APES Report: APES-20-2000
Inside MAC and FC

Ian Philip Gent and Patrick Prosser

May 11, 2000

Abstract

We report on the behaviour during search of MAC and FC, the two most popular algorithms for solving constraint satisfaction problems (CSP's). We do so on random CSP's and a jobshop scheduling problem. We make a number of observations based on our experiments. First, MAC can undergo a 'propagation cascade' in which the size of the problem collapses dramatically at a certain depth in search and after which propagation is unnecessary to solve problems. This is not seen in FC. However, this cascade is not so dramatic when search uses a good variable ordering heuristic, and we suggest that the absence of a collapse can be a good thing, preventing too early commitment. We measure other aspects of problems as well. As search progresses in MAC, constraints become looser. This acts to balance the decrease in size. We also compare two measures of constrainedness, that of the solubility of the problem and its arc consistency (AC). The AC constrainedness continues to rise after solubility constrainedness falls, reflecting the fact that propagation becomes more effective deeper in search.

1 INTRODUCTION

How can we understand the behaviour of search algorithms as we vary the problems we try to solve? We suggest that one method is study the internal behaviour of search. In this study we examine how problems change at different depths in search. We do so using the two most popular algorithms for solving CSP's, MAC [11] and FC [8]. To avoid over-generalising, we study a variety of random problems with different natures, as well as a realistic jobshop scheduling problem.

We demonstrate a number of aspects of search inside the two algorithms, focussing particularly on the behaviour of MAC. We show that behaviour is quite different on dense and sparse problems. As might be expected, our results show that generalisation is dangerous: as previous researchers have shown, there is no single 'champion' algorithm. A picture of which algorithm is likely to win in different circumstances can be obtained by studying their behaviour during the search process.

1.1 The algorithms investigated

We focus on two algorithms, forward checking (FC) [8] and maintaining arc-consistency (MAC) [11]. More specifically, we implement MAC using AC3 [10] (and we might call this MAC3). To instantiate the current variable V_i with the value x MAC does the following.

1. The current domain CD_i is set to the singleton $\{x\}$.
2. The constraints from the future incident on V_i are added to the AC3 queue of constraints (Q) and each of those constraints $C_{j,i} \in Q$ is revised, removing unsupported values from CD_j
3. If a constraint $C_{j,i}$ is revised (i.e. unsupported values are removed from CD_j) the constraints from the future incident on V_j are added to Q, if they are not already on Q.

FC can be thought of as a degenerate form of MAC because it ignores step 3. Consequently FC removes from the domains of adjacent future variables values that conflict with the current instantiation. FC and MAC will have the same behaviour in two situations. First, when the instantiation of V_i is compatible with all future values, and second, when the instantiation of V_i removes values from future variables but those future variables have no constraints with any other future variables. In both these situations MAC does not add constraints to Q in step 3 and propagation stops automatically. This raises two questions. When does propagation take place in MAC, i.e how does the algorithm behave? And secondly, how does domain filtering in FC and MAC change the future subproblem during search?

1.2 The problems investigated

The problems studied are predominately random binary constraint satisfaction problems[1] (CSP), in particular flawed model-B [9]. The problems will typically be represented as a 4-tuple $\langle n, m, p_1, p_2 \rangle$, where there are n variables each with a domain of m values, the proportion of constraints in the problem is p_1 and p_2 is the proportion of conflicting tuples in a constraint. By varying p_1 and p_2 we can control the solubility and difficulty of problems.

One measure of a problem is its constrainedness κ [6]. This is defined for model-B problems as follows:

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

where C is a set of constraints, p_c is the tightness of constraint c, V is a set of variables, and m_v is the domain size of variable v. When $kappa = 0$ problems are trivially soluble, when $kappa = \infty$ problems are trivially insoluble, and when $kappa \approx 1$ problems are on the knife edge of solubility and are typically hard.

[1] ... although later on we will look at a jobshop scheduling problem.

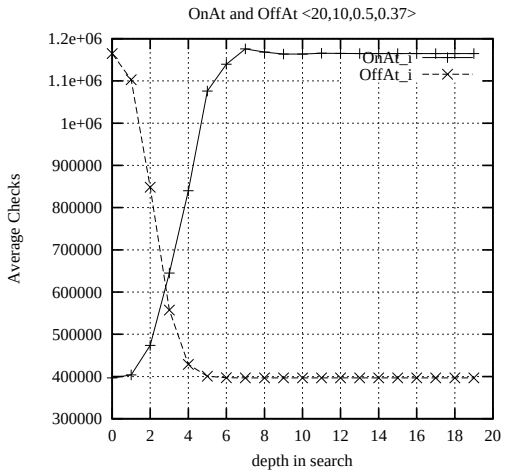

Figure 1: Search effort for $OnAt_i$ and $OffAt_i$ over $\langle 20, 10, 0.5, 0.37\rangle$ for varying values of depth in search i

We can measure the future subproblem during search. When a variable is instantiated we will have one less variable in the future subproblem (n will reduce), domain filtering will reduce domain sizes (m will vary across future variables) and this may change the proportion of conflicting tuples to allowed tuples in a constraint (i.e. p_2 may vary across future constraints). Therefore, we might expect that as search progresses to greater depth in the search tree problem features will change, and this will impact constrainedness, and this in turn may influence algorithmic behaviour.

2 PROPAGATION IN SEARCH

When does propagation take place in search? To answer this question two generic algorithms were developed $OnAt_i$ and $OffAt_i$. $OnAt_i$ enables step 3 above when the search process is at depth $= i$. Therefore $OnAt_0$ is MAC and $OnAt_n$ is FC. Symmetrically, $OffAt_i$ disables step 3 when search is at depth $= i$, and $OffAt_0$ is FC and $OffAt_n$ is MAC. Therefore we have $2 \times n$ actual algorithms to investigate[2]. If algorithm $OffAt_i$ is more efficient than algorithm $OffAt_j$, for all $j > i$, then we might conjecture that there is no propagation beyond depth i. Similarly, if $OnAt_i$ is more efficient than $OnAt_h$, for all $h < i$, we might conjecture that there is no propagation taking place before depth i.

The algorithms were applied to 100 instances of $\langle 20, 10, 0.5, 0.37\rangle$, hard problems close to the solubility phase transition. Figure 1 plots average search effort (measured as consistency checks) against depth in search where propagation was turned on

[2]A similar study was performed by John Gaschnig in his PhD thesis [4] where his algorithm $DEEB_i$ is similar to $OnAt_i$, switching between simple backward checking to MAC at depths beyond a threshold. His experiments used random n-queens problems.

($OnAt_i$) or off ($Off At_i$). That is, a point on the $OnAt_i$ ($Off At_i$) contour shows the average search effort expended by an algorithm that turns propagation on (off) at depth i. The $OnAt_i$ contour shows that for this problem it is best to turn on propagation as early as possible. The $Off At_i$ contour shows that we get the same performance for all values of i greater than 5. That is, for this class of problem a MAC algorithm performs no propagation beyond depth 5 in the search tree. This begs a question "What does the problem look like when propagation stops?".

3 PROBLEM SIZE AT DEPTH

One obvious problem feature to measure is size. The size of a CSP is taken as the product of the domain sizes (the size of the state space). In fact we compute problem size as the sum of the logarithms of the domain sizes, and this is the denominator of equation (1)[3]. As the search process moves to greater depths the size of the future problem decreases. There are two reasons for this. First, as search advances to the next depth we remove the current variable from the future problem. Second, propagation takes place from the current variable removing values from future variables. Therefore problem size must fall for any backtracking algorithm, but this fall will vary depending on the degree of propagation used by the algorithm.

3.1 The size of random problems

We examine 87 soluble random problems $\langle 20, 10, 0.5, 0.36 \rangle$ drawn from 100 instances near the phase transition region. Two algorithms were used, FC [8] and MAC [11],. The size of the future subproblem was measured at each internal node of the search tree, where an internal node corresponds to the search process selecting a new current variable when all future variables have non-empty domains. On completion of search the average size was computed at each depth in the tree (i.e. the sum of the sizes at depth divided by the number of internal nodes at that depth). Figure 2 shows two contours, one for FC the other for MAC (both selecting variables in lexicograpic order), of median problem size at depth. We see that median problem size falls gradually with depth for FC, whereas for MAC there is an abrupt fall in problem size between depths 4 and 6. This collapse in problem size coincides with the end of propagation i.e. step 3 of the algorithm is never performed after the collapse.

3.2 The size of a real problem

Is this collapse only a feature of random problems, or do we see it in real problems? To answer this question we investigated a single 7 by 6 jobshop problem (Figure 3). The problem is posed as a CSP by imposing a release date of zero and some due date. The due date is then reduced until the problem becomes infeasible (i.e. an optimisation problem posed as a sequence of decision problems). The optimal due date is 57.

[3]This is comparable to the measure used by Freeman [3] when studying 3SAT and the DPL search algorithm.

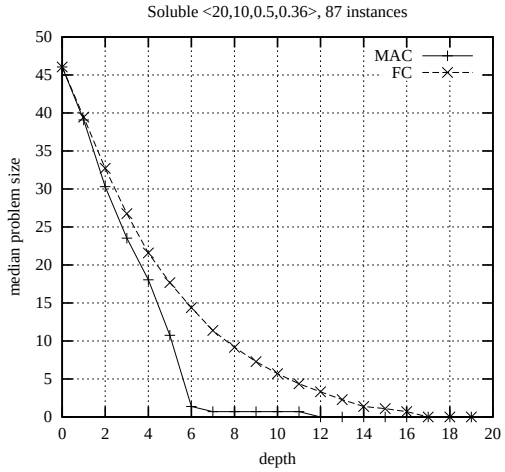

Figure 2: Median problem size at depth in search for MAC and FC for 87 soluble instances of $\langle 20, 10, 0.5, 0.36 \rangle$. No DVO heuristics are used.

```
7 6
2  1  0  3  1  6  3  7  5  3  4  6
1  8  2  5  4 10  5 10  0 10  3  4
2  5  3  4  5  8  0  9  1  1  4  7
1  5  0  5  2  5  3  3  4  8  5  9
2  9  1  3  4  5  5  4  0  3  3  1
1  3  3  3  5  9  0 10  4  4  2  1
2  4  4  4  0  2  3  7  5  2  1  3
```

Figure 3: A 7 by 6 jobshop problem. Each row corresponds to a job (7 off) with 6 operations. Each operation has a resource (0 to 6) and a duration (1 to 10). Operations that execute on the same resource cannot be in process at the same time. The minimum makespan is 57.

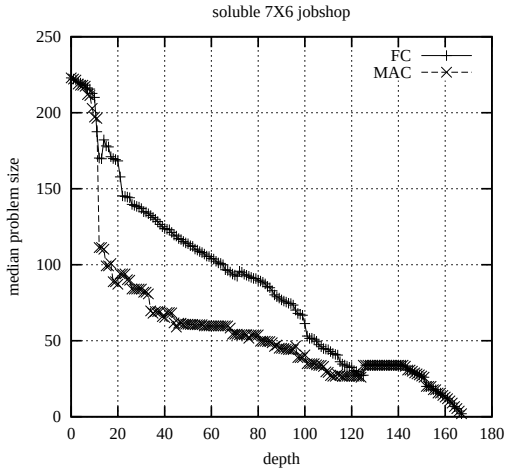

Figure 4: Median problem size at depth in search for MAC and FC for soluble 7X6 jobshop

The problem was represented in a similar way to that of Cheng and Smith [2]. For each resource we have $n(n-1)/2$ decision variable, each associated with the sequencing of a pair of operations on that resource. For example, if decision variable $DV_{i,j}$ is true Op_i will be sequenced before Op_j, if false Op_i will be sequenced after Op_j, and if $DV_{i,j}$ has not been instantiated there will be no sequencing constraints between Op_i and Op_j. In addition we have modelling variables representing the 7 jobs, one variable for each operation. Therefore we have 42 modelling variables and 126 decision variables (168 in all).

The search process instantiates the decision variables, imposing constraints, and maintains arc consistency within the modelling variables. This gives an FC like behaviour, as there is no propagation from the modelling variables back to the decision variables. This was modified to give a full MAC behaviour by adding constraints between the modelling variables and their decision variables. Therefore whenever a modelling variable lost values via propagation a test was then made on the related decision variable to see if either of the resource sequencing constraints were impossible. This might reduce the domain of a decision variable and force a new decision, i.e. full propagation. Figure 4 shows median problem size during search for the FC and MAC behaviour. Both algorithms use the MRV heuristic. We clearly see again an early collapse in problem size at depth round about 10 for the MAC algorithm. And again we see a gradual decline in problem size for FC. It is worth noting that MAC solves the problem with substantially less effort than FC.

4 OTHER MEASURES OF THE PROBLEM

When a variable is selected for instantiation, propagation takes place and all constraints on that variable are removed from the future problem. The number of constraints and their tightness define the numerator of equation (1), and is referred to as *negative log looseness*. If *negative log looseness* falls in value (i.e. the future looses constraints or the constraints become slacker) faster than the reduction in problem size, we should see the constrainedness κ of the future subproblem fall. Alternatively, we might see κ increase, because the fall in problem size out strips fall in the *log looseness*. And of course, the two measures may remain in balance keeping κ constant. So, which one is it?

4.1 κ inside search

At each internal node of the search tree we measured the tightness of each constraint p_c. That is, for each pair of constrained variables we counted the number of remaining compatible pairs of values and divided this by the number of remaining possible pairs of values, to give us a p_c value for each constraint. Combining this with the size of the problem we computed κ at each internal node. We also computed κ_{ac}, the constrainedness of arc-consistency [5], derived from the general definition of constrainedness [6]. If $\kappa_{ac} = 0$ then any selected subset of values from the domains of the variables will be arc-consistent, when $\kappa_{ac} = \infty$ no subset of the domains is arc-consistent, and when $\kappa_{ac} \approx 1$ we expect that we can make half of the problems in an ensemble arc-consistent. In [5] it is shown that at low values of κ_{ac} an arc consistency algorithm, such as AC3, performs very little domain filtering and costs very little to perform. When $\kappa_{ac} >> 1$ arc-consistency processing is again inexpensive, but frequently eliminates sufficient values to prove a problem insoluble. And when $\kappa_{ac} \approx 1$ arc-consistency processing reaches a peak in cost. As a rule of thumb, as κ_{ac} increases in value the amount of propagation increases. Figure 5 shows five contours. The first (solid line with + points) shows the median value of κ at depths in search for FC, out of a sample of 87 soluble problems. FC has a long search, extending to great depths in the search tree (compared to MAC), and we see that κ rises and then falls away. The search tree has greatest width at depth 6, falling away to a single branch at depth 15. Therefore κ falls just as the search tree narrows. This rise and fall of κ is a result of the balance between falling problem size (see figure 2, contour FC) and falling constraint tightness p_c, shown as contour FC-pc (solid line with $\Box$ points).

It must be noted that some data points have to be excluded for FC. During search FC can alter the domains of pairs of constrained future variables such that their constraint tightness $p_c = 1$. When this happens $\kappa = \infty$ and FC blindly advances towards a dead end. When that happens the data point is excluded. Consequently, the FC contours are optimistic in that κ will frequently be off the scale and p_c will frequently be higher than reported.

There are 3 contours for MAC, one for median κ (broken line with $\times$ points), one for median κ_{ac} (dotted line with $\star$ points), and median constraint tightness p_c (broken line with solid $\Box$ points). Again we see κ rise and fall as problem size diminishes faster than constraint tightness. κ then falls away as p_c falls and problem size collapses.

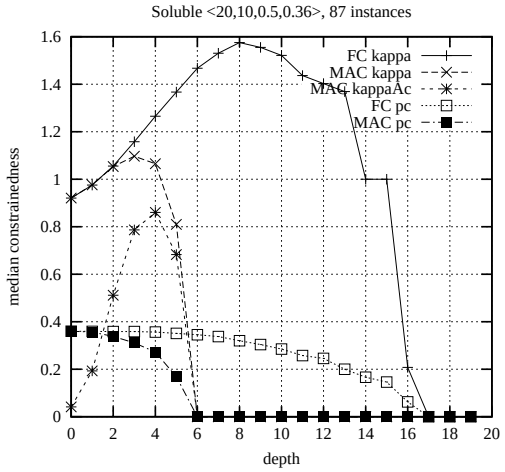

Figure 5: Different measures of constrainedness at depth in search for MAC and FC for 87 soluble instances of $\langle 20, 10, 0.5, 0.36 \rangle$. No DVO heuristics used.

However κ_{ac} continues to climb beyond the depth in search where κ reaches a maximum, and this corresponds to the onset of a cascade in propagation and the collapse in the problem.

5 IS COLLAPSE A GOOD THING?

Is it beneficial to generate an early collapse in the problem, or is it better to delay this? To answer this question we examined the same set of problems but used MAC with a lexicographical ordering of the variables and compared this to MAC with the minimum remaining values (MRV) variable ordering heuristic [8]. Although not shown, MAC produces a collapse in size between depths 4 and 6, whereas MAC-MRV produces a less dramatic effect with a gradual increase in the rate of fall in problem size through depths 7 and 9, but no definite collapse. The explanation for this is almost trivial. The MRV heuristic selects variables with small domains leaving behind future variables with comparably large domains, and this maintains a large sized future problem. This is what is advocated in Haralick and Elliott's fail first principle [8] and the minimise-κ heuristic of [6]. This is shown graphically in figure 6. We see the MAC-MRV contours for κ and κ_{ac} stretched along the x-axis, beyond the basic algorithm. And it appears to be a good thing to do, as this results in a reduction in nodes visited and consistency checks i.e. a reduction in search effort. Clearly, collapse allows the search algorithm to march straight to a solution, but early collapse may be a sign of a poor heuristic.

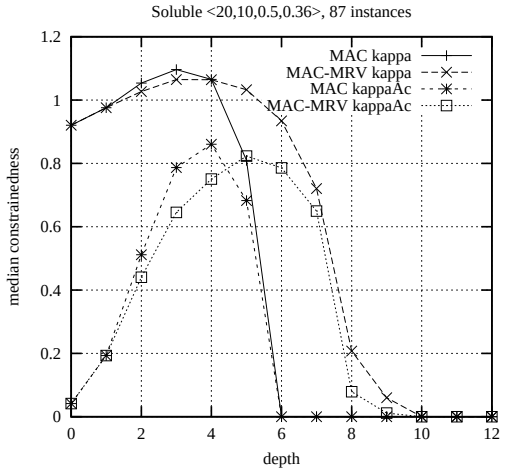

Figure 6: Different measures of constrainedness at depth in search for MAC and MAC-MRV for 87 soluble instances of $\langle 20, 10, 0.5, 0.36 \rangle$. The MRV heuristic maintains a large loose future and delays collapse.

6 IS MAC BETTER THAN FC?

Two reports might lead us to believe that MAC is better than than FC. In [11] experiments are performed on 50 variable problems, uniform domain size of 8, but graphs are sparse. To reach the solubility phase transition constraint tightness is relatively high, and this leads to strong propagation from MAC over few constraints. Sabin and Freuder reported that MAC was frequently many times better than FC with respect to CPU time. However, they also note that "The advantage of MAC ... increases as we move towards less dense, more tightly constrained ...". The down side of this, although not explicitly reported, is that FC has an increasing advantage as problems are dense and constraints are loose.

In [1] again experiments were carried out on large sparse problems, with relatively large domains, and again high constraint tightness (in order to achieve hardness). They reported that "... FC-CBJ becomes less and less worse as the number of constraints grows till the complete graph." Again, this can be expressed positively as saying that FC performs well on dense problems. They also observe that MAC performs well on hard problems with large domains. Again, these are problems with tight constraints where propagation will be effective.

It should be noted that in all of the experiments performed here on $\langle 20, 10, 0.5 \rangle$ problems FC-MRV was more efficient than MAC-MRV, when effort is measured as consistency checks. In all cases MAC-MRV had a smaller search tree, visiting fewer nodes.

One set of experiments from [11, 1] was repeated, namely $\langle 50, 10, 0.1, 0.56 \rangle$. These problems exist at the crossover point where 50% are soluble, and MAC and FC perform similarly (with respect to consistency checks). 50 problems were generated and the av-

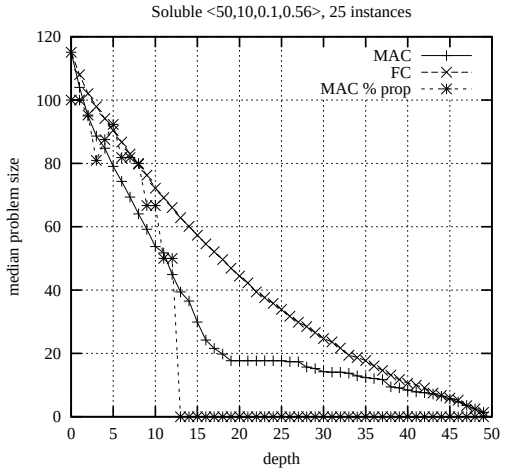

Figure 7: Median problem size at depth in search for MAC and FC for 25 soluble instances of $\langle 50, 10, 0.1, 0.56 \rangle$

erage (median) number of consistency checks performed by MAC-MRV was 738873 (338461) and for FC-MRV 703359 (362486). These figures are in reasonable agreement with the experiments reported by Grant [7] (page 103). The 25 soluble instances were examined as before but with only two algorithms, MAC and FC, both with the MRV heuristic (and we now drop the MRV suffix). Figure 7 shows the median size of the problem at depth in search, along with a contour for the percentage of time propagation occurred within MAC at depth in search (i.e. the percentage of times step 3 of the algorithm was performed at a given depth). The collapse in size is not so marked for MAC but we see that propagation cuts out at depth 13, high up in the search tree. The fall in MAC's problem size beyond depth 13 is due to two things: removal of variables from the future and filtering to adjacent variables as the algorithm locks onto a single solution. When filtering does take place there is no propagation because of the lack of future constraints. FC's behaviour is indistinguishable from the smaller 20 variable problems examined earlier. Figure 8 shows an analysis of various measures of constrainedness within the search process for MAC and FC (both using MRV), over the 25 soluble instances. Again we see κ_{ac} still rising as κ starts to fall away for MAC, and this is happening near to the point where propagation cuts out. The κ contour for FC is similar to that seen previously. Also shown are contours for average constraint tightness p_c for both algorithms. MAC shows median p_c falling and then following a serrated shape. The serration is a result of constraints being removed from the future, causing the median value to rise. On the other hand FC hardly changes p_c. And again the κ and p_c contours for FC are optimistic as there are numerous data points omitted when $\kappa = \infty$ and some future constraint has $p_c = 1$, a situation that cannot occur with MAC.

The experiments here and in previous sections should convince the reader that in hard, large, sparse problems with tight constraints MAC tends to dominate FC, but

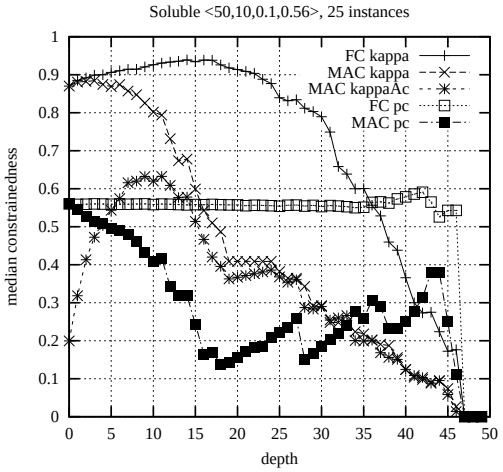

Figure 8: Different measures of constrainedness at depth in search for MAC and FC for 25 soluble instances of $\langle 50, 10, 0.1, 0.56 \rangle$.

in hard dense problems with loose constraints FC will frequently win. We leave the last words on this topic to Grant [7] (page 116) "All of these observations appear to reinforce the increasingly prevalent view that *champion* algorithms which perform extremely well on all types of problem do not exist."

7 CONCLUSION

What has this investigation shown? First, just as we cannot expect a single champion algorithm, we cannot expect a single type of behaviour of an algorithm in different situations. Second, studying the behaviour inside search can help to understand why one algorithm outperforms another on particular problems. Features such as collapse in problem size, increasing looseness, changing constrainedness and a peak in constrainedness of arc consistency, can all be used to gain a picture of how algorithms are likely to perform. As Walsh has shown [12] this can be done by sampling a subset of branches. In the future, this should make it possible to make predictions on algorithms' performance without running them to exhaustion. We are only beginning to unravel the behaviour of the search process, identifying some measurable and changing problem features. One goal is to be able to predict when we are approaching the end of search, or when we should increase or decrease propagation.

Acknowledgements

We would like to thanks our friends and colleagues for their encouragement and contributions.

References

[1] C. Bessiere and J-C Regin, 'MAC and combined heuristics: Two reasons to forsake FC (and CBJ?) on hard problems', in *Principles and Practice of Constraint Programming - CP96*, pp. 61–75. Springer, (1996).

[2] C-C Cheng and S.F. Smith, 'Generating feasible schedules under complex metric constraints', in *Proceedings of AAAI94*, pp. 1086–1091. AAAI Press, (1994).

[3] J.W. Freeman, 'Hard random 3-sat problems and the davis-putman procedure', *Artificial Intelligence*, **81**, 183–198, (1996).

[4] J. Gaschnig, 'Performance measurement and analysis of certain search algorithms', Technical report CMU-CS-79-124, Carnegie-Mellon University, (1979).

[5] I.P. Gent, E. MacIntyre, P. Prosser, P. Shaw, and T. Walsh, 'The constrainedness of arc consistency', in *Proceedings of CP-97*, pp. 327–340. Springer, (1997).

[6] I.P. Gent, E. MacIntyre, P. Prosser, and T. Walsh, 'The constrainedness of search', in *Proceedings of AAAI-96*, pp. 246–252, (1996).

[7] S.A. Grant, *Phase Transition Behaviour in Constraint Satisfaction Problems*, Ph.D. dissertation, The University of Leeds, 1997.

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

[9] E. MacIntyre, P. Prosser, B.M. Smith, and T. Walsh, 'Random constraint satisfaction: Theory meets practice', in *Proceedings of CP-98*, pp. 325–339. Springer, (1998).

[10] A.K. Mackworth, 'Consistency in networks of relations', *Artificial Intelligence*, **8**, 99–118, (1977).

[11] D. Sabin and E.C. Freuder, 'Contradicting conventional wisdom in constraint satisfaction', in *Proceedings of ECAI-94*, pp. 125–129, (1994).

[12] T. Walsh, 'The constrainedness knife edge', in *Proc. AAAI-98*, pp. 406–411, (1998).