University of Leeds
SCHOOL OF COMPUTER STUDIES
RESEARCH REPORT SERIES
Report 97.45

Trying Harder to Fail First

by
Barbara M Smith and Stuart A Grant

November 1997

Abstract

Variable ordering heuristics can have a profound effect on the performance of backtracking search algorithms for constraint satisfaction problems. The smallest-remaining-domain heuristic is a commonly-used dynamic variable ordering heuristic, used in conjunction with algorithms such as forward checking which look ahead at the effects of each variable instantiation on those variables not yet instantiated. This heuristic has been explained as an implementation of the fail-first principle, stated by Haralick and Elliott [7], i.e. that the next variable selected should be the one which is most likely to result in an immediate failure.

We calculate the probability that a variable will fail when using the forward checking algorithm to solve a class of binary CSPs. We derive a series of heuristics, starting with smallest-remaining-domain, based on increasingly accurate estimates of this probability, and predict that if the fail-first principle is sound, the more accurate the estimate the better the performance should be. We describe experiments applying these heuristics, in conjunction with the forward checking algorithm, to a large set of randomly-generated problems from the same class. Our predictions are not borne out by the experimental results: putting more effort into estimating the probability does not in general pay off. Our results thus refute the fail-first principle and show that the success of smallest-remaining-domain and related heuristics must be explained in some other way.

1 Introduction

A constraint satisfaction problem (CSP) consists of a set of variables, each of which has a set of possible values, known as its domain, and among which a set of constraints are imposed. A solution to a CSP is an assignment of a value to every variable such that no constraints are violated. Many types of practical problem, including many varieties of scheduling problem, can be formulated as CSPs, so that algorithms which can solve CSPs, or prove that no solution exists, are of practical importance. Since the CSP is an NP-complete problem, techniques which improve the performance of these algorithms are highly desirable.

CSPs are commonly solved using systematic backtracking search algorithms, which repeatedly choose a variable which is currently unassigned, attempt to assign a value to it, and then either proceed to choose another variable or, if a failure occurs, backtrack. The way in which the next variable is chosen can have a significant effect on the size of the search space explored by the algorithm. Either the variables can be considered in a predetermined order (a static variable ordering, or SVO) or the effect of the assignments

already made can be taken into account when choosing the next variable, giving a dynamic variable ordering, or DVO, strategy, which could potentially lead to a different variable ordering along each branch in the search tree.

In this paper we are concerned with DVO heuristics. A possible basis for devising DVO heuristics is the 'fail-first' principle, introduced by Haralick and Elliott [7], who in the same paper introduced the forward checking and full-lookahead algorithms. This style of algorithm, which 'looks ahead' after each variable assignment, makes DVO particularly appropriate, since information on the effects of the past assignments is readily available.

Haralick and Elliott expressed the fail-first principle as 'To succeed, try first where you are most likely to fail'. They implemented it, in conjunction with a lookahead algorithm, by a DVO heuristic which chooses next the variable with smallest remaining domain. 'Smallest remaining domain' is still a popular variable ordering heuristic, and is often seen as synonymous with fail-first; we shall term it FF in this paper, but it should be emphasised that this is not the only way of implementing the fail-first principle as a DVO, and conversely, as we shall discuss, its success may not necessarily be due to the fact that it implements fail-first.

The study reported in this paper began as an attempt to find improved variable ordering heuristics based on the fail-first principle for a class of binary CSPs. This attempt failed, which lead us to some unexpected conclusions about the fail-first principle itself.

2 The Fail-First Principle

Our discussion and experiments are based on the forward checking algorithm: whenever a variable is assigned a value, values of uninstantiated or *future* variables which conflict with this assignment are removed from their domains. The assignment fails if, as a result, some future variable has a domain wipeout, i.e. the assignment causes its domain to become empty; the values removed are then restored and an alternative value tried instead. We shall be concerned solely with binary CSPs, i.e. CSPs in which each constraint affects two variables. The future variables, with their remaining domains and the constraints between them, then form a future subproblem which is itself a binary CSP.

In discussing the effect of changing the order in which the variables are assigned values, Haralick and Elliott [7] make the assumption that *'the best search order is the one which minimizes the expected length or depth of each branch'*. They show that the expected branch depth can be minimized by choosing at each level the variable which has the smallest probability of succeeding, and therefore the smallest probability of becoming a parent node.

The fail-first principle therefore says that we should choose next the variable whose success probability is smallest, in order to minimize the expected branch length and thereby minimize the search cost.

For forward checking, the success probability is the probability that at least one of the remaining values of a variable does not cause a domain wipeout. Haralick and Elliott show that under certain assumptions, the success probability is minimized by choosing the variable with smallest remaining domain. In this paper, we calculate the success probability more precisely for a class of randomly-generated binary CSPs and hence derive new heuristics implementing the fail-first principle.

3 Variants of 'Smallest-Remaining-Domain'

As well as new fail-first heuristics, later in the paper we evaluate a number of existing variants on the FF heuristic. These variants express the intuitive idea that a variable which constrains many future variables is also likely to cause a domain wipeout, so that the degree of the variables should be taken into account as well as their domain sizes.

- When all variables have the same initial domain size, a variant of the FF heuristic is used by Frost and Dechter [4] which selects the *first* variable to instantiate as the one with the highest degree, i.e. the one constraining the largest number of other variables. Thereafter, the 'smallest remaining domain' strategy is used. We term this DVO heuristic FFdeg, for fail-first with initial degree ordering[1].

- A DVO heuristic originally developed for graph colouring problems by Brélaz [2] can also be applied to CSPs. The Brélaz heuristic (BZ) selects the variable with the smallest remaining domain and breaks ties by selecting the variable with the highest *future degree*, i.e. the one constraining the largest number of future variables.

- Bessière and Régin [1] show that the SVO which considers variables in descending order of degree gives good results in comparison with FF when the constraints are sparse, but performs very badly on complete constraint graphs, when it degenerates to lexicographic ordering. Conversely, FF does much better when the constraints are dense, since the fact that it ignores the degrees of the variables becomes less important. They proposed a heuristic, dom/deg, which combines the two and minimizes the ratio of current domain size to (original) degree. Here we

[1] Frost and Dechter referred to this heuristic simply as DVO.

consider a variant of this which minimizes the ratio of current domain size to future degree.[2]

4 New Fail-First Heuristics

In deriving the 'smallest-remaining-domain' heuristic (FF) as an implementation of the fail-first principle, Haralick and Elliott [7] assume that the probability that the assignment of a value to a variable fails (in the context of forward checking, results in a domain wipeout) is the same for all available values of all unassigned variables. On that assumption, the probability that the variable chosen will fail (i.e. the probability that every value will lead to a domain wipeout) is maximized by choosing the variable with smallest domain. However, it is clear that other factors, such as the number of future variables which each variable constrains, also affect this probability. The variants of FF already discussed take some account of the future degree of each variable. However, if we want to follow the fail-first principle, it would be better to incorporate these other factors when calculating the probability of failure.

We calculate the failure probability for a class of binary CSPs. We assume that there are n variables, that the set of pairs of variables which have constraints between them is known, and that each variable has m possible values. When there is a constraint between two variables, the constraint tightness, i.e. the probability that two values are inconsistent, is a constant p_2 for all constraints. Suppose that after a number of successful past assignments, we have a future subproblem consisting of a set F of unassigned variables, each variable $v_i \in F$ having current domain size m_i. If there is a constraint between two of these variables, v_i and v_j, then due to the values which have been removed from their domains by the past instantiations, the current tightness of this constraint is p_{ij}, measured by the proportion of the remaining pairs of values which are not allowed.

The fail-first principle says that we should choose next the variable in F which is most likely to fail, i.e. which maximizes the probability that every one of its possible values will result in a domain wipeout.

If we consider a variable $v_i \in F$ with current domain size m_i,

$$\Pr\{\text{every assignment of } v_i \text{ fails}\} = (\Pr\{v_i = x \text{ fails}\})^{m_i}$$

where x is any value in the current domain of v_i, assuming that the failure of the assignment $v_i = x$ is independent of the failure of any other value for v_i.

[2]Bessière and Régin considered this heuristic and found its performance roughly similar to that of dom/deg.

If there is a constraint between $v_j \in F$ and v_i and the current tightness of this constraint is p_{ij},

$$\begin{aligned}\Pr\{v_i = x_i \text{ is consistent with at least one value of } v_j\} \\ &= 1 - \Pr\{v_i = x_i \text{ is inconsistent with every value of } v_j\} \\ &= (1 - p_{ij}^{m_j})\end{aligned}$$

approximately, if we take the current constraint tightness p_{ij} as a probability applying independently to each pair of values. If there is no constraint between v_i and v_j then $p_{ij} = 0$.

Using the above,

$$\begin{aligned}&\Pr\{v_i = x_i \text{ fails}\} \\ &\quad = 1 - \prod_{v_j \in F, j \neq i} \Pr\{v_i = x_i \text{ is consistent with at least one value of } v_j\} \\ &\quad = 1 - \prod_{v_j \in F, j \neq i} (1 - p_{ij}^{m_j}) \qquad (1)\end{aligned}$$

Therefore, to choose the variable that is most likely to lead to failure in the future subproblem, we should choose the variable v_i which maximizes

$$(1 - \prod_{v_j \in F, j \neq i} (1 - p_{ij}^{m_j}))^{m_i} \qquad (2)$$

Depending on how much we estimate versus how much we accurately measure in the environment of each variable v_i, this gives us a series of heuristics.

First, if we assume, as Haralick and Elliott did, that the term (1) is the same for every value of every variable v_i, then to maximize (2) we should minimize the number of such terms (since they are all < 1) and hence minimize m_i. Thus, we choose the variable that has the smallest current domain, giving the FF heuristic.

Secondly, we could estimate the current tightness of the constraints between v_i and the other future variables by their original tightness (i.e. 0 or p_2) and use the initial domain size, m, as the estimate for the current domain size of each future variable. Then, to maximize (2), we maximize:

$$(1 - (1 - p_2^m)^{d_i})^{m_i}$$

where d_i is the degree of v_i in the future subproblem, i.e. the number of future variables that it constrains. This gives a heuristic that, like BZ and DD, chooses the next variable to instantiate on the basis of both its domain size and its future degree.

Thirdly, we could use the true current domain size of all future variables, but estimate the current constraint tightness by p_2. Then we want to maximize:

$$(1 - \prod_{v_j \in F, v_i \text{ constrains } v_j} (1 - p_2^{m_j}))^{m_i}$$

If two variables have the same current domain size, this leads us to prefer the one which minimizes $\prod(1 - p_2^{m_j})$. As well as maximizing the number of terms in the product, maximizing this expression will tend to maximize $p_2^{m_j}$, for each j, and hence minimize m_j. Thus, we favour variables *adjacent to future variables with small domains.*

Finally, we can also measure the current tightness of the constraints, and calculate (2) accurately for each variable. Then to maximize $p_{ij}^{m_j}$ we should maximize p_{ij} (as well as minimizing m_j). This chooses a variable *involved in tight constraints*, other things being equal.

It is intuitive that if we want to choose a variable such that all of its available values are likely to cause a domain wipeout in some future variable we should look for a variable which has few remaining values and which is involved in many tight constraints with future variables which themselves have few remaining values. The final heuristic will choose such a variable if there is one.

We term the second, third and fourth heuristics FF2, FF3 and FF4 respectively, and in the following sections we present experimental evidence on their performance relative to FF.

Since FF, FF2, FF3 and FF4 are based on incorporating successively more information about the current subproblem into the estimate of the probability that a variable will fail, we should expect, if the fail-first principle is sound, to see this reflected in decreasing search effort on the part of the forward checking algorithm for CSPs of the type on which the probabilities are based. The heuristics are, of course, also increasingly expensive to apply (and we have ignored this cost in the experiments reported below) but we might hope that FF2 or FF3 provides good performance without the necessity of recalculating the tightness of every constraint after each instantiation, as required for FF4, which is particularly time-consuming.

We might also expect to see FF2, for instance, perform better than the variants of FF (FFdeg, BZ and DD) which take into account the same characteristics of the future variables, but from the point of view of the fail-first principle are less accurate approximations to maximizing the failure probability.

On the other hand, should the proposed heuristics not perform well on this class of problem, this would call into question the fail-first principle itself. Our experiments will therefore not just give us a ranking of the heuristics under investigation, an approach criticized by Hooker [8] since it tells us

"which algorithms are better, but not why". Instead, we have been able to make predictions about the relative performance of the heuristics, which we can now put to the test; the result will either tend to confirm, or will refute, the fail-first principle.

5 The Random Generation Model

The experiments reported here use randomly-generated binary CSPs. Each set of problems is defined by the 4-tuple $\langle n, m, p_1, p_2 \rangle$, where n is the number of variables; m is the size of each variable's domain; p_1, the constraint density, is the proportion of pairs of variables which have a constraint between them; and p_2, the constraint tightness, is the probability that a pair of values is inconsistent, given that there is a constraint between a pair of variables. This is the same model as used for the derivation of the new fail-first heuristics. The problem generator ensures that all problems are generated with connected constraint graphs, so that the resultant problem cannot be decomposed into smaller components.

We use the 3-tuple $\langle n, m, p_1 \rangle$ to denote a problem class in which p_2 is allowed to vary. Each of the experiments reported below is carried out on a $\langle 20, 10, p_1 \rangle$ problem class, with $p_1 = 0.2, 0.5$ or 1.0. For each problem class, we vary the constraint tightness, p_2, in steps of 0.01 over the range [0.01..1] and generate 1,000 random CSPs with each tuple of $\langle 20, 10, p_1, p_2 \rangle$ parameters. This ensures that we see problems from the under-constrained *'easy-soluble'* region when p_2 is small, the over-constrained *'easy-insoluble'* region when p_2 is large, and the hard *phase transition* region corresponding to intermediate values of p_2. The phenomenon of the phase transition in NP-complete problems was discussed by Cheeseman, Kanefsky and Taylor [3], and studies of the phase transition in binary CSPs have been carried out [11, 13]. In comparing the performance of heuristics, it is important to make the comparison across the three regions defined by the phase transition; it is conceivable, for instance, that a heuristic could perform relatively well in one region but not in another. It is also vital that problems from the phase transition are fully represented in the experiments, since for these problems the median cost is highest.

Each problem is solved using the forward checking algorithm combined with each of the DVO heuristics under test in turn. The cost of each search is measured in terms of the number of consistency checks made by the algorithm in either finding a solution or proving that none exists. As already stated, this measure ignores the cost of implementing the DVO heuristic, which for some of the heuristics under investigation can be considerable. The chief aim is not to test the implementation efficiency of the heuristics, but only their effect on the search process.

For each problem class and each value of p_2 we calculate the median cost of solving each problem in the ensemble or proving that it has no solution. We plot the cost against the *general constrainedness parameter*, κ, introduced in [6], rather than against p_2. κ generalizes the specific parameters defining constraints in several classes of NP-complete problem, such as CSPs, SAT and graph colouring. The phase transition is predicted to occur when the expected number of solutions is 1, corresponding to $\kappa = 1$, and this prediction has been shown to be reliable for many random binary CSP problem classes, with the exception of those with low constraint density [11, 13]. Plotting the cost against κ rescales the horizontal axis and simplifies the comparison between problem classes with phase transitions at different values of p_2.

The software used to perform the experiments is implemented in C and runs over a network of 75 Silicon Graphics Indigo workstations under a UNIX environment. The next section presents the experimental results for the new fail-first heuristics presented in section 4.

6 Results for the New Heuristics

In section 4, we predicted that we should see successive improvements in performance from the heuristics FF2, FF3 and FF4 compared to FF: FF4, which puts the greatest effort into accurately calculating the probability of failure for each variable, should give the greatest improvement in search cost. Figure 1 shows the actual results for the four heuristics, used in conjunction with the forward checking algorithm and applied to three problem classes, each with 20 variables and 10 values per variable. In each plot, we show the median cost for each heuristic for a range of values of p_2 centred on the phase transition. The key indicates the ranking of each heuristic at the phase transition.

Unfortunately, the ranking is almost the reverse of what we predicted; FF4 is particularly bad, being always worse than FF2 and FF3. For the $\langle 20, 10, 1.0 \rangle$ problems, FF4 is hugely expensive compared to the other heuristics[3]. The original heuristic, FF, does relatively badly when the constraints are sparse, since it ignores the degrees of the variables; when $p_1 = 1$, it is identical to FF2, since all variables then always constrain the same number of future variables. FF3 should in theory be identical to FF and FF2 when $p_1 = 1$, but when there are several variables with the same minimum domain size, it occasionally chooses a different variable from the other two heuristics, due to the imprecision of floating point arithmetic.

FF4 does out-perform FF on the easiest problems, in all three problem classes, as can be seen at the left edge of each of the plots in Figure 1. If the

[3].. and remember that Figure 1 shows only the search cost, not the cost of estimating the probability of failure for each variable. Because FF4 is so expensive for the $\langle 20, 10, 1.0 \rangle$ problems, we have not completed the experiments.

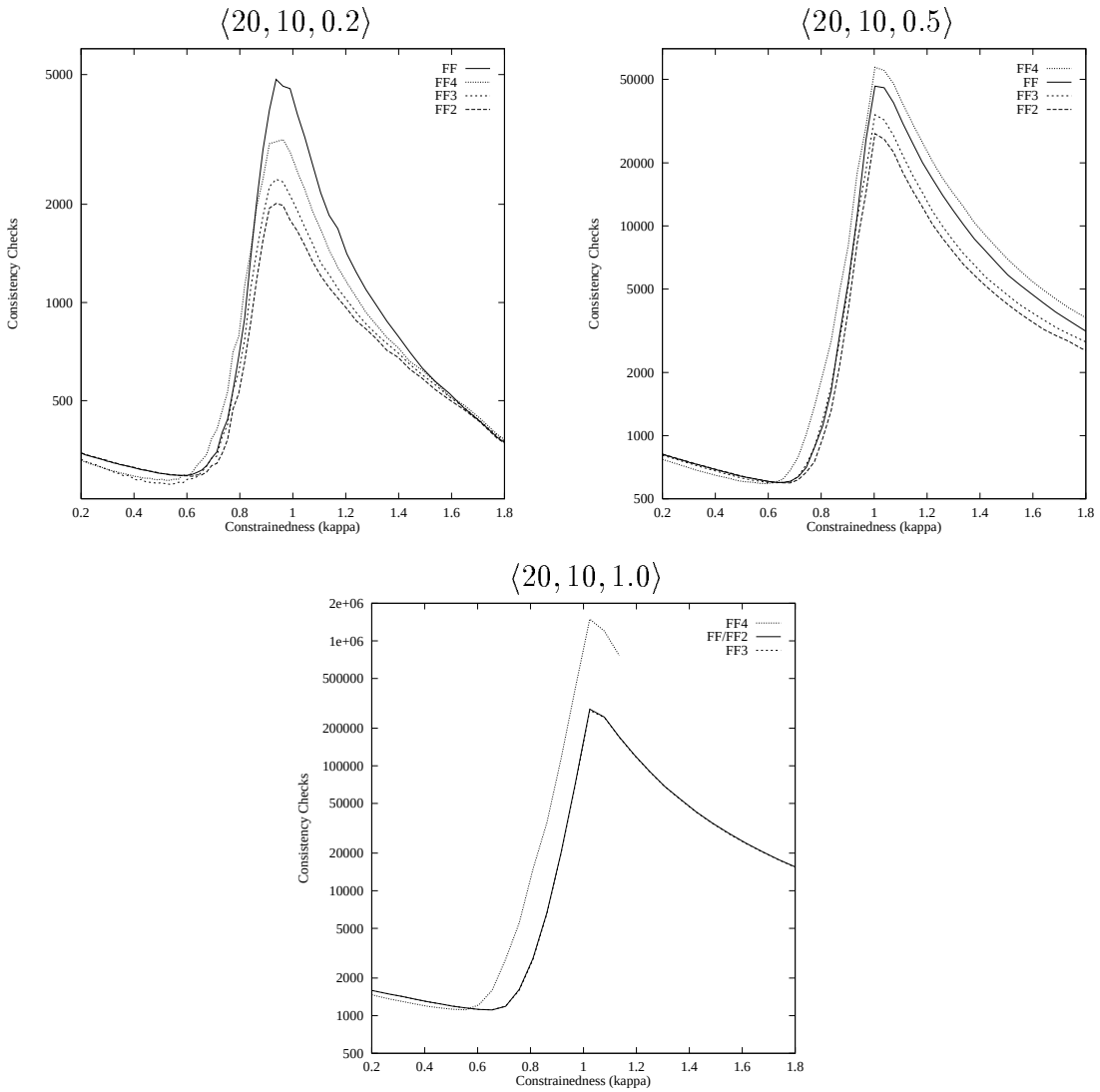

Figure 1: The new fail-first heuristics on $\langle 20, 10 \rangle$ problems.

constraints are very loose, a solution can be found without ever backtracking to a previous variable; in this region, the cost of forward checking decreases as κ increases, since more pruning of the domains of future variables can be done. FF4 is the best of the four heuristics (by a narrow margin) so long as it can find a solution without backtracking, but it has to start backtracking at smaller values of κ, as shown by the fact that its cost is increasing while that of the others is still decreasing. Moreover, a slight reduction in the cost of solving the easiest problems is not very helpful.

A possible, though hopefully wrong, explanation for the poor performance of the new heuristics, which would still leave the fail-first principle intact, might be that they do not in fact increase the probability of failure. All the heuristics are for various reasons only approximations to equation (2): FF, FF2 and FF3 because they use limited information about the current subproblem and FF4 because it treats p_{ij} as the probability that a value for v_i is inconsistent with a value for v_j when in fact it is the proportion of inconsistent pairs of values, so that the independence assumption is not strictly valid. It is therefore conceivable that we are not increasing the probability of failure in our successive heuristics, and therefore not decreasing the average branch length.

To test this possibility, we looked at the depth in the search tree at which failures occur, for the four heuristics. (For this purpose, we used FFdeg rather than FF, because, as will be seen in the following section, simply making a better choice for the first variable improves the performance of FF considerably, and provides a better comparison with FF4). Figure 2 shows the mean number of failures at each depth in the search tree for the four heuristics at the crossover point for $\langle 20, 10, 0.5\rangle$ problems, i.e. when $p_2 = 0.37$. This is the point where the probability that a problem has a solution is 0.5; the median cost of solving the problems is highest; and the differences in cost between the four heuristics are greatest. A failure is counted whenever a variable domain is exhausted and the algorithm moves back to the previous variable; hence a failure does not necessarily occur at the end of a branch, but could occur when the algorithm has backtracked to this variable and there are no other values which have not already been tried.

Figure 2 suggests that the new heuristics do result in successively smaller average branch depth; FF4, for instance, fails most often at depth 5 or 6, and has fewer failures at depth 8 or below than the other heuristics, whereas FFdeg has its maximum number of failures at depth 7, and has many more failures than FF4 further down the tree. However, the fact that FF4 fails earlier is outweighed by the fact that it fails much more often, so that its overall cost is higher.

Figure 3 gives a different view of the work done by FFdeg and FF4 at different depths. It shows the mean consistency checks for the $\langle 20, 10, 0.2\rangle$ problem class for a range of values of the constraint tightness, p_2, across

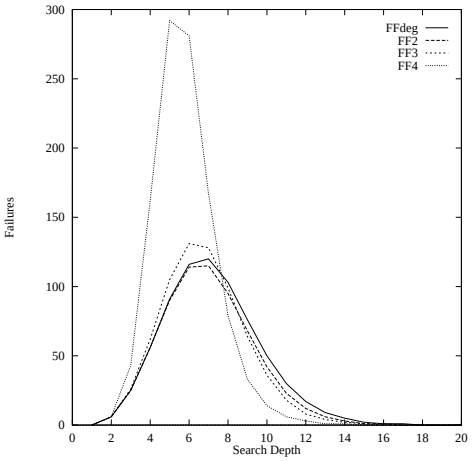

Figure 2: Mean failures at each depth for 1000 $\langle 20, 10, 0.5, 0.37 \rangle$ problems

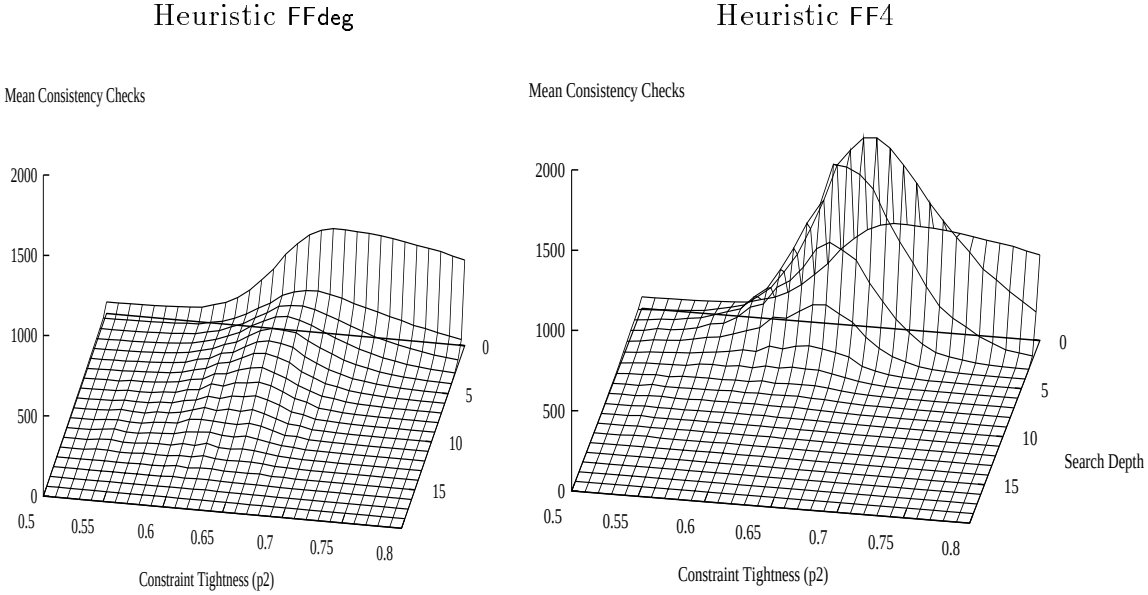

Figure 3: Search profiles for FFdeg and FF4 on $\langle 20, 10, 0.2 \rangle$ problems

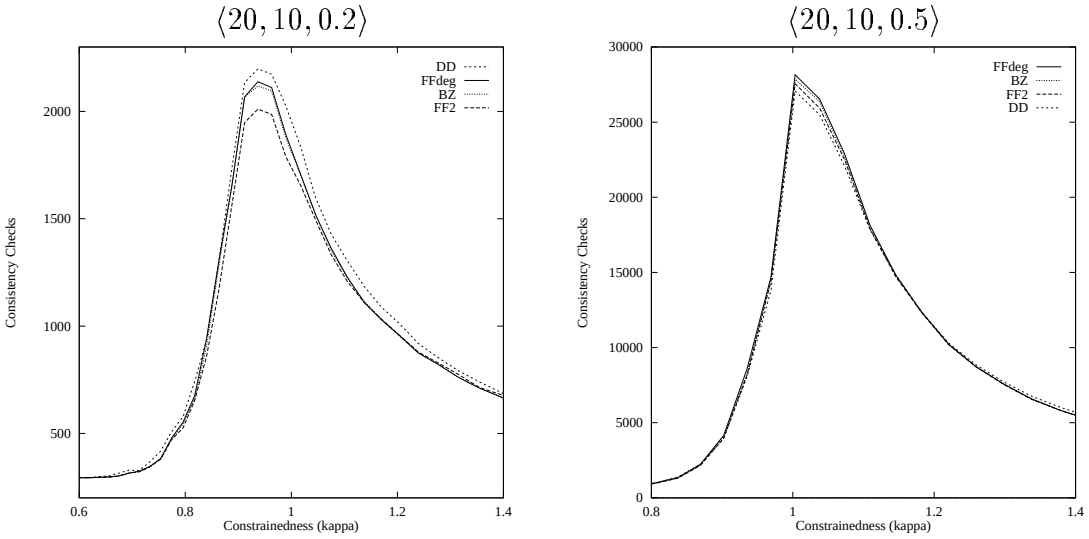

Figure 4: Four DVO heuristics on $\langle 20, 10 \rangle$ problems.

the phase transition at $p_2 = 0.66$. The two heuristics do the same amount of work at the top level of the tree, since they choose the same starting variable. FF4 does almost all its work in the top levels of the tree, and virtually no work at depths 10 or below, whereas around the phase transition, FFdeg does significant work at lower depths. Again, however, FF4 is doing much more work than FFdeg at the top levels of the tree, and this outweighs the benefit of rarely creating long branches.

7 Heuristics Using Future Degree

The only one of our proposed heuristics which merits further consideration is FF2, since, as well as being the cheapest of them to implement, it is the best of the heuristics shown in Figure 1, except when $p_1 = 1$. In this section we compare this heuristic with FFdeg, BZ and DD, the other heuristics in our study which use both the domain size and future degree in selecting the next variable.

Figure 4 compares FF2, FFdeg, BZ and DD; we plot the median cost of solving $\langle 20, 10, 0.2 \rangle$ and $\langle 20, 10, 0.5 \rangle$ problems using the forward checking algorithm with these heuristics. The $\langle 20, 10, 1.0 \rangle$ class has been omitted since these heuristics are identical to FF when the constraint graph is complete. Note that a linear scale is used on the vertical axis, rather than a log scale as in Figure 1, to show the differences in cost more clearly.

The four heuristics give very similar performance. FF2 is marginally the best when $p_2 = 0.2$, but DD is better when $p_2 = 0.5$. It is noteworthy that the simplest of the heuristics, FFdeg, which uses degree information only in

selecting the first variable, is competitive with the others. This suggests that that the poor performance of the plain FF heuristic on sparsely constrained problems can be explained by the fact that it tends to make a wrong initial choice. Simply making a better choice initially and thereafter choosing any variable with smallest remaining domain gives, on these classes of problem, performance comparable with the more sophisticated algorithms which take account of the future degree of each variable throughout.

Figures 1 and 4 show that, at least for this class of random binary problems, taking account of the future degree of variables gives better performance than just choosing the variable with smallest domain, when the constraints are sparse. However, exactly how the future degree is incorporated into the heuristic seems to make very little difference to the search cost; the simplest heuristic, which only takes account of future degree in selecting the first variable (and which is also the cheapest to implement) is nearly as good as the more complex heuristics.

8 The Failure of Fail-First

We have developed new variable ordering heuristics by taking the fail-first principle seriously and calculating the probability that a variable will fail, for a class of randomly-generated CSPs. The fail-first principle is based on the assumption that choosing the variable which is most likely to fail will result in reduced average branch length. We found that our improved heuristics do seem to result in lower average branch length. The more information the heuristic uses to estimate the failure probability, the less often the lower depths of the tree are reached and the greater the proportion of failures that occur high in the tree. However, although our new heuristics appear to fulfil the aim of the fail-first principle by reducing average branch length, this is not reflected in lower search costs. If we ignore the plain FF heuristic, whose costs are biased by poor initial variable choices when the constraint graph is sparse, and the very easy problems, then the more accurately we estimate the failure probability, the greater the search cost.

Reducing the average branch length may not reduce the number of nodes in the search tree, if the average branching factor increases at the same time. This seems to be what is happening with our new heuristics: although failures occur higher in the tree, there are many more of them. The new heuristics must sometimes choose a variable which does not have the smallest domain, because it is calculated to have a higher probability of failure than those which do. If the chosen variable does not fail, more branches are likely to be created, all of which might need to be explored. Although these branches can be expected to terminate more quickly than those resulting from choosing one of the variables with smallest domain, this may not outweigh the fact

that there are more of them.

Hence we claim that our experiments have refuted the fail-first principle: maximizing the failure probability in order to reduce the average branch length does not result in lower search costs. As a corollary, the success of the heuristics based on domain size and degree must be explained in some other way.

We have, of course, only tested our new heuristics on one class of problem, for which we could calculate the failure probability. This would have been a serious disadvantage if our heuristics had done well, since they would not necessarily have done equally well on other classes of problem. However, the fact that they have done badly on this class of problem proves that the fail-first principle is not generally applicable, although it does not rule out the possibility that there might be types of problem for which it pays off to calculate the failure probability more accurately.

Hooker and Vinay [9] report a similar investigation into branching rules used with the Davis-Putnam algorithm for solving satisfiability problems. Branching rules are analogous to variable ordering heuristics for CSP algorithms. Hooker and Vinay took the accepted explanation for one branching rule and using this explanation, derived a new branching rule. Assuming the explanation was correct, the new rule should have been superior to the original rule in terms of the number of nodes generated. They carried out an experiment to test this hypothesis. The new rule proved to be worse than the original, thus refuting the explanation as the true reason for the success of the original rule. They then went a stage further than we have done, by proposing an alternative explanation for the rule, making new predictions on the basis of this explanation and showing that these predictions were borne out by experiment, thus providing evidence in favour of the new explanation, as well as giving a new branching rule which was superior to the original.

9 Explaining Good Heuristics

As we showed in section 7, the heuristics which use both domain size and future degree when selecting the next variable are the best of those we have studied. However, the fail-first principle is no longer an adequate explanation of their performance. We need to understand why these heuristics perform well, in order to be able to improve them, and to devise different heuristics for the problems where smallest-remaining-domain does not give good results. For instance, when the constraint graph is sparse, it is usually a good strategy to instantiate variables connected to those already instantiated. With the $\langle n, m, p_1, p_2 \rangle$ model, the variable with smallest remaining domain tends to be a variable constrained by the largest number of past variables (because such a variable has had its domain reduced most often). However, if the

initial domain sizes are not uniform, then the second variable chosen might be selected because its initial domain size is small and not because its domain has been reduced by the instantiation of the first variable; hence the second variable might not be constrained by the first, and would not be a good choice[4].

A different explanation for the success of heuristics such as BZ is provided by the most-constrained-variable and most-constraining-variable heuristics, described by Russell and Norvig [12], for instance: the most constrained variable is the one with smallest remaining domain and the most constraining variable is the one with maximum future degree. BZ chooses the most constrained variable and breaks ties by choosing the most constraining variable. Russell and Norvig's justification for these heuristics is that the first tends to minimize the branching factor at the current node and the second attempts to reduce the branching factor at future nodes.

By minimizing the branching factor we might hope to minimize the number of nodes visited in the search tree. Nudel [10] proposes trying to minimize the number of nodes in the search tree directly. In choosing the next variable to instantiate, we should try to choose the one which will minimize the number of nodes visited in the search tree below this variable. He gives expressions for N_k, the expected number of nodes visited at level k in the tree when finding all solutions to a CSP using the forward checking algorithm. He suggests that the next variable selected should be one which minimizes N_1, the number of nodes at the top level of the new subtree: N_2 can then be used as a tie-breaker, and so on. Since N_1 is equal to the size of the first variable's domain, at each level in the tree we should next choose the variable with smallest remaining domain: this is exactly the most-constrained-variable heuristic. Although the expression for N_2 is more complex, for $\langle n, m, p_1, p_2 \rangle$ problems, before any assignments have been made, it reduces to a simpler form which implies that we should choose first the variable with largest degree. This heuristic would therefore choose the same starting variable as FFdeg.

Unfortunately, using this idea to generate better heuristics than FFdeg is not straightforward. Minimizing N_1 and then $N_1 + N_2$ is only an approximation to minimizing the expected total number of nodes visited, but calculating $N_1 + N_2 + .. + N_n$ can only be done for a specific ordering of all the variables. In theory we should calculate this quantity for every possible ordering, assign the first variable in the best ordering, repeat the process for the remaining $n - 1$ variables and so on, but clearly this is not practicable.

The fail-first principle and the most-constrained/most-constraining variable heuristics can be seen as complementary attempts to minimize the size

[4]See also the article by Sunil Mohan on variable ordering heuristics for non-binary CSPs at http://www.cirl.uoregon.edu/constraints/links/heuristics.html

of the search tree. The first aims to reduce the average branch length and the second to reduce the average branching factor. We have suggested that, for $\langle n, m, p_1, p_2 \rangle$ problems, heuristics which try to maximize the failure probability may do so at the expense of increasing the branching factor at the current node, so that although the average branch length is reduced, the overall size of the search tree is not. It might seem that a good compromise would be to always choose a variable with minimum domain size, and to break ties by choosing the variable with maximum failure probability. However, although this might give good results for $\langle n, m, p_1, p_2 \rangle$ problems, it requires subverting the fail-first principle and would give no guidance in cases where smallest-remaining-domain is not a good heuristic.

An alternative principle for variable ordering heuristics is proposed by Gent *et al.* [5], namely that the next variable should be chosen so as to minimize the *constrainedness* of the future subproblem. The hope is that this will guide the search towards under-constrained subproblems, since under-constrained problems tend to have many solutions and be easy to solve. Four new heuristics using different measures of constrainedness were investigated, including one based on κ, the general measure of the constrainedness of combinatorial problems referred to earlier [6]. These heuristics can be seen as choosing the most constrained variable, but take into account other factors than just the domain size of each variable. The smallest-remaining-domain heuristic can be viewed as an approximation to minimizing κ, and when all variables have the same domain size and all constraints have the same tightness, κ is minimized by choosing the variable with largest degree, so that again this heuristic would choose the same starting variable as FFdeg for $\langle n, m, p_1, p_2 \rangle$ problems. The κ heuristic is better than BZ for $\langle 20, 10, 1.0 \rangle$ problems (it reduces the average number of consistency checks by more than 10% at the phase transition peak). For $\langle 20, 10, 0.2 \rangle$ problems, the four proposed heuristics are considerably more costly than BZ, even ignoring their implementation cost. However, since most of the other available heuristics degenerate to FF when the constraint graph is complete (or, in the case of FF4, are much worse) a heuristic which reduces search in this case is noteworthy.

10 Conclusions

The rationale for the fail-first principle is that by choosing next the variable with the greatest probability of failing, the average branch depth in the search tree, and hence the size of the search tree and the cost of finding a solution, will be minimized.

To put this to the test, we calculated the probability that a variable will fail when using the forward checking algorithm to solve binary CSPs using the $\langle n, m, p_1, p_2 \rangle$ model. From the failure probability, we derived a series of

heuristics, FF, FF2, FF3 and FF4, which use progressively more information about the subproblem formed by the currently uninstantiated variables. We showed that our new heuristics do progressively reduce the average branch length, and that FF2, which takes into account the future degree of each variable, is an improvement on FF. However, FF3 and especially FF4 are more expensive than FF2, even if their implementation costs are ignored.

The fail-first principle is therefore shown to be unsound: reducing the average branch length does not, in itself, necessarily reduce the size of the search tree and so should not be the sole aim of variable ordering heuristics.

Although FF2 is much better than FF, our study of other heuristics which similarly take account of the future degree of each variable in some way showed that the crucial feature of these heuristics (at least for this class of problems, where all domain sizes are initially equal) is that the *first* variable they choose is the one with maximum degree. Using the future degree of each variable in making subsequent choices is much less important.

We endorse Hooker and Vinay's approach to investigating heuristics; in order to develop better heuristics we should try to explain the performance of existing heuristics, and test these explanations by deriving new heuristics from them. If the new heuristics do badly, where the explanation predicts that they should do well, we know that the explanation is wrong and that we need to seek a different explanation; if they do well, we have some evidence that the explanation is correct (as well as possibly better heuristics than before). Our results have been negative in that instead of developing better heuristics, we have undermined the fail-first principle. In the light of other explanations for the smallest-remaining-domain heuristic, our experience suggests that to minimize the size of the search tree we should perhaps try to reduce the branching factor as well as, or instead of, the average branch length.

Given a principle for variable ordering which would allow us to make predictions about the performance of proposed heuristics and which, unlike the fail-first principle, would prove to be robust, we might then be able to develop improved variable ordering heuristics. Such heuristics should beat the existing heuristics, such as FFdeg and BZ, on problems where they do well: they should also be more generally applicable, and give good results on problems other than uniform binary CSPs, where existing heuristics often do badly.

Acknowledgements

This work has been partly supported by a studentship awarded to Stuart Grant by British Telecom plc. The authors are members of the APES group[5]

[5]http://www.cs.strath.ac.uk/~apes

and thank the other members of the group, especially Ian Gent, for their constructive comments.

References

[1] C. Bessière and J.-C. Régin. MAC and Combined Heuristics: Two Reasons to Forsake FC (and CBJ?) on Hard Problems. In E. C. Freuder, editor, *Principles and Practice of Constraint Programming - CP96*, volume 1118 of *Lecture Notes in Computer Science*, pages 61–75. Springer, Aug. 1996.

[2] D. Brélaz. New Methods to Color the Vertices of a Graph. *Communications of the ACM*, pages 251–256, 1979.

[3] P. Cheeseman, B. Kanefsky, and W. Taylor. Where the *Really* Hard Problems are. In *Proceedings IJCAI-91*, volume 1, pages 331–337, 1991.

[4] D. Frost and R. Dechter. In search of the best constraint satisfaction search. In *Proceedings AAAI-94*, pages 301–306, 1994.

[5] I. Gent, E. MacIntyre, P. Prosser, B. Smith, and T. Walsh. An empirical study of dynamic variable ordering heuristics for the constraint satisfaction problem. In E. C. Freuder, editor, *Principles and Practice of Constraint Programming - CP96*, volume 1118 of *Lecture Notes in Computer Science*, pages 179–193. Springer-Verlag, Aug. 1996.

[6] I. P. Gent, E. MacIntyre, P. Prosser, and T. Walsh. The Constrainedness of Search. In *Proceedings AAAI-96*, pages 246–252, 1996.

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

[8] J. N. Hooker. Testing Heuristics: We Have It All Wrong. *Journal of Heuristics*, 1:33–42, 1996.

[9] J. N. Hooker and V. Vinay. Branching Rules for Satisfiability. *Journal of Automated Reasoning*, 15:359–383, 1995.

[10] B. Nudel. Consistent-labelling problems and their algorithms: Expectedcomplexities and theory-based heuristics. *Artificial Intelligence*, 21:135–178, 1983.

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

[12] S. J. Russell and P. Norvig. *Artificial Intelligence : A Modern Approach.* Prentice Hall, Upper Saddle River, New Jersey, 1995.

[13] B. M. Smith and M. E. Dyer. Locating the Phase Transition in Constraint Satisfaction Problems. *Artificial Intelligence*, 81:155–181, 1996.