Search in Weighted Constraint Satisfaction Problems

by

Daniel Peter Black

Submitted in accordance with the requirements
for the degree of Doctor of Philosophy.

The University of Leeds
School of Computing

June 2003

Abstract

A wide variety of real-world optimisation problems can be modelled as Weighted Constraint Satisfaction Problems (WCSPs). Such problems are NP-hard and require an exponential amount of time to find the optimal solution.

This thesis concentrates on the University Examination Timetabling Problem. A general abstraction of this problem has been used, as there are many institution-specific rules which could be incorporated into the problem. The use of this problem type allows WCSPs to be investigated using realistic problem data and allows a comparison with previously published results for the problem instances used.

We have examined some existing variable ordering heuristics and defined new ones. An analysis methodology has been defined that allows the characteristics of "good" solutions to be identified. Different methods of identifying difficult to solve sub-problems and the use of such methods in variable ordering has been investigated. Incorporating the weight, or preference, associated with constraints into variable ordering heuristics has been found to be beneficial to finding solutions of low cost. The analysis methodology has been used to examine the relationship between solutions of different quality and the knowledge derived has been used to define, and justify, two new variable ordering heuristics.

The usefulness of different value ordering heuristics has been examined. Value selection on the error incurred with past assignments and the use of "look-ahead" have been investigated. Variable ordering heuristics have been extended to try and exploit the advantages of such value ordering heuristics. The use of stochasticity with such orderings has been investigated and has led to a new class of hybrid value ordering heuristics being defined.

Finally two hybrid search algorithms have been defined that attempt to concentrate search upon the sections of the problem instance which have the largest effect upon the overall quality of solutions found. Such methods are shown to be at least competitive with standard tree based search techniques.

Acknowledgements

I would like to thank my supervisors Professor Barbara Smith and Professor Roger Boyle for helping me throughout my time at Leeds and doing the right thing when things didn't look too good. I would like to thank my family who have put up with me as I've gone through thick and thin. Special thanks goes to the Fenton Academicals and everyone else who has made Leeds a home away from home. Also, to my two best friends during my time in Leeds, Ullli who always made me smile, and Neill, who was always up for a pint and a chat. Finally, thanks goes to those people who have help me out in one way or another during my PhD...

Contents

List of Figures

List of Tables

Glossary

Variable Ordering

- **DEG** - recursively, the variable with the largest *degree* is ordered next; then the variable with the next largest degree; and so on. Tie breaks are made arbitrarily.
- **FD** - the variable with the largest *forward degree* is placed next in the variable ordering. Tie breaks are made arbitrarily.
- **VAR** - the variable with the largest weight is ordered next. Variable weight is defined as the number of students who are required to sit the corresponding exam.
- **CLQ-DEG** - the variables of a *maximum clique* is placed at the start of the variable ordering (ordered arbitrarily). The remaining variables are ordered according to **DEG**.
- **CLQ-FD** - the variables of a *maximum clique* is placed at the start of the variable ordering (ordered arbitrarily). The remaining variables are ordered according to **FD**.
- **CLQ-VAR** - the variables of a *maximum clique* is placed at the start of the variable ordering (ordered arbitrarily). The remaining variables are ordered according to **VAR**.
- **WD** - the variable with the largest *weighted degree* is ordered next.
- **WFD** - the variable with the largest *weighted forward degree* is ordered next.
- **WCLQ-DEG** - the variables of a *maximum weighted clique* is placed at the start of the variable ordering (ordered arbitrarily). The remaining variables are ordered according to **DEG**.

- **WCLQ-FD** - the variables of a *maximum weighted clique* is placed at the start of the variable ordering (ordered arbitrarily). The remaining variables are ordered according to **FD**.
- **WCLQ-VAR** - the variables of a *maximum weighted clique* is placed at the start of the variable ordering (ordered arbitrarily). The remaining variables are ordered according to **VAR**.
- **MV** - variables are ordered using a *weighted clique partition* with the variables of the heaviest clique being ordered first. The clique weight is defined as the sum of the variable weights of clique variables.
- **ME** - variables are ordered using a *weighted clique partition* with the variables of the heaviest clique being ordered first. The clique weight is defined as the sum of the edge weights of the clique edges.
- **BD** - the variable with the largest *backward degree* is ordered next. Tie breaks are made according to the variable with the largest *degree*.
- **WBD** - the variable with the largest *weighted backward degree* is ordered next. Tie breaks are made according to the variable with the largest *weighted degree*.
- **FF** - the variable with the smallest *domain size* is chosen next. Tie breaks are made arbitrarily.
- **BZ** - the variable with the smallest *domain size* is chosen next. Tie breaks are made according to largest *degree*.
- **DD** - the variable with the smallest ratio between *domain size* and *degree* is chosen next. The *domain size* is divided by the *degree* and the variable with the smallest such value is chosen.
- **BD-NTB** - the variable with the largest *backward degree* is chosen next. Tie breaks are made arbitrarily.
- **BD-DTB** - the variable with the largest *backward degree* is chosen next. Tie breaks are made according to the largest *degree*.
- **BD-D** - the variable with the smallest ration between *backward degree* and *degree* is chosen next. The *backward degree* is divided by the *degree* and the variable with the smallest such value is chosen.

- **WSD** - the variable with the highest *weighted saturation degree* is chosen next. Tire breaks are made arbitrarily.
- **WSD-WD** - the variable with the highest ratio between *weighted saturation degree* and *weighted degree* is chosen next. The *weighted saturation degree* is divided by the *weighted degree* and the variable with the highest such value is chosen.
- **WBD-WD** - the variable with the highest ratio between *weighted backward degree* and *weighted degree* is chosen next. The *weighted backward degree* is divided by the *weighted degree* and the variable with the highest such value is chosen.
- **ELA** - variables are ordered to try and exploit the nature of *look-ahead* algorithms. The constraints are considered in decreasing weight order, with the *heaviest* constraint first. For each constraint in order, if none of the constraint variables have been placed in the current variable ordering, then the heaviest constraint variable is placed next in the static ordering.

Value Orderings

- **LOW** - the value assigned to the current variable is the lowest value available (i.e. lowest value which satisfies the hard constraints).
- **RND** - the value assigned to the current variable is chosen randomly and uniformly from the set of values available (i.e. lowest value which satisfies the hard constraints).
- **ERR-LOW** - the value assigned to the current variable is that which will incur the least error with previous assignments. Tie breaks are made on the lowest value.
- **ERR-HIGH** - the value assigned to the current variable is that which will incur the least error with previous assignments. Tie breaks are made on the highest value.
- **ERR-RND** - the value assigned to the current variable is that which will incur the least error with previous assignments. Tie breaks are made randomly.

- **ERR-HYD** - the value assigned to the current variable is chosen using either of the **ERR-LOW**, **ERR-HIGH** or **ERR-RND** heuristics. Which is defined according to the current variable and varies across the variable ordering.
- **IC-LOW** - the value assigned to the current variable is that which will incur the least error with previous assignments combined with the error that is guaranteed to occur in future assignments as defined using *look-ahead*. Tie breaks are made on the lowest value.
- **IC-HIGH** - the value assigned to the current variable is that which will incur the least error with previous assignments combined with the error that is guaranteed to occur in future assignments as defined using *look-ahead*. Tie breaks are made on the highest value.
- **IC-RND** - the value assigned to the current variable is that which will incur the least error with previous assignments combined with the error that is guaranteed to occur in future assignments as defined using *look-ahead*. Tie breaks are made randomly.

Hybrid Search Algorithms

- **HYD-BIG** - a limited complete search is performed on the *critical constraints* using the **ERR-LOW** value ordering. The best solution found by this search is then extended a set number of times using the **ERR-RND** value ordering and a defined variables ordering.
- **HYD-TEN** - a limited complete search is performed on the *critical constraints* using the **ERR-LOW** value ordering. A set of best solutions found by this search are then each extended a set number of times using the **ERR-RND** value ordering and a defined variable ordering.
- **HYD-WBD** - the search procedure follows that of **HYD-TEN**; however, the limited complete search is performed on the set of variables ordered first by the **WBD** variable ordering.

Terms

- **Backward Degree** - the backward degree of a variable is the number of previous variables in the ordering which occur in a constraint with the given

variable.

- **Critical Constraints** - the critical constraints are those which have the biggest impact upon solution quality.
- **Degree** - the degree of a variable is the number of variables which occur in a constraint with the given variable.
- **Domain Size** - the domain size of a variable is the number of consistent assignments which can currently be made to the given variable.
- **Forward Degree** - the forward degree of a variable is the number of future variables in the ordering which occur in a constraint with the given variable.
- **Look-ahead** - look-ahead is a process where the current assignments to variables are propagated. Whether possible future assignments conflict with current assignments and the error that would be incurred between future and current assignments are considered.
- **Maximum Clique** - the maximum clique of a graph is the largest sub-graph that forms a clique, where every vertex is connected to every other vertex.
- **Maximum Weighted Clique** - the maximum weighted clique of a graph is the largest weighted sub-graph that forms a clique, where every vertex is connected to every other vertex and sub-graph weight is defined as the sum of all edge weights in the sub-graph.
- **Weighted Backward Degree** - the weighted backward degree is the sum of the weights associated with edges or constraints which include the given variable and at least one variable which precedes it in the variable ordering.
- **Weighted Clique Partition** - a weighted clique partition breaks the problem graph into sub-graphs each of which forms a clique, where every vertex is connected to every other vertex. Each sub-graph or clique is associated with a weight, defined by either the sum of sub-problem vertex weights or by the sum of sub-problem edge weights.
- **Weighted Degree** - the weighted degree of a variable is the sum of the weights associated with edges or constraints which include the given variable.

- **Weighted Saturation Degree** - the weighted saturation degree of a variable considers the weights of constraints which prune the domain of the given variable. As several constraints can prune the same value, of the given variable, the maximum such weight is recorded and the sum of such maximum weights for each pruned value is the weighted saturation degree.

Chapter 1

Constraint Satisfaction

1.1 What is Constraint Satisfaction

Constraint Satisfaction consists of the set of problems where values must be assigned to a set of variables such that a set of rules, or **constraints**, are obeyed, or **satisfied**. Such problems can take many forms such as colouring every country on a map where neighbouring countries are different colours. Another example are Enigma Puzzles. These take the form of a word sum, e.g. $SEND + MORE = MONEY$, where each letter corresponds to a single digit (a non-zero number in the case of the leading letters S and M). The aim of the problem is to assign a different number to each letter such that the sum is correct. In the example given the assignments $\{(S = 9), (E = 5), (N = 6), (D = 7), (M = 1), (O = 0), (R = 8), (Y = 2)\}$ satisfy the sum because $9567 + 1085 = 10652$.

1.1.1 The N-queens Problem

The N-queens problem is often used to introduce Constraint Satisfaction Problems(CSPs) as most people are familiar with the queen piece of the chess set. The problem consists of placing N queens onto a chess board of size $N \times N$. The constraints of the problem specify that no two queens can take each other. In chess the queen piece can move any number of squares in one particular direction at a time. This direction can be horizontal, vertical or diagonal. In order to satisfy the problem constraints, or rules, no two queens can be placed on the same horizontal, vertical or diagonal line. The aim of the problem is to place all N queens on the board such that the constraints are satisfied. A solution to the 4-queens problem is given in Figure 1.1.

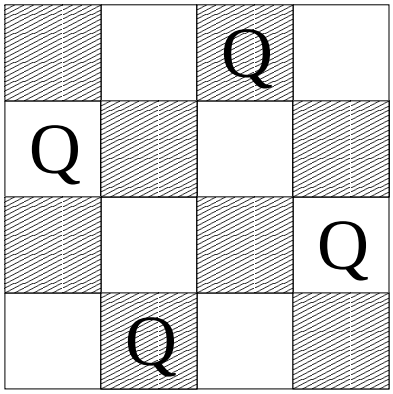

Figure 1.1: A solution to the 4-queens problem.

1.1.2 A Formal Definition

Earlier in this chapter a CSP was defined as the problem of assigning values to a set of variables such that all of the problem constraints are satisfied. A more formal definition of a Classical CSP is as a tuple $< V, D, C >$. The set V describes a set of n problem variables ($V = v_1, ..., v_n$). The superset D is defined as the set of possible variable assignments to each variable ($D = D_1 \cap D_2 \cap ... \cap D_n$) D_i is the set of possible assignments to, or the **domain** of, variable v_i(i.e. $D_i = l_1, ..., l_m$) with each l_j being a possible label to the variable and the number of possible labels, or the **domain size**, being m. A constraint, in the set of constraints C, is a set of **no-goods**. A **no-good** is a tuple of assignments that cannot be made and is of the form ($v_1 = l_2$, $v_3 = l_2$) which states that variable 1 cannot be assigned label 2 if variable 3 has been assigned label 2, and vice versa. No-goods can consist of several violating assignments, the number of which defines the **arity** of, or the number of variables involved in, the constraint (which is 2 above). Therefore, the constraints are used to define the rules of the problem. How the problem at hand is defined in the mathematical framework (as a Classical CSP in this case) is called **modelling**. Problems, particularly more complex problems, can often be modelled in more than one way, each with different advantages.

In the case of the N-Queens problem the variables represent each line of the chess board and the values are the squares of the line where a queen could be placed. Before any queens are placed on the board, each of the variables 1 to N can be assigned a value from 1 to N. The constraints define where a queen cannot be placed. For example, a queen could be placed in the top left hand square of the board ($v_1 = 1$). Constraints are defined so that none of the queens on other lines

$$
\begin{aligned}
C \in \quad & \{(v_1 = 1\,,\ v_2 = 1) \\
& (v_1 = 1\,,\ v_2 = 2) \\
& (v_1 = 1\,,\ v_3 = 1) \\
& (v_1 = 1\,,\ v_3 = 3) \\
& (v_1 = 1\,,\ v_4 = 1) \\
& (v_1 = 1\,,\ v_4 = 4)\}
\end{aligned}
$$

Figure 1.2: Constraints corresponding to the assignment $v_1 = 1$ in a 4-queens problem.

(the variables v_2 to v_N) are in conflict with such a move. The constraints relevant to the assignment $v_1 = 1$, and where $N = 4$, are listed in Figure 1.2. Note that the model used implies no two queens will be placed on the same horizontal line so this rule does not need to be included in the constraint definition. This is one possible way of modelling the N-queens problem and the implication specified in the last sentence reduces the amount of constraint information that needs to be stored.

Classical CSPs take the form of a **decision problem**. The aim of the problem is to determine whether a satisfactory assignment can be made (i.e. find a solution). In the case of the N-queens problem the aim of the problem is to determine whether N queens can be placed on an N by N chessboard with the results being: yes - they can; or no - they cannot.

Research into Classical CSPs tends to focus on artificial problems, such as the N-queens problem. Another example is the Zebra problem. The Zebra problem is of a type that can be found in Logic puzzle books or in the weekend newspaper. It consists of 5 individuals of different nationality who live next to each other in different coloured houses. Each has a distinct pet, favourite drink and smokes a distinct brand of cigarette. The problem gives several statements such as "The Norwegian lives next to the Blue house." or "Kools are smoked in the house next to the house where the Horse is kept.". The aim of the problem is to determine which individual keeps a zebra as his pet. The problem only has one answer which makes it particularly difficult and of interest to research. Although the problem requires a specific answer (the Japanese man) it can still be considered as a decision problem as the aim is still to find a/the solution.

1.1.3 Finding a Solution

In order to find a solution, if one exists, or be sure that a solution does not exist, the approach to solving the problem needs to be performed in a regimented fashion.

The first queen will be placed in a square and only moved once it can be shown that all of the other queens cannot be placed on the board. After the first queen has been placed on the board the second queen is placed on a square that does not conflict with the first queen. This queen is only moved when all possible positions for the remaining queens have been attempted. The search for a solution continues in this manner. Such a search method is guaranteed to find a solution if one exists or prove that a solution does not exist otherwise.

Another approach to finding a solution would be to place all of the queens on the board and move the pieces one at a time to try and reduce the number of queens that can take each other. This process continues until a solution is found or some time limit is reached. Such a search method is not guaranteed to find a solution to the problem and cannot prove that a solution does not exist. However, it may be a quicker method of finding solutions.

Each time a queen is placed on the board or moved to a new square many decisions have to be made. The most common decisions are "which square" and "which queen". When a human attempts to solve this problem (s)he applies a certain knowledge to such decisions, such as "move the queen which is in conflict with the most other queens". One of the main aims of research into Constraint Satisfaction is to incorporate such knowledge into computer algorithms which solve CSPs. Computers will not do this and unless told to do so will attempt to solve a problem with no intuition.

1.1.4 Chronological Backtracking

With respect to solving CSPs the most common complete search methods are based upon Depth First Search (DFS). DFS extends the current partial solution by assigning values to variables one at a time. When a value cannot be assigned to a variable an inconsistency has been found in the current partial solution. In such a case DFS **backtracks**, unlabelling assignments of the partial solution and trying new assignments. The search process forms a tree, known as the **search tree**, an example of which is given in Figure 1.3. Each node represents an assignment to a variable and the corresponding partial solution of the node assumes this and the assignments to all ancestors of the current node. The root node corresponds to a partial solution with no assignments and is where search starts and ends. With respect to the search tree in Figure 1.3, the nodes are searched in the pre-order (i.e. $0, 1, 4, 5, 6, 2, 7, 8, 9, 3, 10, 11, 12, 0$).

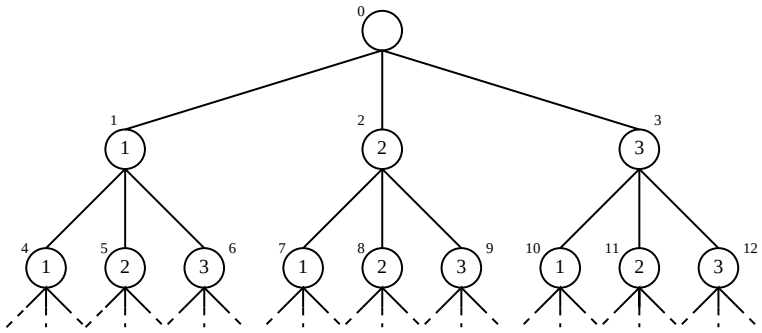

Figure 1.3: Tree based search

Each assignment made by the DFS algorithm is known as an **instantiation** and DFS is also known as **Chronological Backtracking** because the backward moves made by search merely move back to the previous assignment. More intelligent backward moves can be made by search and a brief description of one such method is described later.

1.1.5 Variable and Value Orderings

Other research into CSPs has concentrated upon the order in which the variables are tackled by search. The variable order can be crucial with respect to how much search is required to find a solution, or prove that a solution does not exist. For example, it may be appropriate to make a variable assignment earlier in search if attempting to do so later may be particularly difficult. Variable orderings fall into two categories: static and dynamic. **Static** variable orderings are determined before search begins often using analysis of the problem structure such as the **degree** of variables, where the degree of a variable is the number of constraints it is involved in. **Dynamic** variable orderings choose which variable should be assigned a value next during search. Such methods can utilise information gathered during search such as pruning performed using look-ahead. Commonly used methods include choosing the variable with the smallest domain [37], the smallest domain in proportion to variable degree [3], and more recently, in [35], defining how **constrained** (a combination of graph density and constraint tightness) the remaining problem will become. The aim of such heuristics is to tackle the variables in an order which will result in the remaining sub-problem being easier to satisfy.

When trying to solve a Classical CSP, the aim of search is either; to find a solution, and hence prove the problem has a solution; or to prove that there are

no solutions to the problem. Value ordering cannot affect the number of nodes of a search tree visited in searching for all solutions, merely the order in which they are considered. When searching for a single solution, however, assigning values in a particular order can result in a solution being found quicker. Intelligent value ordering heuristics developed for Classical CSPs work on the principle of choosing the value that seems most likely to lead to a solution. For each variable, assigning a value which maximises the size of the subsequent search sub-tree should minimise the size of search for a single solution. For example, in graph colouring problems this leads to the choice of a value that will minimise the chromatic number of the remaining sub-problem. Bear in mind that the values from some domains will be pruned by the current assignment.

One form of Classical CSP that has received considerable attention is the minimum colour graph colouring problem. Graph colouring consists of assigning values, or colours, to nodes of a graph such that no two connected nodes share the same value. The minimum colour problem consists of minimising the number of colours used in colouring all of the nodes of a graph. A common method of finding solutions to this problem is to assign the lowest available value, entitled the **sequential colouring** method in [55]. This method is often accompanied by a dynamic variable ordering heuristic [8,55,79]. At each stage of search, the variable which has the smallest domain size is assigned a value, on the rationale that the variable with the smallest domain is that which is the most difficult to assign a value to. By assigning the lowest value available, search makes the same value choice for variables with no past neighbours in the variable order. Then as further assignments are made, search attempts to assign values which have already been used for unrelated variables. Using this has the effect of reducing the number of values removed from the domains of future variables because their past neighbours will be assigned as few different values as possible.

For other Classical CSPs, methods have been developed that try and maximise the number of solutions within the future sub-problem. The work of Dechter and Pearl [22] reduces the future sub-problem into a tree structure. Then the number of solutions for this relaxed problem can be calculated in polynomial time, and the best such value assigned. This method was extended by Vernooy and Havens [70] to consider multiple tree-based future sub-problems combined to give the probability that a value is in a solution. Geelen has defined a formula to calculate the **promise** of a variable or a value, with dynamic variable and value orderings being based upon this measure [33].

Frost and Dechter have developed a simpler method of value ordering [29] where values are assigned according to the aim of minimising conflicts. It considers the **pruning** performed by assignments. Each time an assignment is made the possible future assignments which conflict with this assignment can be ignored or **pruned**. By assigning values which prune fewer future variable values, the number of possible future solutions is maintained at as high a level as possible. Such methods are easily applicable to graph colouring problems; although their performance is not known. Graph colouring problems are also related to Frequency Assignment Problems, and variations of value ordering by minimising conflicts can be found in [40] developed (apparently) independently, by Hurley *et al*, of the work of Frost and Dechter [29]. An example of variable and value ordering in action on the 5-queens problem, using the definition from Section 1.1.2, can be found in Figure 1.4.

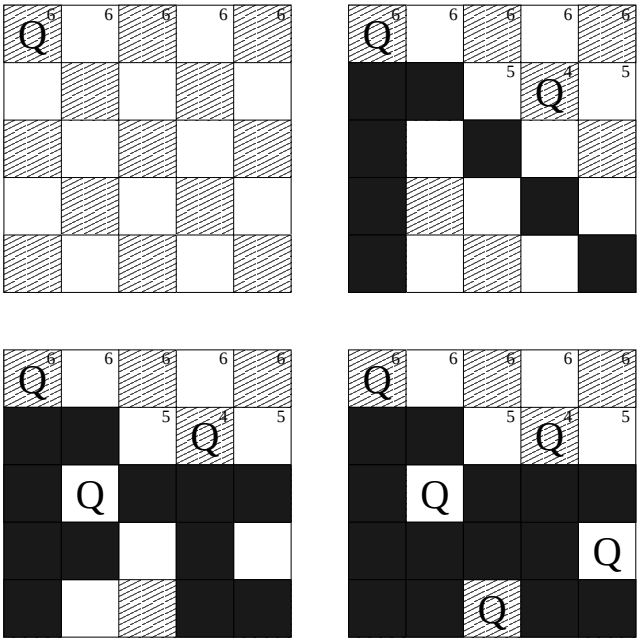

Figure 1.4: Finding a solution to the 5-queens problem.
Variables are ordered by domain size and values are ordered by minimum number of prunes in the future sub-problem (the value of which is given in the top right hand corner of each square). Heavily shaded squares of the chess board are pruned values. Search proceeds as follows.
Top Left Figure: The first variable is chosen arbitrarily as all variables have the same domain size. Equally the assignment to the first variable is chosen arbitrarily.
Top Right Figure: The second variable is chosen arbitrarily; however, it is assigned the 4th value as it prunes fewer future values (i.e. it will shade fewer unshaded squares of the board).
Bottom Left Figure: The 3rd variable is chosen next as it has the smallest domain. It only has one value so there is no need to perform value ordering.
Bottom Right Figure: The 4th variable is chosen next, again arbitrarily. Followed by the only remaining variable. Both assignments to these last two variables are given as each only has one consistent assignment.

1.2 Weighted Constraint Satisfaction

The definition of Weighted CSPs (WCSPs) is an extension to the CSP definition. Each constraint of the problem has an associated error. If a solution violates a constraint, then the error associated with the corresponding no-good is incurred. Every solution has a cost which consists of the sum of all errors incurred by the solution. The aim of the problem is to find the solution of minimum cost.

1.2.1 Search of WCSPs

The search algorithm used takes the form of that used to search Classical CSPs. However, it must be adapted to consider soft constraints. The *search*, *label*, and *unlabel* functions are defined in Algorithms 1, 2 and 3 respectively.

Algorithm 1 is the search function which maintains the current search depth and continues searching until this value is ≤ 0, in which case search has reached the root of the search tree and stops. Line 7 tests if the search depth is $> n$ or whether a full solution has been found. If this is the case the new extant solution cost (the solution will always be extant) is recorded in line 9 and search backtracks and continues in line 10. Line 13 tests whether the current variable can be assigned a value (as recorded by the last call to the *label* or *unlabel* function) and whether the current solution cost is lower than the extant. If both of these assertions are true then an assignment can be made (line 15); if not no assignment is possible and the previous assignment must be undone for search to continue (line 17).

The *label* function (Algorithm 2) makes an assignment in line 5. Then the consistency of this assignment is checked (line 11) with all previous assignments (defined by the loop on line 7). The *consistent* variable maintains the fact that the current assignment must be consistent throughout the process. If the current assignment is consistent (tested in line 14) then the solution cost is updated (line 18), else the current assignment is removed from the set of possible assignments to the current variables (line 16). This process is continued until a consistent assignment is found or the domain of the current variable is exhausted (both of which are tested in line 3).

The *unlabel* function (Algorithm 3) reverses this process. The domain of the current variables is reset (line 4) and the search depth is decrease so that the current variable is the previous assignment (i.e. the one that need to be undone). The current assignment to this variable is removed from the variable domain (line 7); the solution cost updated (line 8); and the assignment is reset (line 9). The function

Algorithm 1 Search

```
1  begin
2    search_depth := 1
3    extant := ∞
4    consistent := TRUE
5    while search_depth > 0
6          do
7          if search_depth > n
8            then
9                  extant := search_cost
10                 consistent := unlabel(search_depth, search_cost)
11         fi
13         if consistent and search_cost < extant
14           then
15                 consistent := label(search_depth, search_cost)
16            else
17                 consistent := unlabel(search_depth, search_cost)
18         fi
19   od
20 end
```

then determines whether a new assignment can be made in line 10. This assertion is passed back to the *search* Algorithm using the *consistent* variable.

Algorithm 2 Label(*search_depth*, *solution_cost*)

```
1  begin
2    consistent := FALSE
3    while | D_search_depth |> 0  and  !consistent
4          do
5          v_search_depth = d,   d ∈ D_search_depth
6          consistent := TRUE
7          for i = 1   to   search_depth − 1
8              do
9              if consistent
10               then
11                     consistent :=!conflict(v_i,   v_search_depth)
12             fi
13         od
14         if   !consistent
15           then
16                 D_search_depth := D_search_depth − d
17            else
18                 solution_cost = solution_cost + cost(v_search_depth)
19         fi
20   od
21 end(consistent)
```

Algorithm 3 unlabel(*search_depth*, *solution_cost*)

```
1  begin
2    if search_depth ≤ number_of_variables
3      then
4            D_search_depth := D
5            search_depth := search_depth − 1
6    fi
7    D_search_depth := D_search_depth − v_search_depth
8    solution_cost := solution_cost − cost(v_search_depth)
9    v_search_depth := ∅
10   if D_search_depth = ∅
11     then
12           consistent := FALSE
13      else
14           consistent := TRUE
15   fi
16 end(consistent)
```

1.3 Overview

In the next chapter research into CSPs that is relevant to this thesis is reviewed. Chapter 3 reviews the literature which focuses upon the University Examination Timetabling Problem from the early beginnings of the problem to the present day. Basic search methods used in this thesis and an analysis of the problem instances on which experiments are performed are given. Chapter 4 investigates the use of variable ordering heuristics in WCSPs and analyses the qualities that make a solution "good". Chapter 5 considers how value ordering can improve the solutions found by search. Hybrid Search for WCSPs is introduced in Chapter 6 and several combinations of search methods have been compared empirically. The work of this thesis is then concluded in Chapter 7.

Chapter 2

An Introduction to CSP Research

Due to the fact that Constraint Satisfaction Problems arise in many areas of Computer Science, and indeed life, they come in many shapes and sizes. The features of some problems may be specific only to that problem, or to very similar problems. The aim of this introduction to the topic is not to list every single type of CSP, but to introduce problems relevant to the work on examination timetabling in this thesis.

2.1 Classical CSPs

2.1.1 Problem Complexity

The complexity of a problem is defined by the complexity of the best method, or algorithm, for finding a solution. The complexity of an algorithm is defined by the amount of time, or number of steps, that it will require to find a solution in the worst case as a factor of the problem size. A **polynomial** time algorithm requires, in the worst case, a time of the order n^k (where k is a constant and $k \geq 1$) to find a solution to a problem of size n, represented as $O(n^k)$.

The class of **NP-hard** problems is defined as those problems that require an exponential amount of time in the worst case, $O(k^n)$ where k is a constant and $k \geq 1$, to find a solution. The solutions to such problems can be checked for correctness using an algorithm. A problem is **NP-complete** if it is NP-hard and the solution checking algorithm requires polynomial time. Most CSP decision problems, where the problem aim is to find a solution such as for N-queens, are NP-complete, while most Weighted CSPs fall into the category NP-hard.

2.1.2 Randomly Generated Problems

A great deal of research into Classical CSPs has concentrated upon randomly generated binary problems. **Binary** problems only constrain assignments between 2 variables. The use of random problems overcomes limits on the number of example instances which research methods can be applied to. For example, the most obvious and vigorously researched feature of Classical CSPs is that of finding solutions using the minimum amount of effort. When comparing methods of finding a solution, more confidence can be gained regarding the superiority of a method as the sample of problem instances increases in size. Random problems are usually defined as a tuple of form $< n, m, p_1, p_2 >$. The number of variables is defined by n; the number of values that can be assigned to each variable by m; the **density** of the problem by p_1; and the **constraint tightness** by p_2. There are many variations upon this definition (a description of some can be found in [62]), some with respect to specific notation, others to whether $p_1$and p_2 define strict proportions or probabilities; however, the general properties remain the same. The **density** of the problem is defined by the proportion, or probability, of constraints occurring between variable pairs. The **tightness** of constraints is defined as the proportion, or probability, of no-goods between two variables. During the rest of this thesis a constraint is defined between a subset of the problem variables (of size 2 in this case) and each consists of a set of no-goods between assignments to the variable subset.

Random binary CSPs can therefore be represented by a graph, known as a **constraint graph**; with nodes representing problem variables; and edges representing constraints. The density, therefore, defines the number of constraints in the problem, or the number of edges in the constraint graph, and the constraint tightness, the number of no-goods in each constraint. The term **over-constrained** is used to describe the situation when the density or constraint tightness of a problem are large enough to ensure that the problem has no solution. The specifics of how random Classical CSPs of this nature are generated is not of relevance to this thesis; however, it should be noted that there are different methods and badly formed methods could lead to misleading results. Figure 2.1 given an example of a constraint graph. The density of this graph is the number of constraints in the graph divided by the maximum number of possible constraints in the graph (i.e. the number of edges in a complete graph), in this case $\frac{11}{15}$ or 0.73. The tightness of constraints within the graph is dependant upon the type of constraints in the problem. It is calculated, in a similar way to density, as the number of no-goods (i.e. pair of inconsistent assignments) divided by the maximum possible number of no-goods.

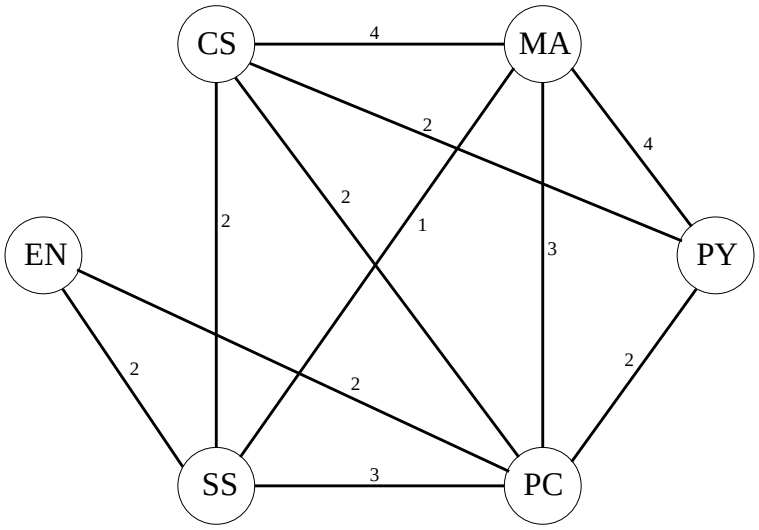

Figure 2.1: Example Problem Graph for the problem defined in Table 3.2.

2.1.2.1 Finding Difficult Random Problem Instances

A large proportion of research has examined the difficulty of finding a solution, or proving that no solution exists, to randomly generated problem instances as the problem parameters are varied. There has been a large amount of research into the phase-transition in random problems (originally discovered by Cheeseman, *et al*, in [18] and developed in terms of random binary CSPs by Williams, *et al*, in [81] and Smith in [66]), where there is a sudden shift between solvable and unsolvable problem instances as the constraint tightness is varied. It has been shown that the most difficult problems (on average) occur for the parameters where the phase-transition occurs. Such work has helped to define specific parameters where problem instances are particularly difficult to solve. Samples of such difficult problems have subsequently been used to analyse the performance of different methods of finding solutions.

2.1.3 Advanced Solution Methods

The most common method of finding a solution to a problem instance is by search. Search methods fall into two categories: **complete** and **incomplete**. Complete search methods will always find a solution if one exists and will prove that one does not otherwise, although they may require an exponential amount of time. Incomplete search methods may find a solution; however, they cannot determine whether a solution exists or not and will search indefinitely unless some stopping

criteria are defined. On the other hand, the advantage of using incomplete search methods is that, if correctly defined, they may be able to find solutions considerably quicker than complete methods. There now follows an overview on complete search methods with respect to classical CSPs; incomplete search methods will be discussed later.

2.1.3.1 Incorporating "Look-Ahead" Into Search

There have been many improvements made to the basic search algorithm with attempts to add some intelligence. One such improvement is the use of "**look-ahead**" developed by Haralick and Elliott in [37]. "Look-ahead" examines the implication of an assignment or potential assignment with respect to the problem constraints. An example of this is **Forward Checking**(FC). Each time an assignment is made the search algorithm can look ahead and calculate the effects upon variables yet to be assigned a value, or **un-assigned** variables. **Related variables**, which occur within the scope of a constraint (i.e. both are connected by an edge in the constraint graph) along with the current variable, can be pruned where they conflict with the current assignment. This is merely **pruning** the domain, i.e. removing a possible assignment, of un-assigned variables when the conflicting assignment is made and not when the later assignment is attempted. This allows search to identify when there are no satisfying assignments to an un-assigned variable, with respect to the assignments made so far, without wasting time assigning values to un-related variables. The process of pruning all of the values in the domain of a variable is called **domain wipe-out** and using look-ahead in this manner helps to reduce the amount of search performed by backtracking as soon as a domain wipe-out is discovered. The only cost is a more complex method of maintaining the domains of un-assigned variables which is often little in comparison to savings made. An example for the propagation used by FC can be found in Table 2.1.

A more intelligent form of "look-ahead" is to enforce **arc-consistency**(AC) during search. A sub-problem consisting of the future (i.e. un-assigned) variables and their remaining domains is arc-consistent if all pairs of possible future assignments are consistent with respect to the binary problem constraints. For example, if a possible variable assignment is inconsistent with every possible assignment to another future variable, then the given assignment can be eliminated or pruned. When combined with a search algorithm such a procedure forms a **Maintaining Arc-Consistency**(MAC) algorithm.

	Remaining Domain values				
Assignment	CS	MA	EN	SS	PY
PC=1	2,3	2,3	2,3	2,3	2,3
CS=2	-	3	2,3	3	3
MA=3	-	-	2,3	$\emptyset$	$\emptyset$

Table 2.1: An example of the effects of Forward Checking propagation.
The effects of 3 consecutive assignments to the variables of the problem in the graph of Figure 2.1 when domain size is limited to 3. After these 3 assignments no possible assignment can be made to the variables SS and PY. Forward Checking will propagate each assignment as it is made (i.e. calculate the information in the table) and will be able to ascertain that this partial solution cannot be extended. Without such a look-ahead technique search may attempt to assign a value to EN and in such a case would not discover the inconsistency until SS or PY are considered for assignment.

2.1.3.2 Intelligent Backtracking

Incorporating intelligence into search has also been applied to the process of backtracking (performed by the *unlabel*() function in the description above). The drawback of the Chronological Backtracking procedure is that it does not necessarily move back to the actual cause of the inconsistency. If such a cause were to be an assignment made near the start of search, chronological backtracking would exhaustively search the search sub-tree below this assignment for no gain. This process is known as **thrashing**.

Intelligent backtracking includes methods such as Gaschnig's Backjumping (BJ) [31]; however, one of the most advanced methods is Conflict-directed Backjumping (CBJ), devised by Prosser [61]. CBJ records which assignments have caused a value to be pruned from the domain of an un-assigned variable (this is regardless of whether look-ahead detects the inconsistency or if it is detected as an assignment is attempted). When an assignment cannot be made search can jump back and unlabel all variables below the deepest conflicting variable. Because the assignments between the un-assignable variable and its deepest conflicting relation would have been fruitlessly searched, the completeness of the algorithm is maintained. Research has shown that such intelligent backtracking can be extremely useful.

Such a method has been extended by Ginsberg to form Dynamic Backtracking [36]. Dynamic backtracking does not require that variable assignments made between the current un-assignable variable and its deepest related variable be unlabelled. The variable order is altered so that the conflicting related variable is moved

down the variable order such that it directly precedes the current un-assignable variable. This prevents the search process repeating the assignments made to the variables between the conflicting variables. Provided pruning through look-ahead is carefully maintained the method retains the completeness of search.

2.1.3.3 Alternative Tree Traversal Techniques

Harvey and Ginsberg have developed Limited Discrepancy Search [38], or LDS, which employs a different approach to how the search tree is searched. It has already been mentioned, in this section, how altering the variable ordering so that the difficult to assign variables are placed near the start can reduce search effort. However, such a variable ordering also has the drawback that if such a difficult to assign variable was to be the cause of inconsistencies deeper in search, a large amount of search effort may be required to reassign such a variable even if intelligent backtracking methods were used. In order to combat such "bad" moves early in search LDS systematically searches all paths that differ from the current partial solution by one assignment. If, subsequently, a solution is yet to be found, the process is repeated for solutions that differ by two assignments. Then, by three assignments, and so on. Provided the search is performed systematically completeness is maintained by the algorithm. The nature of this search means that a small number of "bad" assignments that may occur near the root of the search tree can be altered without the need for exhaustive search.

Meseguer's Interleaved Depth-First Search [57], or IDFS, provides a similar search method. When a domain wipe-out is encountered, rather than backtrack to the previous (conflicting) variable, search backtracks to the first variable and attempts an alternate assignment. Once all of the first variable assignments have been attempted, search considers an alternative second variable assignment. Then, the various assignments to the first variable are repeated. This form of search is actually analogous to Breadth First Search (BFS); however, at each node of the search order it performs a depth first traversal of the search sub-tree. Breadth First Search searches the search tree a level at a time, rather than a branch at a time like DFS. As search moves deeper every consistent solution at that depth is considered and extended in the next level of search. With respect to the search tree in Figure 1.3, the nodes are visited in the order $0, 1, 2, 3, 4, 5, 6, 7, ...$ As with BFS problems regarding storage space are encountered by IDFS; however, the DFS element of the search means that solutions to the problem may be available as the BFS progresses.

Walsh's Depth-bounded Discrepancy Search(DDS) [76] combines both LDS and

IDFS incorporating the best features of both. This search method limits the number of discrepancies considered during each iteration of the algorithm by limiting the depth at which they can occur. Search starts by concentrating on those paths to the bottom of the search tree which differ for assignments made to variables near the start of the variable ordering. As more iterations of the algorithm are performed, paths which differ by more and more variable assignments are considered. The LDS element allows the search to explore paths down the search tree which differ from the first path by a limited number of discrepancies (by 1 discrepancy to begin with, followed by 2 discrepancies, followed by 3,...). The IDFS element of the search limits the maximum depth at which these discrepancies can occur. At the start of search discrepancies are limited to the top of the tree where conventional wisdom would suggest the mistakes are most likely to have been made; however, as search progresses this maximum depth is increased. Experiments in [76] demonstrate empirically the superiority of DDS over both LDS and IDFS.

2.2 Partial CSPs

Partial CSPs, or **PCSPs**, were first defined by Freuder and Wallace in [27] as an extension to the Classical CSP framework. When problems are over-constrained, search should try and find a "good" partial solution to the problem. What constitutes a "good" partial solution can vary from one problem to another. In some cases it may consist of finding a full solution that satisfies as many constraints as possible; while in another finding as large a partial solution as possible that does not violate any constraints may be the order of the day. In either case, PCSPs correspond to optimisation problems. Such problems have had a great deal of research performed on them with reference to specific real world problems (a brief description of three such problems is given in section 2.3); however, it is an important issue to link the advances made on artificial problems with those made in the real world. Different types of Partial CSP arise depending upon what is being relaxed in the search for a "good" solution. A brief overview of these are given in Section 2.2.1.

Higher level frameworks have been defined that encompass all types of CSP. Semi-ring-Based [4] and Valued-CSPs [65] have been developed separately and subsequently shown to be interchangeable in [5]. Both define the possible tuple costs, and operators to compare and combine costs. In the case of Classical CSPs, tuples are associated with the boolean values $TRUE$ and $FALSE$ corresponding to whether the tuple is a **good** or a **no-good**. They are combined using the logical

and ($\wedge$) operator; however, different definitions allows different problem types to be defined. These definitions are made at an abstract level that allows any form of CSP to be defined. Such a framework can also help to compare CSP types and allow definitions such as look-ahead to be extended to other CSP types where the definition may not be so obvious.

2.2.1 Problem Types

A Probabilistic CSP, as defined by Fargier and Lang [25], is a CSP where each constraint has an associated value. This value specifies the probability that the constraint is in the problem. Such uncertainty about a constraint is evident in real world problems, e.g. the possibility of road works in the Travelling Salesman Problem. Given a problem instance with a number of constraints which might need to be considered, a whole range of sub-problems provide possible perturbations of the problem. Each of these sub-problems will have a probability of occurring. The aim of the problem is to find a solution with the maximum probability of being the solution to the actual problem. This is given by the probability that the constraints which a solution violates are in the actual problem. In this model the preference defines the extent to which the constraint is applicable to the problem being defined. In the following models a preference is associated with each constraint which defines the constraint importance when finding a solution.

A Fuzzy CSP can be used to define a real world problem where an amount of tolerance is allowed in finding satisfying valuations for the problem constraints. In [24], Dubios, *et al*, define the order of preference by a **fuzzy relation** which assigns a value in the range [0..1] to each satisfying tuple, or labelling of the variables, of a constraint. The variable labellings associated with the constraint are; completely valid if the value is 1; invalid if the value is 0; or valid to a certain extent otherwise. So the tuples of constraints have varying amounts of preference, depending upon how well the constraint is satisfied. Each constraint is said to have a preference distribution. The problem can also model required constraints by only using 1 and 0 as levels of preference for the relevant tuples. Then the aim, as defined in [24], is to find a solution, or full set of labellings, which maximises the minimum constraint preference that is achieved. [24] extends CSP techniques such as arc-consistency to Fuzzy CSPs. Such a method is similar to Possibilistic CSPs as defined by Schiex in [64].

Lexicographical CSPs have been introduced in [26], by Fargier and Lang, to try

and combat the fact that in Fuzzy CSPs only minimum constraint preference for a solution is considered in calculating the solution preference. In fact, in a large real world problem, this preference or error may be unavoidable to a certain degree and the comparison of solutions on this value may not give a true comparison. As with Fuzzy CSPs all of the constraints have a preference distribution. The preferences of the constraint tuples referred to by a solution are placed into a list. Then these lists of preferences can be ordered and compared by counting the number of constraints with a particular level of preference from the least preferred upward. When they differ, the solution which has fewer constraints of that level of preference is the better as in general its constraints are solved better when looking from bottom up-wards.

Hierarchical CSPs encompass the notion that constraints should be grouped according to their importance. In many real-world problems constraints are defined as different types of constraints and not merely the collection of no-goods as used in random CSP models. Therefore, constraint types often come in an order of preference and can be grouped together in a **constraint hierarchy**, a term originally used by Wilson and Borning in [82]. Within the hierarchy, constraints can still be assigned weights and preferences; however, when used strictly a constraint is infinitely more important than another if it belongs to a higher level of the constraint hierarchy. Solution costs are subsequently given as a tuple such that each number represents the solution cost for each level of the hierarchy. Provided that each level is distinct the hierarchy can consist of different types of CSP. For example, a Classical CSP could be used at the top level to define the rules of the problem, with different Partial CSP models being used at lower levels depending upon the real-world constraint being modelled. The Constraint Hierarchy has been extended by Kam and Lee [43] to consider "fuzzy" boundaries between each level, forming a **Fuzzy Constraint Hierarchy**. Such an adaptation extends the framework such that a large error to an important constraint of one hierarchy level may encroach upon the importance of a less important constraint of the preceding level.

Maximal-CSPs (MAX-CSP) and Weighted CSPs (WCSP) are two other Partial CSPs of more relevance to the rest of this thesis. They are described in more detail below.

2.2.1.1 MAX-CSPs

The MAX-CSP was originally proposed in [27] as a form of PCSP and has the same structure as a Classical CSP; however, the objective is to satisfy as many constraints as possible. The cost of a (partial) solution is the number of constraints that have not

been satisfied. As an extension of Classical CSPs, a large amount of work has been produced extending the research into Classical CSPs to MAX-CSP. An example is the extension of look-ahead. Look-ahead in MAX-CSP performs less pruning than for Classical CSPs. It is mainly used to increase the lower bound of partial solutions by counting the minimum number of constraint violations caused when assigning a value to a **future variable** (i.e., a variable that has not been assigned a variable before). This minimum cost will always be incurred when assignments are made to this future variable, with respect to the current partial solution, and so can be included in the lower bound. Arc-consistency has been extended, in the form of Directed Arc-inconsistencies (defined below), to consider possible errors that will be incurred between pairs of future variables.

Look-ahead for MAX-CSP was first defined in [27] and later optimised by Larrosa and Meseguer [47]. The initial work of Wallace and Freuder concentrated upon the extension of techniques applied to Classical CSPs. In [73] and [49] new variable ordering heuristics are proposed for MAX-CSP using several extensions of algorithms used to search Classical CSP. This work was extended by Wallace [71] by considering methods to increase and decrease the lower and upper bounds respectively. By doing so the evidence shows that less search effort is required to search the problem. Such work with regard to bounds has been continued in several directions.

The use of Directed Arc-inconsistency Counts (DACs) to increase the lower bound, as defined in [71], has been improved several times. DACs are an extension of "look-ahead" information and are used to calculate whether any possible assignment that can be made to a future variable must incur some error. The minimum such error must always be incurred when extending the current partial solution and so can be added to the partial solution cost, which acts as the current lower bound. The use of DACs has been extended by Larrosa and Meseguer [45, 46, 48].

Several methods of defining the lower bound to a problem instance have been examined by Cabon, *et al* in [14] with a hybrid Russian Doll Search (RDS) proving to be the best. However; they only compare how the lower bound is developed over time and do not compare overall search results. RDS performs searches on sub-problems of increasing size with each larger sub-problem containing the previously smaller one (hence the Russian Doll). This technique has been extended by Meseguer and Sánches and combined with a search algorithm [58]. Their results on random problem instances show that in order for the lower bound to become effective the problem needs to be either complete or very dense.

The remainder of the work into MAX-CSPs by Freuder and Wallace has, in

the main, concentrated upon hill-climbing local search methods [72, 74, 75], which are described in more detail in section 2.3.3. Tabu search, which is similar to hill-climbing and is also described in more detail later, has also been implemented for MAX-CSP by Galinier and Hao [30]; however, it is only compared to hill-climbing which limits any conclusions.

2.2.1.2 Weighted CSPs

Weighted CSPs are very similar to MAX-CSPs. A weight, in the range $[1..\infty]$, is associated with each constraint and if a solution does not satisfy it, then the solution cost includes the constraint weight. A WCSP is equivalent to a MAX-CSP if all constraints have a weight of 1, as the solution cost function will merely count the number of violated constraints. The weight of ∞ defines that a constraint cannot be violated by any valid solution, and so a WCSP is equivalent to a Classical CSP if all constraints have the weight ∞. This leads to the definition of **hard** and **soft** constraints. A hard constraint is a constraint that cannot be violated, as it has a weight of ∞. A soft constraint can be violated, but at a cost given by the constraint weight. Generally, the hard constraints are used to define the rules of the problem while the soft constraints define the optimisation criteria of the problem.

WCSP constraints can be considered in two ways. Each constraint can be associated with a weight or each constraint no-good can be associated with a weight. Both cases are equivalent. Obviously the first constraint definition is encompassed by the second definition. The second definition can be defined by the first, by using multiple constraints to define the various errors between two variables. In practice the second definition is used. However, when talking about WCSP constraints it is good to distinguish between the hard and the soft constraints. Therefore, in this thesis when a reference is made to the hard constraints it refers to the hard no-goods of a constraint.

2.2.1.3 WCSPs in the Real-World

The WCSP model is widely used both implicitly and explicitly in solving real-world problems. The use of WCSP as a framework for research has the drawback that random problems require a constraint weight set which needs to be distinct so as to; not provide the same results as random MAX-CSP instances; and be similar to weights found in the real-world. However, the similarity between MAX-CSP and WCSP means that random MAX-CSP instances can be used for exhaustive testing

while real-world problems modelled using WCSP can be used to confirm results. This has been the case when search method have been implemented for MAX-CSP as in [14] and [58] where after testing for random MAX-CSP problem instances the Frequency Assignment Problem has been used as a real-world comparison problem.

2.3 Real World problems

The motivation behind constraint satisfaction work is that it can be applied to real-world problems. Solving such real-world problems can increase efficiency in industry, distribute work between workers evenly or maintain achievable schedules. In this section two real-world problems are introduced. It should be noted that the following problems are given to provide a flavour of the kind of problems that occur in the real world. Although the vast majority of real-world optimisation problems can be modelled using WCSPs or Constraint Programming in general, other approaches to solving the problems (such as Integer Linear Programming) may be both more suitable and more commonly used. These problems will be used to illustrate some of the advanced optimisation methods which are currently used on real-world problems.

2.3.1 Nurse Rostering

The problem of Nurse Rostering, as investigated in exists in [1], all major hospitals and other similar institutions. The problem consists of producing a 24 hour schedule such that a specific number of nurses are required to be present in a ward at any time. The number can vary as, for example, more nurses may be required during the day than during the night. Also, nurses can have specific skills or rank. Subsequently, a matron may have to be on duty at all times during the day or a nurse with a specific skill may need to be on hand at all times. There are also the preferences of the nurses to consider. For example, one nurse may require a particular day off. Rules or preferences concerning whether nurses have to work equal numbers of each shift over a longer period of time may also arise. The core problem forms a decision problem; however, the addition of preferences result in a optimisation element to the problem. An approach to solving the problem as a CSP could consider each shift in turn and assign personnel to that shift such that all skill requirements are met and nurse roster patterns are within an acceptable level of the desired preference for each member of staff. In such an approach variable ordering would correspond to the order in which shifts where considered with the actual variables recording each

staff position on the shift. It would be desirable to consider shift which are hard to assign staff to first such as the night shift. Value ordering would correspond to which staff are assigned to each shift with each value representing a staff member.

2.3.2 Car Sequencing

Car Sequencing problems are used to defined the optimal order of cars in a production line [21, 69]. As cars are passed along a modern production line they move continuously and mechanics work on them while they move along. Each group of mechanics work on a specific **feature**(e.g. sunroof, rear spoiler, etc...) of the car and need to stay within a distance of a base where they access parts and tools. Therefore, each feature of a car can only be "fitted" on a sub-part of the production line. The over-constrained nature of this problem arises because each part can only be fitted to so many cars at once. This is not a problem if the cars move down the production line at a slow rate. If the production line speed is slow enough to allow the most time consuming feature to be fitted to every single car then no cost will be incurred from having to pause the production line. However, different cars in a sequence will have different features or specifications. The aim of the motor company is to run the production line at as fast a speed as possible without requiring it to be paused. Constraint Satisfaction methods are used to order the cars such that the production line can run at a fixed speed, which forms a decision problem, or to order the cars such that the speed of the production line can be maximised, an optimisation problem. Placing the cars with the most time consuming feature across the ordering will mean that the workers who fit that feature may have enough time to fit that feature. The problem becomes more complicated as more features need to be fitted to each car.

2.3.3 Advanced Optimisation Methods

The optimisation search methods which are commonly used on real-world problems tend to be incomplete (i.e. they are not guaranteed to find an optimal solution). The size of the search space of large problems means that it is rare that complete search methods visit a significant proportion of the search tree. In tree-based search it is difficult for search to jump from one section of the tree to another that may offer significantly different, and possibly better, solutions, without losing the property of completeness. To this extent **stochastic** methods are of more interest when searching in the real-world. **Stochastic search** incorporates a random element

into search such that the same search method can be applied several times with different results. Several common optimisation methods are described below.

2.3.3.1 Local Search

Local search techniques avoid the problems of tree-based search. By allowing changes to any variable assignment, as opposed to only the last assignment, the search procedure can focus on those variables which are involved in a significant proportion of the solution error at each search iteration. Such search methods can also be called "hill-climbing" methods. The position in a landscape is defined by the assignments made to the variables. The height of the current point in the landscape is given by the solution quality. Hill-climbing methods always try to climb upward. The drawback of such methods is that they can often lead to dead-ends where the current solution cannot be improved at all. Such points of the landscape are known as **local-optima**. Such a situation would occur if a search method were to climb and reach the top of a hill that is not the highest hill in the landscape. There have been many attempts to get around this, such as restarting search and the use of **tabu lists**. By restarting search the hill-climbing procedure may start climbing from another part of the landscape. **Tabu lists** remember a number of the last moves made by search. By doing so it can attempt to move away from a previously search-area and move away from the local optimum. Davenport and Tsang [21] use a variant of the hill-climbing approach to search on the car sequencing problem. The cars of the sequence are moved to positions where they result in a lower cost. They develop a new method of escaping from local optima by increasing constraint weights and show that it is superior to similar methods of escaping local optima. Tabu search has been applied to examination timetabling by Di Gaspero and Schaerf [32]. In such an implementation exams are moved from one timetable slot to another such that a cost is reduced. They have shown that a standard Tabu search can compete with more traditional Constraint Programming methods.

2.3.3.2 Genetic Algorithms

Genetic Algorithms provide a method of searching evenly across the whole solution space while being able to concentrate on improving the solution cost. One of the common, yet not so well cited, drawbacks of GAs are that they require experimentation to perfect the parameter settings. This, however, is countered by the fact that a basic implementation should provide reasonable results and a more problem specific

implementation should improve results even more. GAs operate on a population of elements. In a simple implementation each element is a, possibly inconsistent, solution to the problem. A fitness operator decides which elements are "fitter" or better than others and the fitter elements are combined, or mate, to form a new population. Two parent solutions mate in two main ways. A **combination** function combines two elements together. Such a method can be as simple as chopping solutions in half and recombining them. **Mutation**, as the name suggests, applies a random mutation function to the assignments of a solution. As the population number increases GAs can converge toward a population of fit elements. Implementations with respect to exam timetabling have to overcome the large obstacle of fitting two half schedules together. GAs have been applied to Nurse Rostering by Aickelin and Dowsland [1]. Each solution is represented as a sequence of shift patterns for a set of nurses. The crossover operator combines two sequences of shift patterns; the mutation operator randomly reassigns a nurse to a different shift pattern; and the fitness function determines whether a solution is feasible and how closely it satisfies the nurses preferences. The sequences that make up a population are ordered according to fitness and then chosen for recombination according to their position in this order. In [1] such a method is extended further to take advantage of problem structure; however, the overlying principle of GAs remains.

2.3.3.3 "Squeaky-Wheel" Optimisation

"Squeaky Wheel" optimisation [42] is an optimisation method that re-applies search continually re-ordering the problem variables in order to tackle the variables which are harder to find assignments for. This idea behind this incomplete optimisation technique is that it is only by applying search that the difficult parts of the problem can be identified. After search has been applied to the problem, the results of search are examined and each variable has its contribution to the solutions found assessed. The variables are then re-ordered so that those variables that have a large negative effect upon the solution quality are moved up the variable order so that they are tackled earlier in search, hopefully resulting in better assignments by search. This method has not been applied to university examination timetabling; however, it is of relevance to such a real-world problem where small parts of the problem may have a big effect upon the solution quality. "Squeaky-wheel" optimisation provides a method of identifying such parts of the problem. "Squeaky-wheel" optimisation has been applied to a variety of problems both real-world and artificial. An application to scheduling a production line is shown to be superior to Tabu search and Integer

Programming in [42], where a comparison with graph colouring methods is also provided. However, possibly due to the vast amount of research into solving graph colouring problems, "Squeaky-wheel" optimisation is only shown to be competitive at best.

2.4 Summary

In this section I have introduced the research into Constraint Satisfaction Problems and the terms used that are relevant to this thesis. Extensions to the basic search methods used, which attempt to instill intelligence, to find a solution to a problem have been described. Several examples of real-world problems have been introduced to familiarise the reader with the wide variety of CSPs that occur in the real-world. Incomplete search methods which are commonly used to solve such problems have also been covered.

Chapter 3

University Examination Timetabling

The University Examination Timetabling Problem (UETTP) is a problem encountered by universities around the world; that of trying to examine all students for the courses they have chosen to take while satisfying the conditions which define an acceptable examination schedule. With respect to this investigation, an acceptable examination timetable is one where no student is required to sit two exams at any one time. This is the most common requirement of a solution to the UETTP because if such a situation where to arise the student involved would be required to sit the two exams in succession and either in a separate room from other students or under strict supervision from an examination invigilator. Such an occurrence is not pleasurable to the student under the stress of examination; neither to the university which is required to instigate the special circumstances for the examinations. In most cases this requirement is achievable and often, as is the case in this investigation, is a requirement of any valid solution to the problem. With respect to Weighted CSPs the constraints which model such a rule are hard constraints.

This problem can usually be solved very easily and has many solutions. Therefore, the problem is often extended to consider criteria that are to be optimised. These criteria can take many forms and vary from one institution to another; however, the most common are **spacing** or **proximity** constraints. Proximity constraints attempt to spread out the timetable slots of the exams which a student sits. In the model used in this investigation, defined and used in [16, 17, 32], a cost is associated with each occurrence where a student sits x exams within y timetable slots. In order to keep the problem simple only binary constraints are considered. This leads to the definition of a **proximity weight set** (pws). This is tuple of values that define the error incurred when a student sits 2 exams within 2, 3, 4 or 5 timetable slots. The **pws** is therefore a set of 5 values, for example $\{5, 4, 3, 2, 1\}$. In

exam timetabling it is assumed that not sitting x exams in y periods is of greater, or at least equal, importance than not sitting x exams in $y+1$ periods. The error incurred is restricted to sitting x exams in y periods exactly; otherwise, sitting 2 exams in 2 consecutive slots would incur the error of all **pws** values as the 2 exams are also within 3, 4 and 5 slots. The **pws** size of 5 has been used in keeping with the work of [16, 17, 32]; however, it is worth noting that if the **pws** is too large then search will rarely consider distant proximities, or constraints with a large y, resulting is redundancy.

When defining experiments to be conducted it must be considered which proximity weight set should be used. The work of Carter, *et al* [17] uses the **pws** of $\{16, 8, 4, 2, 1\}$, yet does not explain why this set has been chosen. A major consideration is how much more important is sitting 2 exams in y periods than 2 exams in $y+1$ periods? In the **pws** of [17] the answer is: twice as important. Using $\{16, 8, 4, 2, 1\}$ as the **pws** is, in fact, a justifiable choice as it both reflects a substantial increase in importance with proximity, while avoiding a situation whereby this increase is so marked that it effectively leads to a hierarchy of distinct groups of constraints. In the latter situation the over-emphasis on the closer proximity constraints may lead to the more distant proximity constraints, between timetable slots distant in time, rarely being considered; which would question their use in the problem model.

The overall cost of a solution is therefore the sum of the proximity weights incurred for each student (defined in 3.2). In practice the problem consists of a file which defines the **pws**, a file that contains a list of each course and the number of students who are registered for it, and a file where each line consists of the course choices of a particular student. In the next section some mathematical terms that are used in this thesis are defined.

Other constraints can be introduced to the model, such as limits on the number of students who can sit exams at the same time or a maximum limit on the number exams assigned to a single slot; however, in keeping with the model of [16, 17, 32] and to prevent the model becoming too complicated, these have not been considered in this investigation.

Early attempts to solve the problem concentrated on graph colouring. Ensuring that no student is required to sit 2 exams at once means that no pair of exams which are both selected by the same student can be assigned to the same slot of the exam timetable. This is analogous to a graph colouring constraint. Therefore, a problem consisting only of such constraints forms either a minimum colour graph

colouring problem, if the aim is to minimise the length of the exam schedule; or a fixed k-colour graph colouring problem, if the aim is to find a feasible schedule for a fixed exam period of length k. Such work includes that of Mehta [56] and Leighton [50] whose work was inspired by applying graph colouring research into real-world problems. From such early beginnings modelling exam timetabling problems has included many other features. Later research has concentrated upon specific instances of the problem at a particular institution. This has led to many varieties of problem aims being considered. Such investigations use a range of methods to model and find solutions to the problem. The most popular method in early attempts is Linear Programming which has been used in [41, 44, 52]. More detailed overviews into earlier work can be found in [15].

As with any real-world problem, exam timetabling problem instances can vary with respect to the specific aims from one institution to another. Burke and Elliman [12] give an overview of such aims derived from the results of a questionnaire sent to many British universities. This can be termed as the "real world politics" of the problem. Differences can include; the importance of spacing out the exams sat by students; regulations regarding exam invigilation; whether room assignments need to be considered; whether certain exams need to be sat in a particular order; and so on. Much of the research into exam timetabling at a particular institution considers a specific problem with respect to such politics. The aim of research, however, should be to concentrate upon the problem in general. By comparing and improving techniques to the general problem such techniques can subsequently be extended to specific problems.

Such a framework was originally proposed by Carter, *et al.* in [17], derived from earlier work in [44], and has been used in other work since. This framework has been used in this thesis as it can be expressed as a Weighted CSP and results obtained can also be compared with the work of others. A Weighted CSP model of the problem has been used by Lim, *et al* [51] where a greedy search heuristic has been applied to a specific problem instance. The actual methods examined by Carter, *et al* [17] consider various variable orderings in conjunction with a constructive search method. Their work has been extended into a commercial timetabling package which is used to schedule the examination timetables of institutions across the world. The research in [17, 51] provides a good starting point for looking at applying Constraint Satisfaction techniques to the UETTP; however, such work has not been extended to cover issues such as intelligent Constraint Programming search methods.

The most interesting features of the less general and more institution-specific

work includes how the problem is broken down into phases such as assigning exams to timetable slots and assigning exams to rooms. Such a staged approach to the problem can be found in [44, 52, 80] and allows the problem to be broken down into problem stages that can be modelled in a simpler fashion. Other methods include the use of Constraint Logic Programming (CLP) [6] where assigning exams to slots, to rooms, and allowing preferences have all been incorporated into a single model. Such work demonstrates the usefulness of constraint programming packages; however, it does not provide any suggestions as to the development of new problem solving techniques and concentrates upon examining how a UETTP can be modelled. The job-shop scheduling problem has also been used to model examination timetabling [60], with advantages in the use of previously discovered optimisation techniques for job-shop; however, again the focus of this work is how a UETTP can be modelled.

The use of TABU search to find solutions has received attention in [32, 80]. The work of Di Gaspero and Schaerf [32] applies TABU using the standardised framework to the suite of problem instances given in [17]. The results obtained are shown to be better than those of [17] only in some cases. Schaerf has also provided a detailed overview of recent work in [63]. White and Xie extend TABU to incorporate a long term memory that prevents updating to the most actively assigned variables. However, they only compare results for two problem instances providing little insight into the performance of their methods with other work in the field.

The major source of recent developments can be attributed to the work of the Nottingham based ASAP group. In [13] Burke, *et al* have used constraint programming and incorporated a stochastic element into dynamic variable ordering by choosing for assignment during search the best variable, as defined by a variable heuristic, from a randomly chosen sub-set of the variables. The use of a small stochastic element in variable ordering has also been briefly investigated by Carter[1] and this may be an interesting line of future research. The majority of the research performed by this group has used evolutionary based methods to seek results. In [11], and later [10], Burke and Newall use a "Memetic" algorithm to search for solutions. Such an algorithm uses evolutionary based operations, the mutation of solutions in this case, with a local search method, a hill-climbing algorithm. Such methods are used to combine evolution with methods that correspond to an organism's ability to adapt to its environment.

In [53] Marin, *et al* use a Genetic Algorithm to evolve the search strategy used by a more traditional search method. Each element of their sample population en-

[1]From personal communication.

codes combinations of search methods, variable and value orderings. Marin has also implemented a GA which evolves solutions to the problem (i.e. each element of the sample population is a solution) [54]. He tackles the problem of two parent solutions combining to make a child that is an inconsistent solution by use of a clique based crossover operator. A set of assignments that form a clique in the constraint graph are forced upon a child which starts with the assignments of its other parent. When a forced assignment results in an inconsistency with another variable assignment, which shall be called the **inconsistent variable**, the inconsistent variable takes the value that was forced off the variable that it was inconsistent with. If this forced assignment results in another inconsistency the process continues. Such an approach to avoiding inconsistencies is similar to that used in [17] where inconsistent variables are either given a new assignment or de-assigned. The results of such GA approaches to the problem are often not comparable with those found in [17] as, although the same sample problem instances are used, often different problem parameters or problem definitions are used.

Some of the previous work into exam timetabling has attempted to use random problems. As mentioned in previous sections, random problems are extremely useful for research. The drawback of using random problems for real-world problems is that it can be hard to replicate the general problem structure that can be found in the real-world. A number of students could randomly choose a set of courses; however, in reality course selection will be dependent upon a whole variety of factors that may be hard to model. Random problems tend to produce a constraint graph with similar degrees for each variable. Exam timetabling graphs have been shown, by Walsh [77], to show a small-world structure whereby many nodes of the constraint graph are clustered locally with a small number of constraints linking such clusters. In the real-world a small number of exams tend to be related to a large number of others while most exams tend to have very low degrees. A similar pattern occurs for the constraint weights of the problem with a small number of constraints having large weights and a larger number with increasingly smaller weights. The work of Coudert [20] has shown that real-world graph colouring problems tend to contain a large clique whose size defines the chromatic number. The work of [20] has shown that by labelling the variables of such a clique first an optimal solution can be found easily. Random problems would need to incorporate phenomena such as "small world-ness" and large cliques so that more advanced search techniques that consider such problem features will be tested fully. Random graphs with a small-world structure can be produced; however, problems would still arise when assigning

suitable constraint weights.

3.1 Problem Definitions

Problems are defined as a set S of student combinations $\{S_1, S_2, ..., S_n\}$ of exams from an exam set E. Each S_i represents the exams which student i is required to sit.

$$S = \{S_i : S_i \subseteq E\} \tag{3.1}$$

The variables $L = \{l_1, l_2, ..., l_m\}$ record the timetable slot of each exam, where an exam is in the range 1 to m. The $pws(l_i, l_j)$ function defines the proximity weight between labels i and j. The solution error $COST(L)$ is defined thus:

$$COST(L) = \sum_{\forall S_i \in S} pws(l_x, l_y) : \forall x, \forall y \in S_i \tag{3.2}$$

A constraint (or exam combination), C_i, is defined as a set of exams which occur in a student combination.

$$C_i \subseteq S_j \, , \; 1 \leq j \leq n \tag{3.3}$$

Each C_i can be considered as the set of variables (i.e. exams) to which it applies.

$$C_i = \{L'\} \; where \; L' \subseteq L \tag{3.4}$$

The set of all constraints (exam combinations) C cover all existing combinations, such that:

$$\bigcup_i C_i = \bigcup_j S_j \tag{3.5}$$

Two constraints (exam combinations) are said to be related if they contain common variables (exams) in their scope.

$$C_i \cap C_j \neq \emptyset \tag{3.6}$$

The neighbourhood of a variable (exam) is defined as the set of exams which occur in a student combination with the variable (exam) in question.

$$N_x = \bigcup_{i\,,\, x \in S_i} S_i \tag{3.7}$$

The degree of a variable (exam) is defined as the size of its neighbourhood.

$$d(x) = \mid N_x \mid \tag{3.8}$$

The weight of a variable (exam) is the number of students required to sit the related exam.

$$W_x = \mid \{S_i \,:\, S_i \in S \;\; and \;\; x \in S_i\} \mid \tag{3.9}$$

The neighbourhood of a constraint (combination of exams) is the set of related constraints (exam combinations):

$$N_{C_i} = \{C_x \,:\, C_x \in C - C_i \;\; and \;\; C_i \cap C_x \neq \emptyset\} \tag{3.10}$$

The degree of a constraint (combination of exams) is defined as the size of its neighbourhood.

$$d(C_j) = \mid N_{C_j} \mid \tag{3.11}$$

Constraints can be partially ordered using the following $<$ definition and where $O_a < O_b$ defines the variable order (i.e. variable a occurs earlier than variable b).

$$C_i < C_j \;\; \Leftrightarrow \;\; max\, O_a < max\, O_b\,, \;\; a \in C_i \;\; and \;\; b \in C_j$$

The **neighbourhood** of a constraint C_i is the set of constraints that are related to it. A backward neighbour of a constraint C_i is a related constraint C_j such that $C_j < C_i$. The constraint backward neighbourhood is the set of all backward neighbour constraints.

$$P_{C_i} = \{C_x \,:\, C_x \in C - C_i \;\; and \;\; C_i \cap C_x \neq \emptyset \;\; and \;\; C_x < C_i\} \tag{3.12}$$

Subsequently, the constraint backward degree is:

$$CBD_{C_i} = \mid P_{C_i} \mid \tag{3.13}$$

Note that in the definitions of (3.12) and (3.13), several of the constraints in the constraint backward neighbourhood ($C_j \subseteq P_{C_i}$) may include the same past variables

and will be counted more than once in CBD_{C_i}. This is reasonable because each of these constraints will have an effect on how well the constraint will be satisfied.

The set of students involved in a constraint (exam combination) is the set of students that select every variable (exam) within the constraint scope (exam combination).

$$W_{C_j} = \{S_k \ : \ S_k \in S \ \ and \ \ C_j \subseteq S_k\} \tag{3.14}$$

The weight of a constraint is the size of this set (i.e. the number of students who select every exam from the combination.

$$w(C_j) \ = \mid W_{C_j} \mid \tag{3.15}$$

As students are added to an empty problem, each student combination that includes the constraint variables (exam combination) will include other variables (exams) which occur in one of the other constraint student combinations, but may also include variables (exams) unique to its combination.

$$\forall C_j \in C \ , \ \forall S_k \in S \ , \ \bigcup_{S_i \in W_{C_j}} S_i \ \supseteq \bigcup_{S_i \in W_{C_j} - S_k} S_i \tag{3.16}$$

The same relationship applies when considering how the variable (exam) degree increases as more students are required to sit a course.

$$\forall x \in E \ , \ \forall S_k \in S \ , \ \bigcup_{S_i \in S \ , \ x \in S_i} S_i \ \supseteq \bigcup_{S_i \in S - S_k \ , \ x \in S_i} S_i \tag{3.17}$$

3.2 A Search Algorithm

3.2.1 Backtracking

Backtracking within the problem can be for two reasons. Either the current partial solution is no better than the extant solution or a hard constraint has been violated. I shall discuss each of these situations separately. In the case of the first instance, which I shall call a **soft backtrack**, search can move back until the current partial solution cost, or the lower bound, is less than the extant solution cost. Figure 3.1 demonstrates how such backtracking can be used to lower the extant solution cost. As search time, measure on the x-axis, increases the best solution found so far, or **extant**, improves, as measured by the y-axis. However, a non-intelligent,

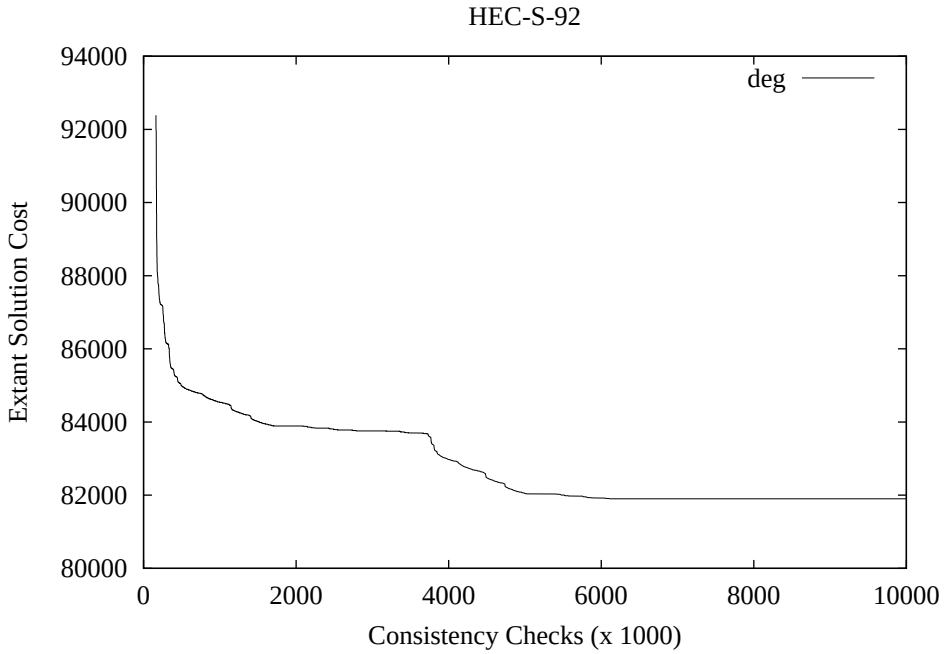

Figure 3.1: How the solution quality improves as search progresses. Extant Solution Cost versus Consistency Checks. The search algorithm was stopped after 10,000,000 checks.

or dumb, value ordering heuristic has been used in the experiment in Figure 3.1. When searching WCSPs, it is very important to use a value ordering heuristic that attempts to apply some criteria as to which value is best. The value order in Figure 3.1 uses the lowest available value as commonly used when searching Classical CSPs.

When search uses a more intelligent value ordering once it finds a solution it rarely finds a better solution without backtracking by a significant amount. This phenomenon is due to the fact that the value ordering will assign values which incur an optimal, or near optimal, error cost with respect to the current partial solution. To improve upon these assignments a large amount of work needs to be undone. This leads to thrashing behaviour. An attempt was made to incorporate CBJ into the search procedure; however, as most constraints connected to variables assigned at the bottom of the search tree incur some error, this provided little, if any, gain. Therefore, DFS of the search tree could only provide a single solution.

Finding a solution also requires the satisfaction of the hard constraints. Again chronological backtracking required large amounts of time to resolve inconsistencies, if it could resolve them at all. An implementation of CBJ specific to the hard constraints did not show any noticeable improvement. As the hard constraints are

graph colouring constraints an implementation of CBJ, which backtracks to the deepest variable that prunes a value from the current un-assignment variable, will usually backtrack to the deepest related variable, in a manner similar to graph based backjumping (GBJ defined in [23]). Added that domain wipe-out will only occur in constraints with a degree larger than the domain size, and each past variable can only prune at most one value, then a large proportion of the previously assigned variables will be candidates to backtrack to. When considering intelligent variable orderings, in general, variables with a large degree will be assigned first. Therefore, any intelligent backtracking method, such as GBJ, is unlikely to back-jump further than 1 or 2 variables.

Therefore, the backtrack method described in Algorithm 4 has been used to find consistent solution when searching on a problem with a low domain size. This backtrack procedure is the one used by Carter *et al* [17], Morgenstern [59] and Larporte and Desroches [44], being applied to timetabling and graph colouring problems. When a variable x cannot be assigned a consistent value, this function forces an assignment on this variable and undoes previous assignments that are in conflict with it. A value is selected for assignment to x from the original domain D (as x's domain D_x has no consistent values) in line 3. The exact method used to choose d can vary; an example being, choosing the d that will cause the minimum number of re-shuffles in lines $13-23$. Then in lines 4 to 26 the effects of this choice are instigated. Every past neighbour of x, where $o_1, ..., o_n$ defines the place in the variable ordering of each variable, that has been assigned the same value d (v_i is the value that variable i has been labelled with) is processed (line 4). Depending upon whether other consistent values can be assigned to these past neighbours (as determined in line 8) the variable is either re-assigned (line 10), or unlabelled (line 12) and re-shuffled. Lines 13 to 23 move the recently unlabelled variable i down the variable order until it is directly after x (i.e. all re-shuffled variables are tackled next by search). Note that Algorithm 4 does not specify how the current partial solution cost is updated. An example of this Algorithm in action is given in Table A.1 of Appendix A.

During search, any variable that is unlabelled will be involved in re-assignment, whether immediately, in line 10, or later in the search after re-shuffling; however, it is important to make the distinction as to when the re-assignment occurs. To this end the term "re-assignment" has been used when a new value is immediately re-assigned (in line 10) and the term "re-label" has been used to describe the fact that a variable has been assigned a new value at some point in search (in either line

Algorithm 4 When the current variable 'x' cannot be assigned a consistent value:

```
2  begin
3    v_x := d, ∃d ∈ D
4    for i := 1 to n
5        do
6        if v_i = d ∧ o_i < o_x
7          then
8                if | D_i |> 0
9                  then
10                       v_i := e, ∃e ∈ D_i
11                 else
12                       v_i = ∅
13                       top := o_i
14                       bottom := o_x
15                       o_i := o_x
16                       for j := 1 to n
17                           do
19                           if bottom ≥ o_j ≥ top
20                             then
21                                   o_j = o_j − 1
22                           fi
23                       od
24               fi
25       fi
26   od
27 end
```

10 or after a re-shuffle). In either the "re-assignment" or the "re-shuffle" case the variable will be "unlabelled" whether re-labelled immediately or later on in search.

3.2.2 Exploitation of problem structure during complete search

Thrashing behaviour in complete search, i.e. when attempting to improve upon the initial solution, can occur for several reasons. For example, if 2 variables at the end of the variable order are unrelated then every minimal error incurring assignment of each (where $nomin_i$ is the number of minimal error incurring assignments that can be made to variable i) will need to be compared together for no gain (a total of $nomin_{n-1} \,.\, nomin_n$ needless assignments). Such a waste of computation is avoided because the value ordering heuristic which has been used in these experiments selects one of the values with the lower error cost. Therefore, there is no need to try

alternative assignments provided the variable in question has no effect on future assignments in the complete search. A **stable set**, of size ≥ 1, can then be defined at the end of the variable ordering, or the portion of it to which complete search is applied. A **stable set** is a set of nodes of a graph; none of which are connected to each other in the graph. When backtracking from variables within this stable set, search can jump back to above the variable at the start of the stable set. Such a move is equivalent to **CBJ** [61] although conflict occurrence between variables is assumed as problems are over-constrained.

The effect of such a search can be amplified by ordering the variables such that a large stable set occurs at its end. As variables in such a stable set will have low degrees such an ordering may complement variable ordering methods which order such variables near the end of an ordering. I have defined a variable ordering heuristic that uses a greedy heuristic to place a large stable set at the end of the variable order for the critical sub-problem. Such a heuristic, in practice, does little when searching large problems and so the results presented in this thesis do not try and improve upon the first feasible solution found. However, in Chapter 6 complete search of small sub-problems is performed. In such cases, where the number of variables in the stable set represents a large proportion of all variables, a reduction in search effort can be made.

3.3 Problem Features and Parameters

problem	#courses	density	#comp	max.comp.	max.clique	low	high
HEC-S-92	81	0.42	1	81	17	18	23
STA-F-83	139	0.14	3	62	13	13	18
KING-96	141	0.07	3	136	7	8	12
YOR-F-83	181	0.29	1	181	18	21	30
UTE-S-92	184	0.08	2	177	10	10	14
EAR-F-83	190	0.27	1	190	21	24	31
TRE-S-92	261	0.18	3	259	20	23	30
LSE-F-91	381	0.06	4	378	17	18	22
KFU-S-93	461	0.06	22	434	19	20	24

Table 3.1: Problem parameters.

Table 3.1 lists the problem parameters of the nine smallest problems for which all of the experiments have been performed on before the results of these experiments

are then extended to larger problems. These problems are part of a suite of problems first used in [17] and now available on the internet as benchmark problems[2]. They come from a variety of different institutions from around the world. The **#courses** column defines the number of problem variables or the problem size. As the problem size increases, search methods that are not purely tree-based become the only usable search methods. The research of this thesis examines search using a high domain size (the size of which is given in the **high** column) then applies the results to the low domain (given in the **low** column). The high domain sizes have been chosen to be large enough so that intelligent search is not required, or at least hardly required, to find a solution. Experiments have been performed on the high domain size so analysis into the performance of search without the use of intelligent backtracking can be performed. The same experiments are performed using the low domain size so that the effect of the intelligent backtracking method can be included in any conclusions. Subsequently any improvement or degradation of solution quality shall be defined as being as the result of the search methods used and/or the use of intelligent backtracking. The low domain size can be seen to be close to the maximum clique size (given in column **max.clique**). A clique is a complete sub-graph within a problem. The maximum clique of a problem is the largest sub-set of problem variables such that a constraint exists between every pair of variables in the sub-set. The maximum clique size defines a lower bound on the number of colours required to colour the constraint graph and, in the case of real-world problems, tends to indicate how many colours will be required to colour the overall problem [20]. The maximum clique size, which is independent of proximity constraints, certainly defines the minimum number of colours required to form a graph colouring because a different colour is required by each node in the clique. As the domain size approaches this lower bound, i.e. the number of slots required, finding a solution becomes more difficult.

The problems consist of two files: a student data file and a course data file. The student data file consists of a list of student course choices. Each line corresponds to a single student and consists of the codes of the courses taken by that student. From such data the problem constraints can be defined. The course file lists the course codes and the number of students registered to each course on a line per course basis. This file is not actually needed; however, it has been used for error checking. A small example problem can be found in Table 3.2 which is translated into the graph shown in Figure 2.1.

[2]ftp://ftp.mie.utoronto.ca/pub/carter/testprob

Dan	CS	MA	PY
Kevin	CS	MA	PY
Debbie	CS	MA	PC
Andy	CS	MA	SS
Neill	CS	PC	SS
Mani	MA	PY	PC
Iain	MA	PY	PC
Martin	PC	SS	EN
Simon	PC	SS	EN

Table 3.2: Student Course Selections

Table 3.1 also records the number of problem components that exist within each problem (in column #**comp**). The components of the problem are sub-problems that are connected, i.e. there is a path from every sub-problem variable to another, yet dis-connected from every other component, i.e. there is no path from one component to another in the problem constraint graph. The statistics show that the majority of the problem instances have more than one component. Therefore, such instances could be broken down into sub-problems and tackled independently; however, the size of the maximum component (in column **max.comp.**) shows that in all but one case the components consist of one very large sub-problem with those remaining being very small. Therefore, breaking the problem instances into sub-problems would not reduce computational effort by any significant amount.

3.4 The Experiments

The WCSP model was coded using C++ on a Linux OS. The experiments were performed on a Pentium 2, 350 MHz Linux computer with 512KB of RAM. Although optimisation was not the single issue when implementing the model efficient code was strived for. There are several CSP toolkits available; however, none was used during this investigation due to its specific nature with respect to problem types. The amount of CPU time required to find a solution to a problem instance varied from about 70 CPU secs. for the smallest problem to about 200,000 CPU secs for the largest(although the largest problem was exceptionally large). For the majority of problems a number of CPU secs in the thousands was required to find a solution.

3.5 Sample Sizes

Later in this thesis stochastic search techniques are used. When using methods with a random element it is important to take a sample of runs of sufficient size. The samples help to establish the confidence in search heuristics and would be used in the real-world to find a solution of lowest cost. Figure 3.2 compares the mean and standard deviation of the solution costs of an example sample. As can be seen by 100 samples any bias in sampling has generally been avoided. The sample size of 100 has therefore been used. When comparing two distributions it is important to prove, to a certain level of confidence, that there is a significant difference between the samples. To this end, the Z Test shall be used. Based upon the assumption that the distribution is normally distributed, the Z value (described by equation 3.18) is a measure of how far away the mean of a sampled distribution($\overline{x}$) is from another(μ) with respect to the standard deviation(σ) and the sample size(N) [19].

$$Z = \frac{\overline{x} - \mu}{\sigma/\sqrt{N}} \tag{3.18}$$

For the two distributions to be considered similar to a significance of 0.05 then $-1.96 \leq Z \leq 1.96$. The sample size of 100 was often large enough to show differences between samples to the given level of confidence.

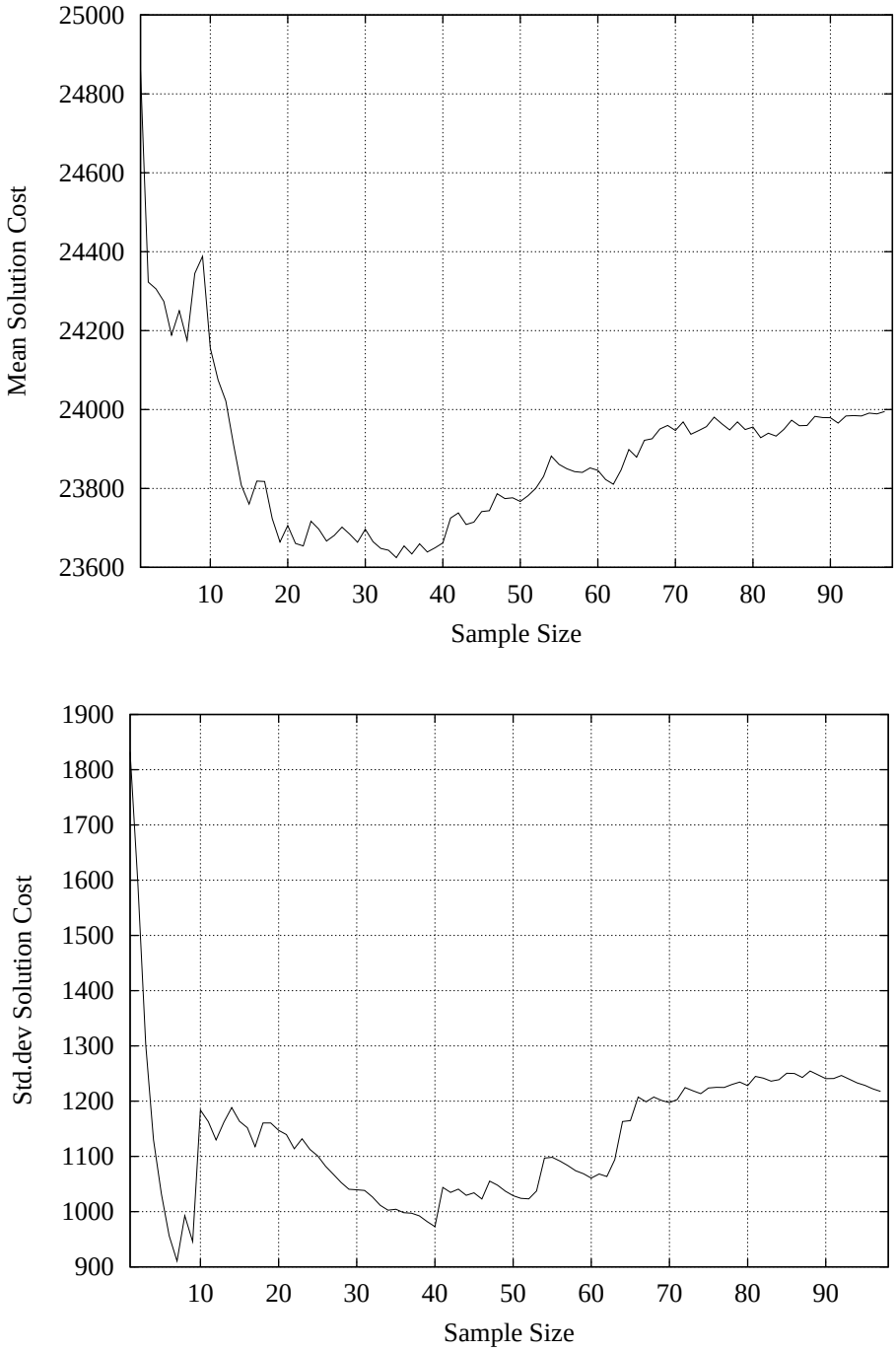

Figure 3.2: An example of the mean and std.dev. of increasingly larger sample sizes.

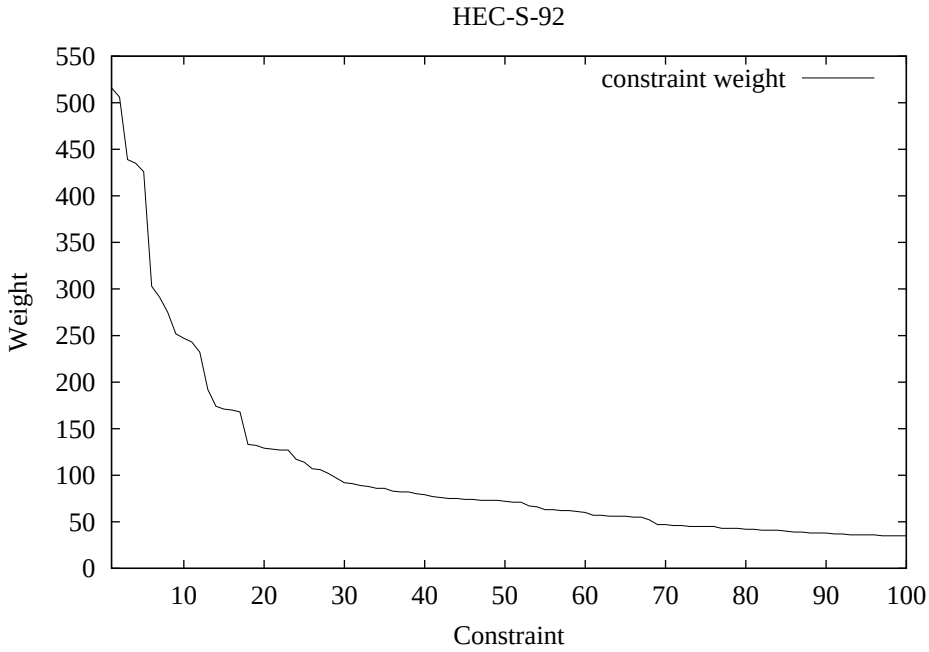

Figure 3.3: The distribution of constraint weights for the **HEC-S-92** problem. The **heaviest** constraints are to the left and the **lightest** to the right. Note that the right hand tail is not shown; however, it merely continues in a gradually decreasing fashion.

3.6 Problem Constraint Weights

Figure 3.3 gives an example of the distribution of constraint weights. The weight of a constraint is the number of students who are registered for all of the combination of courses covered by the constraint. The distributions for all of the problems follow this pattern of a few constraints with very large weights and increasingly more constraints as the constraint weights get **lighter**. The terms **heavy** and **light** are used when comparing constraint weights. A constraint is heavier than another if it has a larger weight and is lighter if it has a smaller weight. The distribution shown occurs as a result of the general problem structure.

Many real world problems, and exam timetabling in particular, show **small-world** behaviour such that the problem structure tends to consist of localised cliques, sub-graphs where every variable is connected to every other variable within the clique, in addition to problem edges which link the various localised parts of the problem [78]. This problem structure occurs in UETTPs because institutions tend to consist of several faculties or schools, e.g., English, Maths, Computing, Music, etc... Students who study at an institution tend to study within one particular

school, which corresponds to the localised nature of small-world problems. However, students can often take electives, where they study a course in a totally different school to their own, or study a joint honours course, where a combination of two, or possibly more, schools contribute to the choice of courses taken by a student. Hence the localised parts of the problem are connected together. The many light constraints of the constraint weight distribution correspond to these non-localised constraints. The localised parts of the problem correspond to the heavier weights. The heaviest weights correspond to compulsory courses in large schools.

Chapter 4

Variable Ordering Heuristics

4.1 Introduction

For any CSP the order in which search tackles variables can be crucial with respect to the difficulty in finding an initial solution and the quality of this solution. For example, it may be appropriate to make a variable assignment earlier in search if attempting to do so later may be particularly difficult. Equally, if a variable is connected by heavily weighted edges (in examination timetabling, edge weight is the number of students sitting that combination of exams), then it may be appropriate to assign it a value near the start of search to avoid incurring a big error cost. In this section, I intend to compare several variable ordering heuristics and determine which are preferable with respect to the problem and value ordering heuristic used. The choice of value ordering heuristic used will be justified in the next chapter. Comparisons are made between; existing variable ordering heuristics used on Classical CSPs; heuristics which have already been proposed for this and related real-world problems; and new heuristics, which I have developed, extended from reasoning behind the heuristics in the previous two categories. An analysis method has been defined and used in comparing the results of different search methods.

Variable ordering heuristics can be classed as either **static** or **dynamic**. The static class of heuristics define the ordering before search begins, while the dynamic methods use data generated during search to determine which variable search should assign a value to next. Both methods have advantages and disadvantages. The obvious advantage of dynamic methods is that they can estimate whether the expansion of the current partial solution will cause a problem for a particular future variable. While a static ordering may be able to determine when such a situation is probable (e.g. by comparing variable degrees), static methods cannot consider difficult future

assignments which are particular to the current partial solution. In general static variable ordering heuristics are quicker to apply than dynamic methods because dynamic methods must be re-applied throughout search, as opposed to one application before search.

The motivation behind variable orderings with respect to Weighted CSPs falls into two aims. The first aim is to find a feasible solution, i.e. satisfying the hard constraints, with the minimum number of backtracks. A great deal of research has been conducted into this area on a wide variety of Classical CSPs [22,23,28,34,35,61, 62,66,67,69]. The second aim is to minimise the error incurred by the solutions found (i.e. minimise the violations of the soft constraints). As the problems considered are over-constrained (with respect to soft constraints), this takes the form of trying not to violate soft constraints with large weights over those with smaller weights. When comparing solutions both of these aims need to be considered. The extremes of considering these being; searching for an optimal solution to the soft constraints with no regard to the amount of time required by search; and searching for a solution to the hard constraints with no regard to the soft constraints. The two methods are incomparable if one is faster but does not satisfy the soft constraints well, and the other finds good solutions to the soft constraints but is slow.

4.2 Static Heuristics

Commonly used heuristics order the variables based upon the problem structure. Measures such as variable degree can suggest which assignments will have a big impact upon whether search will find a solution (Classical CSP). The work of Carter *et al.* [17] states that placing the largest clique at the start of the ordering, and arranging the rest of the variables according to a heuristic, aids search. This view is extended by Coudert [20], to a greater extent, where graph colouring problems are solved optimally using the maximum clique. The rationale behind ordering variables to solve the hard constraints as quickly as possible also applies, to some degree, to minimising the error in the soft constraints. A hard constraint becomes difficult to satisfy as the values of related variables are pruned. This results in less choice when assigning values so as to minimise the error in the related soft constraint. As shown later in section 4.3, this is not a direct relationship as constraint weights can vary considerably.

In the experiments two value ordering heuristics have been used. These are:

- **ERR-LOW**: the value which incurs the least increase in error with respect

to the past variables is selected, tie-breaking upon the lowest value(i.e. the earliest such slot in the timetable).

- **ERR-RND**: the value which incurs the least increase in error with respect to the past variables is selected, tie-breaking randomly.

Non-stochastic value orderings (**ERR-LOW**) will always find the same initial solution (with respect to a problem instance and a non-stochastic variable ordering). The quality of this solution can vary, although it is later shown (see Section 5.3.5) that it is usually worse than the mean found by the equivalent stochastic method. The non-stochastic results give an indication of which variable ordering is best; however, the multiple samples of the stochastic value ordering gives better confidence in the results.

4.2.1 Analysis Method

When comparing the solutions found using different variable orderings we try to define a set of **critical constraints** that have the largest impact on the solution quality. Different ways of defining the critical constraints can be used. For example, the heaviest constraints could be deemed critical as they have the potential to incur a large amount of error. However, such a simple definition does not consider sub-problem difficulty. A heavy constraint may be considered not critical if the variables in its scope have a low degree. In such a case, despite the constraint having a heavy weight, the sub-problem structure in the vicinity of the constraint is sparse, resulting in the constraint being satisfied easily. On the other hand, identifying some sub-problem features can be a difficult, or even NP-complete, task. In such a case, the amount of effort required to process a problem before search may not be repaid by improved search.

The variable order used can be considered when defining the critical constraints. Such critical constraints will be variable order specific and will allow a definition that considers how the constraints will be encountered by search. An example of variable order specific information is variable forward degrees, which can be used to order the variables before search. All critical constraint definitions that use information from search, as is the case when comparing the solutions found using different variable orderings (as in this chapter), will be variable order specific. A further subset of these variable order specific critical constraints are search specific critical constraints. As critical constraints defined in this way are based upon a set of problem solutions they will differ from one set of search solutions to another.

Once these constraints have been found, the properties they hold under different variable orderings can be identified and used to shed light on why one ordering satisfies critical constraints better than another. The critical constraint set has been used because non-critical constraints will tend to have less impact upon solution quality. Including too many non-critical constraints in the critical constraint set should be avoided.

The question of which constraints are critical to finding a good solution cannot be fully answered without performing search. However, if a general characteristic of the set could be identified (i.e. one that is independent of any variable order), then an ordering based upon this characteristic should result in a better variable ordering strategy.

The comparison method has been broken down into two cases; general comparison between several variable orderings; and direct comparison between two orderings.

4.2.1.1 General comparison between several variable orderings

This method is used when comparing the solutions found using several unrelated variable orderings. In this case the critical set is defined as those constraints that incur the most error. An example can be found in Table A.2 of Appendix A. The critical constraint set is search specific and will depend upon the variable ordering. An example of the error that occurs across the constraints of a problem is given in Figure 4.1. Each constraint is a discrete point on the x-axis with constraint positions ordered by the error incurred by each constraint. The y-axis shows the error incurred by each constraint of the x-axis. The shape of the error distribution, as in Figure 4.2, can vary considerably. In some problem solutions only a few constraints may incur the majority of the error leading to a relatively flat line with a sharp jump at the the rightmost end. On the other hand, the error may be more evenly spread leading to a gradual increase with a small jump at the rightmost end.

The size of the critical set is defined as a percentage of the total number of constraints. In Figure 4.1 this corresponds to a set point on the x-axis. Other possible definitions include; all constraints that incur more than a percentage of the maximum error incurred, defined by a set point on the y-axis of Figure 4.1; or those constraints that incur a percentage of the total error, defined by a proportion of the area under the line in Figure 4.1. It will be assumed that constraints toward the left side of the error distribution are non-critical. In comparisons between the critical sets for different variable orderings, it is important to use a common critical set definition because critical sets may include non-critical constraints. Moreover, when

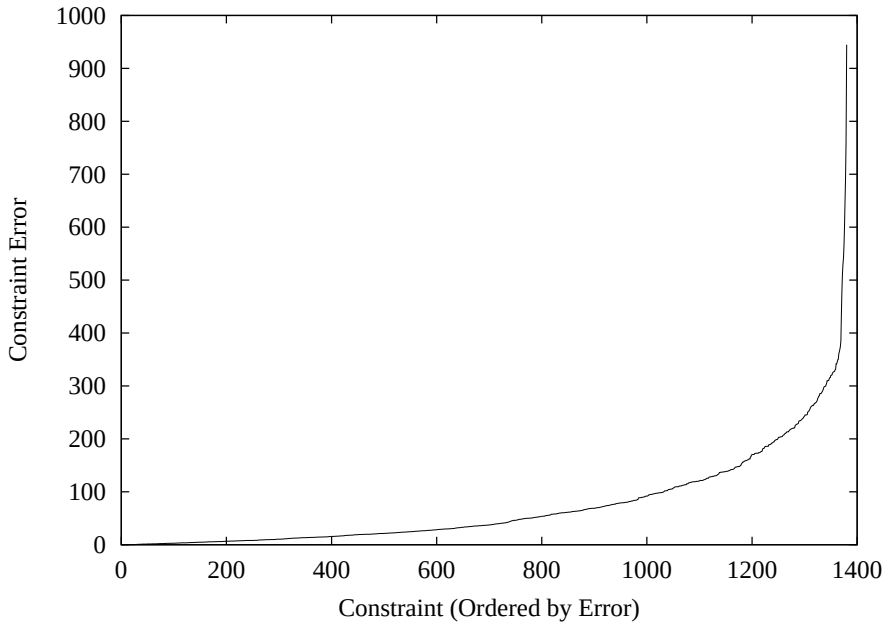

Figure 4.1: Distribution of error from constraint to constraint.
Constraints are ordered by the size of the error incurred in the example solutions.

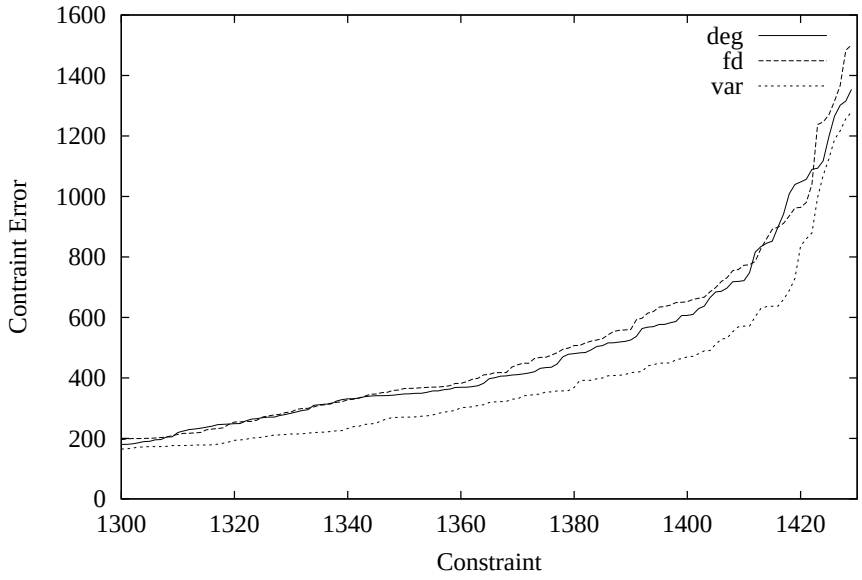

Figure 4.2: Distribution of error for three different variable orderings, focusing on the constraints which incur the most error.

comparing the solutions found by different variable orderings, the error distribution shape may differ from one ordering to another, as in Figure 4.2 where different orderings have different rates of increase. Under such conditions, an error-based critical set may include many more of the non-critical constraints for one ordering than for another. The advantage of using a critical set size of a fixed proportion of the total number of constraints is that it is less susceptible than other methods to bias caused by differences in the shape of the error distribution.

The percentage used is 1% and can be so small because each problem consists of thousands of constraints. Furthermore, the distribution of constraint weights has heavy outliers which suggests that only a small number of constraints will incur a large error. Experiments have shown that the conclusions are not affected by a small change in the % defining the critical constraint set. Whether the error incurred by the constraints of the critical set maintains the same order of magnitude as the overall error from one variable ordering to another can be used as a guideline to whether the percentage used is suitable.

4.2.1.2 Direct comparison between two orderings

This analysis method has been developed to compare two variable orderings that are related in some way (e.g. degree and weighted degree). In this case, the overall aim is to establish where and why one ordering improves over another. The critical constraints are defined as those that contribute the most to the improvement, incurring the biggest saving. The critical set will therefore be specific to the two variable orderings being compared in each comparison (in fact, to the sample of solutions for each ordering). An example can be found in Table A.3 of Appendix A. Figure 4.3 shows an example distribution of constraint improvements for two related variable orderings. Under the direct comparison, the constraints of the problem fall into three general groups:

1. those that incur a larger error for the superior ordering.
2. those that show little difference with respect to error incurred.
3. those that contribute most to the saving made by the superior ordering with respect to the inferior ordering.

All three groups occur when comparing solutions of differing cost; group 3, as a large improvement must be made somewhere; group 2, as a large number of lighter constraints will not incur any noticeable error; and group 1, as any improvement is

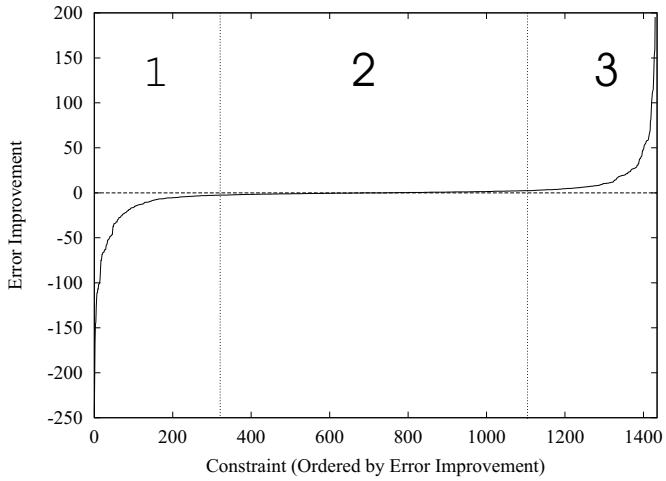

Figure 4.3: Distribution of improvement in error incurred by different constraints. The constraints are ordered according to the improvement. The numbered sections of the graph refer to the different groups of constraints, as explained above.

usually made at the expense of some increase in error from related constraints. In addition, group 2, which consists of lighter constraints, will have had little impact on the path taken down the search tree. Therefore, they will tend be satisfied with no reference to the potential error that they could incur. It is important that, when defining the size of the critical constraint set, a large number of constraints from this second group are not included as most of these constraints will be non-critical.

Group 3 can be very small with only a few heavy constraints constituting the difference between the solutions found by the two orderings. Only the constraints that provide the most improvement should be included in the critical set, so including constraints from group 2 should be avoided. The critical set size only needs to be large enough to avoid bias and to show a significant difference in the analysis of the constraints of the set as effected by the variable orderings. The set size is initialised to an arbitrary size of 10 then, if required, increased until the measures used in the comparison differ significantly. A big increase may lead to the inclusion of constraints from the group 2. However, this can be taken into consideration when debating the results of any analysis. Also, the two orderings in question may not differ significantly at all, in which case the analysis would be inconclusive for any size set. The same critical set size need not be used from one comparison to another because each comparison is different. Each comparison is derived from the distinct relationship between the pair of variable orderings being compared.

4.2.2 Standard Methods

The variable ordering heuristics that have been compared initially are:

- Degree (**DEG**): variables are ordered by decreasing degree, such as **LF** in [50]. A variable with a large degree will have a larger impact on the search than a variable with a smaller degree, as it interacts with more variables. An example can be found in Figure 4.4.

- Forward Degree (**FD**): variables are ordered on decreasing forward degree. The forward degree of a variable represents the number of future neighbours or the degree within the future sub-problem. This is the opposite of node width, as defined in [69], which is the degree of the past sub-problem. A variable with a large forward degree will render more possible future assignments invalid than a variable with a smaller forward degree. An example can be found in Figure 4.5. The calculations used to establish the ordering in Table 4.1.

- Variable Weight (**VAR**): variables are ordered on decreasing variable weight. Variable weight corresponds to the number of students sitting the exam that the variable represents. By assigning values to heavily weighted variables first, search attempts to maximise the available values for these variables and hence reduce the error incurred by them. An example can be found in Figure 4.6.

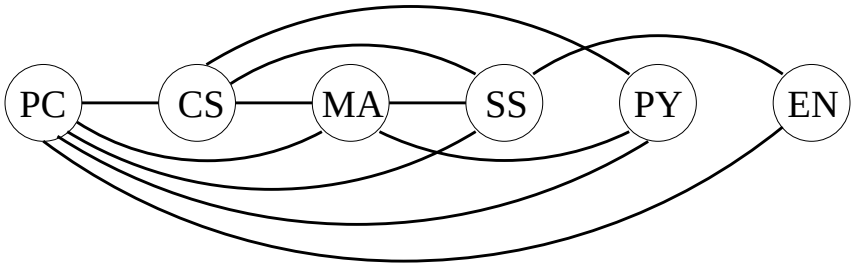

Figure 4.4: A degree ordering for the problem given in Figure 2.1 and Table 3.2.

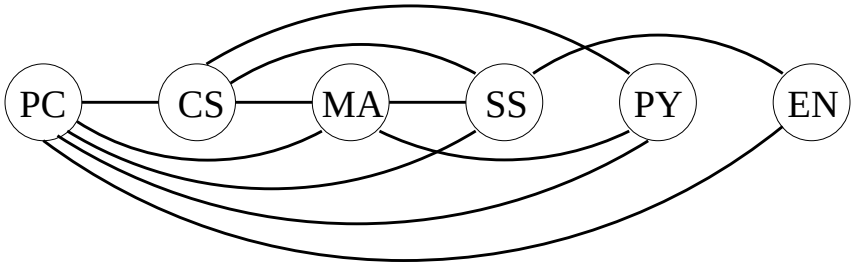

Figure 4.5: A forward degree(**FD**) ordering for the problem given in Figure 2.1 and Table 3.2. The calculations are given in Table 4.1.

		f.d. at iteration						
node	degree	1	2	3	4	5	6	order
CS	4	4	3	-	-	-	-	2
MA	4	4	3	2	-	-	-	3
PY	3	3	2	1	0	0	-	5
PC	5	5	-	-	-	-	-	1
SS	4	4	3	2	1	-	-	4
EN	2	2	1	1	1	0	0	6

Table 4.1: How the variable forward degrees (f.d.) vary as the **FD** ordering of Figure 4.5 is constructed.

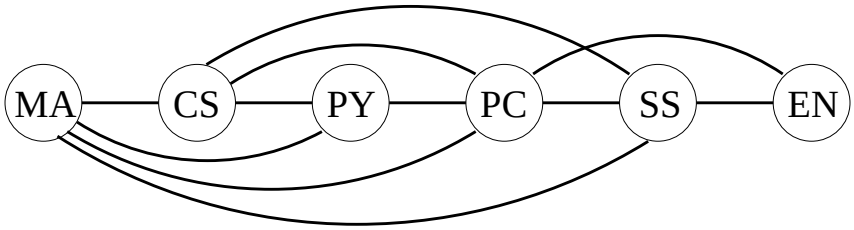

Figure 4.6: A variable weight ordering(**VAR**) for the problem given in Figure 2.1 and Table 3.2.

Table 4.2 compares the costs of the initial solutions found by the **ERR-LOW** value ordering, with the highlighted costs being the lowest for a problem. Ordering variables according to variable weight (the **VAR** column) always results in a lower cost solution in comparison with degree and forward degree orderings in the results shown. This observation is also backed up by search with the related stochastic algorithm (**ERR-RND**) where mean solution costs are compared using the Z test. Degree ordering results in a lower initial solution cost than forward degree ordering

problem	variable ordering		
	DEG	FD	VAR
HEC-S-92	23984	22890	**19455**
STA-F-83	63678	66541	**61971**
KING-96	15880	16258	**13557**
YOR-F-83	22512	22589	**21067**
UTE-S-92	41860	51098	**39522**
EAR-F-83	29419	31273*	**27469***
TRE-S-92	27375	30345*	**24602***
LSE-F-91	27100	28004*	**22355**
KFU-S-93	64902	65845	**52014**

Table 4.2: Comparison of initial solutions found by the **ERR-LOW** value ordering for various problems and variable orderings. Solutions marked with a * are those for which search required re-assignments to find a solution. The lowest solution costs for each problem are given in **bold**.

for all but three of the problems. In the exceptional cases the costs cannot be distinguished at the 5% level of significance.

By observation successful searches require very similar numbers of consistency checks as, in most cases, no backtracks are performed and each constraint is only referred to once and the number of values pruned across the whole search is similar for different variable orderings. These consistency checks(there are several per assignment because value ordering requires that several assignments be tested) are performed when the last of the constraint variable, as specified by the variable ordering, is assigned a value. Therefore, measures of the search effort have not been compared for the high domain size.

A general analysis has been performed for each problem across the three variable orderings. The critical constraints are defined as those constraints which incur the most error. Features of the constraints in the critical constraint set are compared for the three different orderings on the **HEC-S-92**, **YOR-F-83**, and **UTE-S-92** problems in Table 4.3. Several properties of these set have been compared across the three variable orderings (defined by the *var. ord.* column). The error incurred by the constraints in this set (labelled *error*) indicates whether these constraints are satisfied better from one ordering to the next. The mean and standard deviation of the constraint backward degrees, defined as formula 3.13 in Chapter 3, are also given (*mean b.d.* and *std.dev. b.d.*). Previous related assignments prune the domains of

problem	var. ord.	error	mean b.d.	std.dev. b.d.
HEC-S-92	DEG	4889.09	27.36	8.81
	FD	4502.86	23.5	9.44
	VAR	3427.49	11.86	7.87
YOR-F-83	DEG	1886.52	38.33	8.56
	FD	1984.11	38.21	8.51
	VAR	1582.84	22.94	8.44
UTE-S-92	DEG	8331.8	18.47	6.56
	FD	11589.7	29.2	10.47
	VAR	6827.33	7.2	3.34

Table 4.3: Features of the critical constraint set for the **HEC-S-92**, **YOR-F-83** and **UTE-S-92** problems over the **DEG**, **FD** and **VAR** variable orderings. The total error along with the mean and standard deviation of distribution of constraint backward degrees are recorded. These 3 problems instances show results characteristic of all instances from the problem set used.

the constraint variables so that a large constraint backward degree increases the likelihood that the constraint will incur a large error. However, the amount of domain pruning that occurs cannot be determined without performing search, so the constraint backward degree has been used as a substitute.

The differences in error incurred by the critical constraints correspond to the differences in overall solution cost, with the ordering of variable ordering superiority being the same. This is particularly notable for the big differences between the orderings on the **UTE-S-92** problem.

Where the error incurred by the critical constraints is lower for one variable ordering than another, there is a similar relationship in the mean backward degree of the constraints in the two sets. The Z test has been applied to establish, for each problem, where the difference in constraint backward degree is significant between variable orderings. This is so for all but the **DEG** and **FD** orderings respectively of the **HEC-S-92** and **YOR-F-83** problems. The lower error may correspond to a reduction in value pruning of the domains of the critical constraint variables, which leads to the lower mean constraint backward degree. However, it could also be the case that when the error is lower, the lower mean backward degree is a result of different constraints being in the critical constraint set. To eliminate this possibility, Figure 4.7 compares the mean backward degree (on the y-axis) of the **heaviest set**. The **heaviest set** is defined as the set of constraints with the greatest weight. The mean backward degree across the whole problem may be higher(the left most side

of the graph where the heaviest set consists of all constraints). However, the mean backward degree for the heaviest constraints is lower when the total solution cost is lower. The exception is that the two heaviest constraints of the **YOR-F-83** problem have a slightly higher backward degree for the **VAR** ordering, however, the general trend remains. The fact that the heaviest constraints, on average, have a lower backward degree, with the opposite being the case for lighter constraints, concurs with the notion that it is these heavy constraints for which the **VAR** ordering improves upon **DEG** and **FD** (i.e. the critical constraint sets tend to consist of heavy constraints) because for a small set of the heaviest constraints (the right hand side of the graph) the mean backward degree of the set is lower. Consequently, if this is true, the difference in the backward degree of the critical constraint sets for each variable ordering are not as a result of each consisting of distinct sets of constraints.

The observation that **VAR** results in a lower solution cost does not necessarily apply when the domain size used by search is lower. The results in Table 4.4 show that this is in fact the case. The lower the domain size, the more important the hard constraints become. In order for search to proceed when a domain wipe-out is discovered it must first go back and undo the previous assignments that have caused the domain wipe-out (i.e. the process of re-labelling), as described in Algorithm 4. When a lower domain size is used, a domain wipe-out is more likely as there are fewer values to be removed, hence pruning of a domain is more critical. Also, there are fewer values that past assignments can take, so the probability of a variable's past neighbours being assigned all possible values increases. In order for search to continue, the past variable assignments that have caused one of the values to be pruned must be either re-assigned or re-shuffled (i.e. undone and considered later by search). Re-assignment and re-shuffling of a variable are both classed as variable unlabels as both require that the value of a previously assigned variable (or a label) is reversed. Although such moves are necessary for search to continue, they are considered bad with respect to minimising constraint error.

Search cost is also an important consideration. Table 4.4 also shows the median number of unlabels used to find a solution (ignoring searches that find no solution). A comparison between these values and the average solution error show that **VAR** finds better solutions even when it requires more unlabels than **DEG** and **FD**. Search with an ordering that requires more unlabels does not always result in a higher solution cost if the constraints involved in the unlabelling are lighter, and so generally incur less error. Equally, the re-labels may only result in a small error

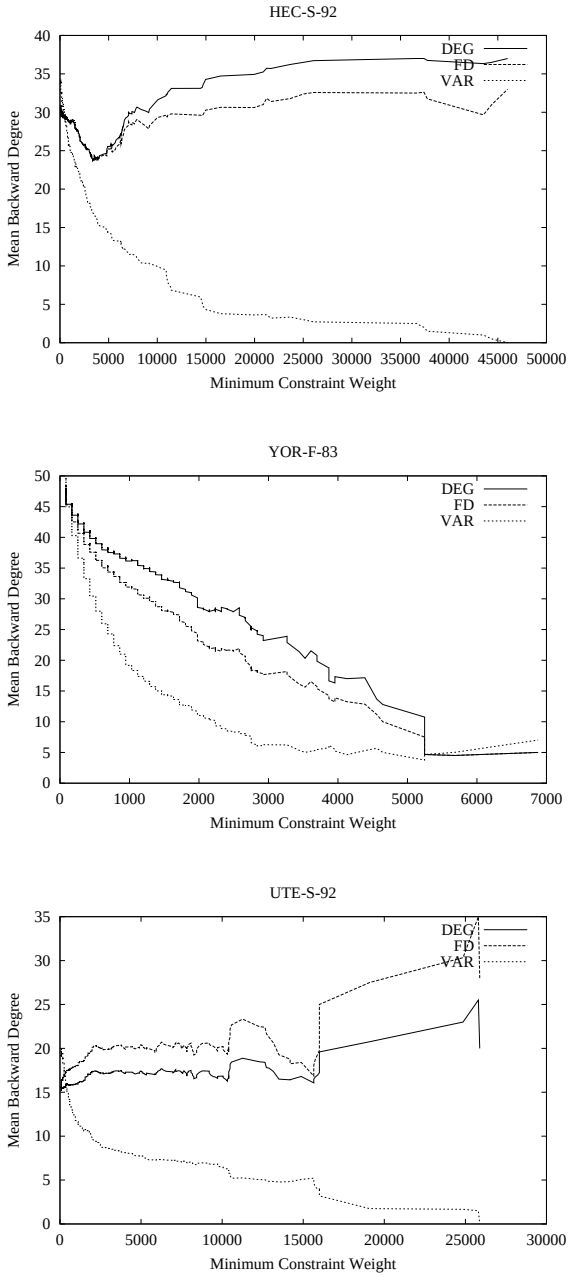

Figure 4.7: Mean constraint backward degree of the heaviest set for the **HEC-S-92**, **YOR-F-83**, and **UTE-S-92** problems. Each point on the x-axis defines the minimum weight of constraints in the heaviest set. For 0 this is the set of all constraints and for the largest constraint weight the set only contains this constraint. These results show that the constraints in the heaviest set tend to have a lower backward degree.

increase if only a few extra variables need to be considered. The same type of general analysis as used in Table 4.3, for the higher domain size, resulted in the same correlation between constraint backward degree and constraint error. When the number of unlabels is high, the constraints whose variables are the subject of re-labelling may also have an impact upon the error cost of solutions.

	mean solution cost			median # unlabels		
	variable ordering					
problem	DEG	FD	VAR	DEG	FD	VAR
HEC-S-92	43187.6	43869.8	**39123.5**	8	19	47
STA-F-83	112507	109879	**103798**	2	7	1
KING-96	34022	35245.6	**32573**	0	4	5
YOR-F-83	43022.5	43389.9	**42483.1**	61	95	77
UTE-S-92	93755.7	97102.9	**81395.7**	5	16	9
EAR-F-83	49355.3	50486.2	49394.6	11	24	34
TRE-S-92	45262.5	46405.7	**42291.3**	12	37	7
LSE-F-91	38221.7	39851.4	**35254.8**	18	63	7
KFU-S-93	96759.6	99330.5	**88439.2**	8	29	6

Table 4.4: Comparison of mean costs of solutions found and the median number of unlabels required to find them by the **ERR-RND** value ordering for various problems and variable orderings. The low domain size has been used. The bold figures show which variable ordering found significantly better solutions, to a 5% level of confidence, for each problem.

4.2.2.1 Maximum Clique

The work of Carter, *et al*, [17] states that "using a clique strategy is very useful on real problems" with reference to placing the variables of a maximum clique at the start of the variable ordering. This remark is made with reference to minimising schedule lengths and use of this strategy extends to their work on fixed schedule lengths, although a comparison of results is not given. It is known that real-world graph colouring problems, where the minimum number of colours is required, can be solved exactly at little cost by finding the maximum clique of the graph [20]. This is due to the fact that real-world graphs tend to be 1-perfect, i.e. the minimum number of colours required to colour the graph equals the number of nodes in the maximum clique of the graph. In such a case, solutions to the problem are found by assigning colours to the maximum clique and then proceeding with colouring the remaining sub-graph. Such a method does not extend itself directly to exam

	variable ordering					
problem	DEG	CLQ-DEG	FD	CLQ-FD	VAR	CLQ-VAR
HEC-S-92	23995	24096.5	23898.3	24119.4	**21756.1**	22969.8
STA-F-83	65128.1	**63164.7**	64897.8	**63873.2**	63856.1	**61513.7**
KING-96	15510.7	**15288.9**	16021.9	**15774.1**	**14531.9**	14622.6
YOR-F-83	22659.2	22742.5	23096.7	23223.6	21876.3	21888.9
UTE-S-92	49934	50422.3	**54891.8**	57125.6	**40093.3**	41101.3
EAR-F-83	**29248.5**	30923.9	**29482.2**	31003.5	**29038.1**	29837.5
TRE-S-92	26612.2	**26191.8**	27277.7	**26717.3**	**24901.6**	25343.7
LSE-F-91	25179.6	25400.6	26399.3	**25999.5**	**22632.9**	23497.2
KFU-S-93	66944.2	**65771.5**	67904.6	68537.6	**58818.7**	63174.3

Table 4.5: Mean cost of the initial solutions found using the **ERR-RND** value ordering on three (non)-clique based variable orderings. Bold type shows the lower mean cost, to 5% significance, of each pair as defined by the Z test.

timetabling because of the soft constraints, and because the domain size is fixed. However, the notion of placing a maximum clique first could still be used.

In order to test the validity of this idea, I have compared several pairs of variable orderings. Degree (**DEG**), Forward Degree (**FD**) and Variable Weight (**VAR**) orderings have been compared to equivalent orderings which place the variables of the maximum clique at the start of the order (named **CLQ-DEG**, **CLQ-FD**, and **CLQ-VAR**).

The maximum clique was found by a branch and bound method which increases the size of the clique by placing viable variables in it. When the clique cannot be expanded and the clique is the largest found so far it is stored. Then the algorithm backtracks removing the previous variable from the clique. The size of the maximum cliques can be found in Table 3.1.

Table 4.5 shows the mean solution cost results of these orderings when **ERR-RND** value ordering is used. The results show that **VAR** leads to solutions of lower cost in all but one case. With the other two orderings, starting with the maximum clique sometimes improves the solutions found and sometimes not. Overall, **VAR**, or **CLQ-VAR** in one case, remains the best ordering.

In order to establish why the use of the maximum clique can improve search in some cases while not in others a direct comparison between the clique and non-clique variable orderings has been performed.

Table 4.6 compares the pairs of orderings from Table 4.5 (ignoring indistinguishable pairs). For each problem and variable ordering (*var.ord.*); the *best* column states whether the ordering with or without the maximum clique was best (labelled *w* and *w/o* respectively); and the *size* column records the size of the critical constraint set. The mean backward degree of the critical constraints for the (non) clique based ordering is also given (*mean b.d. w/o* and *mean b.d. w*) with the bold values describing which value is significantly lower to the 5% level of confidence. These results show that for all but one case (**STA-F-83** using **DEG**) the dominant ordering of the pair has a significantly lower backward degree over the critical constraints.

When a comparison of mean solution error is performed using the solutions found for the low domain sizes the same pattern holds, ignoring indistinguishable pairs, in all but four cases. The results of this comparison are in Table 4.7. Such similarity between results over different domain sizes is not unexpected. The orderings of each pair are related so any reduction in the quality of solutions found as the domain size is lowered will be similar. Subsequently, the significant improvement required for one ordering to become superior to its counterpart is less likely. Again **VAR** outperforms its clique counterpart (**CLQ-VAR**) in the majority of cases and the superiority within **DEG** based pairs varies from one problem to another. The superiority of **FD** over **CLQ-FD** becomes apparent as the domain size is lowered. Again the use of the maximum clique does not provide enough improvement to alter the ordering of superiority between the **DEG**, **FD** and **VAR** based variable orderings.

These results counter the work in [17] which suggests that use of a maximum clique leads to lower cost solutions. It is worth considering that the experiments performed in [17] do not consider stochastic value ordering, as has been used above. The value assignment scheme used in [17] is more akin to the **ERR-LOW** search method[1]. The equivalent results to Table 4.7 for **ERR-LOW** are given in Table 4.8. The comparison is inconclusive, interestingly for the **VAR** ordering pairs for which **VAR** outperformed **CLQ-VAR** comprehensively under **ERR-RND**. When the lowest error tie-break is used the clique based ordering finds a lower cost solution in more cases than when the random error tie break is used. There are many reasons why the results might differ from those mentioned, although not given, in [17]. Even the most basic search procedures will be encoded differently by different people which could lead to minor discrepancies. As **ERR-LOW** searches only find one solution, re-trials will give exactly the same results preventing re-runs with average results to eliminate chance from the comparison. It is also worth noting that many of the

[1]From person communication with M. W. Carter.

problem	var. ord.	d.o.	size	mean b.d. w/o	mean b.d. w
HEC-S-92	DEG	w/o	10	**18.4**	29.8
STA-F-83	DEG	w	86	**16.88**	18.40
	FD	w	-	-	-
	VAR	w	10	22.3	**10.8**
KING-96	DEG	w	31	9.68	**8.39**
	FD	w	38	10.24	**8.42**
	VAR	w/o	-	-	-
UTE-S-92	FD	w/o	-	-	-
	VAR	w/o	-	-	-
EAR-F-83	DEG	w/o	87	**32.10**	35.83
	FD	w/o	112	**34.62**	38.14
	VAR	w/o	10	**17**	27.6
TRE-S-92	DEG	w	10	44.4	**13.6**
	FD	w	10	47.6	**13.4**
	VAR	w/o	10	**9.7**	22.7
LSE-F-91	FD	w	10	37.9	**21.6**
	VAR	w/o	10	**8.3**	19.5

Table 4.6: Comparison of critical constraint sets under (non)-clique variable orderings. "-" means that there is no significant difference. The *d.o.* column records which was the dominant ordering with respect to solution costs; the **bold** mean backward degree (*mean b.d.*), which of the orderings has such for the critical constraints and; the *size* defines the size of the critical constraint set require to establish this to a 5% level of confidence.

	variable ordering					
problem	DEG	CLQ-DEG	FD	CLQ-FD	VAR	CLQ-VAR
HEC-S-92	43187.6	43285.1	43869.8	**42827.8**	**39123.5**	42513.8
STA-F-83	112507	**106200**	109879	**106233**	103798	**102513**
KING-96	**34022**	34684.9	**35245.6**	35647.5	**32573**	32810.7
YOR-F-83	**43022.5**	43637.9	**43389.9**	44014.2	**42483.1**	43417.2
UTE-S-92	93755.7	93194.5	**97102.9**	98159.4	**81395.7**	82452
EAR-F-83	**49355.3**	51434.3	**50486.2**	52469.5	**49394.6**	51406
TRE-S-92	45262.5	**44868.4**	**46405.7**	47134.8	**42291.3**	44609.2
LSE-F-91	**38221.7**	38664.2	39851.4	39449.5	**35254.8**	36903.5
KFU-S-93	**96759.6**	99252.4	**99330.5**	101649	**88439.2**	95577.7

Table 4.7: Mean cost of the initial solutions found using the **ERR-RND** value ordering on three (non)-clique based variable orderings using the low domain size. Bold type shows the lower mean cost, to 5% significance, of each pair as defined by the Z test.

	variable ordering					
problem	DEG	CLQ-DEG	FD	CLQ-FD	VAR	CLQ-VAR
HEC-S-92	**42520**	46199	**43458**	44454	**41473**	52712
STA-F-83	108173	**102828**	109436	**103974**	111534	**101964**
KING-96	35660	**33062**	**33667**	35116	33058	**31788**
YOR-F-83	44846	**43351**	44874	**44046**	43330	**41253**
UTE-S-92	100199	**92570**	**89319**	98326	83845	**81216**
EAR-F-83	**50983**	54678	**50715**	53905	**47862**	51866
TRE-S-92	**44770**	50165	**42551**	48606	**43333**	43451
LSE-F-91	**39542**	42019	39019	**38321**	**36992**	44915
KFU-S-93	-	102519	95733	**93642**	98416	**93859**

Table 4.8: Mean cost of the initial solutions found using the **ERR-LOW** value ordering on three (non)-clique based variable orderings as performed on the lower problem domain sizes.

problems have more than one maximum clique and which of these is used in the ordering may result in both positive and negative effects. In this investigation the first maximum clique found was used. The **ERR-LOW** results in Table 4.8 could be expected to fall below the equivalent **ERR-RND** mean in Table 4.7 (see Section 5.3.5 in the next chapter). This is not the case in several instances.

The critical constraint backward degree has been calculated with reference to the original variable ordering (i.e. the effect of re-assignments on the ordering has not been considered). This is in keeping with the aim of analysing post-search information to generalise the effects of the decisions made before search begins. Table 4.9 repeats the process of direct comparison between the variable ordering pairs (as performed in Table 4.6). It is not clear from these results whether the deductions made in previous sections apply. As the use of a maximum clique was superior for only 5 instances, little knowledge can be gained.

Table 4.10 compares the number of unlabels required by search for each pair of variable orderings. As with the error cost of the solutions found, the use of a maximum clique can often improve search with respect to the number of unlabels required by search. Using **FD**, which in previous experiments usually requires more unlabels than **DEG** or **VAR**, an improvement can be achieved, most notably the large improvement for the **YOR-F-83** problem. For the **DEG** and **VAR** orderings there is no consistent pattern. Whether the use of a maximum clique reduces or increases the number of unlabels required by search would appear, from these results,

problem	var. ord.	d.o.	size	mean b.d. w/o	mean b.d. w
HEC-S-92	FD	w	14	**26.29**	30.14
	VAR	w/o	10	**10.9**	20.9
STA-F-83	DEG	w	125	**19.50**	20.90
	FD	w	63	**19.35**	21.13
	VAR	w	77	**15.82**	17.44
KING-96	DEG	w/o	42	**8.10**	9.07
	FD	w/o	16	**9**	10.75
	VAR	w/o	-	-	-
YOR-F-83	DEG	w/o	10	**16.4**	30.6
	FD	w/o	10	**16.6**	29.7
	VAR	w/o	10	**9.2**	23.1
UTE-S-92	FD	w/o	-	-	-
	VAR	w/o	-	-	-
EAR-F-83	DEG	w/o	15	**23.6**	29.33
	FD	w/o	10	**19.6**	26.2
	VAR	w/o	10	**11.6**	29
TRE-S-92	DEG	w	10	40.1	**18.1**
	FD	w/o	38	38.24	**30.79**
	VAR	w/o	10	**9**	27.3
LSE-F-91	DEG	w/o	10	**12.9**	24
	VAR	w/o	10	**9.5**	24.5
KFU-S-93	DEG	w/o	52	**33.10**	36.23
	FD	w/o	39	**38.33**	42.54
	VAR	w/o	10	**18.5**	30.7

Table 4.9: Analysis of the mean backward degrees (*mean b.d.*) of the critical constraints for each variable ordering pair. *d.o.* records which is the dominant ordering with respect the the solutions found. The **bold** type emphasises for which ordering the *mean b.d.* is significantly lower.

	variable ordering					
problem	DEG	CLQ-DEG	FD	CLQ-FD	VAR	CLQ-VAR
HEC-S-92	8	2	19	18	47	38
STA-F-83	2	0	7	2	1	0
KING-96	0	0	4	2	5	3
YOR-F-83	61	49	95	76	77	83
UTE-S-92	5	3	16	10	9	5
EAR-F-83	11	14	24	30	34	42
TRE-S-92	12	7	37	22	7	8
LSE-F-91	18	6	63	13	7	7
KFU-S-93	8	10	29	30	6	9

Table 4.10: Median number of unlabels required to find a solution for each variable ordering pair.

to be problem (or institution) specific, rather than depending upon the variable ordering.

4.2.2.2 Summary

This section of work has shown that a static variable ordering based upon variable weight out-performs degree based orderings. The conclusion is based upon samples of 100 searches for each problem, so although only 9 problem instances have been used the conclusion should extend to other problem instances. It also shows that the use of a maximum clique at the start of the variable ordering can usually improve search in terms of the amount of search effort. The results have also shown that the use of a maximum clique in variable ordering can be at the expense of solution cost, most notably when the domain size is lowered, although the superiority/inferiority of the underlying variable ordering is usually maintained. The analysis of each solution set has shown that the lower solution cost is linked to a lower backward degree of the constraints that incur the most error.

4.3 The Effect of Edge Weights

Problem features such as variable degrees and cliques are commonly used to define variable orderings when solving constraint satisfaction problems. Such notions can also be extended to take advantage of the constraint weights used in the Weighted CSP model. An example of such an extension is Weighted Degree Order (also called Largest Weighted Degree [17] or Generalised Largest First [40]). The weighted degree of a variable is the sum of the weights of the edges related to the variable, i.e. the weight of the constraints involving the variable.

It is reasonable to assume that, in real world problems, variables with a large weighted degree will also tend to have a large degree. In addition to the definitions used in Section 3.1, the weighted degree of an exam (variable), where S_i is the set of course selections for student i, is defined as:

$$wd(x) = \sum_{\forall S_i \in S \, , \, x \in S_i} (\mid S_i \mid -1) \tag{4.1}$$

The weighted degree ($wd(x)$ defined in 4.1) and degree ($d(x)$ defined in 3.8) of a variable are both concerned with the set of students that take a particular exam (x). The larger the size of some of the student combinations ($\mid S_i \mid$) in 4.1, the larger $wd(x)$ is. An increase in the number of exams taken by each student will tend to increase $d(x)$ because it will tend to increase the variable neighbourhood size (N_x 3.7, i.e. the number of other exams an exam cannot clash with), and hence the degree.

However, there is no direct correspondence between weighted variable degrees and variable degrees, because increases in either the size of student combinations or variable weight do not directly lead to an increase in $d(x)$ (see Figure 4.8). In the real world, different areas of the problem can have slightly different properties. For example, students from different departments of an institution may have different numbers of courses to choose from. Some departments may have fewer students, yet still maintain a large set of possible course choices. Figure 4.9 shows this relationship in a real world problem. There is a clear correlation between the Degree (on the x-axis) and the Weighted Degree (on the y-axis) with a curve leading from the bottom left hand corner to the top right hand corner of the Figure. However, the data also shows that a large weighted degree does not necessarily imply a large degree. The degrees of the variables with large weighted degrees (greater than 600) are anywhere in the region of 40 to 120.

Problem 1	Problem 2
$S_1 = \{c_1, c_2, c_3\}$	$S_1 = \{c_1, c_2, c_3, c_4, c_5\}$
$S_2 = \{c_1, c_4, c_5\}$	$S_2 = \{c_1, c_2, c_4, c_5\}$
$S_3 = \{c_1, c_6\}$	$S_3 = \{c_1, c_4, c_5, c_6\}$
	$S_4 = \{c_1, c_2, c_3\}$
$wd(c_1) = 5$	$wd(c_1) = 12$
$d(c_1) = 5$	$d(c_1) = 5$

S_i is the set of course choices (from courses c_1 to c_6) for student i.

$d(c_1)$ is the same in both problems, despite the fact that $wd(c_1)$ is larger in the right hand problem.

Figure 4.8: Two example problems.

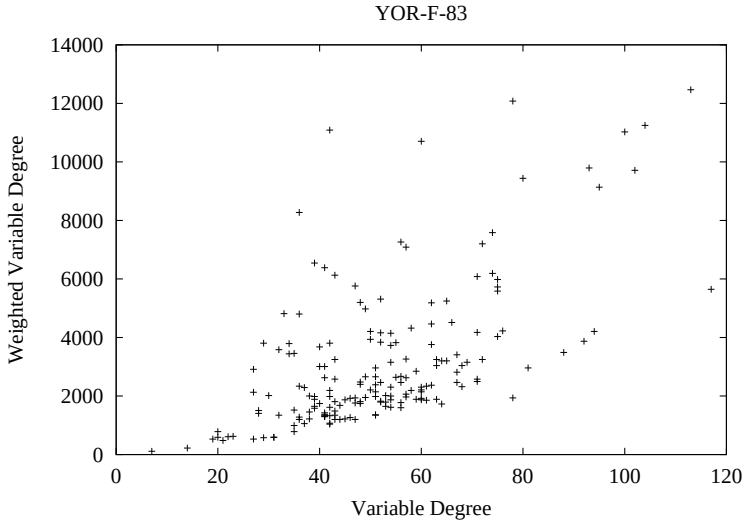

Figure 4.9: Degree versus Weighted Degree in the **YOR-F-83** problem

I intend to examine the relationship between variable orderings based upon other problem features and the corresponding orderings which incorporate edge weights.

4.3.1 Weighted Degrees

The two pairs of orderings that shall be compared are:

- degree ordering (**DEG**) and weighted degree ordering (**WD**).
- forward degree ordering (**FD**) and weighted forward degree ordering (**WFD**)

Examples of these two variable orderings can be found in Figures 4.10 and 4.11.

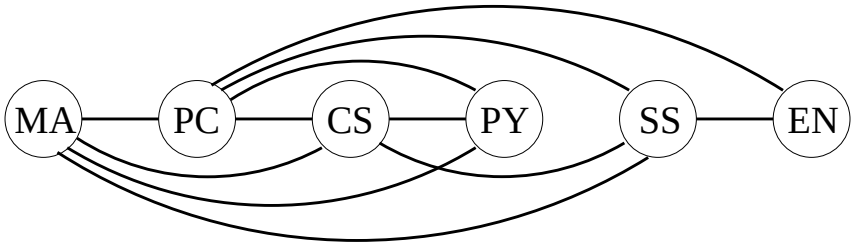

Figure 4.10: A weighted degree ordering(**WD**) for the problem given in Figure 2.1 and Table 3.2.

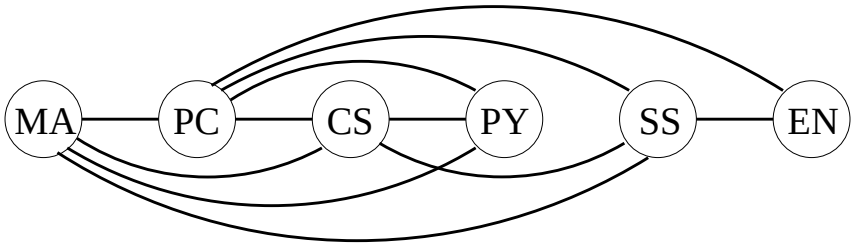

Figure 4.11: A weighted forward degree ordering(**WFD**) for the problem given in Figure 2.1 and Table 3.2. The calculations are given in Table 4.11.

	variable ordering			
problem	DEG	WD	FD	WFD
HEC-S-92	23995	**21216.8**	**23898.3**	24735.5
STA-F-83	65128.1	**62875.7**	64897.8	**63774.4**
KING-96	15510.7	**14558.1**	16021.9	**15885.8**
YOR-F-83	22659.2	**21922.5**	23096.7	**22570.4**
UTE-S-92	49934	**39433.2**	54891.8	**44710.4**
EAR-F-83	29248.5	29004.2	**29482.2**	29949.9
TRE-S-92	26612.2	**24684.8**	27277.7	27142
LSE-F-91	25179.6	**22228.4**	26399.3	**24421.6**
KFU-S-93	66944.2	**60026.4**	67904.6	**64713.5**

Table 4.12: Mean solution cost when comparing (forward) degree and weighted (forward) degree using the **ERR-RND** value ordering and high domain sizes. Bold type shows the lower mean cost, to 5% significance, of each pair as defined by the Z test.

		w.f.d. at iteration						
node	weighted degree	1	2	3	4	5	6	ord
CS	10	10	6	4	-	-	-	3
MA	12	12	-	-	-	-	-	1
PY	8	8	4	2	0	0	-	5
PC	12	12	9	-	-	-	-	2
SS	8	8	7	4	2	-	-	4
EN	4	4	4	2	2	0	0	6

Table 4.11: How the variable weighted forward degrees (w.f.d.) vary as the **WFD** ordering of Figure 4.11 is constructed.

Table 4.12 compares the initial solutions found using these variable orderings using the **ERR-RND** value ordering. The superiority of **DEG** over **FD** extends to the equivalent weighted orderings where **WD** outperforms **WFD**. Search using **WD** clearly outperforms **DEG** and **FD** is outperformed by **WFD** on all but the **HEC-S-92** and the **EAR-F-83** problems. To find out why this is the case, a direct comparison between the solutions found by each pair of variable orderings has been performed.

As with the comparison method in 4.2.2.1, the critical constraint set is defined as those constraints that offer the most improvement under the dominant ordering of the pair. The initial critical constraint set size is set to 10. Table 4.13 compares the mean backward degree of constraints in the critical constraint set for the (weighted)

degree orderings. The *mean b.d.* is significantly lower for **WD**. This is as expected, as a lower constraint backward degree generally implies that a constraint variable will be pruned less, which, in turn, generally implies less error will be incurred by a constraint. Table 4.14 shows the same results when the same method is applied to the comparison of the (weighted) forward degree orderings. Again the ordering which results in a significantly lower constraint backward degree, on average over the critical constraints, corresponds to the dominant ordering, which in this case does vary.

prob	d.o.	size	DEG b.d.	WD b.d.
HEC-S-92	WD	10	28.9	**8.8**
STA-F-83	WD	10	16.2	**7.4**
KING-96	WD	10	8.2	**3.5**
YOR-F-83	WD	10	35.6	**9.6**
UTE-S-92	WD	10	21.4	**6.6**
TRE-S-92	WD	10	40.4	**17.2**
LSE-F-91	WD	10	38.5	**16.5**
KFU-S-93	WD	10	31.7	**10.4**

Table 4.13: Comparison of the mean constraint backward degree, for (weighted) degree variable orderings (column *DEG/WD b.d.*), of constraints in the critical set of *size*. The column *d.o.* indicates which ordering was dominant. **Bold** values show which mean constraint backward degree was significantly lower for each variable ordering pair.

prob		size	FD b.d.	WFD b.d.
HEC-S-92	FD	14	**25.71**	33.93
STA-F-83	WFD	10	17	**4.5**
KING-96	WFD	10	9.8	**4.7**
YOR-F-83	WFD	10	34.6	**16.4**
UTE-S-92	WFD	10	33.3	**6.6**
EAR-F-83	FD	218	**39.13**	41.43
LSE-F-91	WFD	10	39.9	**16**
KFU-S-93	WFD	10	30.6	**14.6**

Table 4.14: Comparison of the mean constraint backward degree, for (weighted) forward degree variable orderings, of constraints in the critical size.

Table 4.15 compares the mean solution costs found when searching on problems using the lower domain sizes. Again **WD** outperforms the related un-weighted

	variable ordering			
problem	DEG	WD	FD	WFD
HEC-S-92	43187.6	**38716**	43869.8	43170.7
STA-F-83	112507	**103919**	109879	**103968**
KING-96	34022	**33802.8**	35245.6	**34944.8**
YOR-F-83	43022.5	**42556.3**	43389.9	**42584.6**
UTE-S-92	93755.7	**82348.1**	97102.9	**88109.9**
EAR-F-83	49355.3	49844.9	50486.2	50357.8
TRE-S-92	45262.5	**41784.6**	46405.7	**45513.5**
LSE-F-91	38221.7	**34896.2**	39851.4	**37455.9**
KFU-S-93	96759.6	**89562**	99330.5	**97681.4**

Table 4.15: Mean solution cost when comparing (forward) degree and weighted (forward) degree using the **ERR-RND** value ordering and lower domain sizes. Bold type shows the lower mean cost, to 5% significance, of each pair as defined by the Z test.

ordering (**DEG**) with the exception of the **EAR-F-83** problem when the solution costs are indistinguishable at the 5% level of confidence. This is the same pattern that occurs when searching using the high domain size. The **WFD** ordering shows a general superiority over **FD**; yet again they are indistinguishable in two cases. Table 4.16 gives the number of unlabels required by search for each problem/variable ordering combination. As with the comparison between **DEG** and **WD**, a high number of unlabels required by search, when using the lower domain values, result in search considering heavier constraints first, which may not be the best strategy as this will lead to unlabelling of variables in the scope of these constraints. Between **DEG** and **WD** neither ordering is superior over all the problems with respect to unlabels required by search. The use of **WD** over **DEG** is justifiable provided a small increase in search is acceptable. An analysis between **FD** and **WFD** shows that **WFD** requires fewer unlabels in the majority of cases. Coupled with the lower solution costs resulting from using **WFD** this shows **WFD** can be considered to be superior to **FD**.

The direct comparison method is repeated in Tables 4.17 and 4.18, where the weighted orderings (**WD** and **WFD**) prove to have lower backward degrees for the defined critical constraints. This reinforces the explanation of the superiority of the weighted orderings.

	variable ordering			
problem	DEG	WD	FD	WFD
HEC-S-92	8	33	19	33
STA-F-83	2	1	7	3
KING-96	0	3	4	3
YOR-F-83	61	74	95	86
UTE-S-92	5	8	16	20
EAR-F-83	11	35	24	57
TRE-S-92	12	7	37	34
LSE-F-91	18	6	63	39
KFU-S-93	8	8	29	21

Table 4.16: Median number of unlabels required by successful **ERR-RND** searches on problems using the lower domain sizes.

problem	d.o.	size	DEG b.d.	WD b.d.
HEC-S-92	WD	10	31.5	**6.8**
STA-F-83	WD	10	14.7	**6.2**
KING-96	WD	10	8.9	**4.6**
YOR-F-83	WD	10	39	**10.1**
UTE-S-92	WD	10	21.9	**6**
TRE-S-92	WD	10	41.3	**9.5**
LSE-F-91	WD	10	37.4	**12.6**
KFU-S-93	WD	10	30.3	**7.1**

Table 4.17: Direct comparison of the mean constraint backward degree, for (weighted) degree variable orderings (*DEG b.d.* and *WD b.d.*), of constraints in the critical set of *size*. *d.o.* records which ordering was dominant with respect to solution costs. The solutions examined are those found when searching using the lower domain sizes.

problem	d.o.	size	FD b.d.	WFD b.d.
HEC-S-92	WFD	10	29.5	**12.6**
STA-F-83	WFD	10	16.5	**3.8**
KING-96	WFD	10	7.8	**2.8**
YOR-F-83	WFD	10	29	**8.7**
UTE-S-92	WFD	10	26.8	**4.3**
TRE-S-92	WFD	10	49.8	**15.4**
LSE-F-91	WFD	10	40.2	**15.1**
KFU-S-93	WFD	10	29.8	**15.6**

Table 4.18: Direct comparison of the mean constraint backward degree, for (weighted) forward degree variable orderings, of the critical constraints. The solutions examined are those found when using the lower domain sizes.

4.3.2 Maximum Weighted Cliques

The weight of a weighted clique is defined as the weight of all of the variables within it. The motivation behind placing a maximum weighted clique, defined as a clique of any size which has a maximum weight, at the start of the ordering is not just that of assigning values to variables that may encounter a large amount of pruning (as with un-weighted cliques of section 4.2.2.1), but also that the weighted clique variables may be critical to the error of the resulting solution. I have compared weighted and un-weighted cliques to develop an insight as to whether this is the case.

Table 4.19 compares **DEG**, **FD** and **VAR** with related orderings which either have a maximum clique, or a maximum weighted clique at the start of the ordering for stochastic search (**ERR-RND**). Some of the pairs of orderings are indistinguishable from each other at the 5% level of confidence, for example all of the **HEC-S-92** results. Where this is the situation the pairs of distributions are considered to be too similar to make any accurate judgements. Those mean errors that are significantly lower have been highlighted.

	variable ordering pairs					
problem	CLQ-DEG	WCLQ-DEG	CLQ-FD	WCLQ-FD	CLQ-VAR	WCLQ-VAR
HEC-S-92	24096.5	24032.9	24119.4	24423.4	22969.8	23139.6
STA-F-83	**63164.7**	65441.9	**63873.2**	65705	**61513.7**	62762.7
KING-96	15288.9	**14917.3**	15774.1	**15502.3**	14622.6	14537.2
YOR-F-83	**22742.5**	22940.3	**23185.2**	23421.5	**21888.9**	22145.9
UTE-S-92	50422.3	**44343.1**	57125.6	**51271.2**	**41101.3**	42003.4
EAR-F-83	30923.9	**30613.7**	31003.5	30821.3	29837.5	30005.6
TRE-S-92	26191.8	26135.4	26717.3	26671.9	25343.7	**25124.4**
LSE-F-91	25400.6	**25053.9**	25999.5	25834	23497.2	**23132.8**
KFU-S-93	**65771.5**	69495.1	**68537.6**	70517.7	63174.3	**60794**

Table 4.19: Mean solution costs when comparing the (weighted) clique based orderings using the **ERR-RND** value ordering heuristic. Bold type shows the lower mean cost, to 5% significance, of each pair as defined by the Z test.

The results of Table 4.19 show that neither of the **CLQ** or **WCLQ** orderings is consistently better for all problems or all base variable orderings. Examining the superiority across each problem, however, shows that for some problems **CLQ** is superior to **WCLQ** whether combined with **DEG**, **FD**, or **VAR**, and for some problems the opposite is the case.

prob	var.ord.	d.o.	#	CLQ b.d.	WCLQ b.d.
STA-F-83	DEG	CLQ	10	**8.2**	12.3
	FD	CLQ	10	**6.6**	10.9
	VAR	CLQ	10	**4.9**	9.9
KING-96	DEG	WCLQ	10	9.5	**7**
	FD	WCLQ	12	8.33	**6.42**
YOR-F-83	DEG	CLQ	10	33.1	**18**
	FD	CLQ	10	32.2	**18**
	VAR	CLQ	11	24.64	**16.82**
UTE-S-92	DEG	WCLQ	10	20.1	**14.2**
	FD	WCLQ	10	29.8	**22.5**
	VAR	CLQ	-	-	-
EAR-F-83	DEG	WCLQ	10	46.7	**29.8**
TRE-S-92	VAR	WCLQ	18	26.72	**22.39**
LSE-F-91	DEG	WCLQ	-	-	-
	VAR	WCLQ	-	-	-
KFU-S-93	DEG	CLQ	10	34.4	**22.4**
	FD	CLQ	10	31.5	**20**
	VAR	WCLQ	24	25.79	**19.33**

Table 4.20: Direct comparison of variables orderings which incorporate maximum and weighted maximum cliques when using the high domain size. *d.o.* records which ordering is dominant with respect to solution cost. *CLQ b.d.* and *WCLQ b.d.* record the mean backward degree of the critical constraints, whose number is defined by size, under each ordering. The **bold** type is used to show which mean *b.d.* is significantly lower.

This suggests that superiority of the ordering has more to do with the actual clique used than the resulting order of the remaining variables. The results of a direct analysis between the solutions found by each variable ordering pair of Table 4.19 are shown in Table 4.20.

Of the 3 problems where a maximum clique is superior to a weighted maximum clique it is for only one of these that the critical constraints in the direct comparison (Table 4.20) have a lower *mean b.d.* for the maximum clique orderings. This suggests that the variable orderings are too similar for the critical constraint definition to clearly identify critical constraints, a hypothesis which is backed up by the number of statistically indistinguishable pairs in Table 4.19.

	variable ordering pairs					
problem	CLQ-DEG	WCLQ-DEG	CLQ-FD	WCLQ-FD	CLQ-VAR	WCLQ-VAR
HEC-S-92	43285.1	**42113.6**	42827.8	42772.4	42513.8	43346.4
STA-F-83	**106200**	111933	**106233**	109803	**102513**	103572
KING-96	34684.9	**33014.7**	35647.5	**34741.5**	32810.7	**32403.5**
YOR-F-83	43637.9	43481.7	44014.2	44150.1	43417.2	**42614**
HEC-S-92	93194.5	**89662.3**	98159.4	**93873.2**	**82452**	88031.7
STA-F-83	51434.3	51093.5	52469.5	**51880.5**	51506	**51004.5**
KING-96	44868.4	**44519.2**	47134.8	**45723.4**	44609.2	**42828.4**
YOR-F-83	**38664.2**	39110.3	**39449.5**	40010.6	36903.5	**36221.4**
UTE-S-92	99252.4	99856.2	101649	102396	95577.7	**91038.1**

Table 4.21: Mean error of solutions found using the **ERR-RND** value ordering on the variable orderings that incorporate maximum and maximum weighted cliques using the lower domain sizes. Bold type shows the lower mean cost, to 5% significance, of each pair as defined by the Z test.

	variable ordering pairs					
problem	CLQ-DEG	WCLQ-DEG	CLQ-FD	WCLQ-FD	CLQ-VAR	WCLQ-VAR
HEC-S-92	2	5	18	16	38	85
STA-F-83	0	2	2	11	0	1
KING-96	0	0	2	3	3	4
YOR-F-83	49	46	76	75	83	65
UTE-S-92	3	2	10	7	5	7
EAR-F-83	14	13	30	28	42	43
TRE-S-92	7	8	22	27	8	7
LSE-F-91	6	7	13	11	7	5
KFU-S-93	10	12	30	27	9	6

Table 4.22: Number of unlabels required by search using the variable orderings that incorporate maximum and maximum weighted cliques using the lower domain sizes.

The same two experiments have been carried out to compare maximum weighted clique ordering with their maximum clique when searching under the lower domain sizes. Table 4.21 compares the mean solution costs of each **CLQ-WCLQ** pair. The pattern of superiority remains the same for all but 2 pairs and there are also more distinguishable pairs. For each general ordering (**DEG**, **FD** or **VAR**) there are only two problems for which **CLQ** out-performs **WCLQ**. These results show that using a maximum weighted clique in a variable ordering will lead to lower cost

prob	var.ord.	d.o.	#	CLQ b.d.	WCLQ b.d.
HEC-S-92	DEG	WCLQ	10	34.1	**13.1**
STA-F-83	DEG	CLQ	525	16.92	**16.31**
	FD	CLQ	340	16.92	**16.18**
	VAR	CLQ	10	**6.4**	10
KING-96	DEG	WCLQ	10	9.5	**7.1**
	FD	WCLQ	61	9.11	**7.98**
	VAR	WCLQ	10	8.1	**5.3**
YOR-F-83	VAR	WCLQ	10	24.8	**20.5**
UTE-S-92	DEG	WCLQ	10	20.6	**12.3**
	FD	WCLQ	10	32.7	**16.4**
	VAR	CLQ	10	**7.4**	9.6
EAR-F-83	FD	WCLQ	14	40.57	**32**
	VAR	WCLQ	10	36	**27.5**
TRE-S-92	DEG	WCLQ	10	33.5	**20.7**
	FD	WCLQ	10	29.8	**20.8**
	VAR	WCLQ	12	25.83	**20.83**
LSE-F-91	DEG	CLQ	-	-	-
	FD	CLQ	-	-	-
	VAR	WCLQ	-	-	-
KFU-S-93	VAR	WCLQ	12	21.17	**13.5**

Table 4.23: Direct comparison between solutions found using **ERR-RND** and variable orderings that incorporate maximum and weighted maximum cliques using the lower domain sizes. *d.o.* records which ordering is dominant with respect to solution cost. *CLQ b.d.* and *WCLQ b.d.* record the mean backward degree of the critical constraints, whose number is defined by size, under each ordering. The **bold** type is used to show which mean *b.d.* is significantly lower.

solutions. With respect to the number of unlabels required by search, Table 4.22 shows that related **WCLQ** and **CLQ** require a similar number of unlabels. This is to be expected due to the similarity between the orderings. A direct comparison between the ordering pairs (Table 4.23) shows a lower critical constraint set *mean b.d.* for the superior ordering for all but 2 comparisons when **CLQ** is the dominant ordering. Little analysis can be done into this as there are only 4 clear comparisons where **CLQ** is dominant.

4.3.2.1 Summary

The results of this section have shown that the solutions found by placing a maximum weighted clique at the start of the variable ordering can improve upon the solutions

found when using a maximum clique. This is most notable for the low domain sizes. However, little information can be gained from this conclusion as the resulting variable orderings are very similar and the superiority of orderings which use a clique over related non-clique based orderings, in general, remains the same regardless of the clique type. Thus the use of maximum weighted cliques over maximum cliques in variable ordering has little effect.

4.3.3 Weighted Clique Partitions

A **clique partition** of a graph is a set of cliques such that each node of the graph is a member of exactly one clique of the set. Clique partitions come from the field of graph theory, where the problem of minimising the number of cliques in the partition is related to graph colouring. Previous results have shown that placing the maximum clique at the start of the variable ordering can aid search with respect to finding a solution of low error cost. This idea can be extended so that once search progresses past the clique at the start of the ordering, the method is re-applied to the subsequent sub-problem. Continued application leads to a clique covering. This clique covering will have large cliques at the start and cliques of size 1 toward the end of the ordering. A **weighted clique partition** of a graph extends this definition to consider the weights of nodes or edges in some way. The drawback to using a variable ordering based upon a clique covering is that finding a maximum clique is a NP-complete problem. Finding the maximum clique of the original problem may be justifiable if a significantly better solution can be found; however, re-applying this process a number of times requires even more time. This time may be better spent running more stochastic searches using more conventional variable orderings.

I have implemented a method that finds a clique covering using a sequential (or greedy) method. The advantage is that the cliques can be found quickly. However, there is no guarantee of the quality of the cliques found in comparison to the maximum weight possible and the starting variable used is critical. I have initially ordered the variables by decreasing degree. The method is described in Algorithm 5.

Algorithm 5 Find clique partition

```
1  begin
2    U := {1, 2, ..., n}
3    while U ≠ {}
4          do
5          degree_order(U)
6          C := {}
7          for v := 1 to | U |
8              do
9              x := u_v , u_v ∈ U
10             in_clique := FALSE
11             for i := 1 to | C |
12                 do
13                 if !in_clique
14                    then
15                          add_to_clique := TRUE
16                          for j := 1 to | C_i |
17                              do
18                              if !E(x, C_{i,j}) then add_to_clique := FALSE fi
19                              if add_to_clique
20                                 then
21                                       C_i ∪ x
22                                       in_clique := TRUE
23                              fi
24                          od
25                 fi
26             od
27             if !in_clique
28                then
29                     i :=| C | +1
30                     C_i := {x}
31             fi
32         od
33         best := 1
34         for j := 1 to | C |
35             do
36             if W(C_j) > W(C_best) then best := j fi
37         od
38         U − C_best
40         for j := n− | U | +1 to n− | U | + | C_best |
41             do
42             o_{max C_best} := j
43             C_best − max C_best
44         od
45   od
46 end
```

The set U, which is first defined in line 2, is the set of variables which have yet to be placed in the variable ordering. The variables in U are ordered by degree in line 5 and considered in turn. For each of these variables, $\{u_1, ..., u_v\}$, an attempt is made to put the current one, x (line 9), in a clique. The *in_clique* variable maintains whether this can be done for each of the cliques in the current clique set C (line 6). Lines 11 to 26 loop round the clique set; line 13 ensures that the current variable is only added to one clique; and lines 15 to 24 attempt to add the current variable to the current clique C_i. In order for the current variable to be added to the current clique it must be connected, $E(x, y)$ must be true (line 18), to each variable in the current clique (the loop at line 16). If the current variable cannot be added to any clique then lines 27 to 31 create a new clique. Lines 33 to 37 find the "best" clique by the clique weight, defined later, as C_{best}. The variables of this clique are then removed from the unordered variables U (line 38) and added to the variable ordering $o_1, ..., o_n$ in lines 40 to 44. After the best clique has been added to the ordering the whole process is repeated again, until all of the variables have been ordered and U is an empty set (line 3).

Table 4.24 compares the solutions found using the clique partition ordering (**MV** as the **m**aximal cliques are weighted by **v**ertex weight) with those found using **VAR** with a maximum weighted clique at the start (**WCLQ-VAR**). A maximum weighted clique ordering was used for comparison because it uses the same information as **MV**, but in a less vigorous way. **VAR** is used because it is the best of the **WCLQ** orderings. The results show that **MV** is always outperformed by **WCLQ-VAR** which demonstrates that the use of maximum weighted cliques works best when the ordering also incorporates another ordering method. A direct comparison of the results, see Table 4.25, shows that **WCLQ-VAR** consistently has a lower mean backward degree (*mean b.d.*) over the critical constraints.

The above analysis also applies when searching using a lower domain size as the solution cost comparison (Table 4.26) and direct critical constraint comparison (Table 4.27) show.

How the cliques are weighted is also relevant. The above results are based upon cliques that have been weighted by variable weight (**MV**). I have also defined a second method that defines the clique weight as the sum of the clique edge weights (**ME**), which reflects the relationship between variables within the clique. A comparison of the solution costs found by each method can be found in Table 4.28. The **MV** ordering is clearly superior to **ME** with Table 4.29 showing a direct analysis. This can also be seen for lower domain sizes in Tables 4.30 and 4.31. The **MV** order

	variable ordering	
problem	MV	WCLQ-VAR
HEC-S-92	24227.2	**23139.6**
STA-F-83	63211.4	**62762.7**
KING-96	15049.7	**14537.2**
YOR-F-83	22525.9	**22145.9**
UTE-S-92	45100.4	**42003.4**
EAR-F-83	29965.1	30005.6
TRE-S-92	26488	**25124.4**
LSE-F-91	24518.4	**23132.8**
KFU-S-93	67782	**60794**

Table 4.24: Mean solution costs of each variable ordering for higher domain sizes. Bold type shows the lower mean cost, to 5% significance, of each pair as defined by the Z test.

problem	d.o.	size	MV b.d.	WCLQ-VAR b.d.
HEC-S-92	WCLQ-VAR	10	32.8	**9.2**
STA-F-83	WCLQ-VAR	10	13.2	**6.1**
KING-96	WCLQ-VAR	10	8.1	**4.9**
YOR-F-83	WCLQ-VAR	10	38.3	**16.5**
UTE-S-92	WCLQ-VAR	12	15.17	**8.17**
TRE-S-92	WCLQ-VAR	10	41.6	**20.2**
LSE-F-91	WCLQ-VAR	10	38.2	**20.1**
KFU-S-93	WCLQ-VAR	10	41.6	**11.9**

Table 4.25: Analysis of the mean backward degrees of the critical constraints (the number of which is given in *size*) between **MV** and **WCLQ-VAR**. *d.o.* records which ordering is dominant with respect to solution cost. *WCLQ b.d.* and *MV b.d.* record the mean backward degree of the critical constraints, whose number is defined by size, under each ordering. The **bold** type is used to show which mean *b.d.* is significantly lower.

	variable ordering	
problem	MV	WCLQ-VAR
HEC-S-92	42823.5	43346.4
STA-F-83	109700	**103572**
KING-96	34025.2	**32403.5**
YOR-F-83	44286.6	**42614**
UTE-S-92	89517.6	**88031.7**
EAR-F-83	51891.8	**51004.5**
TRE-S-92	45699.4	**42828.4**
LSE-F-91	37929.2	**36221.4**
KFU-S-93	99576.9	**91038.1**

Table 4.26: Mean solution costs for lower domain sizes. Bold type shows the lower mean cost, to 5% significance, of each pair as defined by the Z test.

problem	d.o.	size	MV b.d.	WCLQ-VAR b.d.
STA-F-83	WCLQ-VAR	10	14.7	**5.6**
KING-96	WCLQ-VAR	10	8.7	**5.9**
YOR-F-83	WCLQ-VAR	10	34.5	**17.6**
UTE-S-92	WCLQ-VAR	21	12.71	**7.95**
EAR-F-83	WCLQ-VAR	11	37.55	**29**
TRE-S-92	WCLQ-VAR	10	47.2	**18.3**
LSE-F-91	WCLQ-VAR	10	32.9	**19.9**
KFU-S-93	WCLQ-VAR	10	39.3	**10.9**

Table 4.27: Analysis of the mean backward degrees of the critical constraints between **MV** and **WCLQ-VAR** using the lower domain sizes.

	variable ordering	
problem	MV	ME
HEC-S-92	24227.2	24140
STA-F-83	63211.4	63143.8
KING-96	**15049.7**	15722.2
YOR-F-83	22525.9	22572.3
UTE-S-92	**45100.4**	47269.1
EAR-F-83	**29965.1**	32187.3
TRE-S-92	**26488**	28603.2
LSE-F-91	**24518.4**	26307.9
KFU-S-93	**67782**	70153.6

Table 4.28: Mean solution costs for higher domain sizes. Bold type shows the lower mean cost, to 5% significance, of each pair as defined by the Z test.

problem	d.o.	size	MV b.d.	ME b.d.
KING-96	MV	10	**5.7**	10.2
UTE-S-92	MV	10	**7.9**	19.3
EAR-F-83	MV	10	**28.9**	51.1
TRE-S-92	MV	10	**37.1**	75.9
LSE-F-91	MV	10	**33.3**	58

Table 4.29: Analysis of the mean backward degrees of the critical constraints using the **MV** and **ME** orderings.

maintains some general problem structure information. In **ME** this information has been removed and the clique weighting concentrates only on the clique variables, resulting in a higher solution cost.

	variable ordering	
problem	MV	ME
HEC-S-92	**42823.5**	44358.3
STA-F-83	109700	109985
KING-96	**34025.2**	34416.9
YOR-F-83	44286.6	44153.6
UTE-S-92	**89517.6**	93161.4
EAR-F-83	**51891.8**	54208
TRE-S-92	**45699.4**	47552.4
LSE-F-91	**37929.2**	40323
KFU-S-93	**99576.9**	105029

Table 4.30: Mean solution costs for lower domain sizes. Bold type shows the lower mean cost, to 5% significance, of each pair as defined by the Z test.

problem	d.o.	size	MV b.d.	ME b.d.
HEC-S-92	MV	-	-	-
KING-96	MV	10	**4.7**	7.1
UTE-S-92	MV	10	**6**	23.2
EAR-F-83	MV	10	**29.7**	52.9
TRE-S-92	MV	10	**29**	63.9
LSE-F-91	MV	10	**23.7**	38.2
KFU-S-93	MV	10	**29.2**	46.2

Table 4.31: Analysis of the mean backward degrees of the critical constraints using the **MV** and **ME** orderings and the low domain sizes.

4.3.4 Summary

I have shown how **DEG** and **FD** can be extended to the WCSP framework by including constraint weight information. Both extensions, **WD** and **WFD**, are superior with respect to searching for lower solution costs; although both generally require more unlabels to find feasible solutions.

I have repeated this extension method to develop orderings which use maximum weighted cliques (an extension from the **CLQ** orderings) and have shown that they have an effect similar to that observed for **CLQ**. There is no clear improvement or degradation of solution or search cost when such cliques are used in a variable ordering. Continued analysis has suggested that improvements due to **CLQ** and **WCLQ** depend on the actual clique used and not to the general method.

I have extended the use of weighted cliques to define 2 new orderings, on weighted clique partition, **MV** and **ME** using 2 different definitions of clique weight. Results have shown that neither ordering leads to better solutions. As the orderings lose general problem information, as **MV** does when compared to **WCLQ-VAR** and **ME** when compared to **MV**, the quality of solutions degrades.

I have continued the comparison between the solution cost attributed to critical constraints and the mean backward degree of such constraints. The results uphold the general hypothesis that increased error in the critical constraints, and subsequently the whole problem, is due to the large constraint backward degree of these constraints.

4.4 Using Constraint Backward Degree In Variable Ordering

A large proportion of the analysis performed in this chapter has focused on the backward degree of the **critical constraints**, i.e. the constraints that affect search the most. It has been shown that when one variable ordering results in solutions with a lower mean cost than another variable ordering, the mean backward degree of the **critical constraints** is lower. The definition of the **critical constraints** has varied depending upon whether a direct analysis between 2 related orderings, or a general analysis between un-related orderings, is being performed. However, the use of constraint backward degree in the analysis remains the same. If the critical constraint set could be defined before search, then a variable ordering that minimises their backward degree could be constructed. Unfortunately, the critical constraints cannot be defined before search, but we can approximate the critical constraint set.

Due to the distribution of constraint weights, where there are a few constraints with very large weights and increasingly more lighter constraints, those constraints which incur the significant cost for a solution will be those with large weights. Therefore, constructing a variable order that minimises the backward degree of these constraints may result in lower mean solution costs. Minimising the backward degree has been suggested by Freuder in relation to backtrack free search [28]; however, in this case it is used to the benefit of search overall. I have developed 2 algorithms based around Algorithm 6 which order variables by sequentially placing the variable with the maximum backward degree (tie breaking on degree) next in the order.

Lines $1-4$ initialise the ordering and record the degree of each variable (stored in the d array). Then the Algorithm sequentially places variables in the final ordering one by one (line 6). The backward degree array (bd) is reset (lines $7-9$) and the current backward degree, with respect to the partially generated ordering (the first l elements of ord), of each variable is calculated (lines $10-16$). Then a sort procedure reorders the elements of the ord array not placed in the new ordering (i.e. elements l to n) in largest bd order first, tie-breaking on largest d, reshuffling bd and d so that equivalent elements correspond to those in the bd array.

An example of a backward degree ordering is given in Figure 4.12.

Algorithm 6 Order the variable array **ord** according to backward degree.

```
1  begin
2    foreach i ∈ 1, ..., n do
3      ord[i] = i
4      d[i] = degree(i)
5    od
6    foreach l ∈ 1, ..., n − 1 do
7      foreach i ∈ 1, ..., n do
8        bd[i] = 0
9      od
10     foreach i ∈ 1, ..., l do
11       foreach j ∈ 1, ..., n do
12         if (edge(ord[i], ord[j])) then
13                 bd[j] + +
14         fi
15       od
16     od
17     sort ord according to bd tie − break on d
18   od
19 end
```

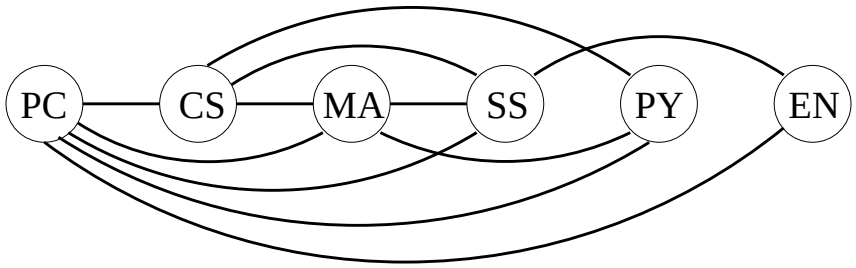

Figure 4.12: A backward degree ordering(**BD**) for the problem given in Figure 2.1 and Table 3.2. The calculations are given in Table 4.32.

		b.d at iteration						
node	degree	1	2	3	4	5	6	order
CS	4	0	1	-	-	-	-	2
MA	4	0	1	2	-	-	-	3
PY	3	0	1	2	3	4	-	5
PC	5	0	-	-	-	-	-	1
SS	4	0	1	2	3	-	-	4
EN	2	0	1	1	1	2	2	6

Table 4.32: How the variable backward degrees (b.d.) vary as the **BD** ordering of Figure 4.12 is constructed.

This backward degree ordering (**BD**) may not place the heaviest constraints of the problem at the start of the ordering; however, it should minimise backward degree across the variable ordering and so help to minimise constraint backward degrees as well as minimise re-labelling. Following my work extending variable orderings to incorporate constraint weight information I have also implemented a second ordering of weighted backward degree (**WBD**), an example of which can be found in Figure 4.13. This ordering is essentially the same; however, instead of defining the degree in line 4 of Algorithm 6 it calculates the weighted degree (i.e. the sum of the weight of all constraints for which the variable is in the scope) and the *weighted bd* is calculated in line 13 by $bd[j] += edgeweight(ord[i], ord[j])$. This ordering should address the problem of minimising the backward degree of the heavy constraints in the ordering. The backward degree of a variable is also known as the width of a node, as used by Tsang [69]; however, **BD** does not correspond to the Minimum Width Ordering (**MWO**) of [69] where the width of a variable ordering, defined as the maximum width or backward degree of a variable, is minimised.

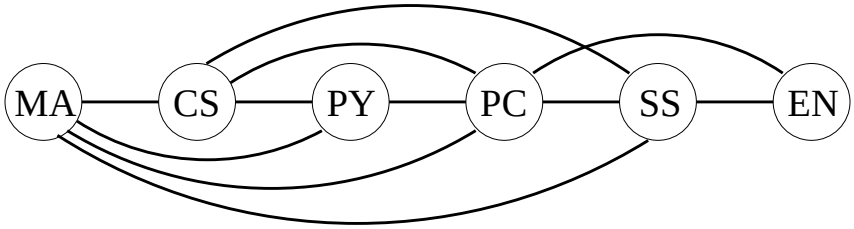

Figure 4.13: A weighted backward degree ordering(**WBD**) for the problem given in Figure 2.1 and Table 3.2. The calculations are given in Table 4.33.

		w.b.d. at iteration						
node	weighted degree	1	2	3	4	5	6	order
CS	10	0	4	-	-	-	-	2
MA	12	0	-	-	-	-	-	1
PY	8	0	4	6	-	-	-	3
PC	12	0	3	5	7	-	-	4
SS	8	0	1	3	3	6	0	5
EN	4	0	0	0	0	2	4	6

Table 4.33: How the variable weighted backward degrees (w.b.d.) vary as the **WBD** ordering of Figure 4.13 is constructed.

Table 4.34 compares the variable weight based ordering (**VAR**) with the two newly proposed orderings. The comparison is made with **VAR** as it is a consistently good ordering compared to **DEG** or **FD**. Although its performance can be improved by the use of cliques these have not been used for simplicity. The results show that **WBD** out-performs both **VAR** and **BD** on all of the given problems. Comparing **BD** and **VAR** separately, **VAR** is significantly superior for the majority of problems; however, this is expected as **BD** does not try to incorporate any constraint weight information at all. A better comparator for **BD** would be the degree ordering (**DEG**), against which **BD** produces significantly better solutions for all but one problem, where they are indistinguishable.

	variable ordering		
problem	VAR	BD	WBD
HEC-S-92	21756.1	23226.3	**20636**
STA-F-83	63856.1	63467.3	**62333.7**
KING-96	14531.9	14656.7	**13265.7**
YOR-F-83	21876.3	22307.6	**20989.4**
UTE-S-92	40093.3	44213.8	**39000.3**
EAR-F-83	29038.1	29427.3	**27581.7**
TRE-S-92	24901.6	25126.7	**23562.9**
LSE-F-91	22632.9	23000.3	**21072.7**
KFU-S-93	58818.7	63774.1	**55886.7**

Table 4.34: Comparison of the mean solution costs using **VAR**, **BD** and **WBD** for **ERR-RND**. Bold values signify significantly lower mean distribution using the Z-test at 5% significance.

When the same experiments are performed with the low domain sizes (in Table 4.35), the same observations hold true, on the whole. Again **WBD** performs best,

	variable ordering		
problem	VAR	BD	WBD
HEC-S-92	**39123.5**	42234.4	**38148.2**
STA-F-83	103798	110166	**102160**
KING-96	32573	32794.4	**30908.2**
YOR-F-83	42483.1	42485.1	**41065.5**
UTE-S-92	**81395.7**	83402.5	**81156.1**
EAR-F-83	49394.6	50000	**47090.3**
TRE-S-92	42291.3	43020.3	**39616.8**
LSE-F-91	35254.8	36744.4	**32695.3**
KFU-S-93	88439.2	93855.4	**85055.1**

Table 4.35: Comparison of the mean solution costs using **VAR**, **BD** and **WBD** for **ERR-RND** using the low domain sizes. Bold values signify significantly lower mean distribution using the Z-test at 5% significance.

except in two problems (**HEC-S-92** and **UTE-S-92**) for which it is not significantly superior to **VAR**. Again **VAR** outperforms **BD**; however, **BD** consistently requires fewer unlabels than either of the other 2 orderings (see Table 4.36). This suggests that **BD** may be a good choice of variable ordering when search finds it extremely difficult to find feasible solutions.

	variable ordering		
problem	VAR	BD	WBD
HEC-S-92	47	7	61
STA-F-83	1	0	1
KING-96	5	0	1
YOR-F-83	77	19	31
UTE-S-92	9	2	7
EAR-F-83	34	3	25
TRE-S-92	7	2	4
LSE-F-91	7	2	5
KFU-S-93	6	1	5

Table 4.36: The number of unlabels required by **ERR-RND** when using **VAR**, **BD** and **WBD** on the low domain sizes.

4.4.0.1 Comparisons With The Work Of Others

The majority of the problems used in this investigation are those used in [17]. This suite of problems can be used as a set of benchmarks on which to test any new search methods. Unfortunately, although these problems are used often, search methods still tend to focus on institution-specific criteria such as maximum room space. Although such criteria are important when assessing search methods, different institutions have different criteria and general comparisons between search methods should concentrate upon the base problem.

The work of Di Gaspero and Schaerf [32] includes a comparison of the results of their method with those produced by the **LBT** method of Carter, *et al* [17]. The **LBT** method is similar in description to the search algorithm used in this investigation; however, few details are given in [17]. Table 4.37 compares the mean and minimum results of the search using **WBD** as developed in this investigation with the results of both [17] and [32]. The results of their **TABU** search method can be seen to be competitive with those of Carter, *et al.* The results of search using **WBD** can be seen to outperform **TABU** in all but one instance (**YOR-F-83**). **WBD** outperforms **LBT** in 5 instances showing that it too is competitive with **LBT**. A full comparison of the results presented in this thesis can be found in Appendix B.

	WBD		LBT [17]		TABU [32]	
problem	wbd mean	wbd min	max	min	mean	min
HEC-S-92	38148.2	33549	56460	**30488.4**	35569.8	35005.2
STA-F-83	102160	**97095**	112485.1	98676.5	101924.8	**98248.8**
YOR-F-83	41065.5	39249	46955.9	39239.7	39616.1	**38581**
UTE-S-92	81156.1	77747	105325	**70950**	86075	79750
EAR-F-83	47090.3	43120	52312.5	**40950**	52437.5	51412.5
TRE-S-92	39616.8	**37317**	47982	41875.2	45801	43620
LSE-F-91	32695.3	30274	36255.8	**28623**	43343.4	42253
KFU-S-92	85055.1	78724	118212.9	**74886**	104305.5	96282
RYE-S-93	132320	111455	126313	**83825.9**	-	-
CAR-F-92	88810.1	**79865**	151035.8	114197.8	103146.4	**95778.8**
UTA-S-92	77147.6	**72783**	136102.4	74431	95597	89317.2
CAR-S-91	100206	**89415**	201419.4	120174.6	110019	**104941.2**
PUR-S-93	143976	139210	150160	**117124.8**	-	-

Table 4.37: Comparison of the results of this investigation with the results from [17] and [32]. These problem correspond to the low domain size. The original results of [17] and [32] were given in proportion to the number of students. For ease of comparison these numbers have been converted into total error form (hence some decimals appear due to rounding).

4.4.1 Summary

The work of this section follows on from the results of previous sections which have shown the solution quality is related to the backward degree of constraints. Variable ordering heuristics have been developed to exploit this fact. Results show that such heuristics are superior to other tested heuristics. Results have also been compared to the published results and the new methods have been shown to at least compete with the state of the art search results.

4.5 Dynamic Heuristics

When searching for a low error-cost initial solution dynamic heuristics should hold an advantage over static variable orderings as they can adapt to the current partial solution (i.e. the solution so far) during search. Here I examine how well dynamic heuristics perform with respect to each other and related static orderings and define under which circumstances methods may encounter difficulty.

4.5.1 Standard Methods

In Classical CSPs the Fail-first, Brelaz and DD heuristics are regularly used when attempting to tackle a random or real-world problem. Each tries to identify which of the unassigned variables will be most difficult to assign a value to. The most commonly used heuristics apply the **Fail-First Principle** [37]. This principle states that the next move made by search should be that which is most likely to result in a inconsistent solution (i.e. failure). The principle works in two ways. First of all, it applies the idea that it is best to assign a value to a variable which has a high probability of failure while a valid assignment still exists as opposed to leaving the assignment until later in search when such a failure may be unavoidable. Secondly, the implementation of the principle implies that search includes look-ahead which identifies a definite failure as soon as it occurs. In fact the work of Smith and Grant shows that the Fail-First principle is flawed [68]; however, they show that choosing the next variable on smallest domain is a good choice. This is due to the fact that it help to reduce the amount of branching, or the number of values a variable can be assigned, nearer the root of the tree.

The heuristic called Fail-First(**FF**), attempts to tackle the problem choosing as next the variable with the smallest domain size. This is based upon the assumption that if a variable has few possible assignments, the reduction in choice will mean that the constraints in which it is involved with will be more difficult to solve. A drawback of this method for the UETTP is that it arbitrarily chooses the first variable as all variables start with the same number of possible assignments. The Brelaz heuristic(**BZ**) [8] avoids this problem by extending this heuristic so that where a tie between variables occurs the variable with the largest degree is chosen. The variable with the largest degree is considered a good choice because it will have a large impact on the rest of the problem, pruning more values, etc. This method has been widely used in graph colouring where it is also known as **DSATUR**. The drawback of these methods is that they fail to consider the number of possible

variable assignment choices in relation to the degree of the variable. A variable with a large degree may have a larger domain size than another with a very small degree, yet the variable with the larger degree is more likely to be involved in a difficult sub-problem. This gives rise to a heuristic which ranks variables based upon domain size divided by degree(**DD**) [3]. It is generally considered, for the reasons given above, that **DD** is better than **BZ** which, in turn, is better than **FF**.

4.5.1.1 Related Static Ordering Heuristics

One of the disadvantages of dynamic variable ordering heuristics is that they have to be applied to the unassigned variables each time an assignment is made. Therefore, this expenditure of effort must be justified with lower solution error costs (i.e. better solutions). To this end, each of the dynamic orderings have been compared to a related static ordering which attempts to simplify the reasoning behind the dynamic selection procedure into the static framework. In each case the calculation of domain size has been generalised to the calculation of the backward degree. The related orderings are as follows:

- backward degree with no tie break (**BD-NTB**): the variables are ordered on the number of past neighbours (which is maximised). There are no tie-breaks so the first variable (when all variables have a backward degree of 0) is variable number 1. This static ordering corresponds to the **FF** dynamic ordering.
- backward degree with degree tie break (**BD-DTB**): as above, bu tie-breaks are decided on the largest degree. The first variable in the ordering is that with the largest degree. This ordering corresponds to **BZ**.
- backward degree over degree (**BD-D**): the variables are ordered upon minimising the backward degree divided by the degree. This ordering corresponds to **DD**.

	variable ordering pairs					
problem	FF	BD-NTB	BZ	BD-DTB	DD	BD-D
HEC-S-92	23395.9	**23184.5**	23475.7	**23226.3**	24062.4	**22291.8**
STA-F-83	62719.2	62476.4	63425.5	63467.3	64814.1	64446.2
KING-96	14675.8	**14558.2**	14754.9	14656.7	15159.5	**14667.5**
YOR-F-83	22644.7	22780.8	22551.7	**22307.6**	**22208**	23886
UTE-S-92	44189.6	44075.5	45145.7	**44213.8**	**46496.7**	45151.6
EAR-F-83	31355.1	**31041.2**	29597.6	29427.3	**29343.2**	33131.6
TRE-S-92	**26010.8**	26473.6	25586.5	**25126.7**	**25619.9**	30103.7*
LSE-F-91	23709.7	23742.2	23107.3	23000.3	**23632.4**	29166.6*
KFU-S-93	65761.4	66313.9	64910.8	**63774.1**	**66952.7**	84652.4*

Table 4.38: Mean solution costs found using the **ERR-RND** value ordering. Bold type shows the lower mean cost, to 5% significance, of each pair as defined by the Z test. Results with a * required re-labelling to find viable results.

4.5.1.2 Search Using Standard Dynamic Methods

The mean solution cost when searching with each dynamic ordering is compared to the related static ordering in Table 4.38. Dynamic searches using **FF** are in the main indistinguishable from those of their related static ordering, **BD-NTB**. In 3 of the 4 cases where the ordering pairs were significantly different at the 5% level of confidence **BD-NTB** was the superior ordering. Similarly, search using **BZ** was outperformed by **BD-DTB** for all but the 4 cases when the ordering pairs were indistinguishable. Both dynamic orderings select the next variable primarily on saturation degree. These results show that when searching under a high domain size, when a simple depth first search is unlikely to encounter any difficulties, the dynamic nature of such heuristics leads to slightly higher solution costs. Coupled with the extra cost of calculating the data required for dynamic selection the related static orderings are a better choice in this case.

The same is not the case when searching using **DD**. In 6 of the 8 significantly different pairs of **DD** and **BD-D**, **DD** is superior. It is worth noting, however, that search using **BD-D** encounters problems finding solutions as demonstrated by the fact that for the **TRE-S-92**, **LSE-F-91** and **KFU-S-93** problems re-labelling is required to find any solutions. Difficulty finding valid solutions is a result of search focusing on an area of the search space which has few solutions. The few solutions found in such areas lead to high solution costs. All 3 orderings use the general strategy of choosing the variable which will prove to be the most difficult to find

a valid assignment for, but the results show that different implementations of this strategy can vary in quality.

The same comparison is made when searching using the lower domain sizes in Table 4.39. The most noticeable feature of the comparison between **FF** and **BD-NTB** at the lower domain sizes is that the pairs of solution sets are distinguishable in all but 3 cases with neither ordering being dominant. The opposite is the case when comparing **BZ** with **BD** where only 4 ordering pairs are distinguishable. Subsequently, no clear analysis of dominance can be made from such a low number of sample cases. The comparison between **DD** and **BD-D** shows similar results as when searching using the higher domain sizes with the static ordering dominating for the smaller problems and the dynamic for the larger. The number of unlabels required by **BD-D**, shown in Table 4.40, are high in comparison to those by **DD**, so it comes as little surprise that **DD** appears superior given the difficulties in finding valid solutions when using **BD-D**. Generally, the dynamic ordering heuristics require fewer unlabels than their static counterparts. This is as expected considering that the general principle of tackling difficult variables first should imply incurring fewer unlabels (most noticeably for the **YOR-F-83** problem). Again the assertion regarding superiority of **FF**, **BZ** or **DD** holds when searching using the lower domain size.

	variable ordering pairs					
problem	FF	BD-NTB	BZ	BD	DD	BD-D
HEC-S-92	39791.5	39413.5	**41059.8**	42234.4	41585.7	**40220**
STA-F-83	101165	**102917**	**105802**	110166	110302	**102376**
KING-96	**32022.9**	33907.8	32783.7	32794.4	33965.9	**32790.7**
YOR-F-83	43704	43468	42388.2	42485.1	**42358.9**	44491.8
UTE-S-92	**86605.8**	88982.8	86189.9	**83402.5**	89129.1	**86954.2**
EAR-F-83	51619.4	**51171.1**	50306.8	50000	**49435.6**	53406
TRE-S-92	**44541.9**	44988.2	43502.7	**43020.3**	**43272.2**	48040.1
LSE-F-91	36251.5	36230.8	36884.5	36744.4	**36750.9**	43658.7
KFU-S-93	96594.1	**94963.1**	93882.4	93855.4	**94642.8**	111624

Table 4.39: Mean solution costs using the **ERR-RND** value ordering on the low domain sizes. Bold type shows the lower mean cost, to 5% significance, of each pair as defined by the Z test.

	variable ordering pairs					
problem	FF	BD-NTB	BZ	BD	DD	BD-D
HEC-S-92	10	11	1	7	2	62
STA-F-83	0	0	0	0	1	14
KING-96	0	0	0	0	0	1
YOR-F-83	7	33	3	19	4	68
UTE-S-92	1	2	1	2	1	3
EAR-F-83	0	4	0	3	0	37
TRE-S-92	0	5	0	2	0	73
LSE-F-91	0	0	0	2	0	115
KFU-S-93	1	4	0	1	2	70

Table 4.40: The median number of unlabels required by **ERR-RND** search when searching using the low domain sizes.

4.5.2 Dynamic methods for Weighted CSP

It has already been shown that static variable orderings can incorporate information regarding the weights associated to constraints to improve the solutions found by search. I have applied the same rationale to dynamic variable orderings. The following 2 dynamic variable orderings are derived as extensions of more conventional dynamic orderings:

- **WSD** - weighted saturation degree (also referred to as Generalised Saturation Degree in [40]). This ordering is derived from **BZ** which is also know as ordering on variable saturation degree. The **wsd** value of an unassigned variable during search is calculated by Algorithm 7, with the next variable assignment being made to the variable with the highest **wsd**. Rather than count the number of domain values that have already been used, as **BZ** does, the maximum weight of the edges which prune a particular domain value is recorded. Line 3 loops through each possible original domain value of the variable j (D_j), i.e. without pruning. Line 6 loops through all possible past variables of j (P_j), which are related to j ($edge(j,k)$) and are assigned the domain value d ($l_k = d$), determining which of the constraints between these variables is the heaviest (recorded in the max_wgt variable). An example of how the wsd of a future variable develops as assignments are made can be found in Table A.5 in Appendix A.

- **WSD-WD** - weighted saturation degree divided by weighted degree. This is a weighted extension of the **DD** dynamic variable ordering. The **wsd** part of this ordering is calculated in the manner above; however, the decision as to which variable will have a value assigned to it next, divides **wsd** by the weighted degree (**wd**). The next variable for assignment chosen is that with the smallest such value. The aim of this extension to the **WSD** ordering is to quantify, in a similar way to the normalisation of data, the *wsd* value of each future variable during search such that the relevance to the problem structure local to the variable is considered.

Algorithm 7 Calculate the wsd for a variable j

```
1  begin
2    wsd = 0
3    foreach d ∈ {d_1, ..., d_m} = D_j
4      do
5      max_wgt = 0
6      foreach k : k ∈ P_j and edge(j,k) and l_k = d
7        do
8        if (edgeweight(j,k) > max_wgt) then
9                  max_wgt = edgeweight(j,k)
10       fi
11     od
12     wsd += max_wgt
13   od
14 end
```

Table 4.41 compares the solution costs of the **BZ** and **DD** dynamic orderings with their weighted counterparts. In these results **WSD** consistently out-performs **BZ** in every case with the consideration of the constraint weight data being complementary to the original ordering. The same cannot be said of the **WSD-WD** ordering which only out-performs **DD** for the 3 smallest problems and requires re-labelling during search to find any solutions when searching the larger problem instances. It would appear that in **WSD-WD** using the constraint weight information interferes with the search for a feasible solution.

The same comparison is performed when searching using the lower domain sizes in Table 4.42. Again **WSD** out-performs **BZ**; however, for 2 problems (**HEC-S-92** and **YOR-F-83**) **BZ** is the superior ordering. This is as a result of search using **WSD** ignoring the primary aim of finding feasible solutions. The number of

	variable ordering pairs			
problem	BZ	WSD	DD	WSD-WD
HEC-S-92	23475.7	**20059.1**	24062.4	**21166.9**
STA-F-83	63425.5	**60898.4**	64814.1	**62623.8**
KING-96	14754.9	**13580.4**	15159.5	**14446.4**
YOR-F-83	22551.7	**21086.9**	**22208**	25764
UTE-S-92	45145.7	**40255.3**	**46496.7**	49528.8
EAR-F-83	29597.6	**28753.1**	**29343.2**	35700.9*
TRE-S-92	25586.5	**23502.1**	**25619.9**	27536*
LSE-F-91	23107.3	**21139.6**	**23632.4**	29315.9*
KFU-S-93	64910.8	**56502**	**66952.7**	97812.4*

Table 4.41: Mean solution costs when searching using the **ERR-RND** value ordering on the high domain size. Bold type shows the lower mean cost, to 5% significance, of each pair as defined by the Z test. Values with a * show when re-labelling was required to find viable solutions.

unlabels required by the searches in these cases, given in Table 4.43, can be seen to be considerably higher for **WSD**. When searching with the **WSD** dynamic ordering it is expected that a higher number of unlabels would be required as domain size is lowered. This can be considered to be a trade-off between the primary search goal of finding viable solutions and the secondary goal of finding solutions with a lower cost. The results show that in most cases an increase in the number of unlabels required when incorporating constraint weight information into **DD** does not lead to major search problems in finding viable solutions; however, if the use of **WSD** leads to problems finding viable solutions it will have a resulting effect on the quality solutions with respect to solution cost. The results of Table 4.42 also confirm the unsuitability of **WSD-WD**, with Table 4.43 showing that this extension to **DD** results in a very large increase in the number of unlabels in most cases.

The two dynamic variable orderings with constraint weight information can also be compared to related static variable orderings. As with the similar comparison for **FF**, **BZ** and **DD**, the saturation degree information can be relaxed to backward degree. In the weighted case, however, weighted saturation degree is relaxed to weighted backward degree (**wbd**). The **wbd** is defined in 4.2 (i.e. the sum of the weight of the constraints (W_{C_j}) between the given variable (x) and its past neighbours(P_x)):

$$wbd(x) = \sum W_{C_j} \quad : \quad x,\ y \in C_j \ \ and \ \ y \in P_x \tag{4.2}$$

	variable ordering pairs			
problem	BZ	WSD	DD	WSD-WD
HEC-S-92	**41059.8**	41933.1	**41585.7**	43733.1
STA-F-83	105802	**99230.2**	110302	**105519**
KING-96	32783.7	32796.9	33965.9	34231.5
YOR-F-83	**42388.2**	43760.9	**42358.9**	48092.4
UTE-S-92	86189.9	**81549.9**	**89129.1**	95292
EAR-F-83	50306.8	50961.7	**49435.6**	59559.3
TRE-S-92	43502.7	**41746.5**	**43272.2**	50435
LSE-F-91	36884.5	**35045.8**	**36750.9**	47389.2
KFU-S-93	93882.4	**84643.2**	**94642.8**	108937

Table 4.42: Mean solution costs when searching using the **ERR-RND** value ordering using the low domain sizes. Bold type shows the lower mean cost, to 5% significance, of each pair as defined by the Z test.

	variable ordering pairs			
problem	BZ	WSD	DD	WSD-WD
HEC-S-92	1	46	2	55
STA-F-83	0	1	1	10
KING-96	0	3	0	2
YOR-F-83	3	91	4	98
UTE-S-92	1	6	1	23
EAR-F-83	0	39	0	77
TRE-S-92	0	9	0	105
LSE-F-91	0	13	0	120
KFU-S-93	0	5	2	76

Table 4.43: The median number of unlabels required by **ERR-RND** when searching using the low domain sizes.

	variable ordering pairs			
problem	WSD	WBD	WSD-WD	WBD-WD
HEC-S-92	**20059.1**	20636	21166.9	**20129.2**
STA-F-83	**60898.4**	62333.7	62623.8	62578.2
KING-96	13580.4	**13265.7**	14446.4	**13843**
YOR-F-83	21086.9	20989.4	25764	**21140**
UTE-S-92	40255.3	**39000.3**	49528.8	**41736.2**
EAR-F-83	28753.1	**27581.7**	35700.9*	**28881.4***
TRE-S-92	23502.1	23562.9	27536*	**23492***
LSE-F-91	21139.6	21072.7	29315.9*	**21794.3***
KFU-S-93	56502	55886.7	97812.4*	**86290.4***

Table 4.44: Mean solution costs when searching using **ERR-RND** on the high domain sizes. Bold type shows the lower mean cost, to 5% significance, of each pair as defined by the Z test. Values with a * show when re-labelling was required to find viable solutions.

The two related static variable orderings are subsequently:

- **WBD** : the weighted backward degree ordered largest first. Corresponding to **WSD**.
- **WBD-WD** : the weighted backward degree divided by the weighted degree ordered smallest first. Corresponding to **WSD-WD**.

The initial comparisons are shown in Table 4.44. Only 5 of solution sets for the **WSD**/**WBD** ordering pairs are significantly different making any analysis difficult. On the whole **WBD** is better in this case as it does not require re-calculation of the heuristic during search. However, **WBD-WD** clearly out-performs its dynamic counterpart with only one exception where the solution sets are not significantly different. It is worth noting that for the larger problems, both **WSD-WD** and **WBD-WD** require re-labelling for search to find solutions. In each of these cases **WBD-WD** outperforms **WSD-WD** considerably suggesting that **WSD-WD** is not the best implementation of its general strategy as a relaxed static ordering interpretation is superior. This general strategy does not compare well with **WSD**/**WBD** as **WBD-WD** nearly always results in higher cost solutions.

As the domain size is lowered and the primary search aim of finding viable solutions becomes more difficult the static orderings become more dominant (as shown in Table 4.45).

	variable ordering pairs			
problem	WSD	WBD	WSD-WD	WBD-WD
HEC-S-92	41933.1	41059.9	43733.1	**38379.1**
STA-F-83	**99230.2**	102418	105519	**102529**
KING-96	32796.9	**31348.2**	34231.5	**31998.4**
YOR-F-83	43760.9	**42193**	45430.9	**35467.8**
UTE-S-92	**81549.9**	83711.6	95292	**85789.8**
EAR-F-83	50961.7	**50180.8**	55793.1	**49669.6**
TRE-S-92	41746.5	**39977**	50435	**39922.3**
LSE-F-91	35045.8	**33884**	43017.7	**33598.2**
KFU-S-93	**84643.2**	87800.7	108937	**86290.4**

Table 4.45: Mean solution costs when searching using **ERR-RND** and the low domain sizes. Bold type shows the lower mean cost, to 5% significance, of each pair as defined by the Z test.

	variable ordering pairs			
problem	WSD	WBD	WSD-WD	WBD-WD
HEC-S-92	46	61	55	63
STA-F-83	1	1	10	1
KING-96	3	1	2	1
YOR-F-83	91	31	98	16
UTE-S-92	6	7	23	10
EAR-F-83	39	25	77	30
TRE-S-92	9	4	105	4
LSE-F-91	13	5	120	6
KFU-S-93	5	5	76	5

Table 4.46: The median number of unlabels required by **ERR-RND** when searching using the low domain sizes.

4.5.2.1 Summary

None of the standard dynamic variable ordering methods has been shown to be superior to any of the others. Static relaxations of **FF** and **BZ** have been shown to be competitive with respect to solution cost. The number of unlabels required when searching using the standard dynamic methods are extremely low which is as expected as this was the primary aim in their development.

Two dynamic variable ordering heuristics that consider constraint weights were compared with related standard dynamic methods. **WSD** found lower cost solutions at the expense of search requiring more unlabels. **WSD-WD** proved not to be a very effective heuristic at all. These heuristics were also compared to static relaxations where **WBD** proved to be competitive with **WSD** and **WBD-WD** unsurprisingly superior to **WSD-WD**. Hence, the static heuristics are preferable in this case because they are cheaper to use.

4.6 Conclusions

In this chapter I have developed a new analysis method to compare the results of search using different static variable orderings (Section 4.2.1). By identifying the **critical constraints** of an ordering, the features of the constraints that have contributed the most to the quality of an ordering can be examined. I have used this general scheme in two ways; to compare the solutions found using several unrelated orderings in a general manner; and to directly compare the solutions found using two related orderings. These analysis methods, which have been applied throughout the chapter, have been used to show that the backward degree of the **critical constraints** is an important consideration when trying to reduce the error cost of the solutions found by search. I have applied this new finding, derived from post-search analysis, to develop a variable ordering that finds solutions with significantly lower costs than other static variable ordering methods.

I have investigated the use of both maximum cliques and maximum weighted cliques in search, with mixed results for the stochastic search techniques used (section 4.2.2.1). Results have suggested that the actual clique used in such methods determines the effect of the method and not merely the general use of maximum cliques. I have extended such methods to consider a static ordering consisting of a clique partition, developing a new greedy algorithm to generate such partitions, with results showing that when search concentrates upon small sections of the problem

at a time, results are worse than if search tackles variables from across the whole problem.

I have shown that by incorporating constraint weight information into static variable orderings used in traditional Constraint Satisfaction Problems, search can find solutions with a lower error cost(section 4.3). This is the first time a direct comparison has been made between weighted and un-weighted variable orderings. I have also analysed the incorporation of weights into commonly used dynamic orderings with results showing that **WSD** can outperform un-weighted dynamic variable orderings with respect to solution cost at the expense of a small increase in the number of unlabels required during search (section 4.5.2).

I have compared the various dynamic variable ordering methods used with related static variable orderings. Results have shown that when finding consistent solutions is not difficult, such static orderings can compete with their dynamic counterparts with respect to solution cost (section 4.5.1).

The key contribution of this chapter is the identification of backward degree as a key factor in how well critical constraints are satisfied and the extension of this knowledge to develop the **BD** and **WBD** variable orderings.

Chapter 5

Value Ordering Heuristics

5.1 Introduction

The use of value orderings can allow search to follow a path down the search tree that is more likely to result in a low cost solution. However, there is no optimal method with regard to the variety of problems and variable ordering methods. Various features of the search that is performed, and the problem which it is applied to, can affect the suitability of the value ordering heuristics available. The aim of this section is to examine the performance of value ordering heuristics in different situations and consider why in some cases one heuristic is better (in terms of finding good solutions) than another.

The value ordering heuristics that are to be investigated fall into three categories described by their selection criteria of; the error produced by the current partial solution (called error based selection); the error of the current partial solution plus the minimum potential error in the future variables, as calculated using look-ahead (look-ahead based selection); and irrespective of current or potential costs (blind selection). The first two categories provide an alternative use of the data that is gathered for pruning by pure DFS, and DFS combined with forward checking, respectively. The last category can be combined with any degree of look-ahead. Different value ordering heuristics can also be broken down into **dynamic** and **static** methods. The difference between the two is that it is possible to determine the static value order before search as they do not change during search, whereas, dynamic methods depend upon information generated during search. Of the methods described above blind selection is static while look-ahead and error based selection are both dynamic.

5.2 Experiments

It is often desirable to compare results to a control value ordering. Such a value ordering uses a random value selection method (called **RND**). The random value selection heuristic is an example of a **stochastic** value ordering heuristic. Such heuristics have some random element in their selection criteria which means that different trials of the search will lead to different solutions. The investigation will first assess the performance of non-stochastic methods against each other.

The non-stochastic value orderings that have been considered consist of methods which are categorised as blind, error based, and look-ahead based selection (as defined before).

- The blind method is a lowest consistent, or available, value selection (or **LOW**). This heuristic corresponds to value ordering as commonly used in the Classical CSP field.

- The error based methods consist of selecting the value which generates the lowest error and tie-breaks on; the lowest such value (or **ERR-LOW**); or the highest such value (or **ERR-HIGH**). An example of search using **ERR-LOW** is given in Table A.4 of Appendix A.

- There are two look-ahead based methods being considered, both of which select the value with the lowest combined error and minimal potential future error, yet each tie-break differently. As above, two variations tie-break on the lowest (**IC-LOW**) and highest (**IC-HIGH**) such value. A third method has been derived as a result of the way in which the **iterative count** (**IC**) values, which are used to record minimum potential future error, are generated. In order to save computation time, these values are updated in a lazy way and so are only calculated, with respect to the future variables, until they are greater than the minimum value already found within the domain of the variable in question.

	value ordering		
problem	LOW	ERR-LOW	RND
HEC-S-92	92386	23984	32654
KING-96	66962	15880	27683
YOR-F-83	60872	22512	33183
UTE-S-92	151777	41860	73545
EAR-F-83	80707	29419	46487

Table 5.1: Comparison of the lowest solution found using the **LOW**, **RND** and **ERR-LOW** value orderings for 5 problems using the high domain size. Experiments on other problem instances show similar results.

5.3 Results

5.3.1 "Blind" Value Ordering

Table 5.1 compares, for a static degree-based variable ordering, the initial solution found by the lowest value selection heuristic (labelled **LOW**) with that found by **ERR-LOW**. As the results show, the lowest value selection always finds an initial solution which is worse than lowest error selection. This is the result of bad selections (in terms of error incurred) throughout search. In assigning a value to a variable, search encounters one of two situations:

1. If the lowest value has been selected by the neighbour of a variable (and is thus inconsistent), then the next consistent value is selected.

2. If the lowest value has not been assigned to any past neighbour then it is assigned to the current variable.

As situation 1 results in students being required to sit consecutive exams, a large proportion of assignments made using lowest value selection to variables with one or more past neighbours will incur some error.

The Table also shows information about the initial solutions found using a random selection value ordering giving the minimum for a sample set of 100 runs. The results show that the lowest value selection method is consistently worse than the random selections. Like the lowest value ordering heuristic, random ordering does not consider the error incurred when assigning a variable and so is always likely to make costly assignments; however, it does not consistently make bad assignments in the same way as lowest value selection has been shown to. Both are called "blind" value ordering methods because neither looks at any search data when selecting a

value. The results show that, for such problems, blind value ordering does not result in competitive results compared to intelligent value ordering. This is as would be expected. Using **RND** as a control will show little as intelligent value ordering will almost always outperform it.

5.3.2 Non-Stochastic Value Orderings

5.3.2.1 Error-based value ordering

The error based value orderings, **ERR-LOW** and later **ERR-RND**, base the selection of the value to be assigned to a variable upon the amount of error an assignment will incur with the previously assigned variables. As shown above, this basic heuristic improves upon blind value ordering. In some cases several variables may incur the same amount of error, or even incur no error at all. In such a case **ERR-LOW** selects the lowest value incurring the minimum error.

5.3.2.2 Incorporating Look-Ahead

The previous section has shown how, using information regarding the error that a value assignment will incur, can improve the initial solution that is found by search. By incorporating extra information, in this case produced by look-ahead, search should be able to detect future error earlier and so find solutions of a lower cost.

Table 5.2 compares the initial solutions found by the two non-stochastic value ordering heuristics on the 9 smallest problems for 3 different static variable orderings. The highlighted values are those of lowest cost with reference to the problem and the variable ordering used. Three of the experiments(marked *) produced no result. This is a consequence of the nature of non-stochastic value ordering, whereby only one search path is considered and if this path results in a sub-problem that is difficult to solve, with respect to the hard constraints, then it is likely that thrashing will occur. In these cases re-labelling has been used to find a solution.

Although the additional look-ahead information should lead to an improved value ordering, and subsequent initial solution, the experiments show that this is not always the case. For the **KING-96** problem, look-ahead based value ordering resulted in an improvement for all of the variable orderings used. On the other hand, for problems such as **YOR-F-83** no improvement has occurred. Between these two extremes lie problems where one variable ordering leads to an improvement while another does not. The first two situations could be the result of problem-specific features, while the latter must also consider the way in which variables have been

		variable ordering		
problem	val.ord.	VAR	BD	WBD
HEC-S-92	ERR-LOW	**19455**	**22897**	**19064**
	IC-LOW	20862	23411	21328
STA-F-83	ERR-LOW	61971	63734	62695
	IC-LOW	**61026**	**61446**	**62355**
KING-96	ERR-LOW	13557	14354	12897
	IC-LOW	**12876**	**13280**	**12708**
YOR-F-83	ERR-LOW	**21067**	**22912**	**20252***
	IC-LOW	23726	25042	24327
UTE-S-92	ERR-LOW	**39522**	45511	42488
	IC-LOW	40847	**43917**	**41977**
EAR-F-83	ERR-LOW	**27469***	**29671**	**26771**
	IC-LOW	31250	33002	29093
TRE-S-92	ERR-LOW	**24602***	**24091**	**23097**
	IC-LOW	28133	28450	28123
LSE-F-91	ERR-LOW	**22355**	**23410**	**21772**
	IC-LOW	27189	27851	26520
KFU-S-93	ERR-LOW	**52014**	**67339**	**54179**
	IC-LOW	71292	75959	68466

Table 5.2: Comparison of **ERR-LOW** and **IC-LOW** searches. The **bold** values highlight which value ordering (*val. ord.*) was superior for each problem/variable ordering combination. Values marked with a * required the use re-labelling to find a viable solution.

ordered. The experiments have been repeated using the lower domain size, in Table 5.3, with similarly inconclusive results.

		variable ordering		
problem	val.ord.	VAR	BD	WBD
HEC-S-92	ERR-LOW	**41473**	41290	**35664**
	IC-LOW	48272	**40558**	35933
STA-F-83	ERR-LOW	111534	104084	**102341**
	IC-LOW	**102865**	**103555**	102409
KING-96	ERR-LOW	33058	32648	**31344**
	IC-LOW	**32978**	**30632**	31932
YOR-F-83	ERR-LOW	**43330**	**42249**	**41629**
	IC-LOW	44671	45925	44668
UTE-S-92	ERR-LOW	**83845**	87370	**78781**
	IC-LOW	90654	**81474**	84499
EAR-F-83	ERR-LOW	**47862**	**47130**	**43946**
	IC-LOW	52646	54267	51137
TRE-S-92	ERR-LOW	**43333**	**44728**	**40700**
	IC-LOW	45785	46512	45802
LSE-F-91	ERR-LOW	**36992**	**36757**	**33863**
	IC-LOW	39064	42083	36839
KFU-S-93	ERR-LOW	98416	**95054**	**88029**
	IC-LOW	**90225**	100521	92507

Table 5.3: Comparison of **ERR-LOW** and **IC-LOW** searches using the low domain sizes.

5.3.3 The Comparison of Value Orderings

By examining which constraints incurred what cost, the portions of the problem which cause one method to be better than another can be ascertained. This may shed some light on what factors contribute to one value ordering heuristic's superiority over another. The definition of critical constraints has been used in the previous chapter to analyse why one variable ordering provides superior results to another. In the case of comparing value orderings, the variable orderings used are the same. However, the solutions found by two value orderings can still be compared with respect to critical constraints that offer the most improvement over the alternate value ordering.

In comparing **ERR-LOW** and **IC-LOW**, the circumstances under which the look-ahead information used by **IC-LOW** will benefit search needs to be determined. Look-ahead in value ordering considers constraints which have one assigned and one un-assigned variable in their scope because all constraint are binary. The calculation performed by look-ahead determines the minimum error that will be incurred by assigning a value to the un-assigned variable of the constraint. This error is guaranteed to be incurred with respect to the current partial solution. When selecting a value for assignment to the current variable, look-ahead can be applied for each possible assignment and such calculations used when choosing a value.

Suppose we have the set of problem variables $V = \{v_1, ..., v_n\}$. We can define the set of intermediate constraints at any point during search, as the set of constraints $(v_i, v_j) \in C \, and \, v_i \in P \, and \, v_j \in F$ (where P is the set of past variables and F is the set of future or un-assigned variables). When the current variable, v_c, is assigned a value, the error incurred by all **intermediate constraints** will be considered when choosing a value. The intermediate constraints of a constraint have the same deepest variable as the given constraint and their remaining variables will be deeper in the variable ordering than that of the given constraint. An illustrative example is given in Figure 5.1 where constraints c_2, c_3 and c_4 are the intermediate constraints of constraint c_1. When variable 1 is assigned a value, constraint c_1 is considered. When assignments are made to variables 2, 3 and 4, constraint c_1 is considered, among others. Finally, when a value is assigned to variable 5, look-ahead has already calculated the costs of the possible assignments. Variables 2, 3, and 4 are called the **intermediate variables**. Therefore, the constraints for which the use of look-ahead will be advantageous will be those with a large number of **intermediate variables**, known as a large **intermediate degree**. The definition of intermediate degree extends the backward degree, or width [69], definition to try and focus on constraints

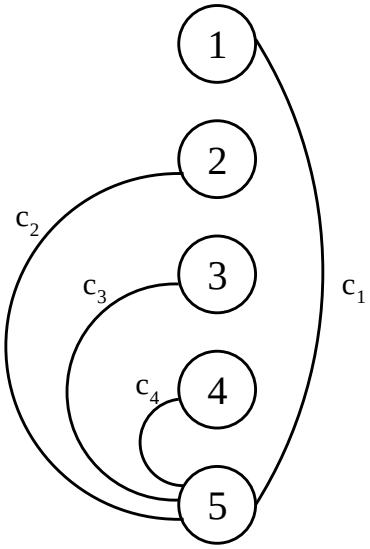

Figure 5.1: An example of the intermediate constraints of constraint c_1 considering the variables 1 to 5.

that will be hard to satisfy when using forward checking. The intermediate degree of a variable is more akin to the **bandwidth** of a variable, defined by Tsang as the maximum distance to a related variable within the variable ordering [69]. However, only past variables within the current variables neighbourhood are considered.

Table 5.4 compares, using the direct comparison method defined in Chapter 4, the mean intermediate constraint degree of the critical constraint set for each value ordering. The critical constraint set is defined as the constraints which incurred the least error for the solution set in comparison to the solution set for the opposite value ordering. Table 5.4 shows results for the **HEC-S-92** and **KING-96** problems using three different variable orderings; however, the results for other problems and using the lower domain size showed the same behaviour. The mean number of intermediate variables was compared and with the size of the critical constraint set being increased from 10 until they differed at the 5% level of confidence. The mean number of intermediate variables for the critical constraint set using look-ahead, in the column **IC-LOW**, is always significantly lower than those for results found without look-ahead, in the column **ERR-LOW**. This data suggests that **IC-LOW** finds it more difficult to satisfy constraints with a high intermediate degree than **ERR-LOW**.

problem	var.ord.	ERR-LOW	IC-LOW	size
HEC-S-92	VAR	8.08	11.65	30
	BD	8.49	14.73	22
	WBD	3.20	4.79	17
KING-96	VAR	13.59	20.96	11
	BD	10.49	17.46	15
	WBD	1.34	3.16	13

Table 5.4: Comparison of the critical constraints of each value ordering with respect to the other for the **HEC-S-92** and **KING-96** problems using various variable orderings (*var.ord.*). The mean intermediate degree of such constraints is shown with the size of critical constraint sets required to show significance (in the *size* column).

5.3.3.1 The cost of look-ahead

Tables 5.5 and 5.6 compare the median number of consistency checks needed to find a solution using the high and low domain sizes respectively. The results show that **IC-LOW** requires at least 5 times as many checks as **ERR-LOW**. This increase in cost is due to the fact that a large proportion of the checks performed by search are solely used to select a value. In order for such an increase in cost to be justifiable, confidence that the search will result in superior results is required.

		variable ordering		
problem	val.ord.	VAR	BD	WBD
HEC-S-92	ERR-LOW	392	384	384
	IC-LOW	2249	2069	1885
STA-F-83	ERR-LOW	306	306	306
	IC-LOW	1698	1471	1425
KING-96	ERR-LOW	95	92	92
	IC-LOW	463	439	405
YOR-F-83	ERR-LOW	1750	1722	943*
	IC-LOW	10065	9728	10439
UTE-S-92	ERR-LOW	257	248	248
	IC-LOW	1052	1209	1095
EAR-F-83	ERR-LOW	1100*	1811	1811
	IC-LOW	9558	10962	11250
TRE-S-92	ERR-LOW	1365*	2243	2243
	IC-LOW	13484	12360	12588
LSE-F-91	ERR-LOW	1223	1223	1223
	IC-LOW	4215	5876	5372
KFU-S-93	ERR-LOW	1732	1732	1732
	IC-LOW	39679	16850	10365

Table 5.5: Comparison of the median number of consistency checks ($\times 1000$) required to find a solution using the **ERR-LOW** and **IC-LOW** value orderings. The value with a * required re-labelling to find a solution.

		variable ordering		
problem	val.ord.	VAR	BD	WBD
HEC-S-92	ERR-LOW	244	171	1187
	IC-LOW	1762	1238	1156
STA-F-83	ERR-LOW	120	116	117
	IC-LOW	935	861	843
KING-96	ERR-LOW	44	33	34
	IC-LOW	220	199	196
YOR-F-83	ERR-LOW	2153	744	992
	IC-LOW	6834	5625	5451
UTE-S-92	ERR-LOW	122	108	101
	IC-LOW	589	608	601
EAR-F-83	ERR-LOW	1026	751	1102
	IC-LOW	7039	6509	7066
TRE-S-92	ERR-LOW	1061	911	1012
	IC-LOW	7905	8115	6298
LSE-F-91	ERR-LOW	617	516	538
	IC-LOW	3818	3844	3391
KFU-S-93	ERR-LOW	1913	749	776
	IC-LOW	7893	6864	6030

Table 5.6: As before the median number of consistency checks required by **ERR-LOW** and **IC-LOW** using the low domain sizes.

5.3.3.2 Summary

In summary, value orderings based upon some measure of the potential error incurred by an assignment lead to solutions of a lower cost. How such a measure is computed can also affect the solution found. The use of look-ahead can improve a solution, but empirical results show that the advantage is not apparent in all cases. However, the comparison is not conclusive. Each result is from a single solution, due to the search being non-stochastic, and there is the chance that the path to this solution may have encountered a problem feature that causes an exceptionally large amount of error to be incurred.

The cost of using look-ahead is high and, in this case, we cannot be confident that it will produce superior results so its use is not justified.

5.3.4 Stochastic Value Orderings

Of the stochastic value ordering heuristics used in these experiments, one is error based and the other is look-ahead based. The error based method (**ERR-RND**) orders values in a similar way to **ERR-LOW**, except that ties are broken randomly rather than on lowest value. The look-ahead based method (**IC-RND**) has a similar relationship with **IC-LOW**. Stochastic value orderings should help to establish, for a given problem and variable ordering, whether the use of look-ahead information improves the solutions found. One of the advantages of stochastic search methods is that they allow an ensemble of different solutions to be found. Then the best solution found, whether defined on solution cost or some user-defined preference, can be selected for use. The ensembles of solutions found in the different experiments will be compared on the mean solution cost as heuristics are best compared using a sample rather than on individual runs.

5.3.4.1 Stochastic Search with "look ahead"

The same experiments were repeated using the equivalent stochastic value orderings. Table 5.7 compares the mean solution costs of the two value ordering heuristics for the three variable orderings using the high domain size. The value ordering which is superior on average to a 5% level of confidence is highlighted in bold type. A comparison with the same results using the non-stochastic variants, in Table 5.2, shows that the better heuristic is different for only 3 instances. Table 5.8 compares the same set of results using the low domain size. Again, the results are different from the equivalent non-stochastic value ordering in only one case. An analysis of the number of intermediate variables for the critical constraint set of each value ordering pair was performed. The results were similar to those for the non-stochastic value ordering pairs and so confirmed the analysis. The median search costs required to find a solution for the stochastic value ordering are given. This resulted in less variation in the search cost for a value ordering from one variable ordering to another, when compared to non-stochastic value ordering. However, the same pattern with respect to the cost of look-ahead was observed, with **IC-RND** requiring at least 5 times more consistency checks than **ERR-RND**. The same assertion from this analysis, that confidence in an improvement due to the use of look-ahead will be required before it can be used, applies.

		variable ordering		
problem	val.ord.	VAR	BD	WBD
HEC-S-92	ERR-RND	**21756.1**	**23226.3**	**20636**
	IC-RND	22676.7	23588.2	22318
STA-F-83	ERR-RND	63856.1	63467.3	**62333.7**
	IC-RND	**63423.2**	63218.9	63345.6
KING-96	ERR-RND	14531.9	14656.7	13265.7
	IC-RND	**13634.1**	**13412.5**	13291.3
YOR-F-83	ERR-RND	**21876.3**	**22307.6**	**20989.4**
	IC-RND	23963.6	25355.6	24182.8
UTE-S-92	ERR-RND	40093.3	44213.8	**39000.3**
	IC-RND	**39732.3**	43849.5	41066.1
EAR-F-83	ERR-RND	**29038.1**	**29427.3**	**27581.7**
	IC-RND	31153.2	33383.7	30941.8
TRE-S-92	ERR-RND	**24901.6**	**25126.7**	**23562.9**
	IC-RND	28929.4	28257.9	28401.7
LSE-F-91	ERR-RND	**22632.9**	**23000.3**	**21072.7**
	IC-RND	25116.8	26118.1	25406.6
KFU-S-93	ERR-RND	**58818.7**	**63774.1**	**55886.7**
	IC-RND	67947.1	69989.2	68410.3

Table 5.7: Comparison of the mean solution cost of **ERR-RND** and **IC-RND** searches using the high domain size. **Bold** values show which value ordering (*val.ord.*) finds solutions of significantly lower cost .

		variable orderings		
problem	val.ord.	VAR	BD	WBD
HEC-S-92	ERR-RND	**39123.5**	42234.4	**38148.2**
	IC-RND	46896	**41260.6**	42749.9
STA-F-83	ERR-RND	**103798**	110166	**102160**
	IC-RND	104735	**106282**	104753
KING-96	ERR-RND	32573	32794.4	**30908.2**
	IC-RND	32451.8	**30644.1**	31937
YOR-F-83	ERR-RND	**42483.1**	**42485.1**	**41065.5**
	IC-RND	48350.4	45765.7	46392.6
UTE-S-92	ERR-RND	**81395.7**	83402.5	**81156.1**
	IC-RND	90266.7	83493.2	89709.3
EAR-F-83	ERR-RND	**49394.6**	**50000**	**47090.3**
	IC-RND	58299.3	53466.4	54584.8
TRE-S-92	ERR-RND	**42291.3**	**43020.3**	**39616.8**
	IC-RND	45960.5	45839.4	44724.6
LSE-F-91	ERR-RND	**35254.8**	**36744.4**	**32695.3**
	IC-RND	38412.4	39000.1	36980.8
KFU-S-93	ERR-RND	**88439.2**	**93855.4**	**85055.1**
	IC-RND	94582	97692.1	102013

Table 5.8: Comparison of mean solution cost of the **ERR-RND** and **IC-RND** value orderings (*val.ord.*) for the low domain sizes.

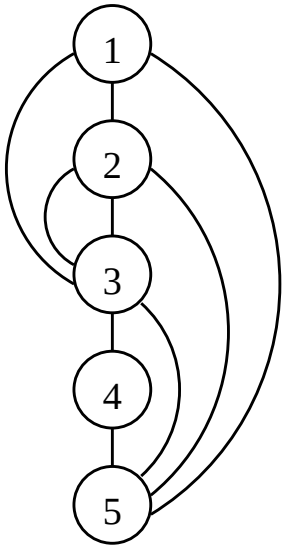

Figure 5.2: An example of shared past neighbour variables.

5.3.4.2 Exploiting Look-ahead in the Variable Ordering

The situation where look-ahead can improve upon purely error-based value selection has already been discussed. However, it is also important to consider when the use of look-ahead may result in a value assignment which ultimately results in a greater error. A **shared past neighbour variable** of a constraint is a variable which is connected to, and precedes in the variable order, all of the constraint variables. Figure 5.2 depicts an example of shared past neighbours. In this example the constraint $(3, 5)$ has two shared past neighbours(variables 1 and 2). Such variables will pose a problem when value ordering during search using the defined look-ahead method because assignments made to these variables (1 and 2 in this case) will be optimised considering the possible error of each constraint variable assignment; however, the error incurred by the constraint $(3, 5)$ will not be considered until one of the constraint variables is assigned a value. This optimisation of the constraints $(1, 3)$, $(1, 5)$, $(2, 3)$ and $(2, 5)$ before constraint $(3, 5)$ may result in the non-optimal assignment to the constraint variables. Therefore, the number of shared past neighbour variables of a constraint should indicate whether the look-ahead based value ordering will be of advantage with respect to a constraint. The number of such variables is called the **shared backward degree**.

Table 5.9 compares, using the direct comparison method, the shared backward degree of the critical constraint sets of each value ordering pair. For the majority of pairs for the smaller problems the shared backward degree is significantly lower

for the **IC-RND** value ordering. For the larger problems the results are mixed. However, as problem size increases the advantage of look-ahead value ordering diminishes.

problem	var.ord.	ERR-RND	IC-RND	size
HEC-S-92	VAR	3	**1.5**	11
	BD	-	-	-
	WBD	5.36	**0.5**	11
STA-F-83	VAR	3.92	**0.33**	12
	BD	5.73	**3**	22
	WBD	4.94	**1.57**	16
KING-96	VAR	-	-	-
	BD	2.09	**1.31**	22
	WBD	-	-	-
YOR-F-83	VAR	**1.59**	2.75	17
	BD	-	-	-
	WBD	4.38	**2.25**	13
UTE-S-92	VAR	**1**	2	12
	BD	2.85	**1.5**	13
	WBD	-	-	-
EAR-F-83	VAR	**2.87**	4.67	15
	BD	**7.33**	13	12
	WBD	**3.5**	6	12
TRE-S-92	VAR	**1.67**	3.33	12
	BD	10	**6.5**	11
	WBD	**1.53**	5	15
LSE-F-91	VAR	**1.31**	2.29	16
	BD	3.09	**1.5**	11
	WBD	**3.09**	8.5	11

Table 5.9: Comparison of the critical constraints of each value ordering with respect to the other. The mean shared backward degree of such constraints is shown. The **bold** values are significantly lower and *size* gives the number of critical constraints required to establish this significance. "-" has been used to represent when the results of search using the value ordering pairs were not significantly different.

A variable ordering, called **ELA**, has been developed to try and exploit look-ahead. It orders variables in a manner such that the shared backward degree of the heaviest constraints (i.e. the pairs of exams which are the most popular combinations) will be minimised while ignoring the number of intermediate variables. The ordering is constructed by considering the constraints in the order heaviest first.

	ELA		WBD
problem	ERR-RND	IC-RND	ERR-RND
HEC-S-92	**20636**	23274.2	22487.2
STA-F-83	62333.7	**60949.9**	61139.8
KING-96	**13265.7**	15147.5	13752.8
YOR-F-83	**20989.4**	22373.2	23745.4
UTE-S-92	**39000.3**	43544.5	41786.2
EAR-F-83	**27581.7**	30510.5	31388.1
TRE-S-92	**23562.9**	26886	27650
LSE-F-91	**21072.7**	25831.1	25623.3
KFU-S-93	60934.8	68644.9	55886.7

Table 5.10: Comparison, using the high domain size, of the mean solution costs found using **ELA** variable ordering with **ERR-RND** and **IC-RND** value ordering. The mean solution cost of **ERR-RND** using **WBD** is also given to compare the overall quality of solutions found using **ELA**.

Then, provided none of the constraint variables have already been placed in the ordering, the constraint variable with the heaviest variable weight is placed next in the order. The shared backward degree of the heaviest constraint is reduced because the heuristic attempts to place at least one of the constraint variables of these constraints near the start of the ordering. By ignoring the remaining variable of a constraint once one variable in its scope has been placed in the order, the number of intermediate variable will increase until another constraint places them in the ordering. Other variable orderings will not attempt to do this.

Table 5.10 compares the performance of the two stochastic value orderings using this new variable ordering. In the majority of cases **ERR-RND** outperforms **IC-RND** suggesting two possibilities about the reasoning behind **ELA**. Either it is false and low shared backward degrees combined with no limitation on the number of intermediate variables do not result in an increased performance for look-ahead; or these constraint features also favour pure error based value ordering. The fact that **ERR-RND** using **ELA** outperforms **ERR-RND** using **WBD** also, suggests that the latter is the case.

The same comparison was made using the low domain sizes. Again **ERR-RND** outperforms **IC-RND** using the **ELA** variable ordering; however, **ERR-RND** using **ELA** is not superior to **WBD**. **WBD** considers the backward degree of constraints, indirectly. On the other hand, **ELA** ignores the remaining variable in a constraint once one constraint variable has been placed in the ordering. Although

	ELA		WBD
problem	ERR-RND	IC-RND	RR-RND
HEC-S-92	42994.9	51311.8	**38148.2**
STA-F-83	**99331.2**	106188	102160
KING-96	34306.9	33621.1	**30908.2**
YOR-F-83	42826.5	49289	**41065.5**
UTE-S-92	86899.7	103712	**81156.1**
EAR-F-83	51635.8	53517.9	**47090.3**
TRE-S-92	44901	51499.3	**39616.8**
LSE-F-91	40417	48717.4	**32695.3**
KFU-S-93	93232	123961	**85055.1**

Table 5.11: Comparison, using the low domain size, of the mean solution costs found using **ELA** variable ordering with **ERR-RND** and **IC-RND** value ordering. The mean solution cost of **ERR-RND** using **WBD** is also given to compare the overall quality of solutions found using **ELA**.

such an action can reduce the shared backward degree of critical constraints it in no way guarantees a reduction in overall backward degree.

5.3.4.3 Summary

In summary, the use of stochasticity by way of random tie breaks provides several advantages. First, when comparing search types stochasticity allows samples of solutions to be compared. This allows confidence in the conclusions derived from empirical analysis. Second, the use of stochasticity can lead to a lower cost solution than non-stochastic methods provided a large enough sample is used. Third, in the real world, solution quality is not so clear cut. Therefore, a set of possible solutions is often provided to the end user for him/her to chose from at his/her discretion. The ensembles of solutions found by each method backs up, in the main, the deductions made using the empirical results for non-stochastic search regarding the superiority of either error based or look-ahead based value ordering. The situations under which constraints are satisfied better and worse when using look-ahead have been examined. This has led to the definition of a variable ordering which attempts to exploit look-ahead; however, results have not been clear.

5.3.5 Tie-breaking During Search

In this section I intend to compare stochastic and non-stochastic value ordering search results and examine when non-stochastic value ordering makes a "good" move and when some element of stochasticity is a better option.

There exists an overall solution space which contains every feasible solution (i.e. satisfying the hard constraints) each with an associated cost. This space would be generated by a blind random value ordering method which attempts to pick solutions at random. Error and look-ahead based value ordering and their tie-breaking methods will explore a smaller solution space. The aim of such methods is to concentrate search on areas of the solution space that contain the best solutions. The size of the solution space depends on the tie-break method chosen. Random tie-breaking leads to a solution space whose size is dependent upon the number of variable assignments that require a tie-break and the number of values involved in the tie-breaks. Provided that more than a few variables are involved in tie-breaks, the size of the reduced solution space is likely to be too large to search in its entirety. Of more concern is traversing this space to one of its lower cost members or to reduce this space yet again, removing higher cost members.

	ERR-RND			ERR-LOW	
problem	mean	std.dev.	min.	cost	prob.
HEC-S-92	20636	1052.74	18480	19064	0.0681
STA-F-83	62333.7	1226.06	59370	62695	0.6141
KING-96	13265.7	371.4	12452	12897	0.1611
YOR-F-83	20989.4	454.571	19744	20252*	0.0526
UTE-S-92	39000.3	2106.3	35418	42488	0.9515
EAR-F-83	27581.7	1052.63	25073	26771	0.2206
TRE-S-92	23562.9	496.729	22406	23097	0.1736
LSE-F-91	21072.7	626.991	19284	21772	0.8686
KFU-S-93	55886.7	2667.13	51872	54179	0.2611

Table 5.12: Comparison of the mean, standard deviation and minimum solution costs found by stochastic search (**ERR-RND**). The cost of the solution found by non-stochastic search (**ERR-LOW**). The probability of a solution of smaller cost occurring in the stochastic solution distribution is also given. These are the results of search using **WBD** and the high domain size. The value with a * required re-labelling to find a solution.

Tie-breaking on the lowest value leads to one solution out of the reduced solution space. If tie-breaking on the lowest value is consistently a "good" move during search

it may provide a quick and easy method of finding a solution to the problem. The results presented in previous sections have shown that lowest value tie-breaking usually provides a solution with a cost below, and if not below not far above, the mean cost of a random sample of the search space. A direct comparison is given in Table 5.12. The reason for this consistency may lie in the assignments made to variables with no past neighbours in the variable ordering. If values in the middle of the current domain are chosen, future variable domains will incur potential errors either side of this value, with reference to the **pws**(proximity weight set, see Chapter 3). Whereas lowest value tie-breaking will mean that half of this potential error is lost.

5.3.6 Hybrid Heuristics

Hybrid value ordering heuristics combine several value ordering strategies, depending upon the depth in search. As mentioned in the previous section, when assigning a value to a variable with no past neighbour, tie-breaking on the lowest value will result in less potential error in future variables. To this end a hybrid heuristic (called **ERR-HYD**) has been developed that tie-breaks on lowest value when there is no past neighbour, and otherwise tie-breaks randomly.

Table 5.13 compares the mean solution cost of the solutions found by **ERR-RND** and **ERR-HYD**, for three variable orderings and using the high domain size. The results show that the use of hybrid value ordering can be useful although this is not always the case. In fact, in 8 cases **ERR-HYD** is the superior value ordering and in 8 cases **ERR-RND** is superior. As the hybrid value ordering uses a lowest value tie-break when a variable has no past neighbours, the number of such variables in the first half of the variable ordering was compared. Only the first half of the variable ordering was considered because for some variable orderings a number of such variables occurred near the end of the ordering. Such variables will have little or no impact upon the effectiveness of hybrid value ordering. The majority of variable orderings only had 1 "no past neighbour" variable which consisted of the first variable or the ordering. However, **STA-F-83** had more than 1 for all three variable orderings and the **VAR** ordering had more than 1 for the **KING-96**, **YOR-F-83**, **EAR-F-83** and **KFU-S-93** problems. It is worth noting that for none of these instances does **ERR-RND** outperform **ERR-HYD**.

Table 5.14 shows the results of the same experiments repeated using the low domain size. These results are similar to those in Table 5.13 such that for approx-

		variable ordering		
problem	val.ord.	VAR	BD	WBD
HEC-S-92	ERR-HYD	**21231.6**	23110.9	20325.6
	ERR-RND	21756.1	23226.3	20636
STA-F-83	ERR-HYD	**61897.9**	**62901.7**	**61669.6**
	ERR-RND	63856.1	63467.3	62333.7
KING-96	ERR-HYD	**14251.3**	15054.2	13221
	ERR-RND	14531.9	**14656.7**	13265.7
YOR-F-83	ERR-HYD	**21544.1**	22267	**20764.8**
	ERR-RND	21876.3	22307.6	20989.4
UTE-S-92	ERR-HYD	40219.4	46613.1	40214.7
	ERR-RND	40093.3	**44213.8**	**39000.3**
EAR-F-83	ERR-HYD	29000.4	29696.3	**26719.8**
	ERR-RND	29038.1	**29427.3**	27581.7
TRE-S-92	ERR-HYD	24912.2	25055.7	23496.7
	ERR-RND	24901.6	25126.7	23562.9
LSE-F-91	ERR-HYD	22868.8	23561.2	21283.4
	ERR-RND	**22632.9**	**23000.3**	21072.7
KFU-S-93	ERR-HYD	58903.5	65346.1	57183
	ERR-RND	58818.7	**63774.1**	**55886.7**

Table 5.13: The mean solution costs found using hybrid value ordering (**ERR-HYD**) versus pure stochastic (**ERR-RND**). These results are for the high domain size.

		variable ordering		
problem	val.ord.	VAR	BD	WBD
HEC-S-92	ERR-HYD	39733.6	**41016.2**	**36953**
	ERR-RND	39123.5	42234.4	38148.2
STA-F-83	ERR-HYD	**103240**	**105725**	**101371**
	ERR-RND	103798	110166	102160
KING-96	ERR-HYD	**32370.6**	32873.6	31000.9
	ERR-RND	32573	**32794.4**	30908.2
YOR-F-83	ERR-HYD	**42003.9**	42900.5	41208.4
	ERR-RND	42483.1	**42485.1**	41065.5
UTE-S-92	ERR-HYD	81267.9	87646.3	81143.6
	ERR-RND	81395.7	**83402.5**	81156.1
EAR-F-83	ERR-HYD	**48063.2**	50310.7	**45842.7**
	ERR-RND	49394.6	50000	47090.3
TRE-S-92	ERR-HYD	42443.3	43101.9	39503.3
	ERR-RND	42291.3	43020.3	39616.8
LSE-F-91	ERR-HYD	35683.9	37035.8	33178.4
	ERR-RND	**35254.8**	36744.4	**32695.3**
KFU-S-93	ERR-HYD	89644.9	95900.9	84820
	ERR-RND	**88439.2**	**93855.4**	85055.1

Table 5.14: The mean solution costs found using hybrid value ordering (**ERR-HYD**) versus pure stochastic (**ERR-RND**) using low domain sizes.

imately half of the significantly different instance pairs **ERR-HYD** is superior to **ERR-RND**, and vice versa for the other half (the ratio is in fact 9 to 7 which favours **ERR-HYD** slightly). Of the 7 instance pairs whose variable orders have more than one "no past neighbour" variable only **KFU-S-93** using **VAR** contradicts the pattern shown when using the high domain size. This suggests that the use of hybrid value ordering is of more use when several such variables occur in the variable ordering.

The lowest solution cost found by **ERR-HYD** was then compared to that found by **ERR-RND** and published results in [17] and [32]. This comparison can be found in Table 5.15. Like **ERR-RND**, **ERR-HYD** outperforms **TABU** in all but one case. However, **ERR-HYD** results in the lowest solution cost in only one case (**CAR-S-91**) as **LBT** provides a better solution for the remaining cases where **ERR-HYD** outperforms **ERR-RND**. In each of the cases where **ERR-RND** is superior to **LBT**, **ERR-HYD** is superior also. This shows that **ERR-HYD** is equally competitive with **LBT**. A full comparison of the results presented in this

problem	ERR-RND	ERR-HYD	LBT [17]	TABU [32]
HEC-S-92	33549	32775	**30488.4**	35005.2
STA-F-83	**97095**	97258	98676.5	98248.8
YOR-F-83	39249	38909	39239.7	**38581**
UTE-S-92	77747	77749	**70950**	79750
EAR-F-83	43120	41346	**40950**	51412.5
TRE-S-92	**37317**	38007	41875.2	43620
LSE-F-91	30274	31085	**28623**	42253
KFU-S-92	78724	79805	**74886**	96282
RYE-S-93	111455	108526	**83825.9**	-
CAR-F-92	**79865**	80269	114197.8	**95778.8**
UTA-S-92	**72783**	73585	74431	89317.2
CAR-S-91	89415	**86894**	120174.6	104941.2
PUR-S-93	139210	139236	117124.8	-

Table 5.15: Comparison of the best results of this investigation with the results from [17] and [32].

thesis can be found in Appendix B.

5.3.6.1 Summary

The relationship between the solutions found when using related stochastic and non-stochastic value ordering heuristics has been examined. This has lead to the development of a hybrid value ordering heuristic(**ERR-HYD**) which combines the best elements of each method. Results show that such a heuristic competes with a pure stochastic method(**ERR-RND**) in general and surpasses **ERR-RND** when the variable ordering used has more than one variable with no past neighbours. A comparison of results with published results shows that **ERR-HYD** is competitive with the current best search method.

5.4 Conclusions

By applying value ordering, search can be guided toward solutions with a low cost. Different methods have been tested, notably both stochastic and non-stochastic, error-based and look-ahead based. Empirical research has shown that error-based value selection improves upon blind selection and that in some cases the use of look-ahead information can further the improvement. Some analysis of why look-ahead does not always enhance search has been performed. Such analysis is vital as it can reduce the workload when solving both this problem and other related real-world problems. The development of a variable ordering that exploits the use of look-ahead has proved inconclusive, although this is a good idea that should be given further consideration. A novell hybrid heuristic has been developed that aims to improve upon the solutions found by stochastic value orderings. Results have shown that such a heuristic is at least competitive with methods in the literature, depending upon the nature of the variable ordering used.

The key contribution of this chapter is the hybrid value ordering heuristic method which has been used to incorporate problem knowledge into solution methods.

Chapter 6

Hybrid Search

The work in Chapter 4 has shown that solution cost is closely related to the cost of satisfying the critical constraints. A complete search of the solution space would reduce the error incurred by the critical constraints and subsequently the overall solution cost. However, it has already been pointed out in Chapter 2, Section 2.3.3, that a complete search of a real-world problem instance will, in practice, never finish. In order to use the power of complete search to reduce the critical constraint error cost, yet avoid the drawbacks of a complete search of the whole problem instance, I suggest a hybrid search method. Hybrid search breaks the search for solutions down into two parts as follows:

- The first part consists of a complete search for the optimal solution to the critical constraints.
- The second part then extends the partial solution found to apply to the whole problem using a stochastic sampling technique.

It should be noted that the optimal solution to a problem does not necessarily contain the optimal solution to the critical constraints; however, as the critical constraints have been shown to have a major effect on overall solution quality, an optimal assignment to the critical constraints will lead to an improved solution to the problem instance in question. The performance of such a search will be compared to a pure stochastic sampling method with the question being: Will finding significantly better solutions to the **critical sub-problem**, i.e. the sub-problem consisting of the critical constraints, lead to better solutions to the overall problem?

The idea of combining search algorithms was initially suggested by Borrett, *et al* [7]. However, their work consisted of swapping between traditional tree searching

algorithms during search. The hybrid search method suggested in this thesis is more akin to the search procedure suggested by Coudert [20] where the maximum clique of a graph colouring problem is found before a sequential colouring is performed, first to the nodes of this clique then to the remaining nodes. However, the two parts of finding a solution in [20] are different types of search; one is a search for a sub-problem with a specific structure; the other a graph colouring assignment problem.

6.1 Related Search Methods

Intelligent backtracking methods have been used in the domain of classical CSPs to jump back up the search tree to areas of difficulty (dynamic backtracking [36], backjumping [31], conflict-directed backjumping [61], etc...). Their main drawback, in the context of what incomplete backtracking tries to achieve, is that because they are complete the search can only ever expect to visit a small proportion of the search tree in large problems. Real world problems are often over-constrained, i.e. all constraints cannot be satisfied. The introduction of soft constraints and the fact that problems are over-constrained mean that a large proportion of variables will be in the scope of constraints that have not been satisfied. Application of these methods to soft constraints results in intelligent backtracking methods failing to jump up the search tree by any significant amount as an assignment responsible for some error will be near by in the variable ordering. Even application to hard constraints has proved to be difficult because the definition of which variable is responsible for an inconsistency is hard to establish.

The motivation behind "Squeaky Wheel" optimisation [42] is that the difficult parts of the problem can be identified. The variables of the problem are re-ordered so that those variables that have a large negative effect upon the solution quality are moved up the variable order. The drawback of this method with respect to exam timetabling is the distribution of edge weights. Such problems only contain a small number of heavily weighted constraints, and increasingly more lighter constraints, it is the heavily weighted constraints that will be critical to solution quality. Work in Chapter 4 has identified these critical constraints and generalised their definition to one that can be applied before search. This is the definition of **critical constraint** used from now on, i.e. critical constraints are defined without any search information. By reducing their backward degree, i.e. by moving the critical constraint variables, or **critical variables**, up the variable order, lower cost solutions can be

found. However, if the critical constraints can be tackled by merely applying this knowledge to the heaviest constraints, then continual definition of a current critical constraint set and re-ordering will require a lot of work for small gains.

Limited Discrepancy Search [38] was introduced in Chapter 2. The principle used is that inconsistencies are usually due to a small number of "bad" assignments near the start of search. This is a useful principle in designing variable ordering heuristics and suggests that they should attempt to place critical variables near the start of the variable order. Application of this method to WCSPs could focus on reassigning the variables in the scope of the critical constraints; however, no consideration of the quality of satisfaction of the critical constraints is used, which could lead to time being wasted on extending solutions where the critical constraints are satisfied poorly.

Interleaved Depth-First Search [57] searches the solution space evenly, such that paths from across the whole search tree are considered during search. This "even-ness" property is useful when critical variables are near the start of the variable order as a whole variety of solutions to the critical constraints will be considered. As opposed to re-assigning the variables at the bottom of the variable ordering like DFS, IDFS re-assigns the first variable, performing a depth first traversal to find a solution, then re-assigns the second variable once all possible assignments to the first variable have been considered. The drawback of this method is that it requires exponential space which is an impossible requirement in the real-world. A usable limited form of IDFS is presented in [57]; however, it still requires partial search trees to be stored, although the storage space required is linear to search depth, and the search does not search the solution space as evenly as IDFS. Limited IDFS trades some of the even-ness of search for efficiency; however, it is concerned with searching the whole solution space which is not feasible when trying to optimise solutions for real world problems.

Depth-bounded Discrepancy Search [76] combines both IDFS and LDS. The use of DDS would encounter the same problems as LDS. Time could be wasted extending partial solutions to the critical constraints which are not very good.

6.2 Hybrid Search Implementation

In the context of hybrid search the critical constraints are defined as those that are heaviest, i.e. have the largest weight. Such a definition allows the critical constraints to be defined before search. The size of the set of critical constraints must not be too

problem	Constraints	Variables
HEC-S-92	9	5
STA-F-83	20	12
KING-96	23	23
YOR-F-83	17	14
UTE-S-92	19	12
EAR-F-83	11	10
TRE-S-92	15	13
LSE-F-91	11	13
KFU-S-93	10	5
RYE-S-93	22	11
CAR-F-92	19	15
UTA-S-92	7	6
CAR-S-91	5	6
PUR-S-93	6	4

Table 6.1: The number of constraints with a weight above 50% of the maximum weight and the number of variables involved in these constraints.

large, because such a definition will not allow the complete search phase of hybrid-search to find or improve partial solutions to the problem, or too small, as there will be little gain from optimising only a small number of critical constraints. The size has been defined in terms of 10 variables for several reasons. A larger size will result in poorer results from the complete search and, as Table 6.1 shows, a set of this size will, in most cases, contain the majority of the variables related to the heaviest constraints. The number of variables required to cover the critical constraints is specific to the distribution of constraint weights in the problem instance.

Two hybrid search procedures have been implemented. The first, referred to as **HYD-BIG**, performs a complete search method on the critical sub-problem and takes a stochastic sample of 100 solutions derived from the best solution found by the complete search.

The second method, **HYD-TEN**, records the last 10 solutions, the **partial solutions**, found by a complete search of the critical sub-problem (the best solution found will improve as complete search progresses) and samples 10 full solutions, the **sample solutions**, from each of these partial solutions stochastically. The number of partial solutions extended has been chosen as 10 because if this number is any larger the number of sample solutions from each partial solution will need to be reduced accordingly to balance the total search effort. The 10 sample solutions which are extended from each partial solution, in practice, provide enough variety

to search the various tie-breaks encountered during value ordering later in search. Most tie-breaks occur for the first few variable assignments, which are tackled by the complete search; therefore, only a small number of runs are required to search the lower tie-breaks sufficiently. Increasing the number of solutions extended from each partial solution, and subsequently decreasing the number of partial solutions extended, will, due to the aforementioned small number of tie-breaks deeper in the tree, result in little improvement upon the best sample solution for each partial solution.

A search limit has been set on the complete search, because even for a sub-problem of just 10 variables, complete search often requires a large amount of time. It was decided that a suitable limit would be the number of consistency checks required when taking 100 stochastic samples of the whole problem instance as this is the search method with which the hybrid search results are compared.

HYD-BIG captures the original motivation of hybrid search which is to exhaustively search for an optimal solution for the critical constraints and extend this to create solutions to the problem as a whole. **HYD-TEN** records a set of such solutions which should be dissimilar. Such a variety of solutions should hopefully lead to low solution costs. Figure 6.1 gives an example of how the extant solution cost is reduced by a complete search. After a period of large reductions, smaller improvements are made gradually as search progresses and as the backtracking reaches the variables near the start of the variable ordering. As more improvements are made the assignments of each improving solution should be more and more dissimilar. Therefore, provided the complete search follows this pattern, and does not find a near optimal solution straight away, **HYD-TEN** will have a variety of good solutions to extend into full solutions

Table 6.2 compares the minimum solution costs of the solutions found when search is limited with the optimal solution of each problem instance (i.e. a complete search of the critical sub-problem). As would be expected, the solution quality is better using a high domain as the problem is easier. These results could be a factored into whether the solutions found by hybrid search are of good quality.

It is worth noting that the structure of the critical sub-problem for each instance may lead to thrashing behaviour, defined in Chapter 2, when performing complete search. Two of the nine problems examined have disconnected critical sub-problems and others may be sparse. Although exploitation of such sub-problem features during the search of a large problem may not offer significant improvement, in complete search of a sub-problem of 10 variables it may allow an improvement.

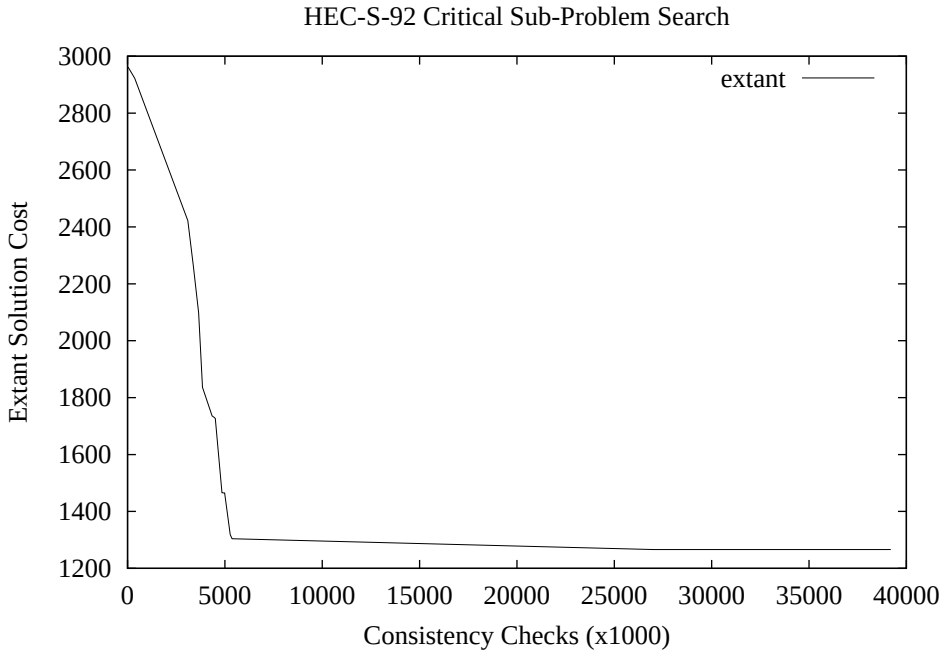

Figure 6.1: An example of the reduction in minimum solution cost as a complete search progresses.

The Stable set method described in Chapter 3 has been used.

6.3 Results

The two hybrid search methods implemented have been tested on each of the sample problem instances using five different variable orderings. The 10 critical variables are placed at the start of the ordering, to be assigned values by complete search, and the remaining variables are ordered according to previously defined static variable heuristics (**VAR**, **BD** and **WBD**, defined in chapter 4). The first sets of experiments were performed using the larger domain sizes and then repeated using the lower sizes.

	low domain		high domain	
problem	limited	un-limited	limited	un-limited
HEC-S-92	4722	4138	1266	1220
STA-F-83	6769	5994	3103	2624
KING-96	1044	1044	137	137
YOR-F-83	229	85	3	3
UTE-S-92	11512	8870	4872	3742
EAR-F-83	573	236	16	14
TRE-S-92	894	520	320	188
LSE-F-91	482	290	160	113
KFU-S-93	5257	4909	947	947

Table 6.2: Comparison of the best solution found by limited and unlimited searches of the critical sub-problem.

	variable ordering					
	VAR		BD		WBD	
problem	mean	min.	mean	min.	mean	min.
HEC-S-92	19111.1	**17283**	18353	18353	19159.5	18849
STA-F-83	59928.5	58939	60928.8	58596	**58811.1**	**57672**
KING-96	13814.9	13242	13931.4	12576	**12169.1**	**11800**
YOR-F-83	20706.3	19951	21644.5	20888	**20017.8**	**19394**
UTE-S-92	**37516.4**	**36146**	42309.9	41758	39203.2	38755
EAR-F-83	27932	26381	26005.4	25984	**25860.1**	**25391**
TRE-S-92	23469.3	23282	24577.7	24331	**22513.5**	**22386**
LSE-F-91	19740.1	19673	21002.7	20846	**18509.1**	**18368**
KFU-S-93	**51202.4**	**50047**	56685.5	56523	52614.3	51672

Table 6.3: Comparison **HYD-BIG** for various variable orderings on the high domain size. The mean and minimum solution costs are given. The **bold** results are those that are the lowest, with significance for the mean costs, for a problem instances.

Table 6.3 compares the solution costs found by **HYD-BIG** using the three variable orderings. The mean and minimum (**min.**) solution costs of each sample are both shown. The minimum solution cost has been shown in this comparison, and those that follow, because the minimum costs of the solutions found by different variable orderings and algorithms are needed to compare the solutions found by a stochastic search with those found by non-stochastic searches. As hybrid search combines both stochastic and non-stochastic search methods it is safer to compare the solution sets using both measures. The results in Table 6.3 show the superiority

of the **WBD** variable ordering method. This is as expected and follows on from the results in Chapter 4. **VAR** has a significantly lower mean, and minimum cost, for **UTE-S-92** and **KFU-S-93**. However, the general trend remains.

	variable ordering					
	VAR		BD		WBD	
problem	mean	min.	mean	min.	mean	min.
HEC-S-92	**18646**	**16600**	19588	18353	**18807.9**	18236
STA-F-83	60033.2	59259	61243.8	58981	**58832.1**	**57878**
KING-96	13814.5	13338	13669.7	12962	**12117.3**	**11800**
YOR-F-83	**20750.5**	20029	21926	20888	**20616**	**19427**
UTE-S-92	**39117**	**36253**	40410.6	37525	**38775.2**	36944
EAR-F-83	27790.9	26017	**26374.9**	25984	**26251**	**25393**
TRE-S-92	23357.6	22669	24577.7	24331	**22513.5**	**22386**
LSE-F-91	**19682.4**	19103	21739.8	20719	**20100.4**	**18120**
KFU-S-93	**52039.5**	**49135**	58610.6	53137	54369.6	51672

Table 6.4: Comparison of the mean and minimum costs of solutions found using **HYD-TEN** for various variable orderings on the high domain size.

The same experiment was carried out using the **HYD-TEN** algorithm. The results in Table 6.4 show that again **WBD** has a low mean solution cost in most cases. The competitiveness of the **VAR** ordering is increased in this set of results, with the solutions found by **VAR** being equally good, with respect to mean solution cost, for about half of the problem instances. However, **VAR** only records the minimum solution cost in 3 instances. Comparison of the average solution cost, by any measure, will be more sensitive for **HYD-TEN** as samples are being taken from several, possibly different, starting partial solutions.

	ERR-RND		HYD-BIG		HYD-TEN	
problem	mean	min.	mean	min.	mean	min.
HEC-S-92	20636	18480	19159.5	18849	**18807.9**	**18236**
STA-F-83	62333.7	59370	**58811.1**	**57672**	**58832.1**	57878
KING-96	13265.7	12452	**12169.1**	**11800**	**12117.3**	**11800**
YOR-F-83	20989.4	19744	**20017.8**	**19394**	20616	19427
UTE-S-92	39000.3	**35418**	39203.2	38755	**38775.2**	36944
EAR-F-83	27581.7	**25073**	**25860.1**	25391	26251	25393
TRE-S-92	23562.9	22406	**22513.5**	**22386**	**22513.5**	**22386**
LSE-F-91	21072.7	19284	**18509.1**	18368	20100.4	**18120**
KFU-S-93	55886.7	51872	**52614.3**	**51672**	54369.6	**51672**

Table 6.5: Comparison of the mean and minimum of the solution samples taken by **ERR-RND**, **HYD-BIG** and **HYD-TEN** using the **WBD** variable ordering on the high domain size.

The two hybrid search algorithms were then compared with each other and the stochastic search algorithm **ERR-RND**, from the previous two chapters, in Table 6.5. All three algorithms use the **WBD** variable ordering; however, **ERR-RND** does not place the critical variables at the start of the variable ordering so the variable orderings will differ. The results show that both hybrid search methods outperform **ERR-RND**. **HYD-BIG** has a significantly lower mean than **ERR-RND** for all but two problem instances; however, **HYD-TEN** is also competitive, outperforming **ERR-RND** for those two instances and equalling **HYD-BIG** for 3 of the other problems.

The above three experiments were then performed using the lower domain sizes. Table 6.6 compares **HYD-BIG** for the five variable orderings; Table 6.7 repeats this process for **HYD-TEN**; and Table 6.8 compare the two hybrid search methods with **ERR-RND**.

	variable ordering					
	VAR		BD		WBD	
problem	mean	min.	mean	min.	mean	min.
HEC-S-92	38481	38481	35921	**31971**	35013	35013
STA-F-83	104478	103004	99490.5	**98815**	102245	102245
KING-96	32610.6	31982	32335	**31073**	**31365.1**	31258
YOR-F-83	41956.1	**39447**	**41436**	41353	**41312.1**	40404
UTE-S-92	81985	80335	**80021.9**	**79819**	81750.3	81543
EAR-F-83	**47379.7**	**45906**	50012.8	49978	49718	46930
TRE-S-92	41759.4	41721	44509.4	44094	**40873.5**	**40501**
LSE-F-91	35481.4	35409	**32827.4**	**32724**	34345.1	34314
KFU-S-93	92289.4	91455	90287.6	83898	**82103.2**	**82058**

Table 6.6: Comparison of the mean and minimum costs of solutions found using **HYD-BIG** for various variable orderings on the low domain size.

	variable ordering					
	VAR		BD		WBD	
problem	mean	min.	mean	min.	mean	min.
HEC-S-92	39320.3	33585	38045.7	**31971**	**36197**	33528
STA-F-83	104497	103004	99577.3	**98815**	102245	102245
KING-96	32816.1	32106	32401.1	30972	**31116.9**	**30826**
YOR-F-83	42350.9	**40228**	42485.4	41353	42513.5	40404
UTE-S-92	83089.4	**78493**	82550.9	78880	**81118.6**	79899
EAR-F-83	49342	45906	49745.8	48205	**47101.7**	**44405**
TRE-S-92	42240.7	39327	43116.1	39383	**40089.5**	**38933**
LSE-F-91	35601.9	33636	34781.7	**32760**	**33722.6**	32909
KFU-S-93	89775.9	86818	97104.8	83898	**82312.7**	**79964**

Table 6.7: Comparison of the mean and minimum costs of solutions found using **HYD-TEN** for various variable orderings on the low domain size.

	ERR-RND		HYD-BIG		HYD-TEN	
problem	mean	min.	mean	min.	mean	min.
HEC-S-92	38148.2	33549	35013	35013	36197	**33528**
STA-F-83	102160	**97095**	102245	102245	102245	102245
KING-96	**30908.2**	**29525**	31365.1	31258	**31116.9**	30826
YOR-F-83	**41065.5**	**39249**	41312.1	40404	42513.5	40404
UTE-S-92	**81156.1**	**77747**	81750.3	81543	**81118.6**	79899
EAR-F-83	**47090.3**	**43120**	49718	46930	**47101.7**	44405
TRE-S-92	**39616.8**	**37317**	40873.5	40501	40089.5	38933
LSE-F-91	**32695.3**	**30274**	34345.1	34314	33722.6	32909
KFU-S-93	85055.1	**78724**	**82103.2**	82058	**82312.7**	79964

Table 6.8: Comparison of the mean and minimum of the solution samples taken by **ERR-RND**, **HYD-BIG** and **HYD-TEN** using the **WBD** variable ordering on the low domain size.

The comparison of **HYD-BIG**, in Table 6.6, is similar to that when using the higher domain values; however, in this case **BD** becomes more competitive, outperforming **WBD** in two cases and equalling **WBD** in one. The most reasonable explanation for this is that **BD** will avoid re-labelling. In fact **BD** requires fewer unlabels in all but one case and for the exception both **WBD** and **BD** require a low number of unlabels. The comparison of **HYD-TEN** in Table 6.7 shows **WBD** to be the superior variable ordering, although it does not have the lowest mean in 5 cases so such a conclusion should be taken tentatively. The comparison of the different algorithms in Table 6.8 shows **ERR-RND** to be superior to **HYD-BIG** in most cases. It is worth noting at this point the results of **HYD-BIG**: **HEC-S-92** and **STA-F-83** and **HYD-TEN**: **STA-F-83**. The solution sets of these searches only contain one solution, regardless of the number of samples taken. This was due to the partial solution found by the complete search of the critical sub-problem resulting in no tie-breaks later on in search. In 4 instances **HYD-TEN** is at least as good as either **ERR-RND** or **HYD-BIG** with respect to the mean solution cost, suggesting hybrid search could be competitive.

problem	ERR-RND	HYD-BIG	HYD-TEN
HEC-S-92	61	80	49
STA-F-83	1	1	1
KING-96	1	1	1
YOR-F-83	31	133	23
UTE-S-92	7	5	6
EAR-F-83	25	4	18
TRE-S-92	4	19	2
LSE-F-91	5	52	9
KFU-S-93	5	50	3

Table 6.9: Comparison of the median number of unlabels required by **ERR-RND**, **HYD-BIG** and **HYD-TEN** using the **WBD** variable ordering.

6.4 Discussion

The results of the previous section show that for a higher domain size the use of hybrid search can lead to better solutions. However, when the domain size is lower such an improvement is less frequent. Table 6.9 compares the number of unlabels required by the search algorithm. As mentioned in previous chapters, as the number of unlabels required by search increases, in general, the quality of the solutions found by search decreases. The most noticeable feature of this Table is that **HYD-BIG** usually requires a larger number of unlabels to find solutions. This explains the poor quality of the solutions found by this search method. However, **HYD-TEN**, more often than not, requires fewer unlabels than **HYD-BIG**. The single solution, to the critical sub-problem, found by **HYD-BIG** must play a large part in the number of subsequent unlabels required by the stochastic second search. As only one such solution is extended into complete solutions it is very important that these assignments do not lead to an increased possibility of unlabels later in search. Because **HYD-TEN** uses a sample of solutions to be extended, except in the case of **STA-F-83** where only one minimal solution is found, when the solutions are extended there will be a lower probability of requiring a large number of unlabels. It should also be noted that **HYD-TEN** always requires fewer unlabels, on average, than **ERR-RND**. This explains the improvement of **HYD-TEN** over **HYD-BIG** and its competitiveness with **ERR-RND**.

	ERR-RND		HYD-TEN		HYD-WBD	
problem	mean	min.	mean	min.	mean	min.
HEC-S-92	38148.2	33549	**36197**	33528	36945.1	**33513**
STA-F-83	102160	97095	102245	102245	101267	**96949**
KING-96	30908.2	29525	31116.9	30826	**30249.2**	**29099**
YOR-F-83	**41065.5**	**39249**	42513.5	40404	**41164.9**	39635
UTE-S-92	81156.1	77747	81118.6	79899	**79031.8**	**77479**
EAR-F-83	47090.3	43120	47101.7	44405	**44930.7**	**41170**
TRE-S-92	**39616.8**	**37317**	40089.5	38933	39928	39123
LSE-F-91	**32695.3**	**30274**	33722.6	32909	34055	32632
KFU-S-93	85055.1	**78724**	**82312.7**	79964	83259.2	78784

Table 6.10: Comparison of the mean and minimum of the solution samples taken by **ERR-RND**, **HYD-BIG** and **HYD-TEN** using the **WBD** variable ordering on the low domain size.

6.4.1 The Effect of Domain Size on Hybrid Search

These results suggest that extending a variety of partial solutions in hybrid search can lead to better complete solutions. However, the lower domain size results in re-labels being made which incur a larger cost than the re-labels performed in pure stochastic search. The variable orderings used by **ERR-RND** and **HYD-TEN**, although they are both based upon **WBD**, differ by the fact that **HYD-TEN** places the critical variables at the start of the variable ordering. The definition of the critical variables is such that only the soft constraints are considered. **WBD** does consider the hard constraints indirectly because a variable with a small weighted backward degree will be connected to only a few past variables. Such a variable ordering has already been shown to be superior to other tested orderings using a low domain size in Chapter 4. To this end, the experiments using **HYD-TEN** on the low domain size were repeated where the definition of critical variables is that defined by the **WBD** variable ordering. In effect the overall variable ordering used was **WBD**, with the first 10 variables being searched using the complete search method.

The results in Table 6.10 show that such a method, labelled **HYD-WBD**, is superior to **HYD-TEN** and subsequently more competitive with **ERR-RND**. In 5 out of the 9 problems **HYD-WBD** finds the lowest cost solution to the problem. With respect to the mean cost of the solutions found using hybrid search, **HYD-TEN** and **HYD-WBD** have the lowest mean solution cost in 6 out of the 9 problem instances.

problem	ERR-RND	HYD-TEN	HYD-WBD	LBT [17]	TABU [32]
HEC-S-92	33549	33528	33513	**30488.4**	35005.2
STA-F-83	97095	102245	**96949**	98676.5	**98248.8**
YOR-F-83	39249	40404	39635	39239.7	**38581**
UTE-S-92	77747	79899	77479	70950	79750
EAR-F-83	43120	44405	41170	**40950**	51412.5
TRE-S-92	**37317**	38933	39123	41875.2	43620
LSE-F-91	30274	32909	32632	**28623**	42253
KFU-S-92	78724	79964	78784	**74886**	96282
RYE-S-93	111455	114766	112267	**83825.9**	-
CAR-F-92	**79865**	81927	81446	114197.8	**95778.8**
UTA-S-92	**72783**	74205	74447	74431	89317.2
CAR-S-91	89415	89832	**87757**	120174.6	104941.2
PUR-S-93	139210	143539	-	117124.8	-

Table 6.11: Comparison of the best results of this investigation with the results from [17] and [32].

Finally, the results gained so far have been compared to the published results of Carter *et al* [17] and Di Gaspero and Schaerf [32] in Table 6.11. Again, as with **ERR-RND**, **HYD-WBD** outperforms **TABU** in all but one problem instance. **HYD-WBD**, however, is the overall superior search method in only two cases. The problem instances for which **HYD-WBD** outperforms **ERR-RND** are often the cases where both methods are inferior to other published methods. However, overall **HYD-WBD** competes with **LBT**. A full comparison of the results presented in this thesis can be found in Appendix B.

6.5 Summary

In this chapter the notion of Hybrid Search has been introduced and been suggested as a method of taking advantage of the critical constraint definition. Three novell and different hybrid search methods have been implemented, each exploiting different features of search. These methods have been shown to be superior to pure stochastic sampling when searching using a high domain size; and competitive when the domain size is lowered. The best method has been shown to outperform the TABU search results of [32] and compete with the LBT search method of [17].

The key contribution of this chapter is the use of complete search to satisfy the critical constraints.

Chapter 7

Conclusions

The contributions and conclusions of the work reported in this thesis are given below. The limitations of the work and possible future work are also discussed.

7.1 Summary of the work presented

The methods in this thesis have been applied to the University Examination Timetabling Problem (UETTP) as modelled by a Weighted CSP. The overall best results found have improved upon the best published for some cases. Even when the best solutions found are not as good as the published results they are still competitive.

The extension of Classical CSP tree-based search methods to Weighted CSPs has been investigated. Methods have been implemented and are valid for all WCSPs. It has been confirmed that complete backtracking, whether chronological or intelligent, rarely leads to; any improvement upon the first solution found by search; or the elimination of any violations of hard constraints. Therefore, all conclusions are drawn from searches where only the first complete solution is sought.

When comparing the performance of search heuristics the quality, or the cost, of the solutions found has been used. However, the work of this thesis has focused upon why one search heuristic leads to better solutions than another. Two analysis methods have been defined, in Chapter 4, to analyse the performance of searches by examining the properties of the solutions found. The general analysis method allows comparison between any number of unrelated searches and the direct analysis allows a comparison between two related searches, related by modified variable orderings or different value ordering heuristics in the case of this investigation. Such analysis methods define a set of critical constraints as those which incurred a large amount of error for the solutions being examined. These critical constraints are then examined

statistically to identify features responsible for low solution costs. In the case of Chapter 4 this amounted to calculating the mean backward degree of the critical constraints, i.e. the number of related constraints connected to past variables in the variable ordering. These new methods can be applied to any CSP although they are most useful to problems which incorporate some measure of preference to constraint satisfaction, as do WCSPs.

7.1.1 Variable Ordering

The order in which variables are tackled by search is of importance to both finding a solution and finding a good solution. Both static and dynamic variable orderings were investigated, in Chapter 4, with reference to these aims.

The aforementioned analysis methods have been applied to examine the performance of various static variable orderings. The constraint backward degree of the critical constraints has been identified as a key factor in the quality of the solutions found by search. This result has been used to justify a new variable ordering, **WBD**, which orders the variables on decreasing weighted backward degree. Search using this heuristic outperforms the tested static variable orderings. Such a variable ordering will apply to all WCSPs, with its effectiveness depending upon the distribution of constraint weights in the problem instance.

As in the work of Carter *et al* [17], the importance of the maximum clique of a problem to search was investigated. Search using a variable ordering heuristic which places the variables of a maximum clique at the start the ordering has been examined empirically. The motivation behind such a variable ordering is that the maximum cliques will be the most difficult parts of the problem for search to find valid and low cost solutions. The use of maximum cliques and maximum weighted cliques in such a way has proved inconclusive. Notably, this conflicts with the conclusions of Carter et al, in [17]. No details of the exact methods used are given in [17] leaving this conflict unexplained although some suggestions have been given in Chapter 4. The most plausible explanation is that search methods have been implemented in a different way. Such work is of relevance to solving real-world problems where finding a good solution to difficult sub-problems can be more important than the search of the remaining problem.

The incorporation of constraint weights into measures used to generate both static and dynamic variable orderings for Classical CSPs has been investigated and found to be useful in finding solutions of low cost although at the expense of some

extra search effort. Although such variable orderings have been used before the direct analysis performed helped to identify the reason for this superiority. This is of particular importance to WCSPs, and other Partial CSP types, where the benefit of incorporating preference information into heuristics may need to be investigated.

Two dynamic variable orderings extended from Classical CSP methods, by incorporating constraint weights, were tested empirically. Both heuristics extended the count of the number of values pruned from a variable's domain (where a previous assignment causes a hard constraint conflict) to consider the maximum weight of the constraints which prune a value (note that more than one previous assignment can conflict with subsequent assignment). This was called the weighted saturation degree. The Brelaz heuristic which, for Classical CSPs, selects as next the variable with the lowest domain size, tie breaking on degree, was found to be inferior to it's weighted extension. However, a related static variable ordering, which relaxed weight saturation degree to weighted backward degree, was found to be equally competitive. The conclusion of this was that static variable ordering heuristics may be more suitable to WCSPs than their dynamic counterparts. This was also shown to be case when the domain size used is close to the minimum required to satisfy the hard constraints. It is in such a case that one would expect dynamic variable ordering heuristics to be superior and so the conclusion of the study would appear to be counter intuitive to conventional variable ordering wisdom.

7.1.2 Value Ordering

Work in Chapter 5 has shown that the ordering which values are chosen is also critical to the cost of solutions. "Blind" value ordering heuristics, that do not take into account in any way the error incurred by an assignment, were shown to be far inferior to more conventional methods. Conventional methods that select a value on the lowest error that will be incurred with respect to the previous assignments has been implemented. Tie-breaking can be performed non-stochastically by the lowest such minimal error value, or stochastically by choosing from the set of minimal error values uniformly.

Both non-stochastic and stochastic value ordering heuristics have been compared. The research in Chapter 5 has shown that the error-based stochastic value ordering can lead to a solution of lower cost given a large enough sample size; although, error-based non-stochastic value ordering tends to lead search to a "good" solution. This applies to searches of all WCSPs of a size large enough to render complete search as

impractical. However, the performance of the non-stochastic lowest minimal error value selection may be specific to the UETTP and the constraint structure used.

The use of look-ahead information in value ordering heuristics has been investigated. In addition to the error incurred with respect to the past assignments, the error that will be incurred with respect to the current variable assignment and future assignments has been considered. Although the full impact of such future assignments cannot be known until they are attempted, a reduced amount of information can be gained. Such information can be seen to be of benefit in some cases; however, no overall conclusion can be ascertained. A variable ordering that attempted to exploit the heuristic was developed, but again no clear conclusion could be made. Empirical results have shown that the use of look-ahead is reasonably expensive and should only be used when there is confidence in its superiority. As this confidence could not be obtained the conclusion that look-ahead is too expensive was derived. Although look-ahead has been used in WCSPs before, it's application to value ordering is novel.

A hybrid value ordering heuristic has been defined which combines several value ordering heuristics. A combination of the stochastic and non-stochastic error based heuristics was investigated. Which heuristic is used when is dependent upon the number of related past variables with respect to the variable ordering used. When a variable has no past neighbours the non-stochastic method is a better choice. Results in Chapter 4 have shown that such a heuristic is superior to a pure stochastic value ordering whenever more than one variable has no past neighbours. Such a method is extendible to all WCSPs and the general idea could be used in any CSP especially for which a good but expensive value ordering heuristic exists. Such an expensive heuristic can be applied when it will be most beneficial with a cheaper heuristic being used for the remaining variables. This method is new and it's application to other WCSP and CSPs is of interest, specifically for problems where sub-problem difficulty can vary across the problem as a whole (as is the case with the UETTP).

7.1.3 Hybrid Search

The motivation behind the analysis of the critical constraints has been extended to form a hybrid search method. This method performs a complete search on the critical constraints, the definition of which is simplified as the heaviest constraints. Then the results of this search, which will be in the form of partial solutions, are extended using a stochastic search. Such a search has been shown to be competitive

with standard search methods and when the domain size used is high, in relation to the minimum domain size possible, it has been shown to be superior. Such a search method is applicable in principle to all WCSPs; however, it requires some definition of critical constraints.

7.2 Major Contributions

In summary the key contributions of this work lie in; the modelling of the UETTP as a WCSP; the identification methods for finding the critical constraints of a problem instance and the specific way they have been used; and the development of hybrid algorithms to exploit the problem structure with respect to the critical constraint in finding quality solutions to the problem.

7.3 Limitations

The work of this investigation is limited to a single WCSP type. This has been justified in that this allows the use of real-world data and the fact that the problem has been abstracted to a general level. As with any research that uses real-world problem instances the issue of whether results are problem specific occurs. This has been considered throughout the work conducted, however, more confidence in the results, conclusions and methods developed would be gained by extending the work to, or to abstractions of, other problem types.

The nature of the problem abstraction used has limited this study to binary constraints. The UETTP, and real-world problems in general, often use non-binary constraints. In principle, the methods described can be extended to non-binary constraints; however, this would require a large amount of work. At a basic level the satisfaction of non-binary constraints might only be considered when the last assignment to a constraint variable is made; however, as the arity of the constraints in the critical constraint set increase, the size of the critical constraint set will have to decrease in order to prevent the number of variables involved in the set becoming too large. A large number of variables involved in the critical constraint set would restrict the effectiveness of hybrid search techniques because the complete search section requires an exponentially increasing amount of time to remain effective. The interpretation of graph based terms such as backward degree would need to be investigated in order to find a extension to the binary definition that is still workable in the non-binary situation with respect to the research of this investigation. Prob-

lems which contain a mix of both binary and non-binary constraints, which UETTPs could conceivably be, would complicate matters even further.

7.4 Future Work

Confidence in results increases as the number of problem instances tested increases; however, one of the major limitations of WCSP research in general is the lack of a large testbed of problems. A random problem generator would address this limitation, to some degree, by providing a limitless set of artificial problem instances. Research into random generation of small-world problem graphs [2, 39, 78] could be extended to generate random problem instances with different constraint weight distributions.

In Chapter 4 the two aims of search, which are to satisfy the hard constraints and optimise the soft constraints, are discussed. Each variable ordering heuristic addresses these aims in different ways. The use of a dynamic variable heuristic which combines two heuristic measures could be investigated. The two problem aims of satisfying the hard constraints and minimising soft constraint error could be combined in a single heuristic measure with precedence of each aim being dependant upon the sub-problem structure.

The use of stochastic value ordering heuristics could be extended to consider near minimal error assignments. The current best method selects the next value for assignment by which has the lowest error and tie-breaks randomly. An extended heuristic would allow an assignment that is non-minimal, with respect to the error incurred, and would tie-break with a probability distribution related to the different errors associated with each possible value selection. The usefulness of small amounts of stochasticity in variable ordering has been observed by Bresina [9] and may also apply to value ordering.

The limitations of using look-ahead in value ordering could be addressed by the use of a hybrid value ordering heuristic. Look-ahead information could be limited to, and hence would only need to be calculated for, variables at the start of search. The remaining variable assignments would be chosen using a cheaper, or less computationally expensive, heuristic. The aim would be to use all available information to optimise the assignments to the critical constraints while not incurring a large computational cost for the rest of search.

Hybrid search could be extended to consider other search methods. A stochastic sample of a larger critical sub-problem could find better partial solutions. The best

of these solutions could then be extended. This would avoid the cost of having to extend every partial solution, as would be the case if a stochastic sample of the whole problem was taken.

Bibliography

[1] Uwe Aickelin and Kathryn A. Dowsland. Exploiting problem structure in a genetic algorithm approach to a nurse rostering problem. *Journal of Scheduling*, 3:139–153, 2000.

[2] A-L. Barabási and R. Albert. Emergence of scaling in random networks. *Science*, (286):509–512, 1999.

[3] Christian Bessiere and Jean-Charles Regin. MAC and combined heuristics: Two reasons to forsake FC (and CBJ?) on hard problems. In *Principles and Practice of Constraint Programming CP-96*, pages 61–75, 1996.

[4] S. Bistarelli, U. Montanari, and F. Rossi. Constraint solving over semi-rings. In *Proceedings of IJCAI-95*. Morgan Kaufman, 1995.

[5] S. Bistarelli, U. Montanari, F. Rossi, T.Schiex, G. Verfaillie, and H. Fargier. Semi-ring-based CSPs and valued CSPs: Frameworks, properties, and comparison. *Constraints*, 4(3):275–316, 1999.

[6] P. Boizumault, Y. Delon, and L. Peridy. Constraint logic programming for examination timetabling. *The Journal of Logic Programming*, pages 217–233, 1995.

[7] J. E. Borrett, Edward P. K. Tsang, and N. R. Walsh. Adaptive constraint satisfaction: The quickest first principle. In *Proceedings of ECAI-96*, pages 160–164, 1996.

[8] D. Brelaz. New methods to color the vertices of a graph. *Communications of the ACM*, 22(4):251–256, 1979.

[9] John L. Bresina. Heuristic-biased stochastic sampling. In *Proceedings of AAAI/IAAI-96, Vol. 1*, pages 271–278, 1996.

[10] E. K. Burke and J. P. Newall. A multi-stage evolutionary algorithm for the timetable problem. *IEEE Transactions on Evolutionary Computation*, 3.1:63–74, 1999.

[11] E. K. Burke, J. P. Newall, and R. F. Weare. A memetic algorithm for university exam timetabling. In E. K. Burke and P. Ross, editors, *The Practice and Theory of Automated Timetabling (PATAT-96)*, volume 1153 of *Lecture Notes in Computer Science (LNCS)*, pages 241–250. Springer, 1996.

[12] E.K. Burke, D.G. Elliman, P.H. Ford, and R.F. Weare. Examination timetabling in British universities - a survey. In E. K. Burke and P. Ross, editors, *PATAT-96*, volume 1153 of *LNCS*, pages 76–92. Springer, 1996.

[13] E.K. Burke, J.P. Newall, and R.F. Weare. A simple heuristically guided search for the timetable problem. In E. Alpaydin and C Fyte, editors, *Proceedings of EIS-98*, pages 574–579. ICSC Academic Press, 1998.

[14] B. Cabon, S. de Givry, and G. Verfaillie. Anytime lower bounds for constraint violation minimisation problems. In M. Maher and J.-F. Puget, editors, *Principles and Practice of Constraint Programming CP-98*, volume 1520 of *LNCS*, pages 117–131. Springer, 1998.

[15] M. W. Carter. A survey of practical applications of examination timetabling algorithms. *Operations Research*, 34(2):193–202, 1986.

[16] M. W. Carter, G. Laporte, and J. W. Chinneck. A general examination scheduling system. *Interfaces*, 24(3):109–120, 1994.

[17] M. W. Carter, G. Laporte, and S. Y. Lee. Examination timetabling: Algorithmic strategies and applications. *Journal of the Operational Research Society*, 47:373–383, 1996.

[18] P. Cheeseman, B. Selman, and W. M. Taylor. Where the really hard problems are. In *Proceedings of IJCAI-91*, pages 331–337, 1991.

[19] P. R. Cohen. *Empirical Methods for Artificial Intelligence*. MIT Press, 1995.

[20] O. Coudert. Exact coloring of real-life graphs is easy. In *Proceedings of the 34th Annual ACM IEEE Design Automation Conference*, pages 121–126, 1997.

[21] A.J. Davenport and E.P.K. Tsang. Solving constraint satisfaction sequencing problems by iterative repair. In *Proceedings of PACLP-99*, pages 345–357, 1999.

[22] R. Dechter and J. Pearl. Network-based heuristics for constraint-satisfaction problems. *Artificial Intelligence*, 34:1–38, 1988.

[23] Rina Dechter. Enhancement schemes for constraint processing: backjumping, learning, and cutset decomposition. *Artificial Intelligence*, 31(3):273–312, 1990.

[24] D. Dubios, H. Fargier, and H. Prade. The calculus of fuzzy restrictions as a basis for flexible constraint satisfaction. In *Proceedings of the 2nd IEEE International Conference on Fuzzy Systems*, pages 1131–1136, 1993.

[25] H. Fargier and J. Lang. Uncertainty in constraint satisfaction problems: a probabilistic approach. In M. Clark, R. Kruse, and S. Moral, editors, *Proceedings of ECSQARU-93*, volume 747 of *LNCS*, pages 97–104. Springer-Verlag, 1993.

[26] H. Fargier, J. Lang, and T. Schiex. Selecting preferred solutions in fuzzy constraint satisfaction problems. In *Proceedings of EUFIT-93*, pages 1128–1134, 1993.

[27] E. C. Freuder and R. J. Wallace. Partial constraint satisfaction. *Artificial Intelligence*, (58):21–70, 1992.

[28] Eugene C. Freuder. A sufficient condition for backtrack-free search. *Journal of the ACM*, 29(1):24–32, 1982.

[29] D. Frost and R. Dechter. Look-ahead value ordering of constraint satisfaction problems. In *In Proceedings of IJCAI-95*, pages 572–578, 1995.

[30] Philippe Galinier and Jin-Kao Hao. Tabu search for maximal constraint satisfaction problems. In *Principles and Practice of Constraint Programming CP-97*, pages 196–208, 1997.

[31] J. Gaschnig. *Performance Measurement and Analysis of Certain Search Algorithms*. PhD thesis, Department of Computer Science, Carnegie-Mellon University,, 1979.

[32] L. Di Gaspero and A. Schaerf. Tabu search techniques for examination timetabling. In *Proceedings of PATAT-00*, pages 76–179, 2000.

[33] P. A. Geelen. Dual viewpoint heuristics for binary constraint satisfaction problems. In *Proceedings of ECAI-92*, pages 31–35, 1992.

[34] I. Gent, E. MacIntyre, P. Prosser, B. Smith, and T. Walsh. Random constraint satisfaction: Flaws and structure. Research report 98.23, School of Computer Studies, University of Leeds, 1998.

[35] I. P. Gent, E. MacIntyre, P. Prosser, B. M. Smith, and T. Walsh. An empirical study of dynamic variable ordering heuristics for the constraint satisfaction problem. In *Principles and Practice of Constraint Programming CP-96*, volume 1118 of *LNCS*, pages 179–193. Springer, 1996.

[36] M. L. Ginsberg. Dynamic backtracking. *Journal of Artificial Intelligence Research*, 1:25–46, 1993.

[37] R. M. Haralick and G. L. Elliott. Increasing tree search efficiency for constraint satisfaction problems. *Artificial Intelligence*, 14:263–313, 1980.

[38] W. D. Harvey and M. L. Ginsberg. Limited discrepancy search. In *Proceedings of IJCAI-95*, pages 607–613, 1995.

[39] T. Hogg. Refining the phase transition in combinatorial search. *Artificial Intelligence*, 81(1-2):127–154, 1996.

[40] S. Hurley, D. H. Smith, and S. U. Thiel. FASoft: A system for discrete channel frequency assignment. *Radio Science*, 32(5):1921–1939, 1997.

[41] D. Johnson. Timetabling university examinations. *Journal of the Operational Research Society*, 41(1):39–47, 1990.

[42] David E. Joslin and David P. Clements. "Squeaky wheel" optimization. *Journal of Artificial Intelligence Research*, 10:353–373, 1999.

[43] R.W.L. Kam and J.H.M. Lee. Fuzzifying the constraint hierarchies framework. In *Principles and Practices of Constraint Programming CP-98*, volume 1520 of *LNCS*, pages 280–294. Springer-Verlag, 1998.

[44] G. Laporte and S. Desroches. Examination timetabling by computer. *Computers and Operations Research*, 11(4):351–360, 1984.

[45] J. Larossa and P. Meseguer. Partition based lower bound for MAX-CSP. In J. Jaffar, editor, *Principles and Practice of Constraint Programming CP-99*, pages 303–315, 1999.

[46] J. Larrosa and P. Meseguer. Exploiting the use of DAC in MAX-CSP. In *Principles and Practice of Constraint Programming CP-96*, pages 308–322, 1996.

[47] J. Larrosa and P. Meseguer. Partial lazy forward checking. URL : citeseer.nj.nec.com/larrosa97partial.html, 1997.

[48] J. Larrosa, P. Meseguer, T. Schiex, and G. Verfaillie. Maintaining reversible DAC for MAX-CSP. *Artificial Intelligence*, 107:149–163, 1999.

[49] Javier Larrosa and Pedro Meseguer. Optimization-based heuristics for maximal constraint satisfaction. In Ugo Montanari and Francesca Rossi, editors, *Principles and Practice of Constraint Programming CP-95*, volume 976 of *LNCS*, pages 103–120. Springer, 1995.

[50] F. T. Leighton. A graph colouring algorithm for large scheduling problems. *Journal of Research of the National Bureau of Standards*, 84(6):489–506, 1979.

[51] Andrew Lim, Ang Juay Chin, and Oon Wee Chong. A campus-wide university examination timetabling application. In *Proceedings of IAAI 2000*, pages 1020–1015, 2000.

[52] Vahid Lofti and Robert Cerveny. A final-exam-scheduling package. *Journal of the Operational Research Society*, 42(3):205–216, 1991.

[53] H. Terashima Marín, P. M. Ross, and M. Valenzuela-Rendon. Evolution of constraint satisfaction strategies in examination timetabling. In *Proceedings of GECCO-99*, pages 635–642. Morgan Kaufmann, 1999.

[54] Hugo Terashima Marín. *Combinations of GAs and CSP Strategies for Solving the Examination Timetabling Problem.* PhD thesis, Instituto Tacnol'ogico y de Estudios Superiores de Monterrey, 1998.

[55] David W. Matula, George Marble, and Joel D. Isaacson. Graph coloring algorithms. In R. C. Read, editor, *Graph Theory and Computing*, pages 109–122. Academic Press, Inc., 1972.

[56] N. K. Mehta. The application of a graph coloring method to and examination scheduling problem. *Interfaces*, 11(5):57–65, 1981.

[57] P. Meseguer. Interleaved depth-first search. In *Proceedings of IJCAI-97*, pages 1382–1387, 1997.

[58] Pedro Meseguer and Martì Sánchez. Specializing Russian doll search. In Toby Walsh, editor, *Principles and Practice of Constraint Programming CP-01*, volume 2239 of *LNCS*, pages 464–479. Springer, 2001.

[59] Craig Morgenstern. Distributed coloration neighborhood search. In David S. Johnson and Michael A. Trick, editors, *Cliques, Coloring, and Satisfiability: Second DIMACS Implementation Challenge, 1993*, volume 26, pages 335–357. American Mathematical Society, 1996.

[60] W.P.M. Nuijten, G.M. Kunnen, E.H.L. Aarts, and F.P.M. Dignum. Examination time tabling: A case study for constraint satisfaction. In *ECAI-94 Workshop on Constraint Satisfaction Issues Raised by Practical Applications*, pages 11–19, 1994.

[61] P. Prosser. Hybrid algorithms for the constraint satisfaction problem. *Computational Intelligence*, 9(3):268–299, 1993.

[62] P. Prosser. An empirical study of phase transitions in binary constraint satisfaction problems. *Journal of Artificial Intelligence*, (81):81–109, 1996.

[63] Andrea Schaerf. A survey of automated timetabling. *Artificial Intelligence Review*, 13(2):87–127, 1999.

[64] T. Schiex. Possibilistic constraint satisfaction problems or "How to handle soft constraints?". In D. Dubois, M. P. Wellman, B. D'Ambrosio, and P. Smets, editors, *Uncertainty in Artificial Intelligence: Proceedings of the Eighth Conference*, pages 268–275. Morngan Kaufmann, 1992.

[65] T. Schiex, H. Fargier, and G. Verfaillie. Valued constraint satisfaction problems: Hard and easy problems. In *Proceedings of IJCAI-95*. Morgan Kaufman, 1995.

[66] B. M. Smith. Phase transition and the mushy region in constraint satisfaction problems. In *Proceedings of ECAI-94*, pages 100–104, 1994.

[67] B. M. Smith and S. A. Grant. Modelling exceptionally hard constraint satisfaction problems. In G. Smolka, editor, *Principles and Practice of Constraint Programming - CP97*, volume 1330 of *LNCS*, pages 182–195. Springer, 1997.

[68] Barbara M. Smith and Stuart A. Grant. Trying harder to fail first. In *Proceedings of ECAI-98*, pages 249–253, 1998.

[69] E. Tsang. *Foundations of Constraint Satisfaction*. Academic Press, 1993.

[70] M. Vernooy and W. S. Havens. An examination of probibilistic value-ordering heuristics. In *Proceedings of IJCAI-99*, pages 340–352, 1999.

[71] R. Wallace. Directed arc consistency preprocessing. In M. Meyer, editor, *Selected Papers from ECAI-94*, volume 923 of *LNCS*, pages 121–137. Springer, 1995.

[72] R. J. Wallace. Analysis of heuristic methods for partial constraint satisfaction problems. In *Principles and Practice of Constraint Programming CP-96*, 1996.

[73] R. J. Wallace and E. C. Freuder. Conjunctive width heuristics for maximal constraint satisfaction. In *Proceedings of AAAI-93*, pages 762–768, 1993.

[74] Richard Wallace and Eugene Freuder. Heuristic methods for over-constrained constraint satisfaction problems. In Michael Jampel, Eugene Freduer, and Michael Maher, editors, *OCS-95: Workshop on Over-Constrained Systems at CP-95*, 18 1995.

[75] Richard J. Wallace and Eugene C. Freuder. Anytime algorithms for constraint satisfaction and SAT problems. In *Working Notes of IJCAI-95 Workshop on Anytime Algorithms and Deliberation Scheduling*, 1995.

[76] Toby Walsh. Depth-bounded discrepancy search. In *Proceedings of IJCAI-97*, pages 1388–1395, 1997.

[77] Toby Walsh. Search in a small world. In *Proceedings of IJCAI-99*, pages 1172–1177, 1999.

[78] D. Watts and S. Strogatz. Collective dynamics of 'small-world' networks. *Nature*, (393):440–442, 1998.

[79] D. J. A. Welsh and M. B. Powell. The upper bound to the chromatic number of a graph and its application to timetabling problems. *The Computer Journal*, 10:85–86, 1967.

[80] George M. White and Bill S. Xie. Examination timetables and tabu search with longer-term memory. In Edmund K. Burke and Wilhelm Erben, editors, *PATAT-00*, number 2079 in LNCS, pages 85–103. Springer, 2000.

[81] C.P. Williams and T. Hogg. Exploiting the deep structure of constraint problems. *Journal of Artificial Intelligence*, (70):73–117, 1994.

[82] M. Wilson and A. Borning. Hierarchical constraint logic programming. *The Journal of Logic Programming*, 16:277–318, 1993.

Appendix A

Examples

	Remaining Domain Values					
Action	PC	CS	MA	SS	PY	EN
-	1,2,3	1,2,3	1,2,3	1,2,3	1,2,3	1,2,3
PC=1	**1**	2,3	2,3	2,3	2,3	2,3
CS=2	1	**2**	3	3	3	2,3
MA=3	1	2	**3**	$\emptyset$	$\emptyset$	2,3
Forced SS=1	1	2	3	**1**	$\emptyset$	2,3
Undo	$\emptyset$	2	3	1	$\emptyset$	2,3
Reshuffle	CS	MA	SS	PC	PY	EN
	2	3	1	$\emptyset$	$\emptyset$	2,3

Table A.1: An example of the backtrack method described in Algorithm 4. In this example and attempt is being made to assign 3 colours to the graph in Figure 2.1. After the first 3 assignments an assignment cannot be made to SS. An assignment is forced upon it and any conflicting assignment is unlabelled (in this case PC= 1). As the variable PC has no alternative assignment it is shuffled down the variable ordering for later consideration. If an alternative assignment to PC had existed, then this assignment would have been made. This method of re-assignment or re-shuffling (depending upon whether consistent assignments exist) is performed on every assignment that conflicts with the forced assignment (SS= 1 in this case).

Constraint	Error
(CS,MA)	8
(SS,PC)	6
(PY,PC)	4
(CS,PY)	2
(CS,SS)	2
(CS,PC)	0
(MA,PY)	0
(MA,PC)	0
(MA,SS)	0
(PC,EN)	0
(SS,EN)	0

Table A.2: The error produced by each constraint for the solution found by the search in Table A.4. The critical constraints, as used in a general comparison, would consist of those with the largest error.

Constraint	DEG	WD	Diff.
(CS,MA)	8	0	8
(SS,PC)	6	3	3
(PY,PC)	4	2	2
(CS,PY)	2	0	2
(CS,SS)	2	2	0
(MA,PC)	0	0	0
(PC,EN)	0	0	0
(MA,SS)	0	2	-2
(CS,PC)	0	4	-4
(SS,EN)	0	4	-4
(MA,PY)	0	8	-8
Total	22	21	1

Table A.3: The difference in error produced by each constraint for the solution found by the search of the Table 3.2 problem using **DEG** and **WD**. The critical constraints, as used in a direct comparison, would consist of those with the largest error difference in favour of the superior ordering.

	Slot	PC=1					
PC	1	**0**					
	2	0					
	3	0					
	4	0					
	5	0					
			CS=4				
CS	1	x	x				
	2	4	4				
	3	2	2				
	4	0	**0**				
	5	0	0				
				MA=5			
MA	1	x	x	x			
	2	8	12	12			
	3	4	12	12			
	4	0	x	x			
	5	0	8	**8**			
					SS=2		
SS	1	x	x	x	x		
	2	6	8	8	**8**		
	3	3	7	8	8		
	4	0	x	x	x		
	5	0	4	x	x		
						PY=2	
PY	1	x	x	x	x	x	
	2	4	6	6	6	**6**	
	3	2	6	10	10	10	
	4	0	x	x	x	x	
	5	0	4	x	x	x	
							EN=5
EN	1	x	x	x	x	x	x
	2	4	4	4	x	x	x
	3	2	2	2	6	6	6
	4	0	0	0	2	2	2
	5	0	0	0	0	0	**0**

Table A.4: An example of search using the **ERR-LOW** value ordering heuristic on the example problem given in Table 3.2 and Figure 2.1. For simplicity a **pws** of $\{2, 1\}$ has been used (i.e. there is an error of 2 for each pair of consecutive exams taken by each student and an error of 1 for each pair of exams separated by only one slot). There are 5 timetable slots and each column shows the error that will be incurred by assigning an exam to a time-slot of each row (with an "x" being used to record that a value has been pruned). The **bold** values define the minimum error that is incurred at the time of the assignment.

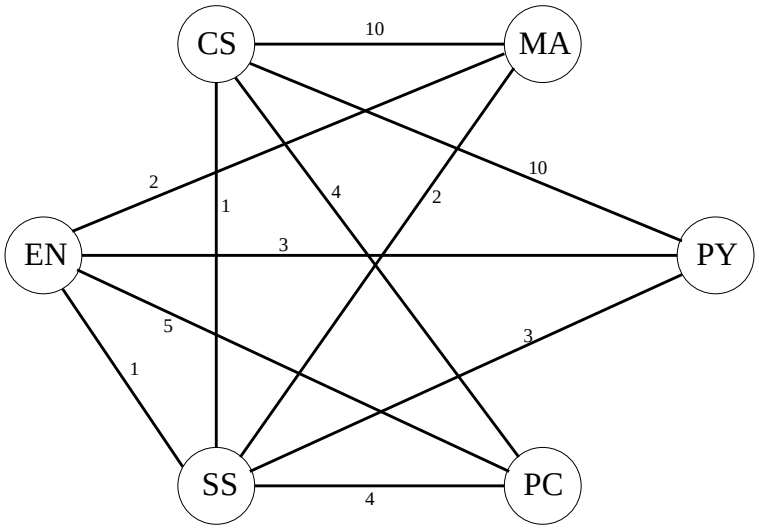

Figure A.1: Constraint graph of a second example problem.

	Domain of SS					
Action	1	2	3	4	5	wsd
-	0	0	0	0	0	0
CS=1	1	0	0	0	0	1
MA=4	1	0	0	2	0	3
PY=4	1	0	0	2,3	0	4
PC=4	1	0	0	2,3,4	0	5

Table A.5: An example of how the *wsd* of a variable develops as assignments are made to related variables.
The domain size of each variable is 5; the problem is that described in Figure A.1; and a **pws** of $\{2, 1\}$ has been used. For each value in the domain of SS the weight of the edges, or constraints, which prune the value is recorded. For example, for the partial assignment $\{CS = 1,\ MA = 4,\ PY = 4\}$, the value 4 is pruned by both MA and PY. The weight of the constraints which prune the values are recorded as 2 and 3 respectively. The *wsd* of the variable is the sum of the maximum pruning constraint weight for each value (i.e. the sum of the maximum values in each element of a table row.

Appendix B

Results Compendium

The following 2 pages given the lowest solution found using the different search algorithms, value orderings and variable orderings on the lower problem domain sizes. The best solution for each problem has been highlighted in **bold**. Where a "-" entry occurs the experiment was not carried out. This situation occurs for the larger problems when the drawbacks of inferior heuristics would be magnified.

Search Alg/ Value Ord.	Var. Ord.	Problem Instance						
		HEC-S-92	STA-F-83	KING-96	YOR-F-83	UTE-S-92	EAR-F-83	TRE-S-92
ERR-RND	DEG	37540	99415	31173	39775	84174	42847	40483
	FD	37007	99939	33134	39672	88269	46307	42276
	VAR	34850	97911	30629	39892	78034	44656	39851
	BD	35629	100594	32212	40000	78307	44493	39477
	WBD	33549	97095	29525	39249	77747	43120	**37317**
	CLQ-DEG	37575	99933	31865	41300	82958	46892	41429
	CLQ-FD	37346	100951	33025	40879	88615	47700	41990
	CLQ-VAR	35953	98554	30721	40157	78309	46671	40991
	WCLQ-DEG	37566	101050	30760	41244	82344	46387	41229
	WCLQ-FD	37963	100127	32357	41015	83650	46933	41828
	WCLQ-VAR	36849	98292	30089	39942	81018	47317	39679
	MV	37131	101503	31959	41510	82830	46807	42537
	ME	37646	101163	32688	41246	83165	49563	43437
	BZ	36357	101888	31159	40076	80576	45378	39777
	DD	35738	101727	31428	**38910**	79949	44720	40278
	WSD	35239	97805	31315	39306	**77221**	43493	38645
IC-RND	VAR	36943	100124	30339	44228	84204	47830	41970
	BD	36879	101776	30593	43374	80962	49652	42119
	WBD	35070	101339	31827	42894	85313	47677	41375
	ELA	39380	100276	30714	42467	89577	47242	42662
HYD-BIG	VAR	38481	103004	31982	39447	80335	45906	41721
	BD	**31971**	98815	31073	41353	79819	49978	44094
	WBD	35013	102245	31258	40404	81543	46930	40501
HYD-TEN	VAR	33585	103004	32106	40228	78493	45906	39327
	BD	**31971**	98815	30972	41353	78880	48205	39383
	WBD	33528	102245	30826	40404	79899	44405	38933
HYD-	VAR	35114	105347	31782	41274	78852	46749	40847
	BD	35312	100534	33329	41195	84583	46478	39860
	WBD	33513	**96949**	**29099**	39635	77479	**41170**	39123

Search Alg/ Value Ord.	Var. Ord.	Problem Instance						
		LSE-F-91	KFU-S-93	RYE-S-93	CAR-F-92	UTA-S-92	CAR-S-91	PUR-S-93
ERR-RND	DEG	34013	89413	-	-	-	-	-
	FD	36077	87901	-	-	-	-	-
	VAR	31771	78384	-	-	-	-	-
	BD	33497	85155	124165	84705	77435	-	143256
	WBD	30274	78724	**111455**	**79865**	**72783**	89415	**139210**
	CLQ-DEG	35265	89799	-	-	-	-	-
	CLQ-FD	35606	91448	-	-	-	-	-
	CLQ-VAR	33934	84745	-	-	-	-	-
	WCLQ-DEG	33847	89559	-	-	-	-	-
	WCLQ-FD	36083	93143	-	-	-	-	-
	WCLQ-VAR	33499	81423	-	-	-	-	-
	MV	34624	91376	-	-	-	-	-
	ME	35505	96100	-	-	-	-	-
	BZ	32721	86603	-	-	-	-	-
	DD	32860	85577	-	-	-	-	-
	WSD	**30212**	**76267**	-	-	-	-	-
IC-RND	VAR	33847	82958	-	-	-	-	-
	BD	35384	88273	-	-	-	-	-
	WBD	34062	87948	-	-	-	-	-
	ELA	36039	93361	-	-	-	-	-
HYD-BIG	VAR	35409	91455	-	-	-	-	-
	BD	32724	83898	-	-	-	-	-
	WBD	34314	82058	-	-	-	-	-
HYD-TEN	VAR	33636	86818	-	-	-	-	-
	BD	32760	83898	-	-	-	-	-
	WBD	32909	79964	114766	81927	74205	89832	143539
HYD-	VAR	32619	88672	-	-	-	-	-
	BD	34531	84466	-	-	-	-	-
	WBD	32632	78784	112267	81446	74447	**87757**	142436