 to be the case where one or more does.

case 1: As with standard backtracking, for backjumping the constraint C_{pq} is first checked when it reaches level q. For all previous search levels, the algorithm visits the same nodes as it would do in a *ZDC* formulation not including C_{pq}. When the constraint is checked at level r, if no further compatible values are found, a backtrack to the previous level, or a backjump to level p will occur in the case where no compatible value at all were found, 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 composition of conflicts with variables p and r will not take place at this point and it is deferred until the search reaches q. As a result, for this situation, the addition of C_{pq} can only result in the *same or fewer* nodes being visited.

case 2: If a constraint exists between a variable i, which is prior to p, and a variable j which lies between q and r, it is always possible that a jump could occur from j back to i, resulting in a section of the search space being eliminated. However, for the case where redundant composition constraint C_{pq} is added, backtracks at level q due to that redundant constraint mean that possible jumps from j to i are avoided, or delayed. As a result, sections of futile search can occur due to a culprit decision at level i and this means that *more* nodes could be visited than would have been the case without C_{pq}.

Given that both case 1 and case 2 are possible situations, the overall effect of redundant composition constraint C_{pq} can be for *the same*, *fewer or more* nodes being visited by backjumping, when a static variable ordering is used. ■

The effect of redundant composition constraint C_{pq} on the number of nodes visited by forward checking is similar to that for standard backtracking. This gives us proposition 6.3;

Proposition 6.3: The addition of redundant composition constraint C_{pq} results in the *same or fewer* nodes being visited by forward checking, 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 forward checking when the search reaches level p. For all previous search levels, the algorithm visits the same nodes as it would do in a *ZDC* formulation not including C_{pq}. When the constraint is checked at level p, values may be pruned from the domain of variable q. If values are removed from the domain of q, then this can result in earlier domain wipe-out in the search levels between p and q, compared to the original *ZDC* formulation. This can only result in the same or fewer nodes being visited. ■

Propositions 6.1-6.3 gives us the effects of redundant composition constraint C_{pq} on the number of nodes visited. The effect of C_{pq} on the of number of constraint checks performed by these three algorithms is less easily identified. If less 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.

A summary of our findings in this section is given in table 6.1

Algorithm	Nodes visited	Compatibility Checks
bt	The same or fewer nodes visited	Can be more or less
bj	The same, fewer of more nodes visited	Can be more or less
fc	The same or fewer nodes visited	Can be more or less

Table 6.1 - The effects of adding redundant composition constraint C_{pq}

6.3.2.2 Effects of Composition Constraint C_{qr}

Referring to figure 6.4, when constraints C_{pr} and C_{pq} exist, we can generate a redundant constraint C_{qr} by composition. In this section we consider the effect of this redundant constraint on standard backtracking, backjumping and forward checking.

For standard backtracking, the introduction of C_{qr} has no effect on the number of nodes visited. This gives us proposition 6.4;

Proposition 6.4: 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 6.4 that we cannot gain in terms of the number of constraint checks when C_{qr} is added.

Proposition 6.5: 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. ■

In a similar way, C_{qr} also has no effect on the number of nodes visited by backjumping. This gives us proposition 6.6;

Proposition 6.6: The addition of redundant constraint C_{qr} has *no effect* on the number of nodes visited by backjumping, when a static variable ordering is used.

Proof: We can apply the same line of reasoning as used in the proof of proposition 6.4. Furthermore, no backjumps from level r to level q can occur as a result of C_{qr}. This is because for values in the domain of r to fail against constraint C_{qr}, they would also have to have failed against constraint C_{pr}. This is the case since C_{qr} is the composition of C_{pr} and C_{pq}. As a result, jumps to p would always take precedence and hence C_{qr} has no effect on the jumping mechanism. ■

It follows from proposition 6.6 that the number of constraint checks can never be less for backjumping when C_{qr} is added;

Proposition 6.7: The addition of redundant constraint C_{qr} can only *increase* the number of constraint checks performed by backjumping, when a static variable ordering is used.

Proof: As for proposition 6.5. ■

The forward checking algorithm also fails to gain from C_{qr}.

Proposition 6.8: The addition of redundant constraint C_{qr} has *no effect* on the number of nodes visited by forward checking, when a static variable ordering is used

Proof: C_{qr} is first checked by forward checking when the search reaches level q. However, any pruning on the domain of r would have already taken place at level p through the combined effect of constraints C_{pr} and C_{pq}. This results in the same performance as would have been seen with the original *ZDC* formulation. ■

Proposition 6.9: The addition of redundant constraint C_{qr} can only *increase* the number of constraint checks performed by forward checking, when a static variable ordering is used.

Proof: As for proposition 6.5. ■

The conclusion of the above analysis is that composition constraint C_{qr} never benefits any of our three algorithms, when a static variable ordering is used. This is summarised in table 6.2.

Algorithm	Nodes visited	Compatibility Checks
bt	The same nodes visited	Can be more
bj	The same nodes visited	Can be more
fc	The same nodes visited	Can be more

Table 6.2 - The effects of adding redundant composition constraint C_{qr}

6.3.2.3 Effects of Composition Constraint C_{pr}

Referring to figure 6.4, when constraints C_{pq} and C_{qr} exist, we can generate a redundant constraint C_{pr} by composition. In this section we consider the effect of this redundant constraint on standard backtracking, backjumping and forward checking.

For standard backtracking, the introduction of C_{pr} has no effect on the number of nodes visited.

Proposition 6.10: 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 6.10 that we cannot gain in terms of the number of constraint checks when C_{pr} is added.

Proposition 6.11: 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. ■

For the backjumping algorithm redundant composition constraint C_{pr} can be beneficial;

Proposition 6.12: The addition of redundant composition constraint C_{pr} result in the *same or fewer* nodes being visited by backjumping, when a static variable ordering is used. C_{pq} never increases the total number of nodes visited.

Proof: When a backjump is detected at level r, if the reason for that backjump is a conflict due to the combined effect of C_{pq} and C_{qr}, then that reason must also be detected in C_{pr}, by composition. In the original *ZDC* formulation, the resulting jump would be from r to level q. However, the addition of C_{pr} means that the jump could take the search all the way back to level p. This means that the nodes that backjumping would have had to cover to get back to that same reason for failure in the original *ZDC* formulation are eliminated. As a result, there is a possible reduction in the number of nodes visited. ■

In terms of constraint checks, we do not know whether the increases in jumping effectiveness will compensate for the cost of checking the additional constraint. The net effect of C_{pr} could be for more or less checks to be performed.

For forward checking, there is also the potential for a reduced number of nodes to be visited when C_{pr} is added.

Proposition 6.13: The addition of redundant composition constraint C_{pr} results in the *same or fewer* nodes being visited by forward checking, when a static variable ordering is used. C_{pr} never increases the total number of nodes visited.

Proof: The constraint C_{pr} is first checked by forward checking when the search reaches level p. For all previous search levels, the algorithm visits the same nodes as it would do in a *ZDC* formulation not including C_{pr}. When the constraint is checked at level p, values may be pruned from the domain of variable r. If values are removed from the domain of r, then this can result in earlier domain wipe-out in the search levels between p and q, compared to the original *ZDC* formulation. This can only result in the same or fewer nodes being visited, since forward checking backtracks chronologically. ■

The effect of C_{pr} on the number of constraint checks performed by forward checking is less well defined. If less nodes are visited after its addition, 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 adding C_{pr} to an original *ZDC* formulation can have both beneficial and detrimental effects on the number of constraint checks performed.

The above analysis is summarised in table 6.3.

Algorithm	Nodes visited	Compatibility Checks
bt	Same nodes visited	Can be same or more
bj	Never more nodes visited	Can be more or less
fc	Never more nodes visited	Can be more or less

Table 6.3 - The effects of adding redundant constraint C_{pr}

6.3.2.4 Summary of Effects of Redundant Composition Constraints

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 for *ZDC* formulation selection which we described in chapter 3, we can regard propositions 6.1-6.3, 6.12 and 6.13 as *ZDC* formulation suggestion heuristics, H_s. They suggest to us when we might gain from using these constraints. In the next section, we describe a set of evaluation heuristics, H_e, which can be used in conjunction with these suggestion heuristics.

6.4 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 adding those redundant constraints which show promise.

In section 6.3 we said that there is a trade off which must be considered before adding a redundant constraint. On the one hand we need to take into consideration the cost of checking the constraint in question, while on the other, we need to assess the amount of search space it is expected to eliminate. One approach which takes both of these factors into account is the theoretical complexity techniques used in chapters 4 and 5. In this section we adopt a similar approach. However, there is a significant complication with redundant constraints which needs to be addressed.

Equation (4-15) showed us that, using the theoretical complexity model, the expected number of solutions is given by;

$$S = \left(\prod_{i \le n} |D_{x_i}|\right)\left(\prod_{i<j \le n} p_{ij}\right)$$

This is an algorithm independent property which must hold for our three algorithms, standard backtracking, backjumping and forward checking. However, when a redundant constraint is added, provided the tightness is non-zero, the $\left(\prod_{i<j\leq n} p_{ij}\right)$ part of equation (4-15) will always be reduced. As a result, the *expected* number of solutions would be reduced. Clearly this should not be the case since redundant constraints, by definition, do not affect the number of solutions in a *ZDC* formulation. A similar effect is seen with the expected number of nodes at a given search level. In this section we overcome this problem by building on the work of chapter 4. The modifications we present, allow us to develop a set of *ZDC* formulation evaluation heuristics for redundant composition constraints, based on theoretical complexity estimates.

6.4.1 An Evaluation Heuristic for Standard Backtracking, ρ_{bt}

In this section we develop an expression for standard backtracking which tells us whether or not a given redundant composition constraint is likely to be useful. In section 6.3 we found that only the redundant composition constraint C_{pq} could have a beneficial effect on the search cost. Our evaluation heuristic therefore applies to this constraint. We call our heuristic ρ_{bt};

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

and

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

Referring to figure 6.4, according to proposition 6.1, when redundant composition constraint C_{pq} is added to a *ZDC* formulation it can only affect the number of nodes expanded by standard backtracking at search levels *q* to *r-1*, relative to the number of nodes expanded using the original *ZDC* formulation. As a result, when calculating the number of nodes expanded for search levels *1* to *r-1* we can use the same approach as (4-7) and simply include C_{pq} in the calculation. We call the number of nodes expanded when C_{pq} is present *n_red*, for "nodes redundant". Using (4-7) we have;

$$n\_red(bt,k) = \left(\prod_{i \in A_k} |D_{x_i}|\right)\left(\prod_{i<j \in A_{k-1}} p_{ij}\right) \qquad \forall k: k < r \qquad \text{(6-5a)}$$

For the case where search has progressed to level *r* and beyond, C_{pq} has no effect on the number of nodes since its effect is the composition of C_{pr} and C_{qr}. We can therefore ignore C_{pq} when calculating the number of nodes at levels greater than or equal to *r*. This gives us;

$$n\_red(bt,k) = \left(\prod_{i \in A_k} |D_{x_i}|\right)\left(\prod_{\substack{\neg(i=p \wedge j=q) \wedge \\ i<j \in A_{k-1}}} p_{ij}\right) \qquad \forall k: k \geq r \qquad \text{(6-5b)}$$

Our expression for calculating the expected number of constraint checks remains unaffected by the redundant constraint - the constraint C_{pq} is simply included in the calculation and hence will reflect the cost element of adding the constraint. As a result we can use (4-8) to calculate this value. We can combine (6-5) and (4-8) to give us an expected search cost, *c_red(bt)*;

$$c\_red(bt) = \sum_{k=1}^{n}\left(c(bt,k) \times n\_red(bt,k)\right) \qquad \text{(6-6)}$$

We can combine our expression for *c_red(bt)* with the expression for *c(bt)* to give us ρ_{bt};

$$\rho_{bt}(C_{pq}) = c\_red(bt) \,/\, c(bt) \qquad \text{(6-7)}$$

Using (6-7), 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.

6.4.2 An Evaluation Heuristic for Forward Checking, ρ_{fc}

The analysis in section 6.3 showed that two redundant composition constraints, C_{pq} and C_{pr}, can have a beneficial effect on the search cost of forward checking. The evaluation heuristic we describe in this section therefore applies to these two constraints. We call our heuristic ρ_{fc};

$\rho_{fc}(c, TYPE) < 1.0 \Rightarrow$ the addition of redundant composition constraint c is likely to be useful

and

$\rho_{fc}(c, TYPE) \geq 1.0 \Rightarrow$ the addition of redundant composition constraint c is not likely to be useful

where *TYPE* indicates the actual constraint being considered, having possible values {*PQ, PR*}.

Considering the complexity estimates we developed for forward checking in chapter 4, the addition of composition constraints affects three main elements of the calculation directly. These elements are the number of nodes visited, the survival probability and the future domain size of a variable. Furthermore, the actual search levels which can be affected by the constraints are from level *p* to *q-1* for both of the possible redundant constraints since at levels higher than *p*, the constraints have no influence and at levels *q* or lower, the redundant information becomes duplicated by the original constraints. This knowledge can be used to modify the theoretical complexity estimates, so allowing us to use the approach after any redundant composition constraints have been added.

We first develop an expression for the number of nodes visited to incorporate the effects we outlined above. We call the number of nodes at level *k* with redundant constraints *n_red*. It is calculated by modifying (4-11) to ensure that the effects of the redundant constraints only occur at the relevant levels;

$$n\_red(fc,k,TYPE) = \left(\prod_{i \in A_k} |D_{x_i}|\right)\left(\prod_{i<j \in A_{k-1}} p_{ij}\right)\left(\prod_{f \in F_k} S(k-1, f, TYPE)\right) \qquad \forall k: k < q \qquad \text{(6-8a)}$$

$$n\_red(fc,k,TYPE) = \left(\prod_{i \in A_k} |D_{x_i}|\right)\left(\prod_{\substack{\neg(i=p \wedge j=q) \wedge \\ i<j \in A_{k-1}}} p_{ij}\right)\left(\prod_{f \in F_k} S(k-1, f, TYPE)\right) \qquad \forall k: k \geq q \wedge TYPE = PQ \qquad \text{(6-8b)}$$

$$n\_red(fc,k,TYPE) = \left(\prod_{i \in A_k} |D_{x_i}|\right)\left(\prod_{\substack{\neg(i=p \wedge j=r) \wedge \\ i<j \in A_{k-1}}} p_{ij}\right)\left(\prod_{f \in F_k} S(k-1, f, TYPE)\right) \qquad \forall k: k \geq q \wedge TYPE = PR \qquad \text{(6-8c)}$$

A similar modification is used to obtain a revised expression for the survival probability of future variables since the redundant constraints can bring forward pruning in future variables. We also change the notation used for survival probability from S_f^k which was used in (4-10) to *s(k, f, TYPE)*. This means the survival probability of future variable *f*, given current search level *k* and the inclusion of redundant composition constraint *TYPE*. (6-9b) and (6-9c) exclude the constraint affecting the calculation since their effect is the composition of two other constraints which are included in the calculation. This gives us;

$$s(k,f,TYPE) = 1 - \left(1 - \prod_{i \in A_k} p_{fi}\right)^{|D_f|} \qquad \forall k : k < q \tag{6-9a}$$

$$s(k,f,TYPE) = 1 - \left(1 - \prod_{\substack{\neg(i=p \wedge f=q) \wedge \\ i \in A_k}} p_{fi}\right)^{|D_f|} \qquad \forall k : k \geq q \wedge TYPE = PQ \tag{6-9b}$$

$$s(k,f,TYPE) = 1 - \left(1 - \prod_{\substack{\neg(i=p \wedge f=r) \wedge \\ i \in A_k}} p_{fi}\right)^{|D_f|} \qquad \forall k : k \geq q \wedge TYPE = PR \tag{6-9c}$$

In order to calculate the reduced domain size of future variable *f*, given search level *k*, we adopt a similar approach to the survival probability. This means we modify (4-12) to ignore the composition constraint when its has no effect. We also change the notation to incorporate the notion of *TYPE*. In place of $|D_k^f|$ we use *ds(f, k, TYPE)* which denotes the reduced domain size of future variable *f* given current search level *k* and redundant composition constraint *TYPE*. This gives us;

$$ds(f,k,TYPE) = \left|D_f\right| \left(\prod_{j \in A_{k-1}} p_{fj} \right) \Big/ S_f^{(k-1)} \qquad \forall k : k < q \tag{6-10a}$$

$$ds(f,k,TYPE)=\left|D_f\right|\left(\prod_{\substack{\neg(j=p\wedge f=q)\wedge \\ j\in A_{k-1}}} p_{fj}\right)\Big/ S_f^{(k-1)} \qquad \forall k: k<q\wedge TYPE=PQ \qquad \text{(6-10b)}$$

$$ds(f,k,TYPE)=\left|D_f\right|\left(\prod_{\substack{\neg(j=p\wedge f=r)\wedge \\ j\in A_{k-1}}} p_{fj}\right)\Big/ S_f^{(k-1)} \qquad \forall k: k<q\wedge TYPE=PR \qquad \text{(6-10c)}$$

The notation used for estimating the number of constraint checks is also modified slightly in order to incorporate these new expressions for the survival probability and the domain size of future variables. We use *c_red(fc, k, TYPE)* to denote the expected number of constraint checks for the forward checking algorithm at search level *k*, given redundant constraint *TYPE*. By modifying (4-13) to this notation we have;

$$c\_red(fc,k,TYPE)=\sum_{i=1}^{|G_k|}\left(ds(g_{ik},k,TYPE)\prod_{j=1}^{i-1} s(k,g_{jk},TYPE)\Big/ s(k-1,g_{jk},TYPE)\right) \qquad \text{(6-11)}$$

We can now complete our expression for the total estimated search cost in terms of constraint checks;

$$c\_red(fc,TYPE)=\sum_{k=1}^{n}\left(c\_red(fc,k,TYPE)\times n\_red(fc,k,TYPE)\right) \qquad \text{(6-12)}$$

We can combine our expression for *c_red(fc, TYPE)* with the expression for *c(fc)* to give us ρ_{fc};

$$\rho_{fc}(c, TYPE) = c\_red(fc,TYPE) / c(fc) \qquad (6\text{-}13)$$

Using (6-13), when the expected cost after adding a redundant constraint is less than the expected cost of the original *ZDC* formulation, we have $\rho_{fc} < 1.0$ which fits the definition of ρ_{fc} given at the start of this section.

6.4.3 An Evaluation Heuristic for Backjumping, ρ_{bj}

As with the forward checking algorithm, the analysis in section 6.3 showed that the redundant composition constraints C_{pq} and C_{pr} can have a beneficial effect on the search cost of backjumping. The evaluation heuristic we describe in this section therefore applies to these two constraints. We call our heuristicρ_{bj};

$\rho_{bj}(c, TYPE) < 1.0 \Rightarrow$ the addition of redundant composition constraint c is likely to be useful

and

$\rho_{bj}(c, TYPE) \geq 1.0 \Rightarrow$ the addition of redundant composition constraint c is not likely to be useful

where *TYPE* indicates the actual constraint being considered, having possible values {*PQ*, *PR*}.

In order to allow us to use the theoretical complexity estimates for backjumping when redundant composition constraints are added to a *ZDC* formulation we must modify the way we calculate the number of nodes expanded at level *k* in the search. Referring to figure 6.4, when a redundant composition constraint is added to a *ZDC* formulation, a constraint of type *PQ* directly affects the number of nodes visited by backjumping at search levels *q* to *r*, relative to the number of nodes visited using the original *ZDC* formulation. However, the effects of constraint *PR* are only seen in terms of jumping, which is reflected in the calculation of $eds(x_i)$, later in this section. This means that PR should not be directly included in the calculation of *n_red(bj, k, TYPE)*, the number of nodes expanded at level *k* for the backjumping algorithm with redundant constraints added. This leads us to modify (4-16) to take these factors into account;

$$n\_red(bj,k,TYPE) = \left(\prod_{i \in A_k} eds(\chi_i)\right)\left(\prod_{i<j \in A_{k-1}} p_{ij}\right) \qquad \forall k: k < r \wedge TYPE = PQ \tag{6-14a}$$

$$n\_red(bj,k,TYPE) = \left(\prod_{i \in A_k} eds(\chi_i)\right)\left(\prod_{\substack{\neg(i=p \wedge j=q) \wedge \\ i<j \in A_{k-1}}} p_{ij}\right) \qquad \forall k: k \geq r \wedge TYPE = PQ \tag{6-14b}$$

$$n\_red(bj,k,TYPE) = \left(\prod_{i \in A_k} eds(\chi_i)\right)\left(\prod_{\substack{\neg(i=p \wedge j=r) \wedge \\ i<j \in A_{k-1}}} p_{ij}\right) \qquad \forall k: TYPE = PR \tag{6-14c}$$

The second modification to our theoretical estimate for backjumping relates to the probability of jumps occurring at level *q* back across search levels between *p* and *q*, and at level *r* back across search levels between *p* and *r*. Both beneficial types of redundant composition constraint can affect backjumping and as a result we must include them in the calculation of *p(jump_at(j,i), TYPE)*, the probability of a jump occurring at level *j*, given current search level *i* and redundant composition constraint *TYPE*. However, we note that the jumping effects of redundant constraint C_{pr} should only by applied in this calculation for jumps to levels between *p* and *q*, since jumps across levels between *q* and *r* already apply to any existing constraint C_{qr}. This means we need to modify equation (4-20);

$$p(jump\_at(j,i),TYPE) = \left(1 - \prod_{l<i} p_{lj}\right)^{|D_{xj}|} \qquad \forall l: l < q \wedge TYPE = PR \tag{6-15a}$$

$$p(jump\_at(j,i),TYPE) = \left(1 - \prod_{\neg(l=p \wedge l<i)} p_{lj}\right)^{|D_{xj}|} \qquad \forall l: l \geq q \wedge TYPE = PR \tag{6-15b}$$

$$p(jump\_at(j,i),TYPE) = \left(1 - \prod_{l<i} p_{lj}\right)^{|D_{xj}|} \qquad \forall l: TYPE = PQ \tag{6-15c}$$

Furthermore we must ensure that the effect of each constraint on the probability of being at a specific level is only applied to the relevant levels of search. This is because the effects of the composition constraint are only different to the original constraints for levels between *p* and *r*. In fact, only C_{pq} has any effect on this probability and this is the case where adverse effects of redundant composition constraint C_{pq}, as highlighted in the proof of proposition 6.2, are taken into account. The expression for *p(at(j,k),TYPE)*, the probability of being at search level *j*, given current search level *k* and redundant composition constraint *TYPE*, is obtained by modifying (4-19) such that the redundant constraints are only included at levels where they have effects;

$$p(at(j,k),TYPE) = \prod_{\substack{k \le l < j \\ 1 < m < l}} p_{l,m} \qquad \forall l: l < r \wedge TYPE = PQ \tag{6-16a}$$

$$p(at(j,k),TYPE) = \prod_{\substack{\neg(m = p \wedge l = q) \wedge \\ k \le l < j \wedge \\ 1 < m < l}} p_{l,m} \qquad \forall l: l \ge r \wedge TYPE = PQ \tag{6-16b}$$

$$p(at(j,k),TYPE) = \prod_{\substack{\neg(m = p \wedge l = r) \wedge \\ k \le l < j \wedge \\ 1 < m < l}} p_{l,m} \qquad \forall l: TYPE = PR \tag{6-16c}$$

The modifications described in (6-15) and (6-16) allow us to calculate the value of $eds(x_i)$. This means we have all the elements for the expected number of nodes as given in (6-14). Using this, we can now complete an expression for the expected number of constraint checks by substituting (6-14) into (4-22) to give;

$$c\_red(bj,TYPE) = \sum_{k=1}^{n} \left(c(bj,k) \times n\_red(bj,k,TYPE) \right) \tag{6-17}$$

We can combine our expression for *c_red(bj, TYPE)* with the expression for *c(bj)* to give us ρ_{bj};

$$\rho_{bj}(c, TYPE) = c\_red(bj,TYPE) / c(bj) \tag{6-18}$$

Using (6-18), when the expected cost after adding a redundant constraint is less than the expected cost of the original *ZDC* formulation, we have $\rho_{bj} < 1.0$ which fits the definition of ρ_{bj} given at the start of this section.

6.5 Evaluation of the ρ Heuristics

Having developed a set of heuristics for evaluating the addition of redundant composition constraints, in this section we assess their accuracy. We have stated in chapter 3 that for any evaluation heuristic, H_e, to be useful, it should have an accuracy of at least 50%. In other words, it should show some improvement on simply making arbitrary decisions about changes to *ZDC* formulations. For the addition of redundant composition constraints this means that we want to selectively add constraints such that an improvement in search efficiency is seen more than 50% of the time. Furthermore, we should also like to see an improvement in performance relative to the uninformed addition of the constraints.

In the remainder of this section we describe a set of experiments, using randomly generated binary CSPs, aimed at testing this criterion. We do so for the evaluation heuristics ρ_{bt}, ρ_{bj} and ρ_{fc}.

6.5.1 Experimental Method

In chapters 4 and 5 we used randomly generated binary CSPs and 3-colouring problems for the purposes of assessing evaluation heuristics. However, for 3-colouring problems and colouring problems in general, all of the constraints are the same - the "not equals" constraint. We notice from this that the composition of two "not equals" constraints is the null constraint. In other words it constrains no tuples and has a tightness of 0. As a result, the addition of redundant composition constraints is never effective in colouring problems and so we only use the randomly generated binary CSPs for our assessment of the evaluation heuristics ρ_{bt}, ρ_{bj} and ρ_{fc}.

Our approach is similar to the one used for assessing the α heuristics in chapter 5. We generated several sets of random binary CSPs, each having different characteristics, using the problem generator described in appendix A.2. 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.

For each instance generated, we call that *ZDC* formulation *R1*. We then applied the redundant composition constraint transformation, given in figure 6.5, to that *ZDC* formulation in order to generate a second *ZDC* formulation, *R2*. The transformation algorithm looks at candidate redundant constraints and adds them according to evaluation heuristic H_e. It was applied using several different instantiations of, H_e, at lines 9 and 14 of the algorithm. Six cases were used;

i. $H_e = 1.0$ - uninformed addition for standard backtracking

ii. $H_e = \rho_{bt}$

iii. $H_e = 1.0$ - uninformed addition for backjumping

iv. $H_e = \rho_{bj}$

v. $H_e = 1.0$ - uninformed addition for forward checking

vi. $H_e = \rho_{fc}$

The reason for including cases *i*, *iii* and *iv* was to test our "informed" evaluation heuristics against this uninformed baseline. We call the *ZDC* formulation resulting from these transformations *R3*.

```
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(v_j, v_k) ∧
8                  ¬CONNECTED(v_i, v_j))
9                 IF H_e < 1.0
10                   ADD_COMPOSITION(v_i, v_j)
11             IF (CONNECTED(v_i, v_j) ∧
12                 CONNECTED(v_j, v_k) ∧
13                 ¬CONNECTED(v_i, v_k) ∧ (alg=fc ∨ alg=bj) )
14                IF H_e < 1.0
15                   ADD_COMPOSITION(v_i, v_k)
16          k++
17       j++
18    i++
19 END
```

Figure 6.5 - The Redundant Composition Constraint Transformation

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

Our expectation was that the use of the uninformed evaluation heuristics 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 evaluation heuristics are effective, then we 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.

6.5.2 Results

For each of our three algorithms, standard backtracking, backjumping and forward checking, we had a cost measure for each problem instance;

cc(R1) - the cost of solving the original *ZDC* formulation
cc(R2) - the cost of solving the output of figure 6.5 using cases *ii*, *iv* and *vi*
cc(R3) - the cost of solving the output of figure 6.5 using the uninformed cases *i*, *iii* and *v*

In a similar way to the approach taken in chapter 5, each of these cost measures was determined for each of the three algorithms, standard backtracking, backjumping and forward checking. 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 heuristics, 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 tables 6.3-6.5

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 tables 6.3-6.5

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 can say that the 50% criterion outlined in chapter 3 is satisfied, which is a good result. Conversely, if the number of category 2 instances is greater than the number of category 1 instances then the 50% criterion is violated and such a 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 asses the performance of the redundant composition constraint transformation when combined with each ρ heuristic used. However, we should also like to be able to assess how much improvement our ρ heuristics provide 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 tables 6.3-6.5. If the ρ heuristics are 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.

As an example of reading the tables of results, let us consider the results for backjumping with problem class <40, 5, 0.1, 0.52> given in table 6.4. For a margin of significance of 5%, we see that for 85% of the problem instances, *ZDC* formulation *R2* gives a better performance than *ZDC*

formulation *R1*, when ρ_{bj} is used. Only 7% cases were found where *R1* is better than *R2*, so the 50% criterion is satisfied. This compares well with the case where uninformed addition of redundant composition constraints was applied, where only 45% of the problem instances resulted in *ZDC* formulation *R3* giving a better performance than *ZDC* formulation *R1* and 49% giving a worse performance. As a result, the uninformed approach actually fails the 50% criterion.

In the tables some of the cells 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. This effectively says that the 50% criterion is violated. At the same time, the results for our ρ heuristics in these cells show much better performance, easily satisfying that criterion.

6.5.2.1 Conclusion

On inspection of the results in tables 6.3-6.5, we see that for some cases, the use of uninformed addition of redundant composition constraints leads to satisfaction of the 50% criterion given in chapter 3. An example of this is for the problem class <20, 10, 0.10, 0.81> where the criterion is satisfied for each of the algorithms considered. 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, 0.60>, for each algorithm. Further examples of these classes are indicated as the dark grey cells in the tables. This contrasts greatly with gains we see from using the ρ heuristics which provide excellent all round performance.

The results are promising and they suggest that our modified theoretical complexity estimates can be used reliably for analysing the effects of adding redundant constraints to *ZDC* formulations for the problem classes considered. We can say this because, with the use of the ρ heuristics, the number of cases where the 50% criterion was violated was reduced to just a single problem class - <20, 10, 0.30, 0.50> for ρ_{bj}, and even then the performance of ρ_{bj} was still an improvement on the uninformed approach. Furthermore, the range of problem classes used includes many which have very low constraint graph densities. This reinforces the observation that there are many cases where the theoretical complexity approach can be used on low density *ZDC* formulations.

		Accuracy with 5% margin (%instances)		Accuracy with 15% margin (%instances)	
Class	Algorithm+ Heuristic	$\frac{cc(R2)}{cc(R1)} \leq 0.95$	$\frac{cc(R2)}{cc(R1)} \geq 1.05$	$\frac{cc(R2)}{cc(R1)} \leq 0.85$	$\frac{cc(R2)}{cc(R1)} \geq 1.15$
<20, 5, 0.10, 0.72>	bt+nat	99 (100)	0 (0)	99 (99)	0 (0)
<20, 5, 0.10, 0.76>	bt+nat	100 (100)	0 (0)	100 (100)	0 (0)
<20, 5, 0.10, 0.80>	bt+nat	100 (100)	0 (0)	100 (100)	0 (0)
<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 6.3 - Results of adding redundant composition constraints using ρ_{bt} - uninformed figures are in brackets

		Accuracy with 5% margin (%instances)		Accuracy with 15% margin (%instances)	
Class	Algorithm+ Heuristic	$\frac{cc(R2)}{cc(R1)} \leq 0.95$	$\frac{cc(R2)}{cc(R1)} \geq 1.05$	$\frac{cc(R2)}{cc(R1)} \leq 0.85$	$\frac{cc(R2)}{cc(R1)} \geq 1.15$
<20, 5, 0.10, 0.72>	bj+nat	97 (98)	2 (1)	96 (98)	2 (1)
<20, 5, 0.10, 0.76>	bj+nat	99 (99)	1 (1)	99 (99)	1 (1)
<20, 5, 0.10, 0.80>	bj+nat	96 (96)	3 (3)	96 (96)	3 (3)
<20, 5, 0.10, 0.84>	bj+nat	97 (97)	3 (3)	95 (95)	3 (3)
<20, 5, 0.30, 0.32>	bj+nat	56 (33)	31 (58)	43 (21)	20 (53)
<20, 5, 0.30, 0.36>	bj+nat	84 (44)	13 (50)	70 (27)	10 (43)
<20, 5, 0.30, 0.40>	bj+nat	92 (47)	5 (38)	87 (43)	2 (26)
<20, 5, 0.30, 0.44>	bj+nat	98 (74)	1 (18)	94 (67)	1 (14)
<20, 10, 0.10, 0.81>	bj+nat	99 (100)	1 (0)	97 (100)	0 (0)
<20, 10, 0.10, 0.86>	bj+nat	98 (100)	1 (0)	98 (99)	0 (0)
<20, 10, 0.10, 0.91>	bj+nat	98 (98)	1 (1)	96 (97)	1 (1)
<20, 10, 0.30, 0.50>	bj+mwo	30 (0)	40 (100)	11 (0)	2 (100)
<20, 10, 0.30, 0.55>	bj+mwo	45 (0)	7 (100)	19 (0)	1 (100)
<20, 10, 0.30, 0.60>	bj+mwo	37 (0)	12 (97)	9 (0)	0 (92)
<40, 5, 0.10, 0.44>	bj+mwo	86 (48)	8 (50)	79 (45)	5 (40)
<40, 5, 0.10, 0.48>	bj+mwo	96 (61)	3 (34)	92 (54)	1 (28)
<40, 5, 0.10, 0.52>	bj+mwo	85 (45)	7 (49)	83 (38)	3 (42)
<40, 5, 0.30, 0.20>	bj+mwo	56 (27)	11 (50)	26 (17)	1 (32)
<40, 5, 0.30, 0.24>	bj+mwo	60 (9)	14 (88)	34 (2)	3 (83)

Table 6.4 - Results of adding redundant composition constraints using ρ_{bj} - uninformed figures are in brackets

		Accuracy with 5% margin (%instances)		Accuracy with 15% margin (%instances)	
Class	Algorithm+ Heuristic	$\frac{cc(R2)}{cc(R1)} \leq 0.95$	$\frac{cc(R2)}{cc(R1)} \geq 1.05$	$\frac{cc(R2)}{cc(R1)} \leq 0.85$	$\frac{cc(R2)}{cc(R1)} \geq 1.15$
<20, 5, 0.10, 0.72>	fc+nat	89 (90)	7 (9)	83 (89)	3 (7)
<20, 5, 0.10, 0.76>	fc+nat	92 (97)	4 (3)	92 (95)	2 (3)
<20, 5, 0.10, 0.80>	fc+nat	86 (94)	2 (4)	84 (91)	0 (2)
<20, 5, 0.10, 0.84>	fc+nat	83 (86)	0 (5)	81 (84)	0 (5)
<20, 5, 0.30, 0.32>	fc+nat	5 (10)	1 (81)	4 (7)	0 (74)
<20, 5, 0.30, 0.36>	fc+nat	1 (21)	1 (70)	0 (9)	0 (58)
<20, 5, 0.30, 0.40>	fc+nat	4 (24)	0 (66)	0 (15)	0 (56)
<20, 5, 0.30, 0.44>	fc+nat	4 (40)	0 (49)	1 (26)	0 (36)
<20, 10, 0.10, 0.81>	fc+nat	88 (91)	3 (8)	87 (90)	1 (5)
<20, 10, 0.10, 0.86>	fc+nat	98 (97)	1 (1)	97 (97)	0 (1)
<20, 10, 0.10, 0.91>	fc+nat	93 (96)	1 (2)	92 (95)	0 (2)
<20, 10, 0.30, 0.50>	fc+mwo	0 (0)	0 (100)	0 (0)	0 (100)
<20, 10, 0.30, 0.55>	fc+mwo	1 (0)	0 (99)	0 (0)	0 (100)
<20, 10, 0.30, 0.60>	fc+mwo	0 (0)	0 (100)	0 (0)	0 (99)
<40, 5, 0.10, 0.44>	fc+mwo	50 (42)	9 (54)	36 (36)	3 (49)
<40, 5, 0.10, 0.48>	fc+mwo	51 (48)	6 (45)	34 (39)	0 (38)
<40, 5, 0.10, 0.52>	fc+mwo	42 (29)	0 (68)	16 (24)	0 (62)
<40, 5, 0.30, 0.20>	fc+mwo	0 (29)	0 (42)	0 (8)	0 (12)
<40, 5, 0.30, 0.24>	fc+mwo	0 (1)	0 (98)	0 (0)	0 (94)

Table 6.5 - Results of adding redundant composition constraints using ρ_{fc} - uninformed figures are in brackets

6.5.3 Quantitative Performance

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 figures 6.6-6.8 for the performance of ρ_{bt}, ρ_{bj} and ρ_{fc} on problem class <40, 5, 0.10, 0.48>. These results show the 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, especially for backjumping. 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. This gives further evidence of the robustness of our approach.

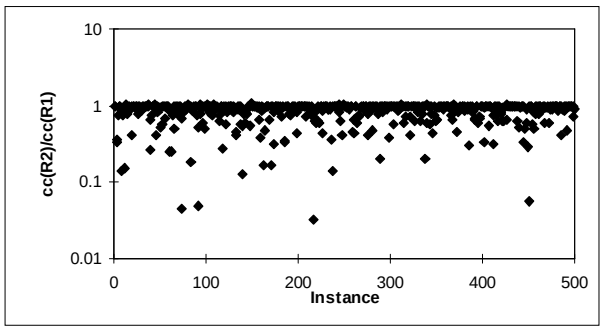

Figure 6.6 - Search costs on *ZDC* formulations *R1* and *R2* using ρ_{bt} - sample size of 500

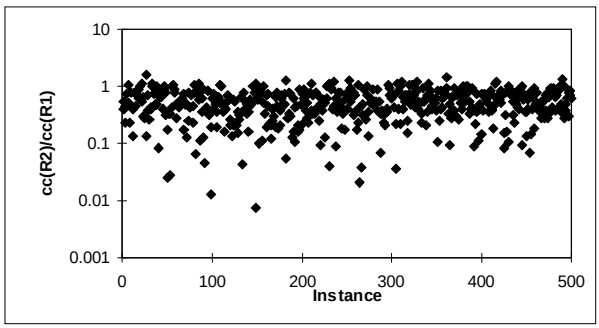

Figure 6.7 - Search costs on *ZDC* formulations *R1* and *R2* using ρ_{bj} - sample size of 500

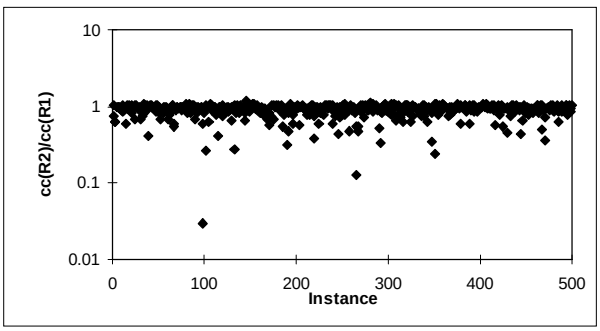

Figure 6.8 - Search costs on *ZDC* formulations *R1* and *R2* using ρ_{fc} - sample size of 500

6.6 Discussion

The results in section 6.5 show that the new evaluation heuristics we have developed in this chapter perform well on randomly generated binary CSPs. Several interesting observations arise from the results of our experiments.

6.6.1 Sensitivity to Class Variations

For a *ZDC* evaluation heuristic to be effective, it must perform well over a wide range of problem classes. Our results demonstrate that the ρ heuristics are sensitive enough to include redundant composition constraints when they are useful, while not including them when they are not useful. We saw this effect in tables 6.3-6.5 where only one pair of cells, class <20, 10, 0.30, 0.5> for backjumping, shaded light grey, show a qualitative performance which violates the 50% criterion outlined in chapter 3. This contrasts with the results obtained when candidate redundant constraints are always added, as indicated by the cells shaded both light and dark grey in the tables. The ρ heuristics prove to be far more selective, demonstrating the flexibility of the theoretical complexity approach.

6.6.2 Accuracy with Low Density Constraint Graphs

In chapter 4 we noted that the theoretical complexity model is less accurate in predicting search costs in CSPs with low density constraint graphs. However, we also noted that this does not necessarily preclude the approach from being useful when comparing *ZDC* formulations since for *ZDC* formulation comparison we are interested in the qualitative relationship in search costs. Many of the problem classes used in this chapter have low density constraint graphs, such as <20, 5, 0.10, 0.72> and <40, 5, 0.10, 0.52> where the constraint graph density is 0.1. Good performance on such problem classes is seen for all three of our evaluation heuristics. This suggests that there the scope of useful application of the theoretical complexity approach can be extended using the techniques we have described.

6.6.3 Searching for a Single Solution

The searches performed in all of the experiments in this chapter were for finding the first solution to the problems in question. This relies on the relaxation of assumption A_4.1. However, despite this relaxation, the ρ heuristics have been successfully applied when searching for a single

solution. This further demonstrates how the scope of theoretical complexity estimates can be extended.

6.6.4 Exceptionally Hard Problems

The quantitative performance of our approach shows a useful and consistent gain in search costs for the transformed *ZDC* formulation, as indicated in figures 6.6-6.8. Furthermore, some significant gains are also seen in some instances within the problem class used. The process of adding redundant constraints nearly always results in a reduction in the number of nodes visited by the search algorithms we have considered, as we showed from our analysis in section 6.3. The reason for this is that redundant constraints, when they are useful, bring forward the elimination of regions of futile search space.

We conjecture that for some classes of CSP, bringing forward the elimination of such regions of futile search space could reduce the frequency of *exceptionally hard problems* (Smith & Grant 1995).

6.6.5 Effective Evaluation Heuristics for Constant Solution Density

The work in this chapter has concentrated on the development of effective suggestion and evaluation heuristics for the addition of redundant composition constraints. An important point to note about the addition of redundant constraints in general is that they have no effect on properties such as such as *T-Factor* or kappa, κ, which are sensitive to the solution density of a *ZDC* formulation. The reason for this is that redundant constraints do not affect the number of solutions in a *ZDC* formulation, or the dimensions of the search space, *S*. Modified theoretical complexity measures such as the ones we have described in this section can overcome this problem and provide a measure sensitive to the variations in this important property.

6.6.6 Extension to of Measure Vector

The evaluation heuristics we have described are based on the use of the single property of theoretical complexity. Our results show that this is a feasible approach. However, we believe that combinations of theoretical complexity with other measures may further enhance the performance of the ρ heuristics.

6.7 Summary

The use of redundant constraints is a well known technique which can bring significant reductions in problem solving efficiency. Methods for automatically evaluating the usefulness of candidate redundant constraints have not previously been developed. Such methods are essential if we are to take full advantage of redundant constraints as a tool for improving search costs since while redundant constraint addition can be good, there are also examples where it can prove detrimental to search cost. In this chapter we have developed a new and original approach which uses the idea of *ZDC* formulation evaluation heuristics. Our work has made several important contributions;

- We have presented a thorough analysis of the usefulness of redundant composition constraints with respect three systematic search algorithms. This analysis forms the basis of a *ZDC* formulation suggestion heuristic, avoiding the futile addition of constraints while identifying the opportunities for gains.

- We have modified the theoretical complexity equations presented in chapter 4 and used these modifications to develop a set of new *ZDC* formulation evaluation heuristics; ρ_{bt}, ρ_{bj} and ρ_{fc}.

- Extensive experiments were performed to evaluate the ρ. These experiments demonstrated that our approach does indeed provide us with a flexible method for selectively adding redundant composition constraints.

The ρ heuristics we have devised demonstrate a how the automatic, a priori, assessment of the likely effect of a given redundant constraint on the search cost can be achieved. Furthermore, the theoretical complexity approach we have adopted provides an accurate, low cost method for achieving this.