Trying Again to Fail-First

J. Christopher Beck[1], Patrick Prosser[2] and Richard J. Wallace[1]

[1]Cork Constraint Computation Center and Department of Computer Science
University College Cork, Ireland
[2]Department of Computer Science, University of Glasgow, Scotland
{c.beck,r.wallace}@4c.ucc.ie, pat@dcs.gla.ac.uk

Abstract. In ECAI 1998 Smith & Grant performed a study [1] of the fail-first principle of Haralick & Elliott [2]. The fail-first principle states that "To succeed, try first where you are most likely to fail." For constraint satisfaction problems (CSPs), Haralick & Elliott realized this principle by minimizing branch depth. Smith & Grant hypothesized that if failing first is a good thing, then more of it should be better. They devised a range of variable ordering heuristics, each in turn trying harder to fail first. They showed that trying harder to fail-first does not guarantee a reduction in search effort. We failed in our attempt to repeat that study. This was due to a implementation error for one of their heuristics. However, with this fix applied we show that Smith & Grant's conclusions were indeed correct: failing earlier does not invariably lead to greater overall efficiency. This is because minimizing branch depth sometimes involves large increases in the branching factor. We also show that the effects of trying harder to fail-first are algorithm dependent. Our results indicate that trying to fail early is an important principle in search, and they suggest an alternative failure strategy that seeks to minimize the number of nodes in a subtree.

1 Introduction

Search is at the heart of many AI approaches to problem solving. Despite this importance, there is no understanding at a foundational level of the behavior of heuristic decision making. The answer to the basic question "Why do some heuristics perform better than others?" remains elusive. One long-standing intuition for heuristic performance is the fail-first principle due to Haralick & Elliott [2] which states: "To succeed, try first where you are most likely to fail." Though initially counter-intuitive, the fail-first principle is widely seen as a useful insight into heuristic decision making. Given a set of inter-related decisions, the fail-first principle suggests that the one that is most difficult should be made first. This is akin to cautious intelligent behavior, focusing effort on critical choices before allowing ourselves the luxury of solving the easy parts of the problem.

In adapting this principle to constraint satisfaction search, Haralick & Elliott made a further critical inference: that minimizing branch length during search should also minimize search effort. Because of this, their subsequent formal analysis of 'fail-firstness' did not pertain directly to discovering and solving difficult subproblems, but simply to finding the quickest way to fail during search. We will refer to this as the "radical fail-first principle" to distinguish it from the original idea. In this analysis, they showed that

the "smallest domain first" (SDF) variable ordering heuristic (choose next to assign a value to the variable with the smallest number of possible values) is a level-one estimate of minimizing branch length.

To test this principle, Smith & Grant [1] created a set of new heuristics designed to aggressively fail early in the search. The hypothesis was that if failing quickly was the explanation for search efficiency, then heuristics that failed earlier should demonstrate lower search effort. Smith & Grant showed that, contrary to expectations, increasing the ability to fail early in the search did not always lead to increased search efficiency. They concluded that the (radical) fail-first principle cannot be the only thing that explains the differences in search cost among heuristics.

Our interest in this problem was sparked by the recent development of a new framework for analyzing heuristic performance [3]. This framework incorporates *fail-firstness* as one of its performance principles. Another, called *promise*, concerns the selection of alternatives most likely to succeed. (This principle should be distinguished from the heuristics of the same name [4]; obviously, however, promise *heuristics* are designed to conform to the promise principle if and when the latter applies.) In this framework, if search deviates from a correct path, then and only then, does fail-firstness come into play.

In this paper, we revisit Smith & Grant's experiments and show that their study was incomplete for a number of reasons. First, we have discovered an error in their empirical results, brought about by a programming error in one of their heuristics. Second, we have to consider the possible role of promise in their results. Third, the measurement used to judge the relative ability of heuristics to fail early in the search was insufficient as it did not isolate the mean branch depth. Fourth, due to the increase in computing power over the last five years, we are now able to look at larger problems and use different algorithms.

Our contributions are to extend the experiments done by Smith & Grant and show that despite the above issues their basic conclusion still holds: for variable ordering there does not appear to be a simple mapping between minimization of branch depth and search efficiency. We show that this is due to the fact that minimizing branch depth can be associated with an increase in the average branching factor, and the latter effect can be large enough to impair search overall. This occurs with forward checking [2], but if we use maintained arc-consistency (MAC) [5] then this tradeoff is mitigated, so that minimizing branch length correlates with reduced search effort. We suggest that the radical fail-first principle can be replaced with an alternative failure principle that takes both the branching and the branch depth factors into account.

2 Trying Harder to Fail First (1998)

The basic hypothesis of Smith & Grant was that if failing first is such a good thing then more of it must be better. Therefore, creating heuristics with a stronger ability to fail early should increase search efficiency. Efficiency was defined as the number of constraint checks required to find a solution to a problem or to prove that no solution exists. Because the focus of the study was on the relation between fail-firstness and search efficiency, the computational effort to make those heuristic decisions was (cor-

rectly) factored out of the experiments. The goal was to understand the relationship between the radical fail-first principle and search efficiency so the computational effort to increase the fail-firstness of a heuristic was irrelevant.

Experiments were performed over randomly generated binary constraint satisfaction problems. Each set of problems was defined by a 4-tuple $\langle n, m, p_1, p_2 \rangle$, where n is the number of variables, m is the uniform domain size, p_1 is the proportion of edges in the constraint graph, and p_2 is the uniform constraint tightness. p_2 can be considered as the probability that when a pair of constrained variables are instantiated they will be in conflict. All their experiments were over problems with $n = 20$ and $m = 10$.

Using the forward checking algorithm [2] and standard chronological backtracking Smith & Grant tested four heuristics engineered for increasing levels of fail-firstness.

- **FF**: FF is the same as SDF (Smallest Domain First): choose the variable with the smallest remaining domain.
- **FF2**: The variable, v_i, chosen is the one that maximizes $(1 - (1 - p_2^m)^{d_i})^{m_i}$, where m_i is the current domain size of v_i, and d_i is the future degree of v_i. The FF2 heuristic takes into account an estimate (based on the initial parameters of problem generation) of the extent to which each value of v_i is likely to be consistent with the future variables of v_i. The FF2 heuristic has the flavor of the Brelaz heuristic [6], in that it involves a tradeoff between domain size and forward degree that favors differences in the former over the latter.
- **FF3**: FF3 builds on FF2 by using the current domain size of future variables rather than m. The variable, v_i, chosen is the one that maximizes the expression (1) below, where C is the set of all constraints in the problem, F is the set of unassigned variables, and $P = p_2$.
- **FF4**: Finally, FF4 modifies FF3 by substituting the current tightness, $P = p_{ij}$, of the future constraints (the fraction of tuples from the cross-product of the current domains that fail to satisfy the constraint) instead of p_2.

$$(1 - \prod_{(v_i, v_j) \in C, v_j \in F} (1 - P^{m_j}))^{m_i} \tag{1}$$

The progression from FF through to FF4 uses more and more information in determining which variable is most likely to fail at any given stage of search. Smith & Grant tested this by measuring the distribution of the depth of backtracks for 1000 problem instances at $\langle 20, 10, 0.5, 0.37 \rangle$. The instances were a mix of soluble and insoluble problems. Their measurements showed that the heuristics performed as expected: FF4 has a distribution skewed to backtracks at a shallower depth in search while the distribution gradually moved to deeper backtracks for FF3, FF2, and FF. Qualitatively, the distributions were similar to those shown in Figure 3 below.

With respect to total search effort, Smith & Grant expected that the heuristics would be ranked as follows: FF $>$ FF2 $>$ FF3 $>$ FF4, where $>$ means "results in greater search effort than". Their results were not as expected. Through a number of experiments, the details of which are reported in the following section, Smith & Grant showed that except on easy problems, FF2 incurred the least search effort, followed by FF3, FF and finally FF4. That is, they observed the following order: FF4 $>$ FF $>$ FF3 $>$ FF2. This

ordering is clearly at odds with the hypothesis of a simple mapping between the ability to fail-first and search effort. Smith & Grant concluded that there must be some other factor at work, perhaps in concert with the fail-first principle, in determining the search efficiency for variable ordering heuristics.

3 A Bug in FF4

In order to continue on from Smith & Grant's work and investigate additional factors that may impact the search efficiency of variable ordering heuristics, our first step was to reproduce their experiments. We were able to reproduce the results for FF, FF2, and FF3, but we were unable to reproduce the behavior of FF4. Our version of FF4 was orders of magnitude *better* than the FF4 reported by Smith & Grant. Our results were tested on a number of independent implementations of the algorithms and problem generators: a solver written in C extending [7], one in Java, one based on ILOG Solver, and one written in LISP. All of our attempts failed to reproduce the reported results.

Barbara Smith and Stuart Grant kindly provided us with the source code of their solver and we found a programming error in their code. When computing the value for p_{ij} in equation (1) they used integer division instead of floating point division. This resulted in p_{ij} being assigned a value of 1 for a variable that is inconsistent with an adjacent variable and 0 otherwise. Therefore the search algorithm will either select a future variable that is arc-inconsistent with some other future variable, generating a dead-end, or selecting any other variable with equal likelihood.

When this error was corrected in their solver we were able to reproduce our results using their solver. In the new results we typically saw the following ordering of the heuristics: FF $>$ FF3 $>$ FF4 $\geq$ FF2.

4 Measuring the Ability to Fail-First

Clearly, to test the fail-first principle it is necessary to demonstrate the extent to which each heuristic does indeed fail first. Smith & Grant measured the mean *backtrack depth* to find a solution to a problem or to prove that no solution exists. We believe there are two weaknesses in this methodology. First, a backtrack was counted whenever a domain was emptied and search returned to the previous variable. If that variable has no more values to try, its domain has been emptied and another backtrack is counted in moving back once again. That is, Smith & Grant measured the average depth of failed leaf nodes *and* failed interior nodes of the search tree explored. The original formulation of the (radical) fail-first principle assumed that minimizing mean *branch depth* would minimize search effort. Therefore, it should be the mean branch depth that is used as a measure of the ability of a heuristic to fail-first. The backtrack depth measure does not do this. For example, imagine a node N that is at the second level of a subtree containing no solutions; that is, node N can be consistently extended by at least one more variable, but not to an entire solution. In proving that there is no solution below the parent of N, many backtracks will be counted in moving from N (and its siblings) back to its parent. Counting these backtracks does not assess branch depth since there is no branch which terminates at N. A more appropriate measure of the mean branch depth is as follows:

whenever a variable is assigned a value and that assignment immediately leads to a domain wipe-out, we count a failure. That is, we measure the average depth of failed leaf nodes in the search tree.

The second weakness with the measurement of a heuristic's ability to fail-first is that by searching only for the first solution to soluble problems the measure of the ability to escape a mistake is contaminated by the heuristic's ability to find a solution (its promise, in the newer formulation). It is at least theoretically possible that a heuristic that is very poor at failing first could be very good at finding a solution when one exists. Therefore, searching for the first solution combines the abilities of a heuristic to escape dead-ends and its ability to find solutions. We are interested in isolating the former ability. For our experiments, we assess the ability of a heuristic to fail-first by measuring the branch depth in searching for *all* solutions or showing that no solution exists.

5 Trying Again to Fail First

We report the results of repeating the experiments of Smith & Grant. Our results were produced by using two solvers coded independently. The first was written in C and extended the CSP solver used in [7] while the second was written in Java and extended the solver used in [8]. As noted, we also confirmed these results using the C++ solver used by Smith & Grant with the integer division error in FF4 corrected. In our implementation of the heuristics, ties (i.e. when more than one variable was judged heuristically best) are broken lexicographically. The problems we investigated were generated at the beginning of our study and stored. These problems were then used by all our solvers, allowing us to reproduce results across two sites.

Problems were generated using a "probability-of-inclusion" model, in which each possible constraint element (domain value, constraint or constraint tuple) is included with a specified probability. The generator allows parameters to be fixed at the expected values given these probability values; in this case a set of elements is generated repeatedly until the cardinality matches the expected value. This allows it to generate problems in accordance with model B [9]. By specifiying a probability of 1 for domain element inclusion, all domains have a size specified by a maximum domain-size parameter. Sets of soluble or insoluble problems were sometimes used; these problems were generated in an identical fashion and filtered on the basis of solution testing.

Figure 1 presents the median number of constraint checks to find a solution or to prove that no solution exists for problems from the set $\langle 20, 10, 0.2 \rangle$. The constraint tightness was varied from 0.01 to 0.99 in steps of 0.01. For each combination of parameters 1000 problem instances are generated. Median checks (the same measure used by Smith & Grant) are plotted against κ (kappa), the measure of constrainedness proposed by [10]. Figure 2 presents the same set of experiments, but for the $\langle 20, 10, 0.5 \rangle$ problem set.

In both graphs, FF clearly incurs the highest number of constraint checks, followed by FF3. In Figure 1, FF2 performs better than FF4 however in Figure 2 there is no discernible difference between FF2 and FF4. Ignoring the FF4 results, these graphs agree with the original experiments of Smith & Grant.

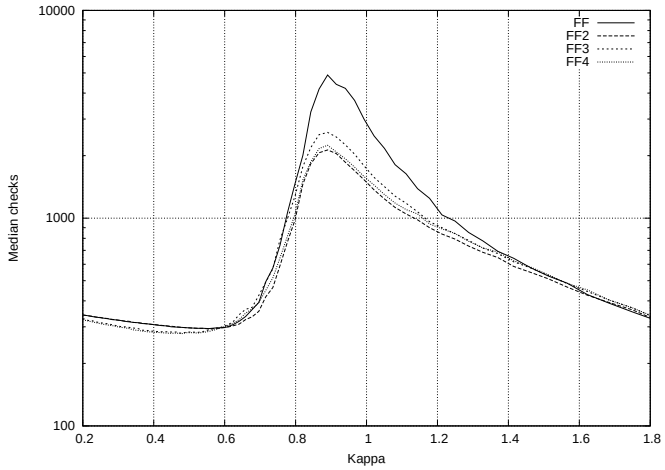

Fig. 1. The median number of checks for the $\langle 20, 10, 0.2 \rangle$ problem set. The heuristics are ranked as follows : FF $>$ FF3 $>$ FF4 $\geq$ FF2. That is, FF2 and FF4 compete as the best heuristics, and FF3 is better than FF.

In Figure 3 we plot the fraction of failed leaves at each depth when searching for all solutions. Qualitatively, these results match what Smith & Grant found with their measure of the ability to fail-first: FF4 does indeed fail higher in the tree (as judged by mean branch depth), followed by FF3, FF2 and FF. Therefore, we can confirm that Smith & Grant did indeed propose new heuristics that progressively increase in their ability to fail-first.

The crux of these experiments is the fact that the ability to fail earlier in the tree does not necessarily translate into better search effort: FF3 incurs a higher search cost than both FF2 and FF4, yet Figure 3 shows that FF3 is between FF2 and FF4 in its ability to fail early in the tree. Stated more precisely: the radical fail-first principle (i.e., branch depth minimization) does not, by itself, account for variations in search efficiency among variable ordering heuristics. We therefore confirm the conclusions of Smith & Grant.

6 The Impact of Promise

Why is FF2 better than FF3? FF3 does better than FF2 at failing first, but FF2 searches more efficiently than FF3. There must be something else driving our heuristics. What might it be?

We have already noted that a second principle of performance has been identified, the promise principle. It has also been shown that variable heuristics can be distinguished by their promise (a less obvious relation than the one between promise and value ordering heuristics) [3]. In other words, if the search is on a path to a solution, a heuristic with high promise will select a variable for which there is a high likelihood that a good value selection can be made. That is, the variable assignment is more

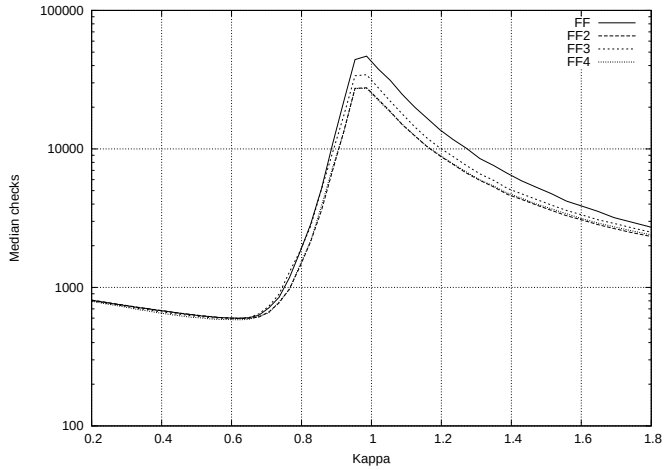

Fig. 2. The median number of checks for the $\langle 20, 10, 0.5 \rangle$ problem set. The heuristics are ranked as follows: FF $>$ FF3 $>$ FF4 $\approx$ FF2.

likely to lead to a new search state that still has solutions in its subtree. Beck et al. [3] demonstrated that for problems with many solutions there is a high inverse correlation between the promise of a variable ordering heuristic and the number of constraint checks required to find a solution.

Given the definition of promise, there is a simple experiment to test if it could possibly account for the discrepancies in the fail-first principle: although FF3 fails more quickly than FF2 might FF2 out-perform FF3 because it is more promising? Promise is only well-defined on problem instances with solutions. If the results above are due to a combination of promise *and* fail-firstness then the relative ranking of the heuristics might differ for soluble and insoluble problems. We therefore refine Smith & Grant's hypothesis and instead hypothesize that on *insoluble* problems as we try harder to fail-first we will reduce search effort. Furthermore, we expect that the ranking of the heuristics might change when we look at soluble problems, as promise may come in to play.

Figure 4 demonstrates that this hypothesis is false. The relative ordering of the heuristics on insoluble problems for the $\langle 20, 10.0.5 \rangle$ problem set is identical to the ordering on mixed problems. Though not shown, this ordering is the same for the soluble problems as well. Furthermore, the branch depth on insoluble (as well as soluble) problems follows the pattern of Figure 3: FF4 fails highest followed by FF3, FF2, and FF. Therefore promise cannot be the "missing" factor explaining the discrepancy identified by Smith & Grant.

7 Extending the Experiments

To this point the experiments above, and those in Smith & Grant, have concentrated on problems of a single size (20 variables, 10 values) and on a single consistency enforcement algorithm (forward checking). It may be that this limited scope is obscuring

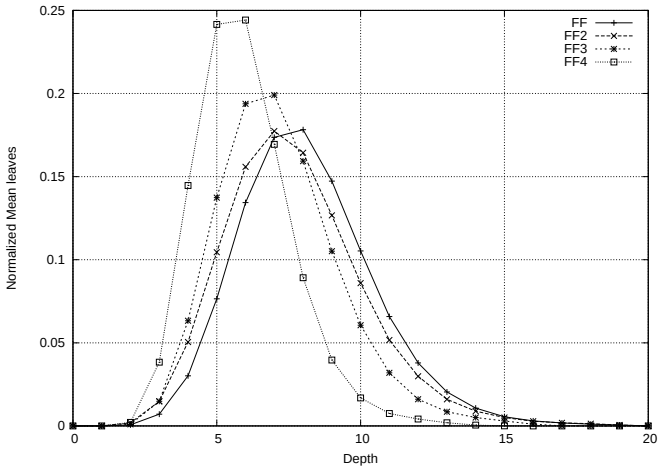

Fig. 3. The fraction of failed leaves at each depth in the tree when searching for all solutions for 1000 problems in the $\langle 20, 10, 0.5, 0.37 \rangle$ set, i.e. the problems at the $\langle 20, 10, 0.5 \rangle$ phase transition.

important information. For example, perhaps the fact that FF2 performs better than expected, based on the fail-first principle, is an artifact of the problem size and we will not observe such behavior on larger problems. In this section we report on our experiments with larger problems (up to 70 variables) and a different algorithm (maintaining arc consistency (MAC) [5]).

7.1 Varying problem size

Do we get the same ranking of the heuristics FF $>$ FF3 $>$ FF4 $\geq$ FF2 when we look at larger problems? We generated random problems of varying size, from $n = 20$ up to $n = 70$. These problems were generated such that in each set of problems variables had the same average degree of 10, i.e. each variable was on average constrained by 10 other variables. Variables had a uniform domain size of 10. Each set of problems was generated such that constrainedness $\kappa \approx 0.9$. Figure 5 gives contours for the four heuristics, with median consistency checks plotted against problem size. We see that the ranking of the heuristics is not influenced by problem size. Problems with more than 60 variables were too hard for the FF heuristic. Figure 5 shows contours for the soluble problems only. Similar experiments were performed with $\kappa > 1.0$ such that all problems were insoluble. Again, we observed the same ranking of the heuristics. Therefore, as far as the relative ranking of our heuristics is concerned, size does not matter.

7.2 Moving to MAC

Might the consistency enforcement algorithm have an effect on the heuristics? To test this out we repeated the above experiments, varying problem size, but this time using

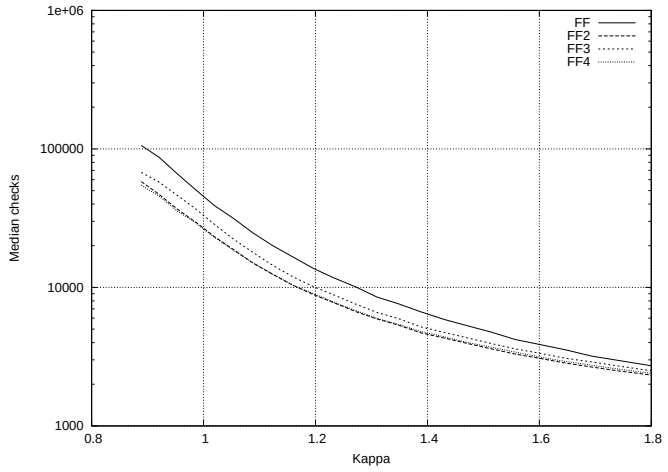

Fig. 4. The median number of constraint checks for the insoluble problem instances in the $\langle 20, 10, 0.5 \rangle$ problem set. The problem sets plotted are those for which at least one instance is insoluble, therefore each κ value may have a different number of instances up to 1000. In particular, the low values of κ are based on few instances.

the maintaining-arc consistency algorithm (MAC) [5]. As the name implies, whenever a variable is instantiated the future sub-problem is made arc-consistent. If this results in a domain wipe-out, a new value is tried, and failing that, backtracking takes place. Our results are presented in Figure 6. The ranking of the heuristics has changed! We now have the order FF $>$ FF2 $>$ FF3 $>$ FF4, suggesting that as we try harder to fail-first we do indeed see a reduction in search effort, but this is algorithm dependent. The results in Figure 6 are for soluble problems, and though not shown, we got the same ranking over insoluble problems. Again, these results were replicated across our solvers and across sites.

We also looked at the failure depths of the heuristics when allied with the MAC algorithm, for large problems ($n = 60$). This is shown in Figure 7. The results are in agreement with those in Figure 3 with FF4 failing earliest, then FF3, FF2, and FF.

8 The Impact of the Branching Factor

As Smith & Grant point out, Russell & Norvig [11] suggest that successful heuristics such as Brelaz and FF can be justified by the minimization of the branching factor in the tree. Perhaps, rather than the radical fail-first principle, an alternative can be suggested that minimizes search effort by minimizing the branching factor or a tradeoff between branch depth and branching factor.

To assess the branching factor of the FFx heuristics, we examined the 50 variable problems used in Figures 5-6 in more depth. In particular, we measured the the mean domain size of the variables selected by each heuristic (denoted "$|\ \mathrm{d}\ |$ chosen") and the total number of failures. Table 1 displays results that support the idea that a tradeoff be-

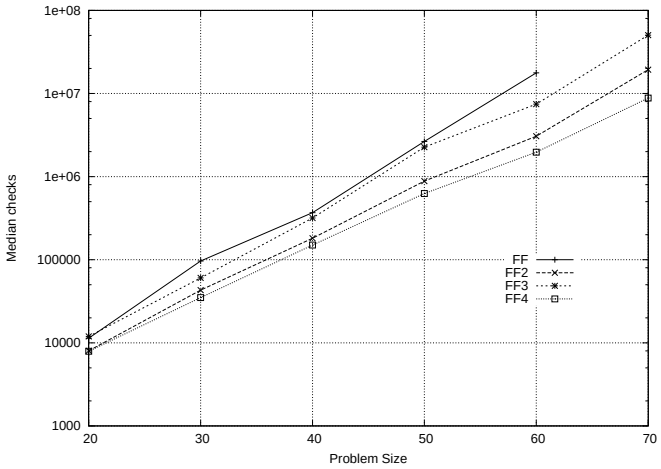

Fig. 5. The median consistency checks for problems with 20 to 70 variables. All problems have 10 values per variable, a density that results in 10 constraints on each variable, and a tightness set so as to give a $\kappa \approx 0.9$. All problems are soluble and at each problem size we have 50 instances.

tween branch depth and branching factor may be the key to the fail first principle. With forward checking, FF4 is better than FF3 on both criteria: smaller branching factor and smaller branch depth. Less search effort is therefore not surprising. In comparing FF2 and FF3, we see that while FF3 achieves a smaller branch depth, the branching factor is significantly larger. (In addition, the *range* of domain sizes also expands dramatically with FF3 and FF4, so that sizes up to 10 are chosen even at moderate depths of the search tree.) We also find that the number of failures increases dramatically with FF3 (cf. Table 1).

We hypothesize that FF3's increase in branching factor overwhelms the modest improvement in the branch depth leading to the better search performance of FF2. That it is a tradeoff between branch depth and branching factor can be seen by comparing the FF and FF4 results with forward checking. If branching factor were the only factor, FF should incur less search effort than FF4. As shown by all our experiments, this is not the case.

Equally importantly for the tradeoff hypothesis is the fact that, with MAC instead of forward checking, the FF2/FF3 conflict over the two factors disappears: there is a consistent decrease in branch depth, branching factor, number of failures, and, as shown in our experiments, search effort, when moving from FF to FF4 under MAC.

Because FF3 uses current values for m_j in equation (1), the values of the product term, $1-p_2^{m_j}$, in this expression are both smaller and more variable than the corresponding values used by FF2. In particular, if a variable with a relatively large domain has variables with small domains in its set of future neighbors, the product of these terms may be smaller than for a variable with a smaller domain that has variables with larger domains in its future set; in this case, the final value of the expression may be larger in the former situation. In addition, because FF3 avoids variables whose neighboring

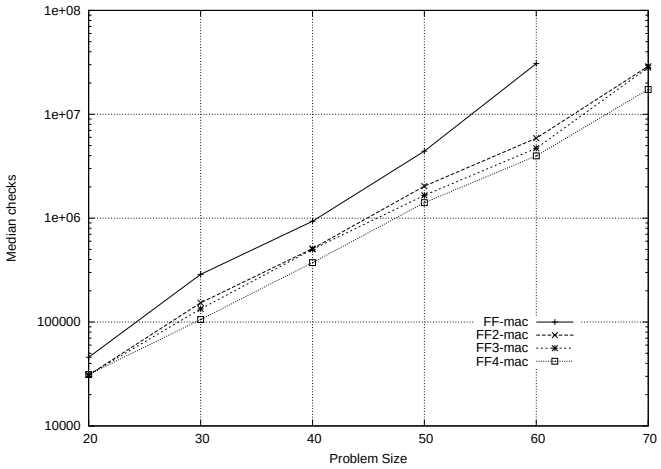

Fig. 6. The median consistency checks for problems with 20 to 70 variables using MAC. All problems have 10 values per variable, a density that results in 10 constraints on each variable, and a tightness set so as to give a $\kappa \approx 0.9$. All problems are soluble and each problem size has 50 instances.

future variables have large domains, the latter may be shielded from propagation. FF4 in combination with forward checking also tends to choose large domains for the same reasons, but because it uses the current tightnesses for P in equation (1), the problems incurred by FF3 are reduced. These problems are also avoided when MAC is used, presumably because the more egregious situations are avoided through more effective filtering. In particular, large domains will be reduced in greater measure if their current constraints are tight.

9 Conclusion

The primary conclusion of this work is that the results, in their entirety, contradict the (radical) fail-first principle proposed by Haralick & Elliott. This means that Smith & Grant's original conclusion is still valid. At the same time, preliminary results show why the radical principle sometimes fails: a reduction in branch length is sometimes accompanied by an increase in the number of branches per node.

Although the radical fail-first principle in this sense is inadequate as an explanation for search efficiency, one should not overlook the more general principle of trying to fail earlier in the search. One direction may be to specify more precisely the conditions where the radical principle is effective. Alternatively, one might redefine failing early to be a measure that combines the branching factor and the branch depth, such as the minimization of the number of nodes in the failed subtrees. In this form, the fail-first principle implies that search should attempt to derive minimal proofs of failure. Something like this was suggested by Nudel [12], who argued that the minimum domain size (FF) heuristic worked by minimizing the size of failed subtrees.

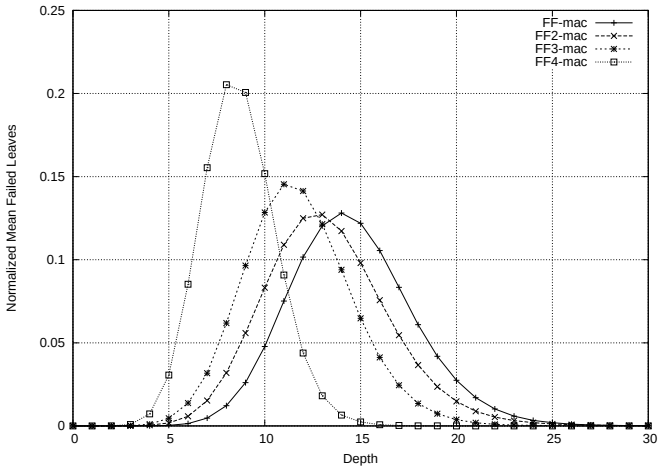

Fig. 7. The distribution of branch depth for soluble problems of size $n = 60$. Qualitatively similar results are seen for the insoluble problems.

Of course, these two ideas are not mutually exclusive, and further work is needed to determine whether either or both approaches will allow us to better understand the basis for search heuristic performance.

Acknowledgement. This work was supported by Science Foundation Ireland under Grant 00/PI.1/C075 and ILOG S.A. We would like to thank Diego Moya for doing some of the coding and especially Barbara Smith for giving us access to the source code and supporting our study.

Table 1. Failure measures and branching factors for fail-first heuristics

	ff	ff2	ff3	ff4
	forward checking			
branch depth	19.9	18.3	16.6	13.7
#failures	600,303	103,365	1,030,002	289,466
$\lvert d \rvert$ chosen	1.35	1.36	2.25	2.18
	MAC			
branch depth	11.4	10.4	9.4	7.1
#failures	126,420	30,642	25,883	14,383
$\lvert d \rvert$ chosen	2.09	1.64	1.37	1.25

Note. Values are means per problem. "#failures" is the number of times an assignment led to a domain wipeout. "$\lvert d \rvert$ chosen" is the mean domain size of variables selected for instantiation.

References

1. Smith, B.M., Grant, S.A.: Trying harder to fail first. In: Thirteenth European Conference on Artificial Intelligence (ECAI 98), John Wiley & Sons, Ltd. (1998) 249–253
2. Haralick, R.M., Elliott, G.L.: Increasing tree search efficiency for constraint satisfaction problems. Artificial Intelligence **14** (1980) 263–314
3. Beck, J.C., Prosser, P., Wallace, R.J.: Toward understanding variable ordering heuristics for constraint satisfaction problems. In: Proceedings of the Fourteenth Irish Artificial Intelligence and Cognitive Science Conference (AICS03). (2003) 11–16
4. Geelen, P.A.: Dual viewpoint heuristics for binary constraint satisfaction problems. In: Proc. 10th European Conf. Artif. Intell. (1992) 31–35
5. Sabin, D., Freuder, E.: Contradicting Conventional Wisdom in Constraint Satisfaction. In: Eleventh European Conference on Artificial Intelligence (ECAI 94), John Wiley & Sons, Ltd. (1994) 125–129
6. Brelaz, D.: New Methods to Color the Vertices of a Graph. Comms of the ACM **22** (1979) 251–256
7. van Beek, P., Chen, X.: CPlan: A constraint programming approach to planning. In: Proceedings of the Sixteenth National Conference on Artificial Intelligence. (1999) 585–590
8. Prosser, P., Stergiou, K., Walsh, T.: Singleton Consistencies. In Dechter, R., ed.: Principles and Practice of Constraint Programming - CP 2000. Proceedings, Springer Verlag (2000) 353–368
9. Gent, I.P., MacIntyre, E., Prosser, P., Smith, B.M., Walsh, T.: Random constraint satisfaction: Flaws and structure. Constraints **6** (2001) 345–372
10. Gent, I.P., MacIntyre, E., Prosser, P., Walsh, T.: The constrainedness of search. In: AAAI/IAAI, Vol. 1. (1996) 246–252
11. Russell, S.J., Norvig, P.: Artificial Intelligence: A Modern Approach. Prentice-Hall (1995)
12. Nudel, B.: Consistent-labeling problems and their algorithms: Expected-complexities and theory-based heuristics. Artificial Intelligence **21** (1983) 263–313