An Empirical Study of Dynamic Variable Ordering Heuristics for the Constraint Satisfaction Problem*

Ian P. Gent[1], Ewan MacIntyre[1], Patrick Prosser[1], Barbara M. Smith[2] and Toby Walsh[3]

[1] Department of Computer Science, University of Strathclyde, Glasgow G1 1XH, Scotland. E-mail: {ipg,em,pat}@cs.strath.ac.uk
[2] Division of Artificial Intelligence, School of Computer Studies, University of Leeds, Leeds LS2 9JT, England. E-mail: bms@scs.leeds.ac.uk
[3] IRST, I38100 Trento & DIST, I16145 Genova, Italy. E-mail: toby@itc.it

Abstract. The constraint satisfaction community has developed a number of heuristics for variable ordering during backtracking search. For example, in conjunction with algorithms which check forwards, the Fail-First (FF) and Brelaz (Bz) heuristics are cheap to evaluate and are generally considered to be very effective. Recent work to understand phase transitions in NP-complete problem classes enables us to compare such heuristics over a large range of different kinds of problems. Furthermore, we are now able to start to understand the reasons for the success, and therefore also the failure, of heuristics, and to introduce new heuristics which achieve the successes and avoid the failures. In this paper, we present a comparison of the Bz and FF heuristics in forward checking algorithms applied to randomly-generated binary CSP's. We also introduce new and very general heuristics and present an extensive study of these. These new heuristics are usually as good as or better than Bz and FF, and we identify problem classes where our new heuristics can be orders of magnitude better. The result is a deeper understanding of what helps heuristics to succeed or fail on hard random problems in the context of forward checking, and the identification of promising new heuristics worthy of further investigation.

1 Introduction

In the constraint satisfaction problem (CSP) we are to assign values to variables such that a set of constraints is satisfied, or show that no satisfying assignment exists. This may be done via a systematic search process, such as depth first search with backtracking, and this amounts to a sequence of decisions, where a decision is a choice of variable and value to assign to that variable. The order in which decisions are made can have a profound effect on search effort. Dechter and Meiri's study of preprocessing techniques [3] shows that dynamic search rearrangement (DSR), i.e. a variable ordering heuristic that selects as next variable

* This research was supported by HCM personal fellowship to the last author, by a University of Strathclyde starter grant to the first author, and by an EPSRC ROPA award GR/K/65706 for the first three authors. Authors listed alphabetically. We thank the other members of the APES group, and our reviewers, for their comments.

the one that has minimal number of values in its domain, dominated all other static orderings. Here, we present three new dynamic variable ordering (dvo) heuristics, derived as a result of our studies of phase transition phenomena of combinatorial problems, and compare these against two existing heuristics.

Tsang, Borrett, and Kwan's study of CSP algorithms [22] shows that there does not appear to be a universally best algorithm, and that certain algorithms may be preferred under certain circumstances. We carry out a similar investigation with respect to dvo heuristics in an attempt to determine under what conditions one heuristic dominates another.

In the next section we give a background to the study. We then go on to describe four measures of the constrainedness of CSP's, and in Section 4 describe five heuristics, based on these measures. The empirical study is reported in Section 5, the heuristics are then discussed with respect to previous work in Section 6, and conclusions are drawn in Section 7.

2 Background

A constraint satisfaction problem consists of a set of n variables V, each variable $v \in V$ having a domain of values M_v of size m_v, and a set of constraints C. Each constraint $c \in C$ of arity a restricts a tuple of variables $\langle v_1, \ldots, v_a \rangle$, and specifies a subset of $M_1 \times M_2 \times \ldots \times M_a$, each element of which is a combination of values the variables are forbidden to take simultaneously by this constraint. In a binary CSP, which the experiments reported here are exclusively concerned with, the constraints are all of arity 2. A solution to a CSP is an assignment of a value to every variable satisfying all the constraints. The problem that we address here is the decision problem, i.e. finding one solution or showing that none exists.

There are two classes of complete search algorithm for the CSP, namely those that check backwards and those that check forwards. In algorithms that checks backwards, the current variable v_i is instantiated and checking takes place against the (past) instantiated variables. If this is inconsistent then a new value is tried, and if no values remain then a past variable is reinstantiated. In algorithms that check forwards, the current variable is instantiated with a value and the (future) uninstantiated variables are made consistent, to some degree, with respect to that instantiation. Chronological backtracking (BT), backmarking (BM), backjumping (BJ), conflict-directed backjumping (CBJ), and dynamic backtracking (DB) are algorithms that check backwards [11, 5, 6, 10], whereas forward checking (FC) and maintaining arc-consistency (MAC) are algorithms that check forwards [13, 18]. This study investigates only forward checking algorithms, and in particular forward checking combined with conflict-directed backjumping (FC-CBJ) [15].

Algorithm FC instantiates variable v_i with a value x_i and removes from the domains of future variables any values that are inconsistent with respect to that instantiation. If the instantiation results in no values remaining in the domain of a future variable, then a new value is tried for v_i and if no values remain for v_i (i.e. a dead end is reached) then the previous variable is reinstantiated (i.e. chronological backtracking takes place). FC-CBJ differs from FC; on reaching a

dead end the algorithm jumps back to a variable that is involved in a conflict with the current variable [15].

In selecting an algorithm we will prefer one that takes less search effort than another, where search effort is measured as the number of times pairs of values are compared for compatibility, i.e. consistency checks. Generally, checking forwards reduces search effort, as does jumping back.

The order in which variables are chosen for instantiation profoundly influences search effort. Algorithms that check backwards tend to use variable ordering heuristics that exploit topological parameters, such as width, induced width or bandwidth, and correspond to static instantiation orders (i.e. they do not change during search) [21]. Algorithms that check forwards have additional information at their disposal, such as the current size of the domains of variables. Furthermore, since domain sizes may vary during the search process, forward checking algorithms may use dynamic variable ordering (dvo) heuristics [17], and it is this class of heuristics that is investigated here.

3 Constrainedness

Many NP-complete problems display a transition in solubility as we increase the constrainedness of problem instances. This phase transition is associated with problems which are typically hard to solve [2]. Under-constrained problems tend to have many solutions and it is usually easy to guess one. Over-constrained problems tend not to have solutions, and it usually easy to rule out all possible solutions. A phase transition occurs in between when problems are "critically constrained". Such problems are usually difficult to solve as they are neither obviously soluble or insoluble. Problems from the phase transition are often used to benchmark CSP and satisfiability procedures [22, 9]. Constrainedness can be used both to predict the position of a phase transition in solubility [23, 20, 16, 7, 19] and, as we show later, to motivate the construction of heuristics.

In this section, we identify four measures of some aspect of constrainedness. These measures all apply to an *ensemble* of random problems. Such measures may suggest whether an individual problem from the ensemble is likely to be soluble. For example, a problem with larger domain sizes or looser constraints is more likely to be soluble than a problem with smaller domains or tighter constraints, all else being equal. To make computing such measures tractable, we will ignore specific features of problems (like the topology of the constraint graph) and consider just simple properties like domain sizes and constraint tightness.

One simple measure of constrainedness can be derived from the *size* of problems in the ensemble. Size is determined by both the number of variables and their domain sizes. Following [7, 8], we measure problem size via the size of the state space being explored. This consists of all possible assignments of values to variables, its size is simply the product of the domain sizes, $\prod_{v \in V} m_v$. We define the size ($\mathcal{N}$) of the problem as the number of bits needed the number of bits needed to describe a point in the state space, so we have:

$$\mathcal{N} =_{\text{def}} \sum_{v \in V} \log_2 m_v \tag{1}$$

A large problem is likely to be less constrained and has a greater chance of being soluble than a small problem with the same number of variables and constraints of the same tightnesses.

A second measure of constrainedness is the *solution density* of the ensemble. If the constraint c on average rules out a fraction p_c of possible assignments, then a fraction $1 - p_c$ of assignments are allowed. The average solution density, ρ is the mean fraction of assignments allowed by all the constraints. The mean solution density over the ensemble is,

$$\rho = \prod_{c \in C} (1 - p_c) \tag{2}$$

Problems with loose constraints have high solution density. As noted above, all else being equal, a problem with a high solution density is more likely to be soluble than a problem with a low solution density.

A third measure of constrainedness is derived from the size and solution density. $E(N)$, the expected number of solutions for a problem within an ensemble is simply the size of the state space times the probability that a given element in the state space is a solution. That is,

$$E(N) = \rho 2^{\mathcal{N}} = \prod_{v \in V} m_v \times \prod_{c \in C} (1 - p_c) \tag{3}$$

If problems in an ensemble are expected to have a large number of solutions, then an individual problem within the ensemble is likely to be loosely constrained and to have many solutions.

The fourth and final measure of constrainedness, κ is again derived from the size and solution density. This has been suggested as a general measure of the "constrainedness" of combinatorial problems [8]. It is motivated by the randomness with which we can set a bit in a solution to a combinatorial problem. If κ is small, then problems typically have many solutions and a given bit can be set more or less at random. For large κ, problems typically have few or no solutions and a given bit is very constrained in how it can be set. κ is defined by,

$$\kappa =_{\text{def}} 1 - \frac{\log_2(E(N))}{\mathcal{N}} \tag{4}$$

$$= -\frac{\log_2(\rho)}{\mathcal{N}}$$

$$= \frac{-\sum_{c \in C} \log(1 - p_c)}{\sum_{v \in V} \log(m_v)} \tag{5}$$

If $\kappa \ll 1$ then problems have a large expected number of solutions for their size. They are therefore likely to be under-constrained and soluble. If $\kappa \gg 1$ then problems have a small expected number of solutions for their size. They are therefore likely to be over-constrained and insoluble. A phase transition in solubility occurs inbetween where $\kappa \approx 1$ [8]. This is equivalent for CSPs to the prediction made in [19] that a phase transition occurs when $E(N) \approx 1$.

4 Heuristics for Constrainedness

Many heuristics in CSPs branch on what can often be seen as an estimate of the most constrained variable [8]. Here, we describe two well known heuristics for CSPs and three new heuristics. We use the four measures of constrainedness described above. These measures were defined for an ensemble of problems. Each measure can be computed for an individual problem, but will give only an estimate for the constrainedness of an individual problem. For example, an insoluble problem has zero solution density and this may be very different from the measured value of ρ. Even so, such measures can provide both a good indication of the probability of a solution existing and, as we show here, a heuristic estimate of the most constrained variable.

Below, we adopt the following conventions. When a variable v_i is selected as the current variable and instantiated with a value, v_i is removed from the set of variables V, constraint propagation takes place, and all constraints incident on v_i, namely C_i, are removed from the set of constraints C. Therefore V is the set of future variables, C is the set of future constraints, m_j is the actual size of the domain of $v_j \in V$ after constraint propagation, p_c is the actual value of constraint tightness for constraint $c \in C$ after constraint propagation, and C_j is the set of future constraints incident on v_j. All characteristics of the future subproblem are recomputed and made available to the heuristics as local information.

4.1 Heuristic FF

Haralick and Elliott [13] proposed the fail-first principle for CSPs as follows: *"To succeed, try first where you are most likely to fail."* The reason for attempting next the task which is most likely to fail is to encounter dead-ends early on and prune the search space. Applied as a constraint ordering heuristic this suggests that we check first the constraints that are most likely to fail and when applied as a variable ordering heuristic, that we choose the most constrained variable. An estimate for the most constrained variable is the variable with the smallest domain. That is we choose $v_i \in V$ such that m_i is a minimum.

An alternative interpretation of this heuristic is to branch on v_i such that we maximize the size of the resulting subproblem, without considering the constraint information on that variable. That is, choose the variable $v_i \in V$ that maximizes

$$\sum_{v \in V - v_i} \log(m_v) \tag{6}$$

where $V - v_i$ is the set of future variables with v_i removed, and is the same as selecting the variable v_i which maximizes the denominator of equation (5).

4.2 Heuristic Bz

The Brelaz heuristic (Bz) comes from graph colouring [1]; we wish to find a colouring of the vertices of a graph such that adjacent vertices have different colours. Given a partial colouring of a graph, the *saturation* of a vertex is the number of differently coloured vertices adjacent to it. A vertex with high saturation will have few colours available to it. The Bz heuristic first colours a vertex of

maximum degree. Thereafter Bz selects an uncoloured vertex of maximum saturation, tie-breaking on the degree in the uncoloured subgraph. Bz thus chooses to colour next what is estimated to be the most constrained vertices.

When applying Bz to a CSP we choose the variable with smallest domain size and tie-break on degree in the future subproblem. That is, choose the variable with smallest m_i and tie-break on the variable with greatest future degree $|C_i|$. In a fully connected constraint graph, Bz will behave like FF, because all variables have the same degree.

4.3 Heuristic Rho

The Rho (ρ) heuristic branches into the subproblem that maximizes the solution density, ρ. The intuition is to branch into the subproblem where the greatest fraction of states are expected to be solutions. To maximize ρ, we select the variable $v_i \in V$ that maximizes

$$\prod_{c \in C - C_i} (1 - p_c) \tag{7}$$

where $C - C_i$ is the set of future constraints that do not involve variable v_i, and $(1 - p_c)$ is the *looseness* of a constraint. If we express (7) as a sum of logarithms, $\sum_{c \in C - C_i} \log(1 - p_c)$, then this corresponds to selecting a variable that minimizes the numerator of (5). Expression (7) gives an estimate of the solution density of the subproblem after selecting v_i. More concisely (and more computationally efficient), we choose the future variable v_i that minimizes

$$\prod_{c \in C_i} (1 - p_c) \tag{8}$$

This is the variable with the most and/or tightest constraints. Again, we branch on an estimate of the most constrained variable.

4.4 Heuristic E(N)

The E(N) heuristic branches into the subproblem that maximizes the expected number of solutions, $E(N)$. This will tend to maximize both the subproblem size (the FF heuristic) and its solution density (the Rho heuristic). Therefore, we select a variable $v_i \in V$ that maximizes

$$\prod_{v \in V - v_i} m_v \times \prod_{c \in C - C_i} (1 - p_c) \tag{9}$$

where $V - v_i$ is the set of future variables with v_i removed, and $C - C_i$ is the set of future constraints that do not involve variable v_i. This can be more succinctly (and efficiently) expressed as choose the variable $v_i \in V$ that minimizes

$$m_i \prod_{c \in C_i} (1 - p_c) \tag{10}$$

The E(N) heuristic has an alternative, intuitively appealing, justification. Let N be the number of solutions to the current subproblem. At the root of the tree, N is the total number of solutions to the problem. If N=0, the current subproblems has no solutions, and the algorithm will at some point backtrack.

If N=1, the current subproblem has exactly one solution, and N will remain constant on the path leading to this solution, but be zero everywhere else. As we move down the search tree, N cannot increase as we instantiate variables. The obvious heuristic is to maximize N in the future subproblem. We use E(N) as an estimate for N, so we branch into the subproblem that maximizes E(N). And this is again an estimate for the most constrained variable, as loosely constrained variables will tend to reduce N most. Consider a loosely constrained variable v_i that can take any value in its domain. Branching on this variable will reduce N to N/m_i. Tightly constrained variables will not reduce N as much.

4.5 Heuristic Kappa

The Kappa heuristic branches into the subproblem that minimizes κ. Therefore, select a variable $v_i \in V$ that minimizes

$$\frac{-\sum_{c \in C - C_i} \log(1 - p_c)}{\sum_{v \in V - v_i} \log(m_v)} \tag{11}$$

Let α be the numerator and β be the denominator of equation (5), the definition of κ. That is, $\alpha = -\sum_{c \in C} \log(1 - p_c)$ and $\beta = \sum_{v \in V} \log(m_v)$. Then we select a variable $v_i \in V$ such that we maximize the following

$$\frac{\alpha + \sum_{c \in C_i} \log(1 - p_c)}{\beta - \log(m_i)} \tag{12}$$

This heuristic was first suggested in [8] but has not yet been tested extensively on a range of CSPs, and depends on the proposal in [8] that κ captures a notion of the constrainedness of an ensemble of problems. We assume that κ provides an estimate for the constrainedness of an individual in that ensemble. We again want to branch on a variable that is estimated to be the most constrained, giving the least constrained subproblem. We estimate this by the subproblem with smallest κ. This suggests the heuristic of minimizing κ.

4.6 Implementing the heuristics

We use all the above heuristics with the forward checking algorithm FC-CBJ. After the current variable has successfully been assigned a value (i.e. after domain filtering all future variables have non-empty domains), the constraint tightness is recomputed for any constraint acting between a pair of variables, v_j and v_k, such that values have just been removed from the domain of v_j or v_k, or both. To compute constraint tightness p_c for constraint c acting between variables v_j and v_k we count the number of conflicting pairs across that constraint and divide by the product of the new domain sizes. This counting may be done via consistency checking and will take $m_j \times m_k$ checks. Constraint tightness will then be in the range 0 (all pairs compatible) to 1 (all pairs are conflicts). When computing the sum of the log looseness of constraints (i.e. the numerator of equation (5)), if $p_c = 1$ a value of $-\infty$ is returned. Consequently, the Kappa heuristic will select variable v_j or v_k next, and the instantiation will result in a dead end.

In the FF heuristic the first variable selected is the variable with smallest domain size, and when all variables have the same domain size we select first the lowest indexed variable v_1. For the Bz heuristic *saturation* is measured as the inverse of the domain size; i.e. the variable with smallest domain size will have largest saturation. Consequently, when the constraint graph is a clique FF and Bz will have identical behaviours.

Search costs reported in this paper do not include the cost in terms of consistency checks of recomputing the constraint tightness. This overhead makes some of the heuristics less competitive than our results might suggest. However, our main concern here is to establish sound and general principles for selecting variable ordering heuristics. In the future, we hope to develop book-keeping techniques and approximations to the heuristics that reduce the cost of re-computing or estimating the constraint tightness but which still give good performance.

5 The Experiments

The experiments attempt to identify under what conditions one heuristic is better than another. Initially, experiments are performed over *uniform* randomly generated CSP. In a problem $\langle n, m, p_1, p_2 \rangle$ there will be n variables, with a uniform domain of size m, $\frac{p_1.n.(n-1)}{2}$ constraints, and exactly $p_2 m^2$ conflicts over each constraint [16, 19]. This class of problem is then modified such that we investigate problems with non-uniform domains and constraint tightness.

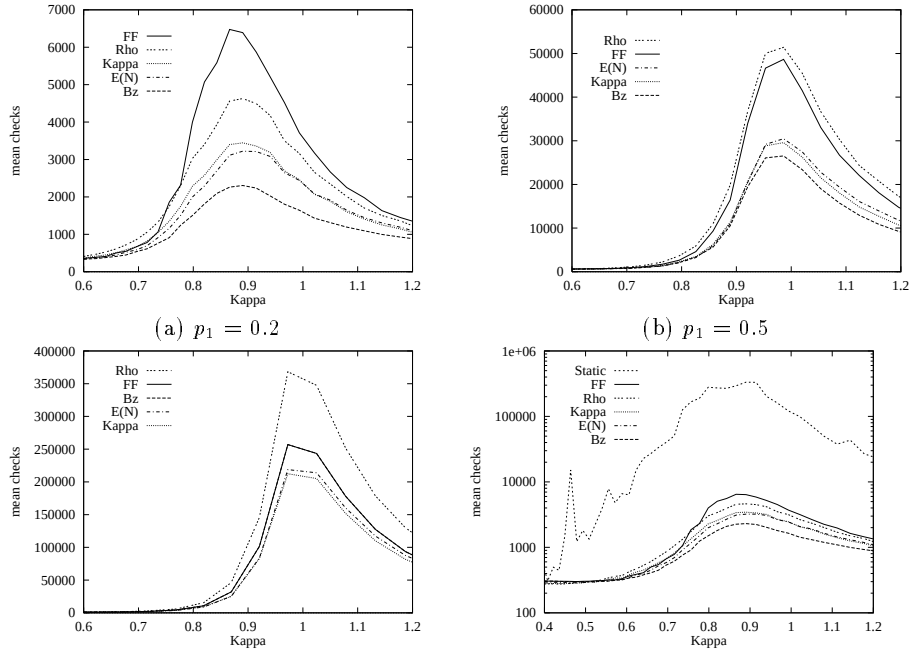
When plotting the results, problems will be measured in terms of their constrainedness, κ. This is because in some experiments we vary the number of variables and keep the degree of variables γ constant, vary the tightness of constraints p_2, and so on. By using constrainedness we hope to get a clear picture of what happens. Furthermore, in non-uniform problems constrainedness appears to be one of the few measures that we can use. It should be noted that in the experiments the complexity peak does not always occur exactly at $\kappa = 1$, and that in sparse constraints graphs the peak tends to occur at lower values of κ, typically in the range 0.6 to 0.9. This has been observed empirically in [16], and an explanation is given by Smith and Dyer [19].

In all of the graphs we have kept the same line style for each of the heuristics. The labels in the graphs have then been ordered, from top to bottom, to correspond to the ranking of the heuristics in the phase transition. The best heuristic will thus appear first.

5.1 Uniform Problems, Varying Constraint Graph Density p_1

The aim of this experiment is to determine how the heuristics are affected as we vary the number of constraints within the constraint graph. The experiments were performed over problems with 20 variables, each with a domain size of 10. In Figure 1, we plot the mean performance for sparse constraint graphs[4] with $p_1 = 0.2$, maximally dense constraint graphs with $p_1 = 1.0$ and constraint graphs of intermediate density $p_1 = 0.5$. At each density 1,000 problems were generated at each possible value of p_2 from 0.01 to 0.99 in steps of 0.01.

[4] Disconnected graphs were not filtered out since they had little effect on performance.



(a) $p_1 = 0.2$ (b) $p_1 = 0.5$

(c) $p_1 = 1.0$ (d) $p_1 = 0.2$, also with static order

Fig. 1. Mean performance of heuristics for $\langle 20, 10 \rangle$

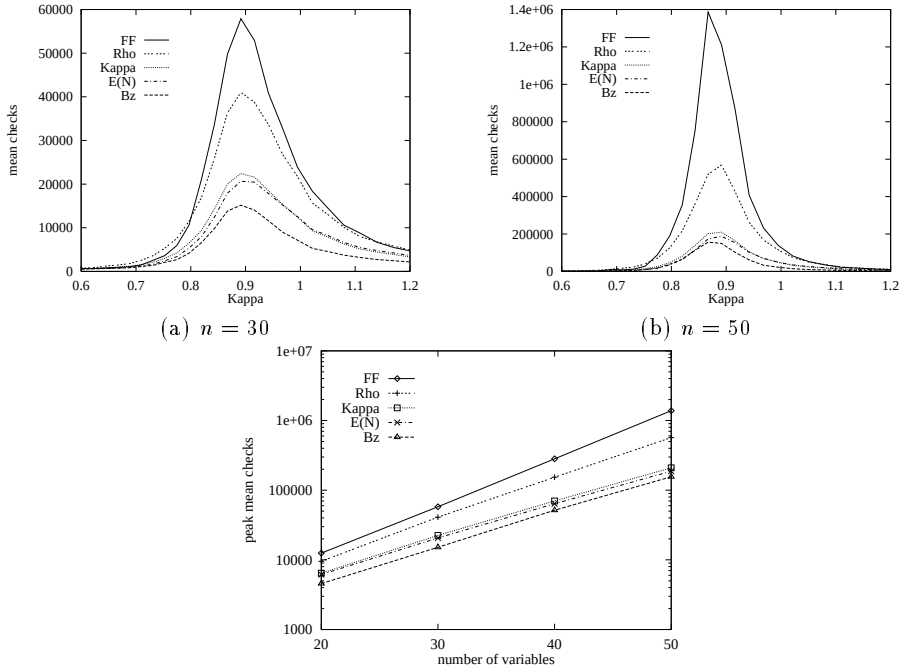
For sparse constraint graphs (see Figure 1(a)), Bz performs best, whilst E(N) and Kappa are not far behind. Rho is significantly worse and FF even more so. Analysing the distribution in performance (graphs are not shown) e.g. the median, 95% and higher percentiles, we observed a similar ranking of the heuristics with the differences between the heuristics opening up in the higher percentiles in the middle of the phase transition. As problems become more dense at $p_1 = 0.5$ (see Figure 1(b)) Kappa dominates E(N). Rho and FF continue to perform poorly, although FF does manage to overtake Rho.

For complete graphs with $p_1 = 1.0$ (see Figure 1(c)), Bz and FF are identical, as expected. (The contour for FF overwrites the Bz contour.) For uniform and sparse problems, Bz seemed to be best, whilst for uniform and dense problems, Kappa or E(N) would seem to be best.

For comparison with the dynamic variable ordering heuristics, in Figure 1(d) we also plot the mean performance of FC-CBJ with a static variable ordering: variables were considered in lexicographic order. Performance is much worse with a static ordering than with any of the dynamic ordering heuristics, even on the relatively easy sparse constraint graphs. The secondary peaks for the static variable ordering at low κ occur as a result of ehps [20], occasional "exceptionally hard" problems that arise following poor branching decisions early in search [9]. The worst case outside the phase transition was more than 14 million checks at $\kappa = 0.46$, in a region where 100% of problems were soluble. This was 5 orders of

magnitude worse than the median of 288 checks at this point.

5.2 Uniform Problems, Varying Number of Variables n



(a) $n = 30$ (b) $n = 50$

(c) Peaks of mean performance over n

Fig. 2. Mean performance for FC-CBJ + heuristics for $\langle n, 10 \rangle$ with $\overline{\gamma} = 5$

The aim of this experiment is to determine how the heuristics scale with problem size. At first sight, this can be simply done by increasing the number of variables n, while keeping all else constant. However, if n increases while p_1 is kept constant the degree γ of a variable (i.e. the number of constraints incident on a variable) also increases. To avoid this, we vary p_1 with n such that average degree $\overline{\gamma}$ remains constant at 5, similar to [12]. To observe a phase transition, 1,000 problems were then generated at each possible value of p_2 from 0.01 to 0.99 in steps of 0.01.

In Figure 2, we plot the performance of each heuristic as we increase n. In Figures 2(a) and (b), we show the mean performance for $n = 30$ and $n = 50$ respectively. The ranking of the heuristics remains the same as in the previous experiment for constraint graphs of intermediate density. Though not shown, we observed similar behaviour in the distribution of performance (*e.g.* median, 95% and higher percentiles). As before, the differences between the heuristics tend to open up in the higher percentiles in the middle of the phase transition.

In Figure 2(c) we plot the peak in average search effort in the phase transition region for each value of n. This then gives a contour showing how search cost increases with n, for this class of problem. The Figure suggests that Bz, Kappa and E(N) scale in a similar manner. Using a least square linear fit on the limited data available, we conjecture that E(N) would become better than Bz when $n > 90$, and Kappa would do likewise when $n > 164$. Further empirical studies on larger problems would be needed to confirm this. However, Rho and FF appear to scale less well. The gradients of Figure 2(c) suggests that FF and Rho scale with larger exponents than Bz, Kappa and E(N).

5.3 Problems with Non-Uniform Constraint Tightness

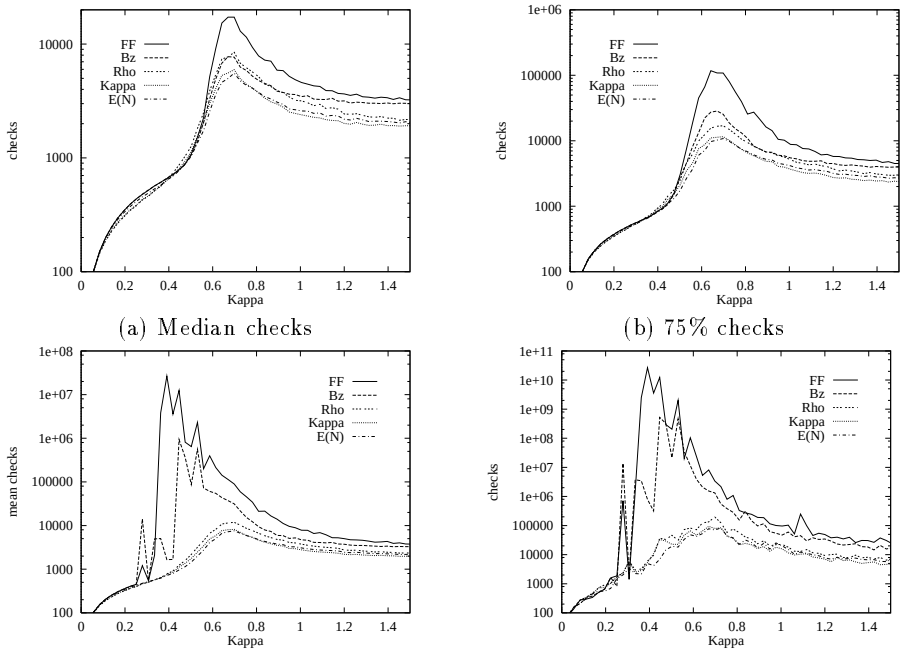
All experiments considered above have constraints generated uniformly. That is, a single value of p_2 describes the tightness of *every* constraint. At the start of search, every constraint is equally tight, so a good measure of the constrainedness of a variable is simply the number of constraints involving this variable (i.e. the variable's degree), together with its domain size. Even as we progress through search and tightnesses vary, this measure should still be reasonably accurate. This might explain why Bz has never been significantly worse in earlier experiments than Kappa or $E(N)$ which undertake the computationally heavy overhead of measuring exact constraint tightnesses.

If we are given a problem with significantly varying constraint tightnesses we must take account of this to measure constrainedness accurately. We therefore expect that Bz and FF may perform poorly on problems with varying constraint tightnesses, while the other heuristics should perform well, because they do take account of constraint tightness. To test this hypothesis, we generated problems with mainly loose constraints, but a small number of very tight constraints. We did this by generating problems with a multiple of 5 constraints, and choosing exactly 20% of these constraints to have tightness $p_2 = 0.8$ (i.e. tight constraints) and the remainder tightness $p_2 = 0.2$ (i.e. loose constraints). We expect Bz to perform poorly on these problems as it will tie-break on the number of constraints and not the tightness of those constraints (the more significant factor in this problem class).

We set $n = 30$ and $m = 10$, and to observe a phase transition we varied the constraint graph density, p_1 from $\frac{1}{87}$ to 1 in steps of $\frac{1}{87}$. Results are plotted in Figure 3. The 50% solubility point is at $\kappa \approx 0.64$ when $p_1 = \frac{23}{87}$.

Median performance, Figure 3(a), shows that as predicted Kappa and E(N) do well. Most significantly, Bz is dominated by all except FF. This is the first of our experiments so far where Bz has been shown to perform relatively poorly.

Figure 3(b) shows the 75th percentiles for the five heuristics (i.e. 75% of problems took less than the plotted amount of search effort) and Figure 3(d) shows worst case. We see that at the 75th percentile there is a greater difference between the heuristics, suggesting a more erratic behaviour from FF and Bz. Mean performance (Figure 3(c)) and worst case performance (Figure 3(d)) shows the existence of exceptionally hard problems for FF and Bz. The worst case for FF was 26,545 million consistency checks at $\kappa \approx 0.39$, in a region where 100% of



(a) Median checks (b) 75% checks

(c) Mean checks (d) Maximum checks

Fig. 3. Performance of heuristics on $n = 30$ and $m = 10$, with $p_2 = 0.2$ for 80% of the constraints, and $p_2 = 0.8$ for the remainder. Note the different y-scales.

problems were soluble. This was 8 orders of magnitude worse than the median of 659 checks at this point, and took 87 hours on a DEC Alpha $200^{4/166}$.

5.4 Problems with Non-Uniform Domain Size

Unlike the other four heuristics, Rho completely ignores the domain sizes and its contribution to problem constrainedness. We therefore expect that the Rho heuristic will do poorly on problems with mixed domain sizes. To test this hypothesis, we generated 20 variable problems, giving each variable a domain of size 10 with probability 0.5 and a domain of size 20 otherwise. We denote this as $m = \{10, 20\}$. To observe a phase transition, we fixed the constraint density p_1 at 0.5 and varied p_2 from 0.01 to 0.99 in steps of 0.01, generating 1,000 problems at each point. We plot the results for mean checks for each of the heuristics in Figure 4. As predicted, the Rho heuristic performs worse than in the previous problem classes. This seems to reaffirm the worth of exploiting information on domain sizes.

6 Discussion

Theory-based heuristics for the binary CSP are presented by Nudel [14], based on the minimization of a complexity estimate, namely the number of compound

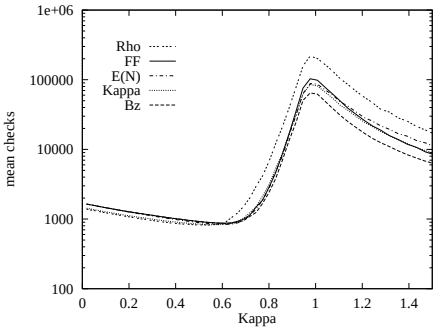


Fig. 4. Performance of FC-CBJ, with $n = 20$, $m = \{10, 20\}$ and $p_1 = 0.5$

labels at a given depth in the search tree. Two classes of heuristic are presented, global and local. Global heuristics fix the instantiation order at the start of search, whereas local heuristics take account of information made available during search, such as actual domain sizes and constraint tightness. Nudel's local heuristics are thus dynamic variable ordering (dvo) heuristics. Three dvo heuristics are presented, IO_2, IO_3, and IO_4. IO_2 chooses *"next below a node, that variable with minimum number m_i of surviving labels after forward checking at the node"*, and is equivalent to FF. Heuristic IO_3 tie-breaks IO_2 by choosing the variable (with smallest domain) that most constrains future variables, and has much in common with Bz. IO_4 stops when any future constraint disallows all tuples across that constraint. As Nudel says, this is not so much a heuristic but an algorithmic step. IO_4 is implicit in heuristics Rho, E(N), and Kappa.

It is interesting to contrast our approach with Nudel's as both give theory-based variable ordering heuristics. Nudel gives measures that estimate the size of the remaining search tree, and then constructs heuristics which seek to minimize these estimates. We have not related our measures directly to the search tree. Instead we have sought to move into areas of the search tree likely to be unconstrained and therefore have solutions. When one makes certain simplifications, both approaches can result in the same heuristic such as FF. However, the detailed relationship between the approaches has not yet been fully analysed.

Feldman and Golumbic [4] applied Nudel's heuristics to real-world constraint satisfaction problems. Three heuristics are presented, one for a backward checking algorithm (BT), and two for a forward checking algorithm (FC1 and FC2). All three heuristics were applied as global/static orderings. Heuristic FC1 selects v_i with minimum $m_i \prod_{i<j}(1-p_{i,j})$, where $p_{i,j}$ is tightness of the constraint acting between v_i and future variable v_j. This corresponds to a global E(N) ordering. Heuristic FC2 takes into consideration all constraints, and selects variable v_i with minimum $m_i \prod_{j \neq i, k \neq i}(1 - p_{j,k})$. As far as we can see, there is no correspondence between FC2 and the heuristics presented here. In their experiments heuristic FC1 dominated FC2 on hard problems.

The new dvo heuristics presented here may be used as global/static vari-

able ordering heuristics. When we have uniform constraint tightness, Rho will correspond to a reverse maximum cardinality ordering [3], suitable for forward checking algorithms. If all variables have the same constraint tightness then E(N) maximizes $\mathcal{N}$ (the FF heuristic), and if all variables have the same domain size E(N) simplifies to maximizing ρ (the Rho heuristic). Like the E(N) heuristic, the Kappa heuristic simplifies to maximizing $\mathcal{N}$ (the FF heuristic) if all variables have the same constraint tightness and to maximizing ρ (the Rho heuristic) if all variable have the same domain size. Clearly, FF and Bz can be considered as low cost surrogates of the minimize Kappa heuristic; both attempt to minimize (11) by maximizing the denominator, and Bz tie-breaks by estimating the numerator of (11) by assuming all constraints are of the same tightness.

7 Conclusions

Three new variable ordering heuristics for the CSP have been presented, namely E(N), Rho, and Kappa. These new heuristics are a product of our investigations into phase transition phenomena in combinatorial problems. The new heuristics have two properties in common. Firstly, they all attempt to measure the constrainedness of a subproblem, and secondly, they attempt to branch on the most constrained variable giving the least constrained subproblem. The heuristics differ in how they measure constrainedness, and what information they exploit.

The new heuristics have been tested alongside two existing heuristics, namely Fail-First (FF) and Brelaz (Bz), and on a variety of uniform and non-uniform problems, using a forward checking algorithm FC-CBJ. On uniform problems, the new heuristics perform similarly to each other and dominate FF. Bz was consistently better on sparse and moderately dense constraint graphs, and was easier to calculate. As constraint graph density increased to the point of becoming a clique, Bz performance degraded to be the same as FF. With respect to problem size, the new heuristics appear to scale better than FF and Bz.

Problems with non-uniform constraint tightnesses exposed poor behaviour from Bz. This was expected, because Bz exploits information from the domain sizes and topology of the constraint graph, but ignores the tightness of constraints. Experiments on problems with non-uniform domains demonstrated that ignoring information of domain sizes results in poor performance.

In some respects the work reported here might be considered as a first foray into a better understanding of what makes heuristics work. Further work could include determining the importance of tie-breaking in the heuristic Bz, compared to simply choosing the *first* variable sensibly. Faster substitutes for the heuristics would allow us to investigate the hypothesis that the new heuristics scale better than the old. Little has been done to compare the ranking of the new heuristics on an individual problem basis. We would also like to investigate the performance of the new heuristics in problems where there is a very large set of different domain sizes at the start of search.

References

1. D. Brelaz. New methods to color the vertices of a graph. *JACM*, 22(4):251–256, 1979.
2. P. Cheeseman, B. Kanefsky, and W.M. Taylor. Where the really hard problems are. In *Proc. IJCAI-91*, pages 331–337, 1991.
3. R. Dechter and I. Meiri. Experimental evaluation of preprocessing algorithms for constraint satisfaction problems. *Artificial Intelligence*, 68:211–241, 1994.
4. R. Feldman and M.C. Golumbic. Interactive scheduling as a constraint satisfaction problem. *Annals of Mathematics and Artificial Intelligence*, 1:49–73, 1990.
5. J. Gaschnig. A general backtracking algorithm that eliminates most redundant tests. In *Proc. IJCAI-77*, page 457, 1977.
6. J. Gaschnig. Performance measurement and analysis of certain search algorithms. Tech. rep. CMU-CS-79-124, Carnegie-Mellon University, 1979.
7. I.P. Gent, E. MacIntyre, P. Prosser, and T. Walsh. Scaling effects in the CSP phase transition. In *Principles and Practice of Constraint Programming*, pages 70–87. Springer, 1995.
8. I.P. Gent, E. MacIntyre, P. Prosser, and T. Walsh. The constrainedness of search. In *Proc. AAAI-96*, 1996.
9. I.P. Gent and T. Walsh. Easy problems are sometimes hard. *Artificial Intelligence*, 70:335–345, 1994.
10. M.L. Ginsberg. Dynamic backtracking. *JAIR*, 1:25–46, 1993.
11. S.W. Golomb and L.D. Baumert. Backtrack programming. *JACM*, 12:516–524, 1965.
12. S. Grant and B.M. Smith. The phase transition behaviour of maintaining arc consistency. In *Proc. ECAI-96*, pages 175–179, 1996.
13. R.M. Haralick and G.L. Elliott. Increasing tree search efficiency for constraint satisfaction problems. *Artificial Intelligence*, 14:263–313, 1980.
14. B. Nudel. Consistent-labeling problems and their algorithms: Expected-complexities and theory-based heuristics. *Artificial Intelligence*, 21:135–178, 1983.
15. P. Prosser. Hybrid algorithms for the constraint satisfaction problem. *Computational Intelligence*, 9(3):268–299, 1993.
16. P. Prosser. An empirical study of phase transitions in binary constraint satisfaction problems. *Artificial Intelligence*, 81(1-2):1–15, 1996.
17. P.W. Purdom. Search rearrangement backtracking and polynomial average time. *Artificial Intelligence*, 21:117–133, 1983.
18. D. Sabin and E.C. Freuder. Contradicting conventional wisdom in constraint satisfaction. In *Proc. ECAI-94*, pages 125–129, 1994.
19. B.M. Smith and M.E. Dyer. Locating the phase transition in binary constraint satisfaction problems. *Artificial Intelligence*, 81(1-2):1–15, 1996.
20. B.M. Smith and S. Grant. Sparse constraint graphs and exceptionally hard problems. In *Proc. IJCAI-95*, pages 646–651, 1995.
21. E.P.K. Tsang. *Foundations of Constraint Satisfaction*. Academic Press, 1993.
22. E.P.K. Tsang, J.E. Borrett, and A.C.M. Kwan. An attempt to map the performance of a range of algorithm and heuristic combinations. In *Hybrid Problems, Hybrid Solutions*, pages 203–216. IOS Press, 1995. Proceedings of AISB-95.
23. C.P. Williams and T. Hogg. Exploiting the deep structure of constraint problems. *Artificial Intelligence*, 70:73–117, 1994.