CHAPTER 5

Evaluation Heuristics for Variable Aggregation

Given a *ZDC* formulation, provided there are at least two variables in *Z*, there is one *ZDC* formulation transformation that can always be applied to it in order to generate further *ZDC* formulations - the *variable aggregation transformation*. As the name suggests, variable aggregation involves the merging of domains of variables and it can result in large reductions in the overall complexity of the search space. This, in turn, can lead to large reductions in the search cost for some problem classes. These characteristics make variable aggregation a useful *ZDC* transformation which has an major role to play in improving the overall cost of problem solving.

In Chapter 3 we said that reducing the search space complexity is a potentially useful *ZDC* formulation suggestion heuristic, H_s. The use of such a heuristic, when traversing the space of possible *ZDC* formulations, would point to the variable aggregation transformation as a suitable move operator. However, as we pointed out in chapter 2, reducing the complexity of the search space, *S*, does not always result in improved search costs. For some problem classes it can actually result in a degradation in search cost. This is to be expected since reducing *S* is only a heuristic. As a result, while aggregating variables can lead to large reductions in search cost, in some cases it can lead to increases. What is required for variable aggregation to be effective is a good evaluation heuristic, H_e, which helps to identify both good and bad aggregations.

In this chapter we investigate evaluation heuristics specific to the variable aggregation transformation. We build on the general evaluation heuristic discussed in chapter 4, and present an important new set of heuristics which use knowledge about the nature of the variable aggregation transformation in order to improve its effectiveness. Our heuristics are designed for *ZDC* formulations that are to be solved with the standard backtracking, backjumping and forward checking algorithms and we present results demonstrating significant improvements in the effectiveness of the transformation.

5.1 The Variable Aggregation Transformation

The basic principle of variable aggregation was described in Chapter 4. It is a simple *ZDC* formulation transformation where a pair of variables, x_a and x_b, in an original *ZDC* formulation f_1, is aggregated to form a single variable, x_{ab}, in the new formulation f_2. The domain of x_{ab} is defined by the set of distinct 2-compound labels formed when selecting a value from each of the original domains. These variables may or may not be constrained to each other. If they are not, then the domain of the new variable becomes the complete set of these 2-compound labels. However, when there is a binary constraint between variables x_a and x_b, the domain of x_{ab} is reduced to the set of legal compound labels which define that constraint. This is illustrated in figure 5.1 where the number of variables is reduced from 3 to 2 and where the search space complexity, *S*, is reduced from 27 to 18.[1]

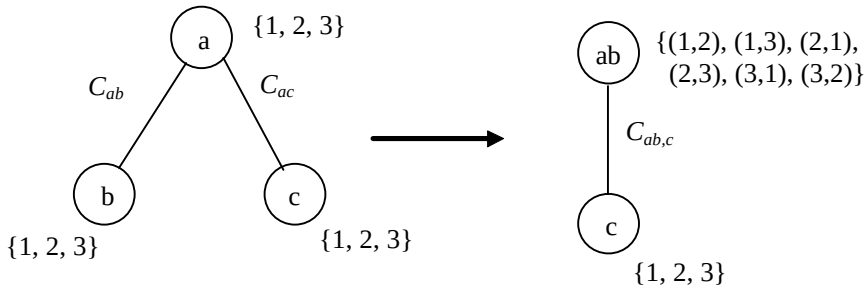

Figure 5.1 - An example aggregation where C_{ab} and C_{ac} are both "not equal".

The reduction in *S* shown in figure 5.1 can be magnified when applied to larger problems where more pairs of variables are available for merging. As a result, as we have previously mentioned, this *ZDC* transformation presents a significant opportunity for improving problem solving

[1] The original value of *S* was $|D_a|\times|D_b|\times|D_c|=3\times3\times3=27$. For the new formulation S is $|D_{ab}|\times|D_c|=6\times3=18$.

efficiency. In order for us to take advantage of such opportunities, the merging of variables can be applied in a systematic fashion. One such algorithm for achieving this is given in figure 5.2.

```
1  Given a CSP(Z, D, C) and a variable x in ordering O:
2  BEGIN
3    FOR i = 1 to i = |Z|-1
4      IF CONNECTED(x_i, x_i+1)
5            AGGREGATE(x_i, x_i+1)
6            i += 2
7            CONTINUE
8      ELSE
9          i++
10         CONTINUE
11 END
```

Figure 5.2 - the basic variable aggregation transformation

For this particular algorithm, pairs of variables are only aggregated if they are connected by a constraint, as defined at line 4. If this were not the case, then there would be no reduction in search space complexity, which is counter to our motivation for carrying out such aggregations.

An example of the application of the basic variable aggregation transformation is with the well known *zebra problem* (Smith 1992) (Prosser 1993). The zebra problem involves solving a puzzle where five people of different nationalities are to be assigned to five different houses, to have a different pet and to be assigned a different drink and brand of cigarettes, all subject to certain constraints. The full problem is given in figure 5.3.

1. The Englishman lives in the red house.
2. The Spaniard owns the dog.
3. Coffee is dunk in the green house.
4. The Ukrainian drinks tea.
5. The green house is immediately to the right of the ivory house.
6 The Old Gold smoker keeps snails.
7. Kools are smoked in the first house on the left.
8. Milk is drunk in the middle house.
9. The Norwegian lives in the first house on the left.
10. The Chesterfield smoker lives nest to the fox owner.
11. Kools are smoked next to the house with the horse.
12. The Lucky Strike smoker drinks orange juice.
13. The Japanese smokes Parliament.
14. The Norwegian lives next to the blue house.

The question to be answered is;

Who drinks water and who owns the zebra?

Figure 5.3 - the zebra problem.

There are many possible *ZDC* formulations of this problem. One natural *ZDC* formulation for the zebra problem is *Z1*;

Z1: *Z*: five variables, a_1-a_5, for the colours
five variables, b_1-b_5, for the nationalities
five variables, c_1-c_5, for the pets
five variables, d_1-d_5, for the drinks
five variables, e_1-e_5, for the cigarette brands

D: the five possible houses that can be assigned, i.e. {1...5}

C: constraints 1 to 14 as defined in figure 5.3
set of binary constraints between all variables a_1-a_5 stating that they have different houses
set of binary constraints between all variables b_1-b_5 stating that they have different houses
set of binary constraints between all variables c_1-c_5 stating that they have different houses
set of binary constraints between all variables d_1-d_5 stating that they have different houses
set of binary constraints between all variables e_1-e_5 stating that they have different houses

We applied the aggregation transformation described in figure 5.2 to *Z1*, so obtaining a second *ZDC* formulation, *Z2*. These were then solved using standard backtracking, backjumping and forward checking combined with the minimum width variable ordering (Freuder 1982)[2]. The results are given in table 5.1. They show that this is a case where aggregation proves to be beneficial.

	Checks for *Z1*	Checks for *Z2*
bt+mwo	8929	6598
bj+mwo	3623	2190
fc+mwo	2936	1375

Table 5.1 - the effects of aggregation on the zebra problem

In terms of our context, which we described in chapter 3, it is possible to categorise the various elements of the aggregation transformation as follows;

[2] The minimum width ordering was used as *O* in the aggregation. To ensure the variables in *Z1* and *Z2* are visited by the algorithms in effectively the same order, we actually use the natural, lexical ordering when solving *Z2*. In this way we see directly the effects of the aggregations.

1. We have a suggestion heuristic, H_s, which tells us to assume that decreasing the size of the search space complexity, *S*, is good.
2. Our move operator for generating new *ZDC* formulations is the aggregation of variable pairs.
3. We have a naive evaluation heuristic, H_e, which simply says that we should *"always accept"* the aggregated output.

This approach can result in substantial improvements in search cost. However, as we showed in chapter 2, there are some problem classes where this is not the case. In other words, the variable aggregation is effective on some classes of CSP but not on others. We can increase its useful scope by developing more effective evaluation heuristics, H_e.

Before we proceed with the design of any new evaluation heuristics we need to have a thorough understanding of the transformation to which it is being applied. In the next section we present a detailed analysis of the effects of variable aggregation.

5.2 Analysis of the Effects of Variable Aggregation

We have seen how there are cases where the effects of the variable aggregation transformation can be both beneficial and detrimental to the cost of search. In this section we undertake a detailed analysis of what actually happens when pairs of variables are merged to form a single variable. Our motivation for this analysis is to gain knowledge that will help us to develop more effective evaluation heuristics, which in turn will lead to improvements in the output of the transformation. By fully understanding the nature of the effects of the variable aggregation, we can ensure that our evaluation heuristics take these factors into account.

The analysis we present identifies ways in which gains and losses can be encountered, when pairs of variables are merged. The aim of this section is to identify the ways in which three algorithms, standard backtracking, backjumping and forward checking, can gain and lose from the aggregation of a given pair of variables in a binary *ZDC* formulation. We demonstrate the usefulness of this knowledge in section 5.4.

In order to aid the clarity of this section we shall refer to the example in figure 5.1. For systematic search algorithms there are two different scenarios in which aggregation can be viewed. These are illustrated in figure 5.4.

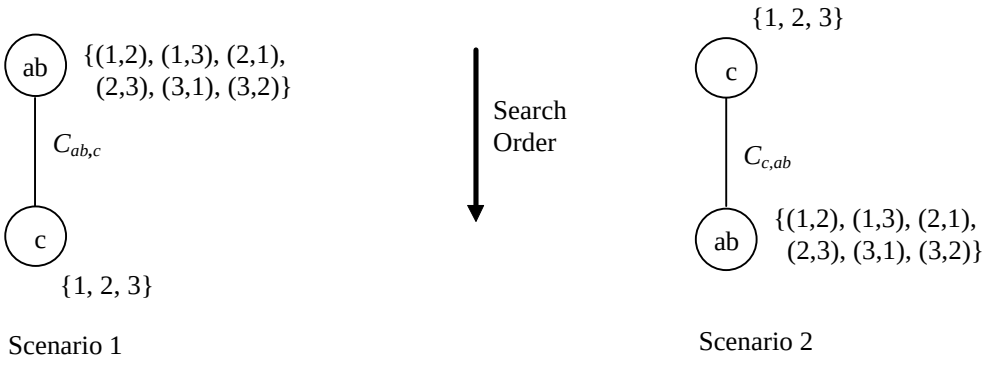

Figure 5.4 - Two search scenarios for the formulation f_2

Furthermore, our analysis is based on the assumption that only variables which are adjacent in the search ordering are aggregated, as described in the algorithm in figure 5.2;

> **Assumption A_5.1**: Aggregated pairs of variables are assumed to be adjacent in the search that would have taken place with the original formulation.

5.2.1 Gains for Standard Backtracking

There are two basic ways in which standard backtracking can gain when two variables are aggregated.

5.2.1.1 Gains Due to Nodal Elimination, $g1_{bt}$

When two variables are aggregated to form a new variable we effectively carry out a once-off, localised, search of the space of possible compound-labels between those two variables. In normal backtracking search in the original *ZDC* formulation we have the potential of visiting $\left|D_{x_a}\right| \times \left|D_{x_b}\right|$ nodes to exhaustively explore this sub-tree. In the aggregated variable, provided it involves a non-null constraint, we will always reduce the number of nodes covered in the new search space attributable to the two variables. An example of this was shown in figure 5.1 where the maximum number of nodes that can be visited in the "*ab*" sub-search space is reduced from 9 to 6. This

represents a potential gain of 3 nodes through the reformulation and applies to both scenario 1 and scenario 2 of figure 5.4, assuming all of the nodes in the *ab* space are visited. It follows that the more often the *ab* subspace is expanded, the more the actual total gain that can be made for any given search. We shall refer to gains attributable to this effect as $g1_{bt}$.

$g1_{bt}$ is an effect relative to the search that would have been carried out had x_a and x_b been adjacent in any search performed in the original formulation f_1, as given by assumption A_5.1. If A_5.1 did not hold then the $g1_{bt}$ can be greater since any intermediate search that would have taken place between x_a and x_b been could have been eliminated. Conversely, there is also the possibility that the search would not have even reached variable x_b.

5.2.1.2 Gains Through Constraint Merging, $g2_{bt}$

In *ZDC* formulation f_1 there is no constraint between variables *b* and *c*. Had this been the case then we would have had a second constraint in the new *ZDC* formulation, f_2. These two constraints would be on the same edge of the constraint graph. Checks could be saved by simply merging the two constraints, resulting in a single, tighter constraint[3]. It is likely that checking against this single constraint would be more efficient than having the two separate constraints[4]. We shall refer to gains due to this effect as $g2_{bt}$. The total gain attributable to $g2_{bt}$ is a function of the number of times that these constraints are checked.

5.2.2 Losses for Standard Backtracking

There are three basic losses which can be incurred by standard backtracking when aggregation takes place. Before we proceed with the description of these losses we need to familiarise ourselves with the important concept of *information potential*.

5.2.2.1 Information Potential

In chapter 2, we discussed the idea of goods and no-goods in the search space of a *ZDC* formulation. One way of viewing goods and no-goods is as pieces of information about the nature of that search space. For example, a constraint between two variables provides us with

[3] This is expected to be the case because it is probable that there will be more no-goods resulting from the merge, and there cannot be less.
[4] This assumes the new constraint is not less efficient to test than the original constraints - see $l3_{bt}$

information about the set of goods between them. More generally, when we consider a set of *k* variables, a possible state in the space of full assignments to those variables is represented by a *k*-compound-label. We call such a compound label a *base unit of information* for that search space;

Definition 5.2 - Given a set of *k* variables, a *base unit of information* is defined to be any possible *k*-compound-label which can be assigned to those variables. ■

For example, if we have two variables x_a and x_b we see that they represent $|D_{x_a}| \times |D_{x_b}|$ different 2-compound-labels. We say that each 2-compound-label $\{<x_a,v_i>, <x_b,v_j>\}$ represents a base unit of information between them.

Let us now consider the variables x_a and x_b under a search ordering *O*. If *O* orders x_a before x_b then it is possible that values in the domain of x_a are eliminated without extending the search further to x_b. For an algorithm such as standard backtracking, if a value in x_a is rejected, then the algorithm does not progress to x_b and another value is tried for x_a.[5] The result of this rejection of a single value in the domain of x_a is that $|D_{x_b}|$ base units in the "x_a-x_b" space are eliminated. We refer to this power of elimination of base units of information as the *information potential* of an assignment;

Definition 5.2 - Given a set of *k* variables under a search ordering *O*, the *information potential* of any *l-compound label*, such that $l<k$, is the number of base information units that can be expressed by extending that compound label to include all *k* variables. ■

In the case of our two variable example with x_a and x_b, we have $k=2$ and $l=1$. This means that there are $|D_{x_b}|$ base units of information that can be expressed by extending the 1-compound label to a 2-compound label.

Information potential is a useful concept in systematic search. For example, it explains why we should like to backtrack high in the search space. This is the case because at higher levels in the

[5] This is known as *a posteriori* pruning in (VanHentenryck 89)

search, assignments have higher information potential in terms of the set of variables in *Z*, hence they have the potential to eliminate greater sections of the search space.

5.2.2.2 Losses Through Reduced Information Potential, $l1_{bt}$

A major inefficiency which arises for standard backtracking following variable aggregation is that due to a reduction in information potential of the combined domain of variables x_a and x_b. It occurs because, assuming an original ordering $x_a<x_b$, in terms of the "x_a-x_b" space, all values in the domain of x_{ab} have an information potential of 1 base unit. This contrasts with the pre-aggregated case where the information potential of values in the domain of x_a is $\left|D_{x_b}\right|$. When there is more than one value in $D_{x_{ab}}$ for each member of the D_{x_a} there is the possibility of the aggregated *ZDC* formulation being less efficient. However, this inefficiency is only realised when members of $D_{x_{ab}}$ are incompatible with previous assignments.

The magnitude of efficiency reductions, due to a decrease in information potential, is dependent on the number of duplications of individual values in $D_{x_{ab}}$ and the tightness of the constraints connecting x_{ab} to previous variables in the search. We refer to this source of loss to standard backtracking as $l1_{bt}$.

5.2.2.3 Other Losses for Standard Backtracking

There are two further ways in which losses can be experienced by the standard backtracking algorithm. The first arises when only one of the merged variables is constrained to another past variable. For example, in figure 5.1 only x_a is constrained by x_c. When considering scenario 2, the constraint C_{ac} must be checked for each of the six values in the domain of x_{ab}. This contrasts with it only needing to be checked against the three values of x_a in formulation f_1. We call losses due to this effect $l2_{bt.}$

A further loss, $l3_{bt}$, is also possible when constraint merging takes place. If constraints are merged it may be the case that the cost of checking the resulting constraint is higher than the cost of checking the individual original constraints.

5.2.3 Gains for Forward Checking

There are three basic gains that can be made by the forward checking algorithm when pairs of variables are aggregated.

5.2.3.1 Gains Through Increased Consistency, $g1_{fc}$

The effect of forward checking on two aggregated variables is actually more than just simple forward checking when mapped back to the original formulation. Whenever a forward check is made against the aggregated variable, the algorithm is effectively checking the path x_c-x_a-x_b or x_c-x_b-x_a for consistency. This can result in backtracks higher up the search tree due to earlier dead-end detection and hence sections of the search space that would otherwise have been visited can be eliminated. The total effect, which we shall refer to as $g1_{fc}$, is a function of the number of forward checks made against the aggregated variable and the amount of search effort eliminated through the early detection of the dead-end.

5.2.3.2 Gains Through Reduced Lookahead Effort, $g2_{fc}$

Considering scenario 2, in the case of the original *ZDC* formulation, f_1, if x_c is assigned first then the forward checking algorithm would make a maximum of $|D_{x_a}| + |D_{x_b}| = 6$ constraint checks against the values in the domains of x_a and x_b. However, if $|D_{x_{ab}}|$ in f_2 is less than $|D_{x_a}| + |D_{x_b}|$, then in the worst case forward checking makes fewer constraint checks than would have occurred in f_1. On further inspection it becomes clear that this effect can also go into deficit and result in a loss (see loss $l1_{fc}$). The total effect, which we shall refer to as $g2_{fc}$, is function of the number of forward checks made against the aggregated variable.

5.2.3.3 Gains Through Constraint Merging, $g3_{fc}$

Again considering our example in figure 5.1, had there been a constraint C_{bc} in our original *ZDC* formulation, then there would have had an additional constraint in f_2. These two constraints would be on the same edge of the constraint graph and checks could be saved by simply merging the two constraints, resulting in a single, tighter constraint, as with $g2_{bt}$. We call this gain $g3_{fc}$.

5.2.4 Losses for Forward Checking

In this section we analyse the possible losses from aggregating pairs of variables. There are three basic losses which can be incurred by forward checking when variable aggregation takes place.

5.2.4.1 Losses Through Increased Lookahead Effort, $l1_{fc}$

As we mentioned in section 5.2.3.2, if $|D_{x_{ab}}|$ in f_2 is greater than $|D_{x_a}|+|D_{x_b}|$, then $g2_{fc}$ actually becomes a loss. We call this type of loss $l1_{fc}$ and it is the corollary of $g2_{fc}$.

5.2.4.2 Losses Through Reduced Information Potential, $l2_{fc}$

When there is more than one element in the domain x_{ab} for each member of the domain x_a we see possible losses due to a reduction in information potential, $l2_{fc}$. This is only realised when members of that domain are incompatible with previous assignments since assignments with the same reason for failure are repeated due to the absence of any information gain. The magnitude of this inefficiency is therefore dependent on the number of duplications of values in x_{ab} and the tightness of the constraints connecting x_{ab} to previous variables in the search.

5.2.4.3 Other Losses for Forward Checking

A third and less significant loss is also possible when constraint merging takes place. If constraints are merged it may be the case that the cost of checking that constraint increases due to increased complexity, as with $l3_{bt}$. We refer to this as $l3_{fc}$.

5.2.5 Gains and Losses for Backjumping

The basic gains and losses for the backjumping algorithm arise in the same way as those for the standard backtracking algorithm. This gives us two gains, $g1_{bj}$ and $g2_{bj}$, corresponding to $g1_{bt}$ and $g2_{bt}$. Similarly, we have potential losses $l1_{bj}$, $l2_{bj}$ and $l3_{bj}$ which correspond to the three losses, $l1_{bt}$, $l2_{bt}$ and $l3_{bt}$. However, other interactions can occur for backjumping following a variable aggregation, whereby the effectiveness of culprit detection is affected. This results in a further potential gain and a further potential loss for backjumping.

5.2.5.1 Gains Through Greater Jumps Back, $g4_{bj}$

In figure 5.5 we show a possible merging scenario. Taking the pre-aggregated case, if we find at some point in the search that all of the values in the domain of x_a are incompatible with the

assignment to x_j, the algorithm will jump back to that past variable since it is the reason for search failing to progress. Suppose also that all the values in the domain of x_b are also incompatible with the assignment to x_i. This does not affect the algorithm in the pre-aggregated case until we find a compatible past assignment for x_a. However, if we take the aggregated case, we have to test all of the values in the new domain before backjumping takes place. This also means that the tuples in the domain of x_{ab} are tested against variable x_i. Since all of these tuples fail against that variable's assignment, a jump can take place back to x_i. As a result, the algorithm could jump back to x_i much earlier than is the case for the pre-aggregated *ZDC* formulation. This has the obvious benefit of eliminating more futile search space and hence reducing the search cost. We refer to this potential gain as $g4_{bj.}$

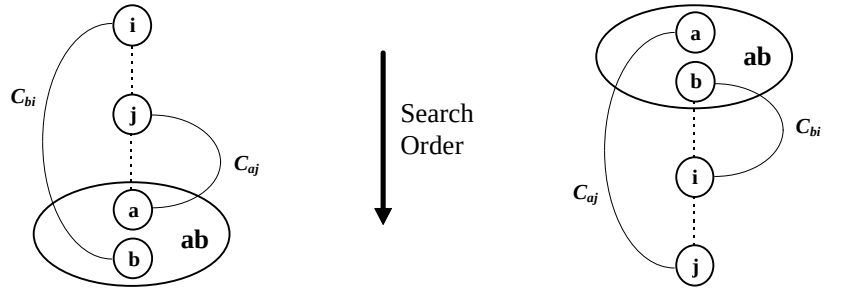

Figure 5.5 - Examples of how jumping efficiency is affected by aggregation

5.2.5.2 Losses Through Smaller Jumps Back, $l4_{bj}$

The second scenario in figure 5.5 shows us the constraints after x_{ab} in the search ordering. If we consider the pre-aggregated case, if no value can be found for variable x_j which is compatible with the assignment of x_a, the algorithm would jump back over x_b and try a new assignment for x_a. However for the aggregated case, when the corresponding jump would take us back to x_{ab} and the algorithm would try a new value from that domain. Since the values from the original domain of x_a can appear in several of the tuples in the domain of x_{ab}, we could end up with an assignment at x_i failing again for the same reason as before. Effectively we could say that we are only jumping back as far as variable x_b if the aggregated case is mapped back to the original *ZDC* formulation. We refer to this potential loss as $l4_{bj.}$

5.2.6 Other Losses

The gains and losses described above are direct effects of the content of the constraints and domains of the variables in the original problem formulation. However, there is a significant additional effect on search performance which should be considered whereby variable ordering heuristics become confused. An example of how this can happen is with the fail first (*ff*) dynamic variable ordering heuristic, based on the smallest domain size (Haralick&Elliott 1980). When an aggregation is carried out between two variables, the result is often a larger domain size. According to the *ff* heuristic this new variable will be selected late on in the search due to its larger domain size, even if a substantial amount of pruning is done. The result can mean a variable which is likely to fail not being assigned early on in the search and a sub-optimal ordering.

A second example of the effect aggregation can have on variable ordering heuristics is with the minimum width ordering (*mwo*) (Freuder 1982). *mwo* uses graph based information to generate a static variable ordering. When aggregation takes place, the structure of the constraint graph is affected. This can be beneficial as shown in (Dechter&Pearl 1989) with the tree clustering technique. However, the aggregation can also confuse graph-based heuristics, resulting in a less effective variable ordering.

The net effect of aggregation in both of the above examples is not easy to quantify. It could outweigh any gains from the other beneficial effects of aggregation, since heuristics play an important part in effective problem solving (Tsang et al 1995). However, the effects may also be beneficial, or they may be insignificant. One possible way to avoid the negative effects on static variable orderings is to set the ordering before aggregation takes place. For the purposes of our work we make the assumption that the same variable ordering is used before and after the transformation.

> **A_5.2**: The same ordering of variables is used before and after the aggregation transformation.

5.3 Theoretical Complexity Based Evaluation Heuristics for Variable Aggregation

Our analysis in section 5.2 showed that there are many different ways in which algorithms can gain and lose from the use of variable aggregation. In order to be able to assess the relative merits of aggregating a given pair of variables we should like to be able to take all of these factors into account. If this can be achieved then we have the basis of an effective evaluation heuristic for the variable aggregation transformation.

One approach to considering the net effect of many different properties of different *ZDC* formulations is to use the theoretical complexity approach described in chapter 4. The use of such estimates is still valid following the aggregation of a pair of variables since no further violations of the assumptions described in chapter 4 are incurred. As a result the use of theoretical complexity estimates can be reasonably applied to both the original and transformed *ZDC* formulations.

In the remainder of this section we use equations (4-9), (4-14) and (4-23) as the basis for three evaluation heuristics which can be applied to standard backtracking, backjumping and forward checking respectively. Using such heuristics, we modify the transformation algorithm described in figure 5.2, by adding line *4b*, so that it is more selective in deciding whether or not an aggregation should be performed. This revised transformation is given in figure 5.6. We investigate the suitability and effectiveness of our heuristics for use with our modified variable aggregation transformation.

```
1  Given a CSP(Z, D, C) and a variable x in ordering O:
2  BEGIN
3    FOR i = 1 to i = |Z|-1
4      IF CONNECTED(x_i, x_i+1)
4b         IF H_e < 1.0
5            AGGREGATE(x_i, x_i+1)
6            i += 2
7            CONTINUE
8      ELSE
9          i++
10         CONTINUE
11 END
```

Figure 5.6 - The modified variable aggregation transformation

5.3.1 Evaluation Heuristic $\alpha 1_{bt}$

By design, we should like our evaluation heuristic, $\alpha 1_{bt}$, to have a value such that it is less than 1.0 for cases where the aggregation of a given pair of variables is expected to prove beneficial to the algorithm. Effectively, it gives us an indication of the expected relative effect of any aggregation on the search cost. This is summarised as;

$\alpha 1_{bt}(x_a, x_b) < 1.0 \Rightarrow$ the aggregation of variables x_a and x_b is likely to be useful

and

$\alpha 1_{bt}(x_a, x_b) \geq 1.0 \Rightarrow$ the aggregation of variables x_a and x_b is not likely to be useful

Let $Cost_{bt}(f_1)$ be a function that uses (4-9) to generate the expected complexity for a given *ZDC* formulation, f_1, and $Cost_{bt}(f_2)$ be the expected complexity after variables x_a and x_b have been aggregated. What we want is for our evaluation heuristic to accept an aggregation if the expected complexity after the aggregation is lower than the expected complexity before the aggregation. We use this idea to give;

$$\alpha 1_{bt}(x_a, x_b) = \frac{Cost_{bt}(f_2)}{Cost_{bt}(f_1)} \qquad (5\text{-}1)$$

If $Cost_{bt}(f_2)$ is less than $Cost_{bt}(f_1)$, then $\alpha 1_{bt}$ has a value less than 1.0.

5.3.2 Evaluation Heuristic $\alpha 1_{fc}$

Following the line of reasoning we did for $\alpha 1_{bt}$, we have

$$\alpha 1_{fc}(x_a, x_b) = \frac{Cost_{fc}(f_2)}{Cost_{fc}(f_1)} \qquad (5\text{-}2)$$

where $Cost_{fc}(f_1)$ is a function that uses (4-14) to generate the expected complexity for a given *ZDC* formulation, f_1, and $Cost_{fc}(f_2)$ is the expected complexity after variables x_a, x_b have been aggregated, for the forward checking algorithm.

5.3.3 Evaluation Heuristic $\alpha 1_{bj}$

Again, following the same line of reasoning as we did for $\alpha 1_{bt}$, we have

$$\alpha 1_{bj}(x_a, x_b) = \frac{Cost_{bj}(f_2)}{Cost_{bj}(f_1)} \qquad (5\text{-}3)$$

where $Cost_{bj}(f_1)$ is a function that uses (4-23) to generate the expected complexity for a given *ZDC* formulation, f_1, and $Cost_{bj}(f_2)$ is the expected complexity after variables x_a, x_b have been aggregated, for the backjumping algorithm.

5.3.4 Effectiveness of Evaluation Heuristics $\alpha 1_{bt}$, $\alpha 1_{bj}$ and $\alpha 1_{fc}$

We have proposed a set of heuristics which evaluate the likely usefulness of aggregating a given pair of variables, for a given *ZDC* formulation. We 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 making an arbitrary choice. For the aggregation of pairs of variables this means that we want to selectively merge these variables such that an improvement in search efficiency is seen more than 50% of the time.

We now describe a set of experiments, using randomly generated binary CSPs, aimed at testing this criterion for evaluation heuristics $\alpha 1_{bt}$, $\alpha 1_{bj}$ and $\alpha 1_{fc}$.

5.3.4.1 The Use of Randomly Generated Binary CSPs

The use of randomly generated binary CSPs as a tool for evaluating new constraint satisfaction techniques has been widely adopted. For example, it has been used in (Prosser 1994) (Gent et al 1996) (Sabin & Freuder 1994) (Smith & Grant 1996) (Sakkout et al 1996) (Williams & Hogg 1995) for investigating various aspects of CSP solving. Clearly such problems are highly abstracted from real life problems. However, they do provide us with a useful, controllable framework in which to evaluate new ideas. As a result we have adopted a similar approach to those researchers in order to demonstrate the principles of our work.

5.3.4.2 Experimental Method

The aim of this experiment was to assess the effectiveness of $\alpha 1_{bt}$, $\alpha 1_{bj}$ and $\alpha 1_{fc}$ over a range of classes of randomly generated binary CSP. We generated our problem sets 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. In order to cover a range of problem characteristics, the problem sets we used included some which were close to the solubility phase transition(Cheeseman et al 1991) and others which were in the so-called "easy" regions.

For each instance generated, we denote the *ZDC* formulation output by the generator to be *AG1*. This formulation was solved using each of our three algorithms, standard backtracking, backjumping and forward checking. We then applied the variable aggregation transformation, described in figure 5.6, to *AG1* in order to generate a second *ZDC* formulation, *AG2*. For each problem instance, the transformation algorithm was applied using $\alpha 1_{bt}$, $\alpha 1_{bj}$ and $\alpha 1_{fc}$ as instantiations of the evaluation heuristic, H_e, at line *4b*. These new formulations were then solved using the relevant algorithm. For the heuristics to have been effective, there must be an improvement in the cost of searching *AG2*, relative to the cost of searching *AG1*. This process was repeated on 100 instances for each problem class considered.

Furthermore, for each of the instances we generated a baseline formulation to see how effective our heuristics were compared to the naive application of variable aggregation. The baseline we chose was to run the algorithm given in figure 5.2 on each instance's *AG1* formulation. The output of this we called *AG3*. We then solved *AG3* using the same algorithms as we used for solving *AG1* and *AG2*.

5.3.4.3 Results

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

cc(AG1) - the cost of solving the original *ZDC* formulation

cc(AG2) - the cost of solving the output of figure 5.6 using the α heuristics

cc(AG3) - the cost of solving the output of figure 5.2 using the uninformed aggregation

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 α heuristics, for each problem class tested, we divided the results into three categories;

cat. 1 - Instances where *cc(AG2)* was less than *cc(AG1)* by a margin of significance - i.e. a benefit was seen from the using the transformation. These results are given in columns 6 and 8 in tables 5.2-5.4

cat. 2 - Instances where *cc(AG2)* was greater than *cc(AG1)* by a margin of significance - i.e. using the transformation resulted in a degradation in performance. These results are given in columns 7 and 9 in tables 5.2-5.4

cat. 3 - Instances where the difference between *cc(AG2)* and *cc(R1)* is within the margin of significance - i.e. using the transformation resulted in no significant change in performance.

By dividing 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.

By processing the results as we have outlined above we can asses the performance of the variable aggregation *ZDC* transformation when combined with each α heuristic used. However, we should also like to be able to assess how much improvement our α heuristics provide over the uninformed aggregation of variables as given by the algorithm in figure 5.2. To see this effect we need to compare the number of instances in each category for *ZDC* formulations *AG2* and *AG3*. The results for *AG3* are given as the figures in brackets in tables 5.2-5.4. If the α heuristics are performing well and providing an improvement on the uninformed approach, then we should see more instances in columns 6 and 8 and fewer in columns 7 and 9, compared with the figures in brackets.

To illustrate how we process the results, let us consider columns 6 and 7 for problem class <20, 5, 0.30, 0.45> in table 5.2. These figures show that, with the 5% margin of significance, 60% of the instances showed that using the variable aggregation transformation in conjunction with $\alpha 1_{bt}$ resulted in an improvement in search cost and only 2% of instances showed a degradation. This contrasts with uniformed aggregation where the figures were 33% of instances showing a gain and 44% showing a loss. Columns 8 and 9 are read in the same way.

In the tables some of the cells are shaded grey. This is to highlight the cases where uninformed aggregation resulted in *cc(AG3)* being higher than *cc(AG1)* more frequently than it was lower. This effectively says that such an approach is failing the 50% criterion. At the same time, the results for our α heuristics in most of these cells show much better performance, easily satisfying that criterion, as indicated by the dark grey. The lighter grey cells indicate cases where this is not the case.

The quantitative performance of our heuristics is shown in columns 3, 4 and 5 of the tables. Here, we present the minimum, mean and median ratio of *cc(AG2)* and *cc(AG1)*. This information is calculated over the entire sample of 100 instances and it gives an indication of how much gain we are likely to achieve from using the variable aggregation transformation. For example, considering the problem class <20, 5, 0.10, 0.80> in table 5.2, there we see that the minimum ratio is 0.21 with a mean of 0.8 and a median of 0.89.

		$\frac{cc(AG2)}{cc(AG1)}$			Accuracy with 5% margin (%instances)		Accuracy with 15% margin (%instances)	
Class	Algorithm +Heuristic	min	mean	med.	$\frac{cc(AG2)}{cc(AG1)} \leq 0.95$	$\frac{cc(AG2)}{cc(AG1)} \geq 1.05$	$\frac{cc(AG2)}{cc(AG1)} \leq 0.85$	$\frac{cc(AG2)}{cc(AG1)} \geq 1.15$
<20, 5, 0.10, 0.70>	bt+nat	0.21	0.90	1.00	35 (40)	2 (9)	25 (27)	0 (3)
<20, 5, 0.10, 0.75>	bt+nat	0.25	0.88	0.95	49 (52)	0 (2)	31 (34)	0 (0)
<20, 5, 0.10, 0.80>	bt+nat	0.21	0.80	0.89	55 (59)	0 (1)	45 (46)	0 (1)
<20, 5, 0.30, 0.35>	bt+nat	0.70	0.96	0.98	29 (13)	0 (71)	4 (2)	0 (62)
<20, 5, 0.30, 0.40>	bt+nat	0.56	0.94	0.96	44 (15)	3 (64)	13 (3)	0 (43)
<20, 5, 0.30, 0.45>	bt+nat	0.51	0.90	0.91	60 (33)	2 (44)	27 (17)	0 (26)
<20, 10, 0.10, 0.80>	bt+nat	0.18	0.91	1.00	31 (28)	4 (2)	23 (15)	2 (1)

Table 5.2 - Results of applying the aggregation transformation using the $\alpha 1_{bt}$ evaluation heuristic

		$\frac{cc(AG2)}{cc(AG1)}$			Accuracy with 5% margin (%instances)		Accuracy with 15% margin (%instances)	
Class	Algorithm +Heuristic	min	mean	med.	$\frac{cc(AG2)}{cc(AG1)} \leq 0.95$	$\frac{cc(AG2)}{cc(AG1)} \geq 1.05$	$\frac{cc(AG2)}{cc(AG1)} \leq 0.85$	$\frac{cc(AG2)}{cc(AG1)} \geq 1.15$
<20, 5, 0.10, 0.70>	bj+nat	0.0013	0.80	0.97	48 (53)	5 (8)	41 (46)	3 (3)
<20, 5, 0.10, 0.75>	bj+nat	0.0053	0.70	0.85	57 (57)	1 (5)	50 (49)	1 (2)
<20, 5, 0.10, 0.80>	bj+nat	0.0010	0.65	0.76	59 (61)	1 (2)	54 (43)	0 (0)
<20, 5, 0.30, 0.35>	bj+nat	0.34	0.92	0.97	45 (39)	12 (41)	22 (20)	4 (31)
<20, 5, 0.30, 0.40>	bj+nat	0.41	0.92	0.97	45 (32)	9 (50)	22 (22)	3 (37)
<20, 5, 0.30, 0.45>	bj+nat	0.19	0.89	0.90	63 (49)	6 (30)	42 (31)	2 (19)
<20, 10, 0.10, 0.80>	bj+nat	0.0013	0.79	1.00	38 (25)	7 (6)	31 (20)	7 (5)
<30, 10, 0.10, 0.70>	bj+mwo	0.077	1.00	1.00	38 (33)	39 (54)	22 (20)	25 (35)
<30, 10, 0.10, 0.75>	bj+mwo	0.091	0.94	0.94	51 (48)	24 (36)	28 (30)	12 (24)
<30, 10, 0.10, 0.80>	bj+mwo	0.48	0.74	0.71	96 (79)	1 (10)	82 (58)	0 (4)
<30, 10, 0.10, 0.85>	bj+mwo	0.45	0.65	0.64	99 (96)	0 (1)	96 (85)	0 (1)
<40, 10, 0.10, 0.60>	bj+mwo	0.30	1.29	1.29	9 (15)	83 (82)	8 (13)	72 (75)
<40, 10, 0.10, 0.65>	bj+mwo	0.74	1.13	1.11	15 (4)	59 (96)	5 (2)	41 (87)
<40, 10, 0.10, 0.70>	bj+mwo	0.52	0.92	0.93	63 (2)	8 (88)	16 (1)	2 (73)
<40, 10, 0.05, 0.80>	bj+mwo	1.4 e-04	0.96	0.97	37 (52)	23 (32)	15 (27)	14 (26)
<40, 10, 0.05, 0.85>	bj+mwo	3.4 e-05	0.82	0.89	67 (72)	15 (18)	35 (56)	9 (10)
<40, 10, 0.05, 0.90>	bj+mwo	0.15	0.50	0.49	100 (98)	0 (1)	100 (98)	0 (1)
<40, 5, 0.10, 0.48>	bj+mwo	0.0014	1.00	1.02	32 (20)	41 (66)	20 (13)	24 (48)
<40, 5, 0.10, 0.52>	bj+mwo	0.47	0.97	0.98	37 (34)	23 (49)	20 (24)	8 (25)
<40, 5, 0.10, 0.56>	bj+mwo	0.44	0.81	0.82	83 (54)	4 (27)	59 (24)	2 (11)
<40, 5, 0.05, 0.62>	bj+mwo	1.2 e-06	1.01	0.95	48 (65)	27 (27)	17 (44)	22 (21)
<40, 5, 0.05, 0.66>	bj+mwo	1.9 e-04	0.86	0.94	57 (75)	12 (19)	21 (54)	7 (14)
<40, 5, 0.05, 0.70>	bj+mwo	1.8 e-04	0.71	0.82	81 (80)	9 (8)	60 (69)	7 (5)

Table 5.3 - Results of applying the aggregation transformation using the $\alpha 1_{bj}$ evaluation heuristic

		$\frac{cc(AG2)}{cc(AG1)}$			Accuracy with 5% margin (%instances)		Accuracy with 15% margin (%instances)	
Class	Algorithm +Heuristic	min	mean	med.	$\frac{cc(AG2)}{cc(AG1)} \leq 0.95$	$\frac{cc(AG2)}{cc(AG1)} \geq 1.05$	$\frac{cc(AG2)}{cc(AG1)} \leq 0.85$	$\frac{cc(AG2)}{cc(AG1)} \geq 1.15$
<20, 5, 0.10, 0.70>	fc+nat	0.0026	0.79	0.99	47 (57)	3 (6)	41 (42)	2 (4)
<20, 5, 0.10, 0.75>	fc+nat	0.012	0.77	0.93	52 (58)	4 (5)	45 (49)	2 (4)
<20, 5, 0.10, 0.80>	fc+nat	0.096	0.79	0.93	53 (66)	1 (2)	40 (54)	0 (1)
<20, 5, 0.30, 0.35>	fc+nat	0.53	0.95	1.00	34 (35)	16 (47)	23 (22)	3 (34)
<20, 5, 0.30, 0.40>	fc+nat	0.56	0.94	1.00	29 (27)	5 (54)	17 (17)	1 (37)
<20, 5, 0.30, 0.45>	fc+nat	0.50	0.92	1.00	33 (33)	6 (51)	26 (24)	0 (37)
<20, 10, 0.10, 0.80>	fc+nat	6.0 e-04	0.77	0.97	48 (40)	8 (6)	44 (35)	2 (1)
<30, 10, 0.10, 0.70>	fc+mwo	0.07	1.01	1.07	30 (36)	51 (54)	23 (25)	32 (43)
<30, 10, 0.10, 0.75>	fc+mwo	0.03	0.92	0.94	54 (32)	27 (52)	34 (27)	14 (33)
<30, 10, 0.10, 0.80>	fc+mwo	0.26	0.78	0.79	90 (51)	4 (34)	63 (31)	1 (23)
<30, 10, 0.10, 0.85>	fc+mwo	0.35	0.70	0.68	88 (81)	2 (11)	84 (72)	2 (5)
<40, 10, 0.10, 0.60>	fc+mwo	0.31	1.22	1.23	16 (29)	71 (65)	10 (26)	55 (55)
<40, 10, 0.10, 0.65>	fc+mwo	0.55	1.05	1.03	19 (14)	39 (80)	8 (7)	23 (73)
<40, 10, 0.10, 0.70>	fc+mwo	0.51	0.95	0.99	39 (6)	16 (88)	20 (2)	3 (73)
<40, 10, 0.05, 0.80>	fc+mwo	1.2 e-05	1.02	1.00	19 (32)	36 (44)	14 (20)	22 (28)
<40, 10, 0.05, 0.85>	fc+mwo	1.0 e-04	0.92	0.97	44 (47)	31 (39)	27 (36)	17 (29)
<40, 10, 0.05, 0.90>	fc+mwo	0.13	0.56	0.54	98 (95)	1 (2)	91 (91)	0 (0)
<40, 5, 0.10, 0.48>	fc+mwo	0.023	0.95	0.99	44 (26)	33 (60)	32 (16)	20 (43)
<40, 5, 0.10, 0.52>	fc+mwo	0.47	0.98	0.99	31 (24)	26 (62)	14 (17)	13 (38)
<40, 5, 0.10, 0.56>	fc+mwo	0.42	0.83	0.86	71 (28)	6 (54)	47 (20)	2 (37)
<40, 5, 0.05, 0.62>	fc+mwo	8.5 e-06	1.03	1.00	22 (43)	35 (41)	9 (18)	19 (32)
<40, 5, 0.05, 0.66>	fc+mwo	3.6 e-04	0.91	0.96	48 (56)	27 (32)	30 (30)	16 (22)
<40, 5, 0.05, 0.70>	fc+mwo	0.001	0.93	0.90	54 (52)	25 (34)	33 (44)	17 (29)

Table 5.4- Results of applying the aggregation transformation using the $\alpha 1_{fc}$ evaluation heuristic

5.3.4.4 Conclusions

The results presented in tables 5.2-5.4 show several interesting aspects. In terms of qualitative performance, the use of the aggregation transformation, when combined with the $\alpha 1$ heuristics, resulted in many more *ZDC* formulations being improved in terms of search cost than being made worse. This is the case for the majority of problem classes considered. Furthermore, the $\alpha 1$ heuristics show significant improvements on the use of uninformed aggregation. This is seen in many problem classes such as <40, 5, 0.10, 0.56>, in table 5.4. There, we see that with a 5% margin of significance, only 28% of instances showed an improvement when naive aggregation was used and 54% showed a degradation in performance. This contrasts with the use of $\alpha 1_{fc}$ which resulted in 71% of instances showing an improvement and only 6% showing a degradation.

In general, the qualitative performance of variable aggregation transformation is seen to be more robust when the $\alpha 1$ heuristics are used. This is seen more clearly if we look at the problem classes where fewer numbers of instances show an improvement than degradation in search cost. For the $\alpha 1$ heuristics, these classes are indicated by the cells shaded light grey in the tables. This contrasts significantly with the number of classes where naive aggregation shows poor performance, as indicated by both the light and dark grey cells.[6]

If we consider the actual values of the ratios of search cost, we note that, quantitatively, there are some spectacular gains to made following the transformation. This is particularly the case for backjumping and forward checking. An example is the problem class <20, 5, 0.10, 0.80> where we see gains of 82% for standard backtracking, 770% for backjumping and 1700% for forward checking in the best case. Gains of this magnitude are exceptional, but they are made possible as a result of improved backjumping with $g4_{bj}$ and through the higher level of lookahead with $g1_{fc}$.

While tables 5.2-5.4 show promising results for variable aggregation combined with the $\alpha 1$ heuristics, there are still some problem classes where performance was actually degraded. It is these inaccuracies, indicated by the cells shaded light grey in the tables, that we attempt to reduce in the next section.

5.4 Knowledge Based Evaluation Heuristics

In the previous section we investigated the use of theoretical complexity equations as the basis for evaluation heuristics to be used with the variable aggregation transformation. Our results showed that the heuristics $\alpha 1_{bt}$, $\alpha 1_{bj}$ and $\alpha 1_{fc}$ prove effective in most of the problem classes tested, but that there were several classes where the performance was not as good as we should like. One possible reason for this is that, as we showed in chapter 4, the reliability of theoretical complexity estimates appears to degrade to some degree for problem classes which have low density constraint graphs.

[6] With the single exception of class <40, 5, 0.10, 0.62>

The use of theoretical estimates as the basis for our evaluation heuristics means that each *ZDC* formulation is analysed without any consideration of the similarities between them. The theoretical complexity model treats the *ZDC* formulations, before and after a pair of variables is aggregated, as distinct problem classes.

However, when using a *ZDC* transformation we have the advantage of knowing about the relationship between the input and output *ZDC* formulations. In particular, for the variable aggregation we have detailed knowledge of its effects on search with certain algorithms, as shown in section 5.2. Using a knowledge free approach ignores this potentially useful relationship.

A possible approach to improving on the pure use of theoretical complexity estimates as the basis for evaluation heuristics is to use a modification which only considers the parts of the *ZDC* formulation which have changed. In this section we adopt such a knowledge based approach and develop a second set of evaluation heuristics for use with the variable aggregation transformation we described in figure 5.6. Our new heuristics are still based on the models described in chapter 4, and we continue to use the assumptions A_4.1 - A_4.4. There is a further assumption we make concerning the merging of constraints;

> **Assumption A_5.3**: There are no losses due to constraint merging. This means that we ignore losses $l3_{bt}$, $l3_{bj}$ and $l3_{fc}$.

We believe this assumption is reasonable since any additional cost that is incurred through merging is likely to be small when compared to the other gains and losses. Furthermore, we are intending to develop heuristics which are, by there very nature, approximations. The key is to develop good approximations.

In the remainder of this section we investigate this new approach to see if improvements on the results in section 5.3 can be obtained. Our new evaluation heuristics, $\alpha 2_{bt}$ $\alpha 2_{bj}$ and $\alpha 2_{fc}$, are designed for use with the variable aggregation transformation for standard backtracking, backjumping and forward checking.

5.4.1 Standard Backtracking

We should like our evaluation heuristic for standard backtracking to take as an input the particular *ZDC* formulation being considered and the pair of variables being proposed for aggregation. We call our heuristic $\alpha 2_{bt}$ such that;

$$\alpha 2_{bt}(x_a, x_b) < 1.0 \Rightarrow \text{aggregation of } x_b \text{ and } x_b \text{ is likely to be useful}$$

and

$$\alpha 2_{bt}(x_a, x_b) \geq 1.0 \Rightarrow \text{aggregation of } x_a \text{ and } x_b \text{ is not likely to be useful}$$

We are interested in developing heuristics based on the differences in the two *ZDC* formulations. This means that we are interested in the cost attributable to searching the variables x_a and x_b in the original *ZDC* formulation, f_1, and variable x_{ab} in the transformed *ZDC* formulations, f_2. To achieve this we use modifications of the theoretical estimates described in chapter 4. The first step in the derivation of our expression for $\alpha 2_{bt}$ is therefore to say that ;

$$\alpha 2_{bt}(x_a, x_b) = \frac{Cost_{bt}(f_2, x_{ab})}{Cost_{bt}(f_1, x_a, x_b)} \tag{5-4}$$

where $Cost_{bt}(f_1, x_a, x_b)$ represents the cost in terms of the expected number of checks to exhaustively cover the variables x_a and x_b in the original *ZDC* formulation, f_1. $Cost_{bt}(f_2, x_{ab})$ represents the expected number of checks to exhaustively cover the variable x_{ab} in the transformed *ZDC* formulation, f_2. This is all we need to consider in order to take account of the effects of the variable aggregation since the sub-search space for variables below x_b or x_{ab} are the same, once the effects of constraint merging have been ignored.

In order to determine the two cost functions, we make use of the ideas and notation described in chapter 4. Given a set of previously assigned variables, the expected number of constraint checks against each value of the domain of a variable x_k is equal to the sum of probabilities of checking each constraint with previous assignments against that value. Following the line of (4-8), this gives us;

$$checks\_per\_value(x_k) = \sum_{i=1}^{|G_k|} \prod_{j=1}^{i-1} p_{g_{jk}k} \quad (5\text{-}5)$$

Assuming we attempt to assign all values in the domain of variable x_k, the total cost in expanding the values of that variable is found by multiplying the checks per value by the domain size of the variable;

$$checks\_per\_variable(x_k) = |D_{x_k}| \times \sum_{i=1}^{|G_k|} \prod_{j=1}^{i-1} p_{g_{jk}k} \quad (5\text{-}6)$$

By substituting x_{ab} into equation (5-6) this allows us to express part of the detail of $\alpha 2_{bt}$, namely the expression for $Cost_{bt}(f_2,x_{ab})$. The final part requires us to develop an expression for the expected number of checks for two adjacent variables. Given that variable x_a is followed by x_b, we can say that the expected cost for covering the whole of the sub-search space between those variables is a function of the expected number of checks against variable x_a, the number of successful assignments at x_a and the expected number of checks against variable x_b;

$$Cost_{bt}(f_1,x_a,x_b) = checks\_per\_variable(x_a) + |Compatible_{bt}(x_a)| \times checks\_per\_variable(x_b) \quad (5\text{-}7)$$

where $Compatible_{bt}(x_a)$ gives us the set of compatible values for variable x_a, since the domain of variable x_b can only be expanded for each successfully assigned value of variable x_a. The expected size of this set of values is determined by multiplying the domain size of x_a by the probability of all constraints with all previous assignments being satisfied . This gives us;

$$|Compatible_{bt}(x_a)| = |D_{x_a}| \times \prod_{i=1}^{|G_a|} p_{g_{ia}a} \quad (5\text{-}8)$$

$Cost(f_1,x_a,x_b)$ can now be expressed fully;

$$Cost_{bt}(f_1,x_a,x_b) = \left(|D_{x_a}| \times \sum_{i=1}^{|G_a|} \prod_{j=1}^{i-1} p_{g_{ja}a}\right) + \left(|D_{x_a}| \times \prod_{i=1}^{|G_a|} p_{g_{ia}x} \times |D_{x_b}| \times \sum_{i=1}^{|G_b|} \prod_{j=1}^{i-1} p_{g_{jb}b}\right) \quad (5\text{-}9)$$

By combining equations (5-4), (5-6) and (5-9) we have the final expression $\alpha 2_{bt}$, for a given ordered pair of variables x_a and x_b.

$$\alpha 2_{bt}(x_a, x_b) = \frac{\left|D_{x_{ab}}\right| \times \sum_{i=1}^{|G_{ab}|} \prod_{j=1}^{i-1} p_{g_{jab}ab}}{\left(\left|D_{x_a}\right| \times \sum_{i=1}^{|G_a|} \prod_{j=1}^{i-1} p_{g_{ja}a}\right) + \left(\left|D_{x_a}\right| \times \prod_{i=1}^{|G_a|} p_{g_{ia}a} \times \left|D_{x_b}\right| \times \sum_{i=1}^{|G_b|} \prod_{j=1}^{i-1} p_{g_{jb}b}\right)} \qquad (5\text{-}10)$$

From inspection we can see that equation (5-10) incorporates all of the gains and losses we have described in section 5.2 for the standard backtracking algorithm.

5.4.2 Forward Checking

As with standard backtracking, we should like to develop an evaluation heuristic which takes as its inputs the particular *ZDC* formulation being considered and the pair of variables being proposed for aggregation. In this section we develop an expression which tells us whether or not we can expect an aggregation to be beneficial for forward checking. It is a conservative heuristic providing a sufficient condition which, if satisfied, tells us that we can expect the aggregation to be beneficial. If the heuristic is not sufficiently positive, then no decision is made. We call our heuristic $\alpha 2_{fc}$;

$$\alpha 2_{fc}(x_a, x_b) < 1.0 \Rightarrow \text{aggregation of } x_a \text{ and } x_b \text{ is likely to be useful}$$

and

$$\alpha 2_{fc}(x_a, x_b) \geq 1.0 \Rightarrow \text{usefulness of aggregation of } x_a \text{ and } x_b \text{ is not determined}$$

In contrast to the approach we took for the $\alpha 2_{bt}$ heuristic we define a sufficient condition for accepting the aggregation of a variable pair. Using the knowledge we gained in section 5.2, this can be achieved if we can show that there are effectively no losses, i.e. they are all less than 1.0.

Following assumption A_5.3, we have two significant loss effects for the forward checking algorithm, $l1_{fc}$ and $l2_{fc}$, which can result in losses. The initial, sufficient condition for aggregation using forward checking is;

$$\alpha 2_{fc}(x_a, x_b) = max(l1_{fc}, l2_{fc}) \quad (5\text{-}12)$$

assuming $l1_{fc}$ and $l2_{fc}$ are expressed as relative losses such that, if they are less than 1.0, they actually result in a gain. In words, provided both $l1_{fc}$ and $l2_{fc}$ are less than 1.0 then we should accept the transformation.

Taking the first of these potential losses, $l1_{fc}$, we have already stated that for the average case, there will only be loss if $\left|D_{x_{ab}}\right|$, in the *ZDC* formulation *f2*, is greater than $\left|D_{x_a}\right| + \left|D_{x_b}\right|$. An expression for $l1_{fc}$ is therefore;

$$l1_{fc} = \frac{\left|D_{x_{ab}}\right|}{\left|D_{x_a}\right| + \left|D_{x_b}\right|} \quad (5\text{-}13)$$

It should be noted that this loss is only applicable if there are constraints connecting the aggregated variable to past variables in the search. In a similar way, $l2_{fc}$ only applies when there are constraints between the aggregated variable and future variables.

In order to obtain an expression for the $l2_{fc}$ we make use of some of the ideas discussed in chapter 4. In a similar way to the expression we devised for $\alpha 2_{bt}$, we begin the derivation of $l2_{fc}$ by saying;

$$l2_{fc} = \frac{Cost_{fc}(f_2, x_{ab})}{Cost_{fc}(f_1, x_a, x_b)} \quad (5\text{-}14)$$

where $Cost_{fc}(f_1, x_a, x_b)$ represents the cost to exhaustively cover the variables x_a and x_b in the original *ZDC* formulation, f_1, in terms of the expected number of checks once search has reached this point. $Cost_{bt}(f_2, x_{ab})$ represents the expected number of checks to exhaustively cover the variable x_{ab} in the transformed *ZDC* formulation, f_2.

Following the line of the previous section, the expected number of constraint checks against each value of the domain of a variable at a search level *k*, is equal to the sum of probabilities of each constraint with future variables being checked. From 4-13 we can say;

$$checks\_per\_value(k) = \sum_{i=1}^{|G_k|}\left(\left|D_k^{g_{ik}}\right|\prod_{j=1}^{i-1} S^k_{g_{jk}} \Big/ S^{(k-1)}_{g_{jk}}\right) \tag{5-15}$$

The total cost attributable to the traversing the entire domain of variable x_a is equal to the expected number of remaining values of that variable times the expected checks per value. The expected number of values remaining in the domain x_a, which we call m_a^a, is given by the probability that all constraints with past variables are satisfied multiplied by the original domain size and the probability that all future variables survived at the previous search level. This gives;

$$m_a^a = \left|D_{x_a}\right| \times \prod_{i \in A_{a-1}} p_{ia} \times \left(\prod_{f \in F_{a-1}} S_f^{(a-1)}\right) \tag{5-16}$$

Combining equations (5-15) and (5-16) we obtain an expression for the number of checks per variable;

$$checks\_per\_variable(x_a) = m_a^a \times \sum_{i=1}^{|G_a|}\left(\left|D_a^{g_{ia}}\right|\prod_{j=1}^{i-1} S^a_{g_{ja}} \Big/ S^{(a-1)}_{g_{ja}}\right) \tag{5-17}$$

We can use this expression to find an expression for $Cost_{fc}(f_2, x_{ab})$ by substituting x_a for x_{ab}.

The final part requires us to develop an expression for the expected number of checks for two adjacent variables. Given that variable x_a is followed by x_b, we can say that the expected cost for covering the whole of the sub-search space between those variables is a function of the expected number of checks against variable x_a, the number of successful assignments at x_a and the expected number of checks against variable x_b;

$$\begin{aligned} Cost_{fc}(f_1, x_a, x_b) = {} & checks\_per\_variable(x_a) + \\ & |Compatible_{fc}(x_a)| \times checks\_per\_variable(x_b) \end{aligned} \tag{5-18}$$

where $Compatible_{fc}(x_a)$ gives us the set of compatible values for variable x_a, since the domain of variable x_b can be expanded for each successfully assigned value of variable x_a. The size of this set of values, is found in the same way as m_a^a except this time we need the probability of survival of all of future variables given that the search was at variable x_a. This gives us;

$$Compatible_{fc}(x_a) = \left|D_{x_a}\right| \times \prod_{i=A_{a-1}} p_{ia} \times \left(\prod_{f=F_a} S_f^a \right) \tag{5-19}$$

Combining equations (5-16)-(5-19) we obtain;

$$Cost_{fc}(f_1,x_a,x_b) = checks\_per\_variable(x_a) + Compatible_{fc}(x_a) \times checks\_per\_variable(x_b) \tag{5-20}$$

Given the series of equations outlined above, we can now fully determine the value of our formulation heuristic $\alpha 2_{fc}(f_1,x_a,x_b)$ for any given pair of adjacent variables for the forward checking algorithm. We expect this to be a conservative estimate because it only considers the losses incurred by forward checking, when a pair of variables is merged.

5.4.3 Backjumping

In Section 5.2 we saw that the gains and losses for backjumping are similar to those experienced by standard backtracking, with the exception of $g4_{bj}$ and $l4_{bj}$, which are related to culprit detection. As a result we adopt a similar approach for designing an evaluation heuristic $\alpha 2_{bt}$. We start by stating that;

$$\alpha 2_{bj}(x_a, x_b) < 1.0 \Rightarrow \text{aggregation of } x_a \text{ and } x_b \text{ is likely to be useful}$$

and

$$\alpha 2_{bj}(x_a, x_b) \geq 1.0 \Rightarrow \text{aggregation of } x_a \text{ and } x_b \text{ is not likely to be useful}$$

Considering the losses due to $l4_{bj}$, we notice that the effect is likely to be relatively small due to two factors. First, any loss only involves the difference in one level of search since $l4_{bj}$ represents the effects of jumping back one level less than would otherwise been the case. The magnitude of any losses are also further reduced by the reduction in the "ab" space. The second reason is that

its effect is conditional on search progressing to levels deeper in the search than that of the aggregation level. As a result, for the purposes of this heuristic, we ignore effects due to $l4_{bj}$.

In contrast to $l4_{bj}$, $g4_{bj}$ can have more significant effects because it can result in jumps across many levels of search. We should therefore consider its effect as part of $\alpha 2_{bj}$. Our approach is to consider the effect at the aggregation level of search only. Analysis of the estimate for backjumping as a whole will show that in fact these jumping effects can affect the number of checks performed at other past search levels. However, for the purposes of our heuristic, provided we include $g4_{bj}$ effects for the aggregation level, we should obtain a good indication of the effects of the aggregation on the backjumping mechanism, and hence the expected search cost. Ignoring the effects on previous levels makes our heuristic more conservative in nature.

The first step in the derivation of our expression for $\alpha 2_{bj}$ is to say that ;

$$\alpha 2_{bj}(x_a, x_b) = \frac{Cost_{bj}(f_2, x_{ab})}{Cost_{bj}(f_1, x_a, x_b)} \tag{5-21}$$

where $Cost_{bj}(f_1, x_a, x_b)$ represents the cost in terms of the expected number of checks to exhaustively cover search over the variables x_a and x_b in the original *ZDC* formulation, f_1. $Cost_{bj}(f_2, x_{ab})$ represents the expected number of checks to exhaustively cover the variable x_{ab} in the transformed *ZDC* formulation, f_2. Unlike the situation with standard backtracking, in order to include the effects of $g4_{bj}$, we need to include the effective domain sizes of previous variables. We therefore use (4-20) as the basis of our calculation of the cost functions. We have;

$$Cost_{bj}(f_1, x_a, x_b) = n(bj, a) \times c(bj, a) + n(bj, b) \times c(bj, b) \tag{5-22}$$

and

$$Cost_{bj}(f_2, x_{ab}) = n(bj, ab) \times c(bj, ab) \tag{5-23}$$

where $n(bj, k)$ and $c(bj, k)$ are taken from chapter 4. We can then use (4-22) and (4-23) to calculate (5-21).

5.4.4 Effectiveness of Evaluation Heuristics $\alpha 2_{bt}$, $\alpha 2_{bj}$ and $\alpha 2_{fc}$

Having devised a second set of evaluation heuristics for use with the variable aggregation transformation we performed a similar evaluation to the one we used for heuristics $\alpha 1_{bt}$, $\alpha 1_{bj}$ and $\alpha 1_{fc}$. In addition we carried out a further evaluation of our heuristics using 3-colouring problems and the *8*-Queens problem.

5.4.4.1 Randomly Generated Binary CSPs

In order to assess the effectiveness of our second set of *ZDC* formulation evaluation heuristics we used the same method and problem classes described in section 5.3. Our results for the three evaluation heuristics are given in tables 5.5 - 5.7.

The results in tables 5.5-5.7 show that the combination of knowledge of the aggregation transformation and the restricted use of theoretical complexity estimates improves the reliability of our evaluation heuristics in terms of the number of classes where more *ZDC* formulations are improved than degraded. This is particularly striking for the backjumping results where there is only one class showing this undesirable effect, <40, 10, 0.05, 0.80>, as indicated by the light grey cells in table 5.6. This gives us further confidence in the correctness of the model we have developed for backjumping in chapter 4.

Similar results are seen for forward checking, with only two problem classes showing more instances where a degradation in performance was observed - <20, 5, 0.30, 0.40> and <20, 5, 0.30, 0.45>. Note also that the conservative nature of $\alpha 2_{fc}$ results in an increase in the number of occasions where no benefit is seen. These cases are indicated in bold type in table 5.7 and are due to the domination of the $l1_{fc}$ term. In general, however, we see that the $\alpha 2$ heuristics perform much more robustly when compared with the naive variable aggregation transformation.

If we look at the quantitative results, we also see a general improvement when compared with the $\alpha 1$ evaluation heuristics used in section 5.3. For example, comparing the results in tables 5.3 and 5.7, if we look at the median performance for standard backtracking, the ratio of *cc(AG2)* and *cc(AG1)* is consistently lower for the $\alpha 2$ case. This means we can expect a greater gain when using the $\alpha 2$ heuristics, for the average case. Similar results are seen for backjumping and forward checking.

		$\frac{cc(AG2)}{cc(AG1)}$			Accuracy with 5% margin (%instances)		Accuracy with 15% margin (%instances)	
Class	Algorithm +Heuristic	min.	mean	med.	$\frac{cc(AG2)}{cc(AG1)} \leq 0.95$	$\frac{cc(AG2)}{cc(AG1)} \geq 1.05$	$\frac{cc(AG2)}{cc(AG1)} \leq 0.85$	$\frac{cc(AG2)}{cc(AG1)} \geq 1.15$
<20, 5, 0.10, 0.70>	bt+nat	0.21	0.89	0.99	39 (40)	0 (9)	25 (27)	0 (3)
<20, 5, 0.10, 0.75>	bt+nat	0.25	0.86	0.95	51 (52)	0 (2)	35 (34)	0 (0)
<20, 5, 0.10, 0.80>	bt+nat	0.21	0.79	0.88	59 (59)	0 (1)	46 (46)	0 (1)
<20, 5, 0.30, 0.35>	bt+nat	0.70	0.95	0.97	39 (13)	0 (71)	8 (2)	0 (62)
<20, 5, 0.30, 0.40>	bt+nat	0.56	0.92	0.94	56 (15)	0 (64)	13 (3)	0 (43)
<20, 5, 0.30, 0.45>	bt+nat	0.51	0.88	0.90	71 (33)	0 (44)	33 (17)	0 (26)
<20, 10, 0.10, 0.80>	bt+nat	0.18	0.89	1.00	33 (28)	1 (2)	23 (15)	0 (1)

Table 5.5 - Results of applying the aggregation transformation using the $\alpha 2_{bt}$ evaluation heuristic

		$\frac{cc(AG2)}{cc(AG1)}$			Accuracy with 5% margin (%instances)		Accuracy with 15% margin (%instances)	
Class	Algorithm +Heuristic	min.	mean	med.	$\frac{cc(AG2)}{cc(AG1)} \leq 0.95$	$\frac{cc(AG2)}{cc(AG1)} \geq 1.05$	$\frac{cc(AG2)}{cc(AG1)} \leq 0.85$	$\frac{cc(AG2)}{cc(AG1)} \geq 1.15$
<20, 5, 0.10, 0.70>	bj+nat	0.013	0.79	0.94	54 (53)	4 (8)	42 (46)	3 (3)
<20, 5, 0.10, 0.75>	bj+nat	0.005	0.68	0.80	62 (57)	0 (5)	52 (49)	0 (2)
<20, 5, 0.10, 0.80>	bj+nat	0.0010	0.59	0.62	68 (61)	1 (2)	62 (43)	0 (0)
<20, 5, 0.30, 0.35>	bj+nat	0.34	0.92	0.97	48 (39)	14 (41)	24 (20)	8 (31)
<20, 5, 0.30, 0.40>	bj+nat	0.32	0.92	0.96	47 (32)	13 (50)	22 (22)	5 (37)
<20, 5, 0.30, 0.45>	bj+nat	0.18	0.85	0.87	67 (49)	10 (30)	46 (31)	5 (19)
<20, 10, 0.10, 0.80>	bj+nat	1.3 e-04	0.77	0.99	44 (25)	9 (6)	34 (20)	8 (5)
<30, 10, 0.10, 0.70>	bj+mwo	0.78	0.86	0.90	67 (33)	12 (54)	37 (20)	5 (35)
<30, 10, 0.10, 0.75>	bj+mwo	0.03	0.82	0.83	79 (48)	6 (36)	54 (30)	2 (24)
<30, 10, 0.10, 0.80>	bj+mwo	0.49	0.71	0.71	99 (79)	0 (10)	93 (58)	0 (4)
<30, 10, 0.10, 0.85>	bj+mwo	0.45	0.62	0.62	100 (96)	0 (1)	99 (85)	0 (1)
<40, 10, 0.10, 0.60>	bj+mwo	0.18	0.98	0.99	19 (15)	15 (82)	8 (13)	6 (75)
<40, 10, 0.10, 0.65>	bj+mwo	0.48	0.95	0.96	46 (4)	13 (96)	13 (2)	0 (87)
<40, 10, 0.10, 0.70>	bj+mwo	0.50	0.89	0.90	79 (2)	3 (88)	25 (1)	0 (73)
<40, 10, 0.05, 0.80>	bj+mwo	8.0 e-06	1.02	1.01	39 (52)	36 (32)	23 (27)	30 (26)
<40, 10, 0.05, 0.85>	bj+mwo	3.6 e-05	0.73	0.80	71 (72)	25 (18)	59 (56)	20 (10)
<40, 10, 0.05, 0.90>	bj+mwo	0.12	0.42	0.43	100 (98)	0 (1)	100 (98)	0 (1)
<40, 5, 0.10, 0.48>	bj+mwo	0.014	0.90	0.96	43 (20)	7 (66)	19 (13)	1 (48)
<40, 5, 0.10, 0.52>	bj+mwo	0.38	0.90	0.91	62 (34)	6 (49)	33 (24)	3 (25)
<40, 5, 0.10, 0.56>	bj+mwo	0.42	0.77	0.78	92 (54)	2 (27)	73 (24)	0 (11)
<40, 5, 0.05, 0.62>	bj+mwo	1.7 e-06	0.96	0.91	60 (65)	27 (27)	36 (44)	22 (21)
<40, 5, 0.05, 0.66>	bj+mwo	1.4 e-05	0.69	0.79	74 (75)	17 (19)	57 (54)	10 (14)
<40, 5, 0.05, 0.70>	bj+mwo	1.5 e-04	0.65	0.71	84 (80)	12 (8)	75 (69)	9 (5)

Table 5.6 - Results of applying the aggregation transformation using the $\alpha 2_{bj}$ evaluation heuristic

		$\frac{cc(AG2)}{cc(AG1)}$			Accuracy with 5% margin (%instances)		Accuracy with 15% margin (%instances)	
Class	Algorithm +Heuristic	min.	mean	med.	$\frac{cc(AG2)}{cc(AG1)} \leq 0.95$	$\frac{cc(AG2)}{cc(AG1)} \geq 1.05$	$\frac{cc(AG2)}{cc(AG1)} \leq 0.85$	$\frac{cc(AG2)}{cc(AG1)} \geq 1.15$
<20, 5, 0.10, 0.70>	fc+nat	0.0026	0.77	0.88	54 (57)	5 (6)	46 (42)	0 (4)
<20, 5, 0.10, 0.75>	fc+nat	0.012	0.76	0.90	53 (58)	4 (5)	44 (49)	0 (4)
<20, 5, 0.10, 0.80>	fc+nat	0.096	0.79	0.92	55 (66)	0 (2)	42 (54)	0 (1)
<20, 5, 0.30, 0.35>	fc+nat	0.88	1.00	1.00	1 (35)	1 (47)	0 (22)	0 (34)
<20, 5, 0.30, 0.40>	fc+nat	0.79	1.00	1.00	4 (27)	7 (54)	2 (17)	4 (37)
<20, 5, 0.30, 0.45>	fc+nat	0.70	1.00	1.00	3 (33)	13 (51)	1 (24)	6 (37)
<20, 10, 0.10, 0.80>	fc+nat	6.0 e-04	0.76	0.96	49 (40)	12 (6)	45 (35)	2 (1)
<30, 10, 0.10, 0.70>	fc+mwo	1.00	1.00	1.00	**0 (36)**	**0 (54)**	**0 (25)**	**0 (43)**
<30, 10, 0.10, 0.75>	fc+mwo	1.00	1.00	1.00	**0 (32)**	**0 (52)**	**0 (27)**	**0 (33)**
<30, 10, 0.10, 0.80>	fc+mwo	0.32	0.88	0.87	63 (51)	22 (34)	44 (31)	11 (23)
<30, 10, 0.10, 0.85>	fc+mwo	0.35	0.72	0.69	86 (81)	12 (11)	72 (72)	2 (5)
<40, 10, 0.10, 0.60>	fc+mwo	1.00	1.00	1.00	0 (29)	0 (65)	0 (26)	0 (55)
<40, 10, 0.10, 0.65>	fc+mwo	1.00	1.00	1.00	0 (14)	0 (80)	0 (7)	0 (73)
<40, 10, 0.10, 0.70>	fc+mwo	1.00	1.00	1.00	0 (6)	0 (88)	0 (2)	0 (73)
<40, 10, 0.05, 0.80>	fc+mwo	1.0 e-05	0.92	0.98	30 (32)	16 (44)	16 (20)	8 (28)
<40, 10, 0.05, 0.85>	fc+mwo	9.6 e-05	0.83	0.93	54 (47)	15 (39)	34 (36)	0 (29)
<40, 10, 0.05, 0.90>	fc+mwo	0.13	0.58	0.59	97 (95)	1 (2)	94 (91)	0 (0)
<40, 5, 0.10, 0.48>	fc+mwo	1.00	1.00	1.00	**0 (26)**	**0 (60)**	**0 (16)**	**0 (43)**
<40, 5, 0.10, 0.52>	fc+mwo	1.00	1.00	1.00	**0 (24)**	**0 (62)**	**0 (17)**	**0 (38)**
<40, 5, 0.10, 0.56>	fc+mwo	0.54	0.98	0.96	47 (28)	40 (54)	34 (20)	28 (37)
<40, 5, 0.05, 0.62>	fc+mwo	8.0 e-06	0.96	0.98	29 (43)	20 (41)	8 (18)	10 (32)
<40, 5, 0.05, 0.66>	fc+mwo	3.6 e-04	0.81	0.93	55 (56)	10 (32)	31 (30)	3 (22)
<40, 5, 0.05, 0.70>	fc+mwo	6.0 e-04	0.83	0.92	61 (52)	10 (34)	32 (44)	3 (29)

Table 5.7- Results of applying the aggregation transformation using the $\alpha 2_{fc}$ evaluation heuristic

5.4.4.2 3-Colouring Problems

Another commonly encountered class of problem is the graph colouring problem, which we also used in chapter 4. These are important problem classes as some real world problems can be mapped to graph colouring. A sub-class of this is the 3-colouring problem. We investigated the validity of our new evaluation heuristics on a sample of 100 instances of the same 3-colouring problem class used in chapter 4, i.e. <50, 3, 4.5>. As with the experiment in the previous section, we used standard backtracking, backjumping and forward checking and we combined them with the minimum width variable ordering. Our results are given in table 5.8.

In terms of the qualitative performance of our heuristics, the results in table 5.8 show how variable aggregation combined with our new evaluation heuristics shows significant improvements in

search costs for all three algorithms, for the class of 3-colouring problem used. If we look at the figures in columns 6, 7, 8 and 9, all of the α heuristics satisfy the 50% selection criterion described in chapter 3.

In terms of the quantitative performance, all of our heuristics display good best case gains of greater than 50%, as can be seen if we look at column 2. Furthermore, in all cases the average gain in search cost obtained is greater than 10%, as seen by the results in column 3.

These results are encouraging because they provide another example of how the theoretical complexity approach can be useful for *ZDC* formulation selection, when applied in the way we have described in this chapter, even on 3-colouring problems which have extremely low density constraint graphs.

		$\frac{cc(AG2)}{cc(AG1)}$			Accuracy with 5% margin (%instances)		Accuracy with 15% margin (%instances)	
H_e	Algorithm+ Heuristic	min.	mean	med.	$\frac{cc(AG2)}{cc(AG1)} \leq 0.95$	$\frac{cc(AG2)}{cc(AG1)} \geq 1.05$	$\frac{cc(AG2)}{cc(AG1)} \leq 0.85$	$\frac{cc(AG2)}{cc(AG1)} \geq 1.15$
$\alpha 1_{bt}$	bt+mwo	0.48	0.90	0.95	52	0	17	0
$\alpha 2_{bt}$	bt+mwo	0.48	0.89	0.93	66	0	20	0
$\alpha 1_{bj}$	bj+mwo	0.47	0.87	0.93	60	0	27	0
$\alpha 2_{bj}$	bj+mwo	0.21	0.87	0.91	68	1	34	0
$\alpha 1_{fc}$	fc+mwo	0.19	0.86	0.94	55	2	27	1
$\alpha 2_{fc}$	fc+mwo	0.19	0.87	0.95	46	31	26	6

Table 5.8 - Results of applying the aggregation transformation to 3-colouring problems

5.4.4.3 The *8*-Queens Problem

As a further test of the effectiveness of variable aggregation combined with our new evaluation heuristics, we investigated the application of the transformation to the *8*-Queens problem. Table 5.9 shows the result of using our more informed application of the aggregation transformation. The results are for the *8*-Queens problem before and after application of the transformation described in figure 5.6.

H_e	Algorithm+ Heuristic	Checks in original *ZDC* formulation	Checks in transformed *ZDC* formulation
$\alpha 1_{bt}$	bt+nat	2438	1810
$\alpha 2_{bt}$	bt+nat	2438	1476
$\alpha 1_{bj}$	bj+nat	2157	1306
$\alpha 2_{bj}$	bj+nat	2157	1306
$\alpha 1_{fc}$	fc+nat	803	760
$\alpha 2_{fc}$	fc+nat	803	803

Table 5.9 - Results of applying the aggregation transformation to the 8-Queens problem

The results in table 5.9 show the search cost required to solve the *8*-Queens problem in terms of the number of constraint checks. Each of our three search algorithms was used to solve the problem before and after the application of the aggregation transformation. The results show that our evaluation heuristics ensure no degradation in performance. In fact, the use of our evaluation heuristics has resulted in improvements in search cost in all but the last row. This compares well with the naive variable aggregation which we used in chapter 2. In that case, a degradation in search cost was seen for all three algorithms.

One interesting feature of these results is the last row, for $\alpha 2_{fc}$. No aggregation takes place using this heuristic and hence no loss or gain in search cost is seen. The reason for this is the very loose constraints in the 8-Queens problem tend to mean that losses due to $l1_{fc}$ dominate by consistently giving a value greater than 1.0, hence pushing $\alpha 2_{fc}$ to a value greater than 1.0. This suggests that the usefulness of $\alpha 2_{fc}$ may be improved by combining it with other measures or properties of CSPs.

5.5 Discussion

The variable aggregation transformation is an important *ZDC* transformation algorithm which presents us with an opportunity for significant gains in search cost. The work in this chapter has been concerned with improving the effectiveness of this transformation by incorporating sophisticated *ZDC* formulation evaluation heuristics into it. Our approach has been to build on the ideas presented in chapter 4, using theoretical complexity models. We have developed heuristics

based on the theoretical comparison of different *ZDC* formulations. We have also used knowledge, specific to the variable aggregation transformation, as a basis for refining our heuristics.

5.5.1 Robustness of Variable Aggregation

One of the problems with uninformed variable aggregation is that, while there are many problem classes where large gains can be made through its application, there are some problem classes where it can result in a degradation in search cost. This phenomenon was seen in table 5.2-5.7 with problem classes where the cells are shaded grey such as problem class <20, 5, 0.30, 0.45> in table 5.5. By using more sophisticated *ZDC* formulation evaluation heuristics, such as the α heuristics we have developed in this chapter, we have improved the robustness and scope of the transformation, reducing the number of problem classes where bad formulations are generated. Our new approach automatically adapts to the problem class being considered.

Further improvements in the performance of the variable aggregation transformation may be possible if further refinements are made to the α heuristics. For example, further properties of CSPs could be incorporated into the measure vector, as we described in chapter 3.

5.5.2 Improved Mean Performance

The $\alpha 2$ heuristics showed a general improvement, in terms of the mean gain in search cost, relative to the $\alpha 1$ heuristics. This suggests that the use of theoretical estimates based on the local changes to *ZDC* formulations can lead to an improvement in the accuracy of theoretical estimates. As a result, we believe the type of analysis and approach used in this chapter could be applicable to other *ZDC* transformation functions.

5.5.3 Exceptionally Hard Problems

From the results in tables 5.2-5.7, when the maximum ratio of search cost is considered, we see savings in search cost of several orders of magnitude are occasionally seen. For example, the minimum ration of *cc(AG2)* and *cc(AG1)* for problem class <40, 10, 0.05, 0.80> in table 5.6 where the difference is six orders of magnitude. This relates to the phenomenon of *exceptionally hard problems* or EHPs (Smith 1994) (Smith & Grant 1996). EHPs are outliers in a class of problems which show search costs several orders of magnitude greater than the median. The use

of the variable aggregation transformation has clearly resulted in some cases where search costs were reduced by several orders of magnitude and we conjecture that the use of the transformation can result in improved performance of some algorithms in terms of the frequency of EHPs.

Avoiding this exceptional behaviour is extremely desirable. The intelligent application of the variable aggregation transformation presents us with an opportunity to reduce their impact, especially for the forward checking and backjumping algorithms.

5.5.4 Theoretical Complexity Model for Backjumping

The performance of the $\alpha 1_{bj}$ and $\alpha 2_{bj}$ heuristics was comparable with the corresponding heuristics for standard backtracking and forward checking. These results give us further confidence in the correctness of the theoretical complexity model we developed for backjumping in chapter 4.

5.5.5 Applicability to Cycle Cutset Method

An interesting observation relating to the variable aggregation transformation in particular concerns the cycle cutset method (Dechter & Pearl 1987). The cycle cutset method is an approach to CSP solving which looks to modify the structure of an original *ZDC* formulation such that we eventually obtain a formulation whose constraint graph is a tree. One part of the method is to join groups of variables. This is done in a naive fashion, without regard to the effects of joining variables that were described in section 5.2. Furthermore, it is often the case that there are several alternative choices that can be made during the problem transformation process, regarding the members of such groups of variables. It is possible that evaluation heuristics similar to the ones developed in this chapter could be used as method for improving the effectiveness of the cycle cutset method.

5.6 Summary

The main contribution of this chapter is the development of an important new set of *ZDC* formulation evaluation heuristics for use with the variable aggregation transformation. This represents a new direction in constraint satisfaction research and the principles we have used to improve the performance of variable aggregation should be applicable to other *ZDC* transformations. Our work also demonstrates how the elements of our context for *ZDC*

formulation selection can be combined and our results show much promise in this area. In particular we have developed;

- a *ZDC* formulation suggestion heuristic, H_s, based on the idea that reducing the search space complexity is a good thing
- a move operator based on the variable aggregation transformation
- a *ZDC* formulation evaluation heuristic, H_e, based significant extensions to the theoretical complexity theory of chapter 4

Our evaluation heuristics were applied to the standard backtracking, backjumping and forward checking algorithms. Results from the experiments we have performed support our claim that theoretical complexity estimates have a significant role to play in the development of *ZDC* formulation evaluation heuristics as large savings in search costs were seen for some classes of problem. The results for backjumping also give additional evidence for the correctness of the new theoretical complexity model which we developed for that algorithm.