

University of Leeds  
**SCHOOL OF COMPUTER STUDIES**  
**RESEARCH REPORT SERIES**  
Report 95.25

**The Phase Transition Behaviour of  
Maintaining Arc Consistency**

by

**Stuart A. Grant and Barbara M. Smith**  
*Division of Artificial Intelligence*

August 1995

## Abstract

In this paper, we study two recently presented algorithms employing a “full look-ahead” strategy: MAC (Maintaining Arc Consistency); and the hybrid MAC-CBJ, which combines conflict-directed backjumping capability with MAC. We observe their behaviour with respect to the phase transition properties of randomly-generated binary constraint satisfaction problems, and investigate the benefits of maintaining a higher level of consistency during search by comparing MAC and MAC-CBJ with the FC and FC-CBJ algorithms, which maintain only node consistency.

The phase transition behaviour that has been observed for many classes of problem as a control parameter is varied has prompted a flurry of research activity in recent years. Studies of these transitions, from regions where most problems are easy and soluble to regions where most are easy but insoluble, have raised a number of important issues such as the phenomenon of exceptionally hard problems (“ehps”) in the easy-soluble region, and the growing realisation that the position of a problem in relation to the phase transition may have a major effect on the relative performance of different algorithms. We therefore apply the algorithms studied to problems covering a range of sizes, topologies and positions in relation to the phase transition, paying particular attention to performance in relation to the occurrence of ehps.

It is shown that increasing look-ahead greatly reduces the areas of the search space which must be explored, enabling backtrack-free searches over a larger range of values of the control parameter, and also greatly reduces the occurrence of ehps. We also observe that the performance of MAC scales much better than that of FC as problem size increases. The addition of CBJ to MAC has little effect on most problems, but further reduces the incidence of ehps to produce stable performance in almost all populations of problems.

## 1 Introduction

Backtracking search algorithms which maintain some level of consistency among the uninstantiated, or *future*, variables during search have become established as the favoured complete methods for searching constraint satisfaction and other classes of hard problems. The most common example of such a technique is the forward checking (FC) [20] algorithm, which maintains node consistency among the future variables by removing values from their domains which are inconsistent with the current partial solution. It has been shown [20, 27, 29] that the additional overheads associated with making more intelligent forward search moves by forward checking are often greatly outweighed by a large reduction in the size of the search tree.

The benefits of achieving a greater level of “look-ahead” capability than FC by maintaining a higher level of consistency among future variables has recently become the subject of a large amount of research, and such an investigation forms the basis of the work presented here. We study a version of the algorithm originally reported by Gaschnig as CS2 [9] and later as DEEB (Domain Element Elimination with Backtracking) [10]. The algorithm re-establishes arc consistency after every instantiation in the sub-problem formed by the future variables and their remaining domains, and following Sabin & Freuder [34] we have termed this algorithm MAC (Maintaining Arc Consistency). Our implementations are based on Mackworth’s AC-3 arc consistency algorithm [24] whereas [34] reports an implementation based on Mohr & Henderson’s AC-4 [26]. As with forward checking, MAC is a chronological backtracker, and so we also investigate the addition of conflict-directed backjumping (CBJ) [29] to MAC to produce the potentially more efficient MAC-CBJ [31].

We study the behaviour of MAC and MAC-CBJ with respect to the *phase transition* behaviour of randomly-generated sets of binary constraint satisfaction problems (CSPs), enabling us to apply the algorithms to many problems covering a range of sizes, topologies and expected difficulties. In order to study the effects of using an increased look-ahead capability, we also compare the performance of MAC with that of FC, and of MAC-CBJ with the equivalent hybrid FC-CBJ [29], over the same sets of problems. In the following subsections, phase transitions and their associated phenomena are introduced, followed by a history of MAC algorithms and an outline of the rest of this report.

## 1.1 Phase transition behaviour

The phenomenon of phase transitions occurring in many types of problem as a control parameter is varied has been recognised and studied extensively in recent years. Cheeseman, Kanefsky and Taylor [5] first reported the phase transition as the interface between a region where almost all problems have many solutions and are relatively easy to solve, and a region where almost all problems have no solution and their insolubility is relatively easy to prove. In this intervening region, the probability of problem solubility falls from close to 1 to close to 0, and the median cost of searching these problems is highest, reaching a peak at the *crossover point* where 50% of problems are soluble. Smith [35, 37] has termed this region the *mushy region*; as problem size increases, the mushy region becomes narrower, and is instantaneous in the limit [45, 46, 47].

Phase transition behaviour has been reported in an increasing number of classes of problem, including satisfiability problems [6, 14, 22, 25], Hamiltonian paths [5], and the travelling salesman problem [16, 17]. Williams and

Hogg [45, 46, 47] have developed approximations to the cost of finding the first solution and to the probability that a problem is soluble, both for specific classes of constraint satisfaction problem (graph colouring,  $k$ -SAT) and for the general case. In the case of binary CSPs, Prosser [30] has carried out an extensive series of experiments investigating the phase transition, and Smith [37] and Smith & Dyer [38] have discussed the extent to which it is possible to predict its location.

The importance of the phase transition cannot be overstated. The increasing set of problem classes for which phase transitions have been observed suggest that they are ubiquitous among NP-complete classes of problem, including “real world” types of problem. Knowledge of the phase transitions for classes of problems has forced a major re-think of the way that algorithm performance is assessed. There has been a rapid move away from attempting to classify algorithms on the basis of limited testing on sets of homogenous problems, such as  $n$ -queens or the zebra problem [8], towards attempting to classify algorithms in terms of performance on large samples of problems of varying size, topology and position in relation to the phase transition (for instance the study by Tsang, et al. [42]). Knowledge of phase transitions should eventually allow problems to be analysed *before* any search is undertaken, and accurate predictions about solution probability, likely difficulty, and most suitable search method to be made.

However, recent studies have highlighted a phenomenon that complicates the phase transition model. The existence of *exceptionally hard problems* (“ehps”) has been reported by Hogg & Williams [21] in graph colouring, Gent & Walsh [12] in satisfiability problems, and by Smith [35] and Smith & Grant [40] in CSPs. These studies show that although there is a well-defined peak in the *median* cost of finding a solution in the region of the phase transition, this is often not where the hardest individual instances occur. Given a large sample of problems, individual problems which are very hard to solve with a particular algorithm may occur in the region where most problems are relatively easy to solve. These searches may be so hard that their cost significantly affects the value of the mean cost; it is for this reason that authors reporting phase transition behaviour have often used the median rather than the mean as a measure of average difficulty.

To date no complete<sup>1</sup> search method has been shown to be completely immune from ehps, although studies of various algorithms [40, 41, 7, 1] have shown that their incidence and magnitude varies greatly between search methods. It is clearly important, therefore, to consider relative ehps behaviour when comparing algorithm performance.

---

<sup>1</sup>To date, no exceptionally hard soluble problems have been reported in studies of incomplete local search methods, for instance those reported in [7], [13] and [21].

## 1.2 A history of MAC

Although MAC is a popular technique employed by the constraint programming community, and is used by many constraint solving tools such as ILOG Solver [32], its application by the constraint satisfaction community has until very recently been passed over in favour of the lesser level of look-ahead provided by FC. A likely explanation for this stems from the previous lack of understanding of the great difference that exists between the behaviour of CSPs with high and low constraint density. More specifically, many of the problems that have until recently formed the test bed for assessing new algorithm performance (in particular  $n$ -queens type problems) have constraint graphs that are cliques (i.e. each variable (queen) is constrained by every other), and therefore have the highest possible constraint density, for which we would expect the look-ahead cost of MAC to be very high.

These expectations are confirmed from the experiments reported by Haralick & Elliot [20] and Nadel [27]. Both of these studies applied various levels of look-ahead to  $n$ -queens type problems, and the results of both showed that the algorithm with the most limited level of look-ahead (that which Haralick & Elliot originally named “forward checking”) performed better on these problems than the algorithms with greater look-ahead capabilities. These results established forward checking as the standard backtracking search strategy by the constraint satisfaction community.

The use of MAC on CSPs covering a range of problem topologies was eventually reported by Sabin & Freuder [34], who presented an implementation based on the AC-4 [26] arc consistency algorithm. They reported significant gains in performance on problems with lower constraint densities by their version of MAC over forward checking, in terms of cpu time, albeit with limited sample sizes of problems.

## 1.3 Structure of this report

In this report we present a major study of the performance of the MAC and MAC-CBJ algorithms over a large range of problem sizes and topologies, showing where the algorithms perform both well and badly, and how their performance compares with the FC and FC-CBJ algorithms employing a lesser level of look-ahead. The relative performance of the algorithms in terms of the incidence and magnitude of exceptionally hard problems is also investigated in some depth.

In the following sections of the report we discuss the method of random problem generation used, review previous research on the exceptionally hard problems that we hope to deal with using increased look-ahead, and describe the AC-3 based MAC and MAC-CBJ algorithms before presenting the major empirical studies that were undertaken.

We study the effects of simple arc consistency preprocessing, building upon previous studies by showing that preprocessing performs little useful work except on tightly constrained problems, and that a phase transition analogous to those occurring between satisfiable and unsatisfiable problem regions is observed for enforcing arc consistency. Study of the performance of MAC at the population level shows that the size of its search trees are much smaller than for FC, that the extra look-ahead produces large regions of backtrack-free search, and that while ehps can still occur, their incidence is greatly reduced from that of FC. It is then shown that while the addition of CBJ to MAC gives little improvement in performance in the average case (an effect similar to adding CBJ to FC), the incidence of ehps is further reduced to the point where only one clear ehp is found among several million candidate sparse problems. A brief study of the microscopic behaviour of a small number of individual ehps is then presented, and we conclude by discussing the series of further issues raised by our results.

## 2 Random problem generation

The experiments reported here were done using sets of randomly-generated binary CSPs, defined by the standard 4-tuple  $\langle n, m, p_1, p_2 \rangle$ , where:

$n$  is the number of problem variables.

$m$  is the number of values in each variable's domain.

$p_1$  is the proportion of pairs of variables which have a constraint between them (i.e. the constraint density)

$p_2$  is the proportion of pairs of values which are inconsistent for a pair of variables, given that there is a constraint between them (i.e. the constraint tightness)

When constructing the constraint graph for a problem we randomly select  $p_1 n(n-1)/2$  from the  $n(n-1)/2$  possible variable pairs as constraints. For each constraint we then randomly select  $p_2 m^2$  from the  $m^2$  possible pairs of values as conflicting pairs. Where necessary, we round to the nearest integer value to give the actual number of constraints and conflicts generated. This random problem generation method corresponds to the "Model B" method used by Smith in [38], where alternative models for random problem generation are discussed in greater detail.

The phase transition from under-constrained to over-constrained problems is observed as  $p_2$  varies, while  $n$ ,  $m$  and  $p_1$  are kept fixed; sets of randomly-generated problems with fixed  $n$ ,  $m$  and  $p_1$  and varying  $p_2$  will be referred to by the tuple  $\langle n, m, p_1 \rangle$ .

Each constraint graph generated is vetted to ensure that it is connected so that the resultant problem cannot be decomposed into smaller components: disconnected graphs are simply discarded and new graphs generated until a connected one is found. For some of the very sparse problems (i.e. with low  $p_1$  and low average degree) not every constraint graph generated at the first attempt is connected, and many graphs may need to be generated and rejected before a connected one is found<sup>2</sup>, making our sparse constraint graphs an unrepresentative sample of random graphs. However, since disconnected constraint satisfaction problems are in practice dealt with using a “divide-and-conquer” strategy, we feel that any conclusions based on disconnected constraint graphs would be flawed, and so for this reason we have excluded them from our study.

Each problem at a particular  $\langle n, m, p_1, p_2 \rangle$  is generated from a single integer seed value, and so single problems or whole sets of problems may be reproduced. This is particularly useful for comparison of algorithms, study of “interesting” problems, etc.

### 3 Tackling the exceptionally hard problems

An obvious aim of implementing and studying new search techniques such as MAC and MAC-CBJ is to attempt to gain improved general performance for certain classes of problem over the techniques that have been employed previously. However, it was the potential impact of extra look-ahead capability on the incidence of exceptionally hard problems that provided the original motivation for the study of MAC, and in this section we review the previous study of this phenomenon, reported in [40]. To recap, we consider an individual CSP to be an exceptionally hard problem (ehp) for a particular search algorithm if:

1. it occurs in the region where almost all problems are soluble, and on average easy to solve (that is, outside the mushy region);
2. it is much more difficult, by at least an order of magnitude, than almost all other problems with the same parameter values;
3. it is more difficult than almost all the problems occurring in the mushy region.

It should be emphasised that this is intended to be a description of ehps, rather than a precise definition. As we reported in [40], individual ehps are highly dependent on the algorithm being used: even a minor change in variable instantiation order may convert an ehp into a much easier problem.

---

<sup>2</sup>Theoretical work on the connectivity of random graphs can be found in [3].

So although individual ehps, and what makes them so difficult for a particular algorithm, should be investigated, we can also ask about ehps in relation to populations of problems and in relation to search algorithms: for instance, do ehps occur in all populations of problems, and are some search algorithms more susceptible to ehps than others?

The previous study of ehps went some way towards addressing these questions by generating and solving large samples of problems, varying problem size, constraint density and constraint tightness. We showed a chronological backtracking algorithm (forward checking with dynamic variable ordering using the “fail-first” principle, or FC-FF) to be relatively susceptible to finding ehps in sparsely constrained samples of problems: in the case of CSPs, these ehps are invariably soluble but can require a search many orders of magnitude more difficult than the median for the given problem parameters. We found no evidence that the incidence and magnitude of ehps is related to the topology of the problems’ constraint graphs, at least for the random graphs, and it was demonstrated that ehp behaviour is maintained as problem size is increased. Another feature of the ehps is that they appear to be exclusive to sparsely constrained problems, at least for forward checking type algorithms: search behaviour becomes extremely regular as constraint density increases, and no suggestion of ehps in problems with dense constraint graphs has been found in our studies to date. However, subsequent studies [41] have shown that densely-constrained ehps may occur with the most naive algorithms having no form of look-ahead capability, for instance plain chronological backtracking [19].

Having examined what might be termed the “macroscopic” behaviour of ehps for this algorithm, we then studied the “microscopic” behaviour of a selection of individual ehps in an attempt to understand the cause of their difficulty. For the chronological backtracker, the first few variable instantiations for each ehp studied leads to an insoluble subproblem, and almost all the search effort is expended in proving its insolubility. Partial solutions involving most of the variables are found in the course of searching the subproblem, resulting in a great deal of backtracking.

As noted in [40], there are, in theory, two potential ways of avoiding ehps which arise through encountering insoluble subproblems early in the search: one is to avoid getting into such subproblems in the first place, and the other is to find some search algorithm which can detect that the subproblem is insoluble more quickly. The previous report looked at the second of these approaches, investigating the effects of adding informed backjumping capability to the FC-FF chronological backtracker (conflict-directed backjumping (CBJ) [29] was added to give FC-CBJ-FF). It was shown that although the addition of CBJ gives little improvement in general over a chronological backtracker which uses a powerful heuristic such as FF (particularly as problem density increases), it can make a huge difference to the difficulty of

the ehps. CBJ allows the algorithm to search insoluble subproblems much more quickly than a chronological backtracker can do. Ehps thereby become much less significant, but do still occur.

The ability of CBJ to crush many ehps suggests that the insoluble subproblems are not generally fundamentally difficult problems, as the insoluble problems in the mushy region undoubtedly are: it seems likely that as a result of the first few instantiations a small subset of the future variables becomes mutually inconsistent, and that those variables happen (in an ehp) to be amongst the last that the algorithm tries to instantiate. In situations such as this, the chronological backtracker will be prone to the most naive form of the pathological behaviour known as thrashing [24]. Failure to find a consistent instantiation for a variable will cause the chronological backtracker to fall back upon the previous variable, which may play no role whatsoever in the cause of the conflict, re-instantiate this variable and proceed to carry out the same set of actions with the same set of outcomes. The backjumper, however, will jump straight back to the variable that is responsible for the conflict (which is, by assumption, one of the earliest to have been instantiated), and so avoid the enormous amount of unnecessary work performed by the backtracker.

It might be argued, however, that rather than employing a more sophisticated form of *backward* search move to escape from these fruitless search paths, an alternative strategy would be to employ a more sophisticated form of *forward* search move which will detect such inconsistencies early and avoid deep search down these paths. We might expect that the technique of increasing look-ahead capability by maintaining arc consistency during search will cause insoluble subproblems to be searched more quickly by failing earlier, perhaps as soon as they are created in some cases. In some respects the improved look-ahead approach may also be viewed as a way of avoiding the hard insoluble subproblems completely by “driving” the search towards fruitful paths, taking advantage of the “fail-first” dynamic variable ordering.

Although MAC requires potentially much more work at each forward search move than forward checking, the greatly enhanced look-ahead may result in even greater savings in search cost, and may have a crucial effect in reducing or even eliminating the occurrence of ehps.

While MAC has been presented here as an alternative approach to back-jumping for tackling ehps, it seems intuitive that combining the two methods could reduce ehp behaviour to a greater extent should they still occur with MAC. Therefore we also study the effects of adding CBJ to MAC, having conjectured that MAC-CBJ might show the most stable performance on sparsely constrained problems of any of the algorithms we have studied so far. We introduce the MAC and MAC-CBJ algorithms in the next section.

## 4 The MAC and MAC-CBJ algorithms

The implementation of MAC and MAC-CBJ that we use is based on Mackworth’s AC-3 arc consistency algorithm [24], and follows the descriptions given by Prosser [31]. Brief descriptions of the AC-3, MAC and MAC-CBJ algorithms are given later in this section: for a more comprehensive description and discussion refer to [31].

It should be emphasised that the decision to base our version of MAC on AC-3 is largely arbitrary. Although there are arguably more efficient, if more complex, arc consistency techniques that could be used<sup>3</sup>, the effect of establishing arc consistency is invariant of the technique used. AC-3 is also a simple algorithm that is easy to follow and, importantly, the effort expended in establishing arc consistency during search using AC-3 can be measured in the same terms as for the backtracking search algorithms themselves: that is, in consistency checks performed. The study by Sabin & Freuder of MAC [34] used an implementation based on Mohr & Henderson’s AC-4 [26]<sup>4</sup>, which performs all consistency checking at the outset when the AC-4 support counters are initialised. During search, all the work of maintaining arc consistency is done by reference to these counters and not to the original constraints. Thus Sabin & Freuder were forced to measure the performance of their MAC implementation in terms of CPU time rather than consistency checks.

**AC-3:** To recap briefly, when establishing arc consistency among a constraint network, we are ensuring that each value in the domain of each variable is *supported* by at least one value in the domain of every variable by which it is constrained. Values which are unsupported are removed from the domains. AC-3 operates on a queue of directed pairs of constrained variables, or *arcs*. When making a directed arc  $(i, j)$  consistent, we prune any values from the domain of variable  $i$  that are unsupported by the current domain of variable  $j$ . If pruning of variable  $i$  does occur, then consistency must be re-established among all arcs incident on  $i$ : that is, each arc  $(x, i)$  that is not already on the queue is added (with the exception of  $(j, i)$ ). This process is repeated until the queue is empty and the network is arc consistent, or domain *wipe-out* of one of the variables occurs, and it is known that the problem has no solutions and no search is necessary. When making an entire problem arc consistent, the queue initially contains every directed arc in the constraint graph.

---

<sup>3</sup>In fact, the improved efficiency of more complex AC algorithms has recently been thrown into doubt [44]

<sup>4</sup>Prosser notes that it may be more appropriate to refer to our version of MAC as MAC3, and to Sabin & Freuder’s version as MAC4, etc., but within the scope of this paper we will continue to refer simply to MAC and MAC-CBJ.

**MAC:** The basis of MAC is to re-establish arc consistency among the uninstantiated variables, or *future* variables, after each variable instantiation, or *forward move*, during search. Having made the entire problem arc consistent initially, when a variable is instantiated its domain can be seen as having temporarily been pruned to a single value. Thus, to re-establish arc consistency following instantiation of a variable  $i$ , AC-3 must be invoked with the queue initially containing all arcs incident on  $i$ : that is, each arc  $(j, i)$  where  $j$  is a future variable constrained to  $i$ . Once again, the effects of pruning values from any of the  $j$  variables must be propagated as before, though only among future variables. If the domain of a future variable is wiped out then we know that the instantiation of  $i$  is inconsistent, and so another value must be tried for it, and the effects of MAC following the abandoned instantiation must be undone. If no more values remain to be tried for  $i$  then the search must backtrack and re-instantiate the previous variable. Prosser notes [31] that if the effects of making the initial queue of  $(i, j)$  arcs consistent are not propagated, then the algorithm becomes Haralick & Elliot's forward checking, where the search only considers the future variables that are directly affected by an instantiation<sup>5</sup>.

**MAC-CBJ:** MAC-CBJ is a more complicated algorithm, which involves the propagation of *conflict sets* as well as constraints. To recap, when using CBJ the conflict set of a variable contains *every* instantiated, or *past* variable, that can have been directly or indirectly responsible for the removal of one or more values from its domain. When MAC-CBJ instantiates a variable  $i$  and the propagated effects of this instantiation result in wipe-out of the domain of a future variable  $x$ , the instantiation and MAC effects must be undone as before, and the conflict set of  $x$  must also be added to the conflict set of  $i$  (excluding  $i$  itself). When no more values can be tried for  $i$ , the search jumps back to the deepest variable  $h$  named in the conflict set of  $i$ , which is added to the conflict set of  $h$  (excluding  $h$  itself), and a new instantiation is tried for  $h$ . To realise MAC-CBJ, when instantiating a variable  $i$  and revising the initial set of constraints  $(j, i)$ ,  $i$  is added to the conflict set of each variable  $j$  that has values removed. Thereafter, when revising an arc  $(k, l)$  where  $k$  and  $l$  are both future variables, the conflict set of  $l$  is added to that of  $k$  if the domain of  $k$  is pruned. If a domain wipe-out occurs, then the identity of the variable and its conflict set are passed back and the actions described above taken. Note that as well as undoing the effects of MAC, the changes to the conflict sets of future variables must also be undone when a variable is uninstantiated. Similarly to MAC, if no propagation of the effects of making the initial queue of  $(i, j)$  arcs consistent is done, then the algorithm reduces

---

<sup>5</sup>Thus, MAC can be viewed as a natural extension of forward checking, or perhaps more correctly FC can be viewed as a degenerate form of MAC!

to FC-CBJ, described by Prosser [29].

**The implementations:** In all of the experiments we report here, both MAC and MAC-CBJ use a dynamic variable ordering heuristic based on the “fail-first” principle: the first variable to be instantiated is the one which is most constrained, and thereafter, the next variable to be instantiated is one with fewest remaining values in its domain. MAC-CBJ has been implemented so that it follows the same search path as MAC, and so will always find solutions in the same order and is always guaranteed to take no more consistency checks than MAC on any individual problem. Note that this also means that MAC-CBJ cannot find a problem exceptionally hard unless MAC does as well.

## 5 The empirical studies

In designing a comprehensive set of experiments to establish the behaviour of the MAC and MAC-CBJ algorithms over large populations of diverse problems, we recognised that it is important to investigate how behaviour (particularly with respect to exceptionally hard problems) changes with both problem *size* and problem *topology*. At first glance, achieving this using the random problems defined by  $\langle n, m, p_1, p_2 \rangle$  appears to be a simple case of varying  $n$  while holding  $p_1$  constant for one set of experiments, and vice versa for another. However, when increasing  $n$  while holding  $p_1$  constant we are in fact increasing the number of constraints (arcs) at a greater rate than we are introducing extra variables (nodes), and so the constraint graph topology changes.

In order to maintain problem topology whilst increasing  $n$ , it is therefore necessary to maintain constant local constraint density, or *average degree* (sometimes referred to as *valency*). Following graph colouring literature convention<sup>6</sup>, we will use  $\gamma$  to refer to average degree.  $\gamma$  is the average number of constraints incident upon each variable, which can be calculated as:

$$\begin{aligned} \text{average\_degree}(\gamma) &= (2 \times \text{num\_constraints})/n \\ \text{where num\_constraints} &= \text{round}(p_1 n(n-1)/2) \end{aligned}$$

Note that the calculation of  $\gamma$  cannot be reduced to  $p_1(n-1)$  without the potential for error, as the number of constraints must be rounded to the nearest integer.  $\gamma$  increases linearly with the number of variables if constraint density is constant, so as  $n$  is doubled with constant  $p_1$ ,  $\gamma$  also approximately doubles (for instance  $\langle 30, m, 0.1 \rangle$  and  $\langle 60, m, 0.05 \rangle$  problems would both have

---

<sup>6</sup>Average degree is in fact a control parameter for graph colouring problems [5]

$\gamma$  around 2.93, and so the latter would give a better indication of the effects of increasing problem size than would  $\langle 60, m, 0.1 \rangle$  problems).

Our previous empirical data suggests that this “local” constraint topology appears to be closely related to the incidence of the exceptionally hard problems: as  $\gamma$  falls, the extremes of problem behaviour become more erratic. It also appears to be the case that ehp behaviour increases with problem size<sup>7</sup>. We therefore aim to design our set of empirical studies so that we may study the independent effects of varying  $\gamma$  and  $n$ .

In the rest of this section, we discuss the issues of measuring search performance and the generation of the problems used. We then describe the major empirical study undertaken, including implementation details, and conclude by outlining the presentation of our results.

## 5.1 Taking search measurements

It is clearly important to take useful measures of performance for individual searches. We take a number of measures for each search, some of which are environment and implementation independent, while others are highly environment and/or implementation dependent. The measures taken are outlined below:

**Consistency checks:** when establishing that a value  $x$  for a variable  $v_i$  is consistent with a value  $y$  for a variable  $v_j$ , a single consistency check is counted. Checks are environment independent, but are highly dependent on implementation efficiency, and the definition of the circumstances under which they are taken<sup>8</sup>. Also note that for some implementations (e.g. ILOG Solver [32]) it may be neither possible or sensible to try and count consistency checks, because of the way the constraints are implemented.

**Nodes visited:** when a search attempts to consistently instantiate a variable by trying one or more values in its domain, until successful or forced to backtrack, each of these *trial instantiations* is counted as a search tree node visited. This definition of nodes visited corresponds to that given in [23]. This measure is both environment and implementation independent, and so allows direct comparison of any MAC implementation. However, it may not reflect the varying levels of search effort required during each trial instantiation, and so is not as analogous to real time as consistency checks.

---

<sup>7</sup>Recent work on scaling effects in CSPs [11] uses a control parameter  $\tau$  whose value is a constant times  $\gamma$ .

<sup>8</sup>For example, whether a check should be counted if the variables are unconstrained: we do not do so.

**CPU time:** the amount of cpu time elapsed during each search is measured approximately. However this measure is highly environment and implementation dependent, and a notoriously difficult measure to take accurately. Circumstances may arise, though, when cpu time is the only suitable measure: for example when trying to estimate overheads that are not reflected in the number of consistency checks or nodes visited, such as the conflict set maintenance cost associated with CBJ.

**Permanent search nogoods:** when establishing arc consistency in an entire problem, the values that are pruned from variable domains can be discarded permanently. The number of such “nogoods” removed is recorded for each search. This measure is both environment and implementation independent.

**Temporary search nogoods:** when re-establishing arc consistency among a subset of problem variables, the “nogoods” cannot be permanently discarded. The total number of these “temporary nogoods” during each search is recorded. This measure is environment independent and largely implementation independent<sup>9</sup>.

The measurement of search effort that is generally favoured is the number of consistency checks made, although cpu time, despite the associated problems, is occasionally favoured by some as the true “bottom line” of search cost. Sometimes, though, cpu time is the only measure that can be taken: the nature of the MAC implementation reported by Sabin & Freuder [34] (described in Section 4) meant that consistency checks could not be counted, and so they report cpu times. However, when presenting the results of our studies on MAC and MAC-CBJ in the following sections, we place an emphasis on nodes visited as a measure of search effort. We suggest that nodes visited is a particularly suitable measure of the performance of all MAC-type search methods as it is completely independent of implementation details, particularly of the underlying arc consistency algorithm on which any version of MAC is based. Search cost in terms of consistency checks is still discussed however, but we choose to omit discussion of the cpu times of searches, chiefly due to the diversity of platforms on which we perform the experimentation.

---

<sup>9</sup>The order in which the arc consistency algorithm propagates constraints may affect the number of temporary nogoods found, as the propagation is halted once any variable domain wipe-out occurs. Hence this measure is partially implementation dependent.

## 5.2 The major experiments

As mentioned earlier in this section, our main experiments to establish the behaviour of MAC and MAC-CBJ were designed to show how behaviour varies as problem size  $n$  and average degree  $\gamma$  are varied. Thus we required two groups of experiments in which we varied  $\gamma$  while holding  $n$  constant, and vice versa. In order to show exactly what the benefits of using the MAC look-ahead technique are, each set of problems was solved not only using MAC and MAC-CBJ, but also using the “underlying” basic algorithms that they can be viewed as extensions of: that is, FC and FC-CBJ<sup>10</sup> respectively, as described by Prosser [29]<sup>11</sup>.

The experiments on each  $\langle n, m, p_1 \rangle$  problem class take the form of generating and searching samples of problems over a range of  $p_2$  (constraint tightness) values, varied in steps of 0.01. For each  $\langle n, m, p_1 \rangle$  problem class, a range of values of  $p_2$  was chosen so as to cover the crossover point and the region of increasing average difficulty leading up to it. In order to have a reasonable chance of seeing the extremes of behaviour, sample sizes of 10000 problems at each set of values of the four parameters were generally used with problem classes having a low  $\gamma$  (where ehps are likely), while sample sizes of 1000 problems were used with problem classes having a high  $\gamma$  (where behaviour is much more regular).

The software to generate problems, search them using various algorithms and collect search statistics is implemented in C, running on a network of around 75 SPARCstation IPX and Silicon Graphics Indigo workstations, and the complete set of experiments outlined below represent an investment of around 1200 days (28000 hours) of cpu time.

**Part 1: varying  $\gamma$  with constant  $n$**  It was decided that for the first set of experiments  $n$  would be held at 30 while  $p_1$  was varied in steps of 0.1 in the range [0.1..1.0] giving a range of values of  $\gamma$ . Following previous experiments, variable domain size  $m$  was held at 10. The set of  $\langle 30, 10, p_1 \rangle$  problem classes studied is listed in Table 1 along with additional background information. The table includes the theoretical critical values of  $p_2$ ,  $\hat{p}_{2crit}$  as presented in [28, 38], at which average search effort is expected to be maximal. This value should also correspond with the crossover point at which 50% of problems are satisfiable, and is calculated as:

$$\hat{p}_{2crit} = 1 - m^{-2/p_1(n-1)}$$

---

<sup>10</sup>Recall that all algorithms use a dynamic variable ordering heuristic using the “fail-first” principle. For brevity we omit the FF tag when naming the algorithms.

<sup>11</sup>We used existing FC and FC-CBJ data developed independently, although we could have generated this data by using MAC and MAC-CBJ with constraint propagation turned off, as described in Section 4.

Problems	$\gamma$	$\hat{p}_{2crit}$	$p_2$ range	samples per $p_2$	Algorithms
$\langle 30, 10, 0.1, p_2 \rangle$	2.934	0.80	0.50-0.90	10000	All
$\langle 30, 10, 0.2, p_2 \rangle$	5.80	0.55	0.20-0.70	10000	All
$\langle 30, 10, 0.3, p_2 \rangle$	8.73	0.41	0.20-0.50	10000	All
$\langle 30, 10, 0.4, p_2 \rangle$	11.60	0.33	0.15-0.40	10000	All
$\langle 30, 10, 0.5, p_2 \rangle$	14.53	0.27	0.10-0.40	1000	All
$\langle 30, 10, 0.6, p_2 \rangle$	17.40	0.23	0.10-0.30	1000	FC/FC-CBJ
$\langle 30, 10, 0.7, p_2 \rangle$	20.33	0.20	0.10-0.30	1000	FC/FC-CBJ
$\langle 30, 10, 0.8, p_2 \rangle$	23.20	0.18	0.05-0.30	1000	FC/FC-CBJ
$\langle 30, 10, 0.9, p_2 \rangle$	26.13	0.16	0.05-0.30	1000	FC/FC-CBJ
$\langle 30, 10, 1.0, p_2 \rangle$	29.00	0.15	0.01-0.30	1000	All

Table 1: The set of  $\langle 30, 10, p_1 \rangle$  problem classes studied.

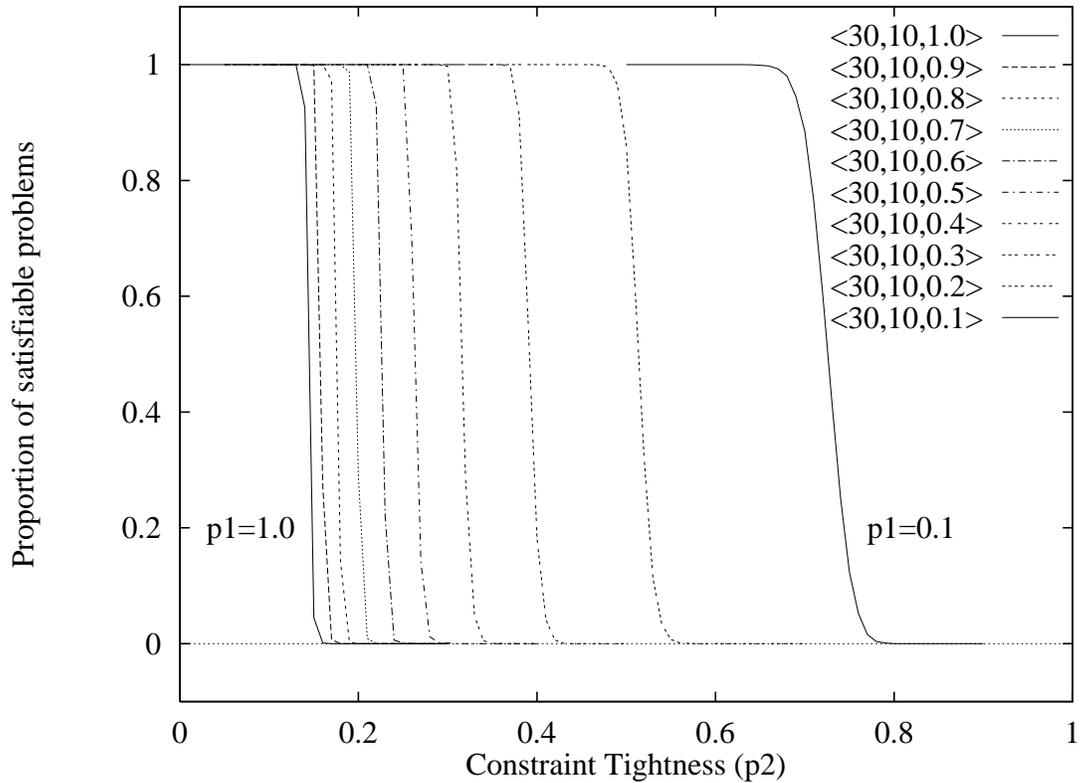


Figure 1: Observed satisfiability curves for the problem classes listed in Table 1.

Figure 1 shows the actual satisfiability curves for each of the ten problem classes, based on the populations of problems generated at each  $\langle 30, 10, p_1, p_2 \rangle$  point. These plots show a similar trend to that reported in [28, 38], in that the predictions are accurate for all values of  $p_1$  except the lowest values where problems are sparsely constrained. There, the theoretical  $\hat{p}_{2crit}$  is an overestimation of the actual critical value.

During the running of the experiments shown in Table 1, it became clear that, as expected, the overhead of MAC becomes very high for densely-constrained problems. For this reason MAC and MAC-CBJ were not run on the classes of problems where  $p_1 = 0.6, 0.7, 0.8$  and  $0.9$ .

**Part 2: varying  $n$  with constant  $\gamma$**  As we are particularly interested in the performance of the algorithms on the exceptionally hard problems, we chose to fix  $\gamma$  at a value where we knew we would find ehps for at least one value of  $n$ . In the study reported in [40] we found a good number of clear ehps for FC and FC-CBJ with the  $\langle 50, 10, 0.1 \rangle$  problem class, which has  $\gamma = 4.92$ . Thus we decided to vary  $n$  in steps of 5 over the range [20..50], calculating  $p_1$  values to give  $\gamma$  as close as possible to  $4.92^{12}$ . The 7 classes of problem studied are listed in Table 2, along with additional background information. The table shows that it is possible to create problems with  $\gamma$  close enough to 4.92 to give the same  $\hat{p}_{2crit}$  predictions for each value of  $n$ , except for  $n = 25$ . Thus we should expect that when plotting the satisfiability curves for the actual problems produced at each value of  $n$ , they should all lie on top of each other. However this plot, shown in Figure 2, shows that the curves do not quite do this, and also do not support the  $\hat{p}_{2crit}$  values. This can be explained by the fact that all of the problem classes are sparsely constrained, and so the  $\hat{p}_{2crit}$  values are likely to be overestimations. As  $n$  decreases,  $p_1$  increases and we see the observed critical values moving slightly closer to  $\hat{p}_{2crit}$ .

It is worth noting that finite size scaling [11] applied to the satisfiability curves, using a rescaled order parameter, would allow the curves to line up exactly, although this would not correct the inaccuracies in predicting the critical value of the control parameter for sparsely constrained problems.

### 5.3 Presentation of the results

We present the results of our studies in the following sections. We begin by examining the effects of simple arc consistency preprocessing on some of the problem classes. The results of the large-scale studies described above

---

<sup>12</sup>Given that  $\gamma$  is based on the discrete number of constraints, and so an exact match may not be possible.

Problems	$\gamma$	$\hat{p}_{2crit}$	$p_2$ range	samples per $p_2$	Algorithms
$\langle 20, 10, 0.2579, p_2 \rangle$	4.90	0.61	0.3-0.8	10000	All
$\langle 25, 10, 0.2067, p_2 \rangle$	4.96	0.60	0.3-0.8	10000	All
$\langle 30, 10, 0.1701, p_2 \rangle$	4.93	0.61	0.3-0.8	10000	All
$\langle 35, 10, 0.1445, p_2 \rangle$	4.91	0.61	0.3-0.8	10000	All
$\langle 40, 10, 0.1256, p_2 \rangle$	4.90	0.61	0.3-0.8	10000	All
$\langle 45, 10, 0.1121, p_2 \rangle$	4.93	0.61	0.3-0.8	10000	All
$\langle 50, 10, 0.1000, p_2 \rangle$	4.92	0.61	0.3-0.8	10000	All

Table 2: The set of  $\langle n, 10, p_1 \rangle$  problem classes studied with constant average degree  $\gamma \approx 4.92$ .

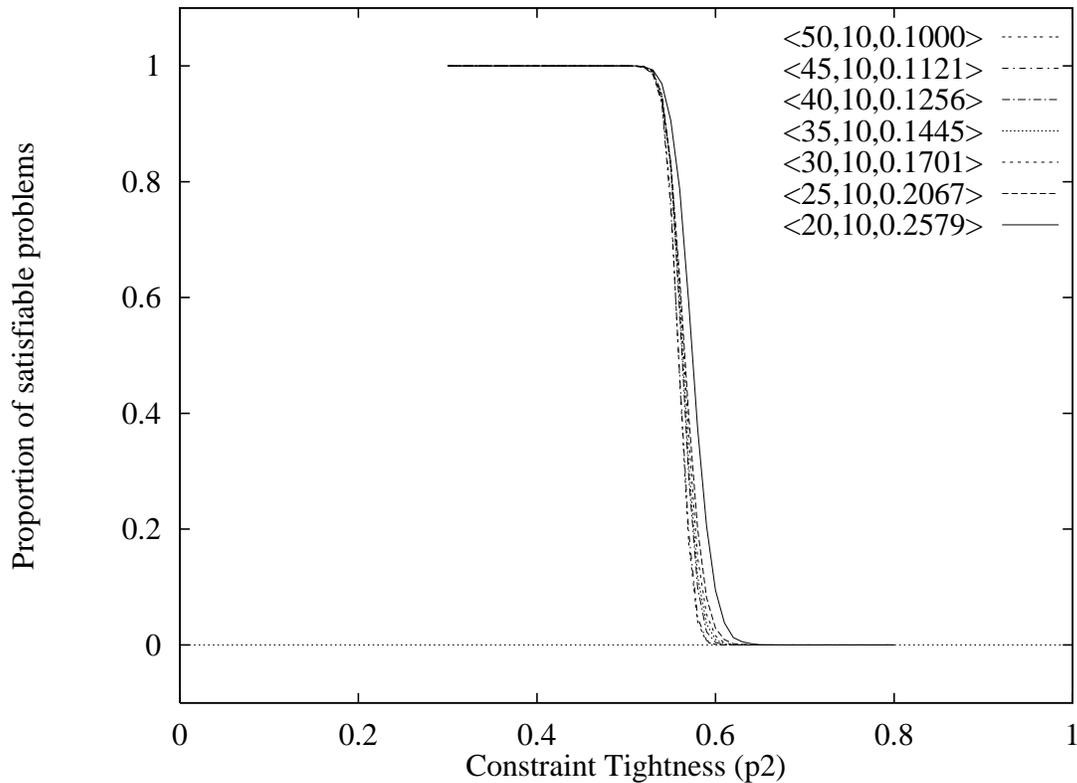


Figure 2: Observed satisfiability curves for the problem classes listed in Table 2.

are then presented, followed by a small-scale study of the effects of the new techniques on some of the individual ehps.

## 6 The effects of AC preprocessing

Following the implementation of MAC reported by Sabin & Freuder [34], we choose to preprocess problems with AC-3 before search by either MAC or MAC-CBJ<sup>13</sup>. Theoretical work on predicting the existing level of consistency in constraint networks has been presented by van Beek [43], and empirical studies of AC preprocessing by Borrett & Tsang [4] has shown that its usefulness is restricted to problems that are over-constrained. A transition has been observed between regions where constraints are very tight and preprocessing eliminates the entire set of variable domains, proving the problems to be insoluble without need for search, and regions where constraints are loose and no pruning of domains occurs. In the intervening region, limited pruning occurs on average, which may sometimes be enough to wipe-out at least one variable domain and prove insolubility, or may partially prune the domains of a number of the problem variables and conceptually make search easier by reducing the potential search space. However, it has been demonstrated [30, 34] that algorithms employing dynamic variable ordering can occasionally perform more poorly as a result of the pruning effects of constraint propagation.

We conducted a brief study of the effects of preprocessing using AC-3 on each of the problem classes listed in Tables 1 and 2. This study is similar in style to that of Borrett & Tsang [4], who reported the effects of preprocessing with AC-6 [2] in terms of the amount of useful work (i.e. the amount of domain pruning) performed on various CSP problem classes. In our study, we look at both the amount of variable domain pruning that occurs and the cost of preprocessing in terms of the consistency checks performed when establishing arc consistency among sets of constrained variables. As we are dealing with the practical application of AC preprocessing, we also terminate the algorithm upon wipe-out of any variable domain, in which case we know that there are no solutions (the preprocessing reported in [4] was always run to completion).

For each  $\langle n, m, p_1 \rangle$  problem class we varied  $p_2$  in steps of 0.01 over the interval  $[0.01, 1.00]$ , generating 1000 problems at each point and preprocessing each problem with AC-3. Figure 3 shows the effects of preprocessing on each of the  $n = 30$  problem classes (left plots) and a selection of the  $\gamma \approx 4.92$  problem classes (right plots), together with the satisfiability curves for the

---

<sup>13</sup>For a simple implementation reason, the AC preprocessing step of MAC and MAC-CBJ is performed by a separate AC-3 procedure which is invoked before starting the main algorithm.

problem classes, as shown in Figures 1 and 2. The plots of the effects of preprocessing show the median curves of the number of values pruned from variable domains, and the median consistency checking effort involved in the preprocessing, both shown on a linear scale.

The curves of the median pruning effects for each problem class show that as constraint tightness increases, the number of values removed rises from zero, increasing slowly at first but then rising sharply to a peak, and then drops sharply down to  $m$  (the variable domain size). As these curves are rising, AC-3 is finding an increasing number of arc-inconsistent values for median problems, although not enough to cause the complete wipe-out of any variable domain. It seems very likely that the peak in each curve corresponds to the point at which a domain wipe-out occurs, and the algorithm starts to terminate early. As the curves fall again, the number of arc-inconsistent values for AC-3 to find is still increasing, and hence domain wipe-outs are occurring more quickly, resulting in the earlier termination of the algorithm (if AC-3 were being run to completion, the curves would rise to a plateau at  $n \times m$ , as those shown in [4] do). When the curves fall to  $m$ , there are no arc-consistent values in any variable domain, and so the algorithm immediately removes the  $m$  values in the domain of the first variable it examines and terminates.

The consistency checking curves show a similar pattern, as might be expected, with a peak in effort coinciding with the peak in “work done” by the algorithm. The patterns of these peaks is the reverse of that for pruning, however: although problems with high constraint density generally require fewer arc-inconsistencies to cause a domain wipe-out, as the effects of removing values tends to be greater, the propagation of these effects has a higher overhead. This results in a greater consistency checking effort on average than for less densely constrained problem classes. This observation suggests that we will see our implementation of MAC perform poorly in terms of consistency checks on problems with high constraint density,  $p_1$ , (and average degree,  $\gamma$ ) due to the higher arc consistency overhead associated with such problems.

It is interesting to note that the behaviour of AC preprocessing observed here is exactly analogous to the phase transitions observed for complete search algorithms. In the case of AC preprocessing, the peak in “effort” occurs between a region where the algorithm can make no conclusions about the satisfiability or otherwise of any problems, and a region where it can show all problems to be insoluble. In the intervening region, it can show a proportion of problems to be insoluble, and the peak in effort would appear to coincide with the point where the algorithm can do this for 50% of problems. We would expect that algorithms enforcing higher levels of consistency between the problem variables would show similar phase transitions at lower constraint tightnesses, naturally culminating with that of enforcing  $n$ -

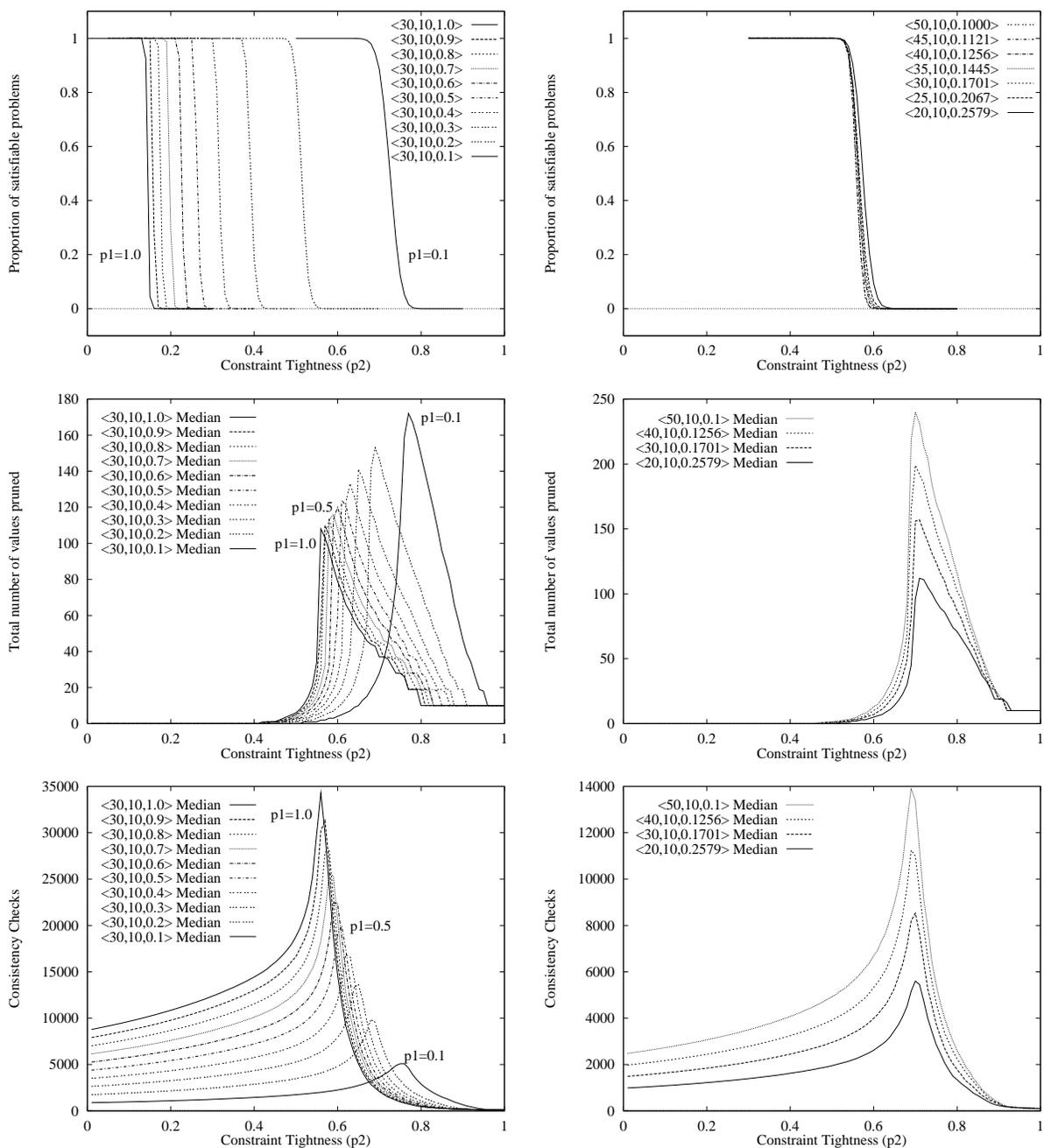


Figure 3: Arc consistency plots:  $n = 30$  on left,  $\gamma \approx 4.92$  on right, showing (top to bottom) the problem satisfiability curves, the median number of values pruned, and the median consistency checking effort used in making the problems arc consistent.

consistency, which would correspond to the phase transition between soluble and insoluble problems.

Observation of the actual positions of the pruning curves agree with the findings reported in [4], in that AC preprocessing has very little effect (in removing values) unless the constraints are tight. For many problem classes, this means that preprocessing only has an effect in the insoluble problem region; it is only for the very sparse problems, where the phase transition occurs at high levels of constraint tightness, that AC preprocessing can affect search on the hard problems in the mushy region. We can see from the plots for the  $\langle 30, 10, 0.1 \rangle$  problems that the point at which the median pruning curve starts to fall (indicating the point at which domain wipe-out occurs for around half of the problems) actually occurs within the mushy region, and for the  $\gamma \approx 4.92$  problems this point occurs slightly into the insoluble problem region. However, as the constraint density (and  $\gamma$ ) of problems increases, the regions where any domain pruning occurs lie further into the insoluble problem region: for  $\langle 30, 10, 0.4 \rangle$  problems no pruning takes place on average inside the mushy region, and for  $\langle 30, 10, 0.7 \rangle$  problems we found no problems in the mushy region to have any arc-inconsistencies at all. These findings suggest that AC preprocessing is only generally effective on insoluble problems. It should be noted, however, that whether or not any values can be pruned depends upon the tightness of individual constraints, and if this is not uniform then AC preprocessing may be worthwhile at lower values of average constraint tightness than is implied by the plots shown.

Although the results of AC preprocessing do not suggest that maintaining arc consistency during search should generally be effective on the types of problems that we are interested in (that is, the hard problems in the mushy region and the ehps in the easy-soluble region), it should be borne in mind that during the process of searching, the sub-problems of uninstantiated variables that we keep arc consistent when using a MAC-type algorithm can be viewed as *new* problems, with different characteristics to the original problem. Thus we might expect these subproblems to include arc-inconsistent values which will be removed, particularly if the subproblem is insoluble. We will investigate this belief in Section 8.

The results of this study of AC preprocessing may be relevant to the notion of a “constraint gap”, proposed by Gent & Walsh [15] as the conditions arising in sparsely constrained problem classes that give rise to the occurrence of ehps. They show that in SAT problems, ehps tend to occur in regions of the control parameter range where the propagation of “goods” and “nogoods” (i.e. values which can be shown to be valid in any solution, and those which can be shown to form part of no solutions) is ineffective. If such a constraint gap exists for CSPs, this AC data (concerning propagation of nogoods) together with data concerning the propagation of goods (such as CSP reduction operators [33]) may provide empirical evidence for it.

## 7 Macroscopic performance of MAC

When analysing the performance of MAC at the population level, we are aiming firstly to gauge the general behaviour of the algorithm in isolation, particularly the extremes of population behaviour such as the occurrence of ehps, and secondly to investigate the comparative performance of MAC and basic FC over the same populations of problems.

To investigate the general behaviour of the algorithm, we plot the median search costs in terms of consistency checks. However, ehps by definition represent extreme behaviour in a population of problems, and so will not be evident in a plot of median cost to find a solution; they will affect the mean cost, but not in a way which elucidates what is happening. For some problem classes, we have therefore chosen to plot the median and higher percentiles, up to the maximum cost, for the sets of problems; this follows graphs shown by Hogg & Williams [21] and Gent & Walsh [12].

### 7.1 General and extreme behaviour

Figure 4 shows the median behaviour of MAC, in terms of consistency checks, on a selection of the  $n = 30$  problem classes, while Figure 5 shows the median and higher percentiles for three of these problem classes. We can see the

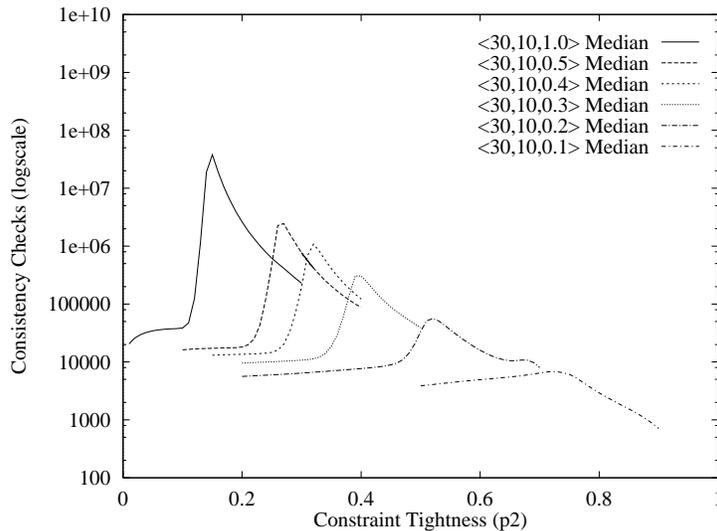


Figure 4: Median performance of MAC on  $n = 30$  series, in terms of consistency checks performed.

clear differences in phase transition behaviour as the constraint density,  $p_1$ , of the problems (and the average degree,  $\gamma$ ) decreases. The behaviour of the  $\langle 30, 10, 1.0 \rangle$  problem class is typical of problems with high constraint density: in this case,  $p_1 = 1$ , i.e. the constraint graph is complete. In terms of median

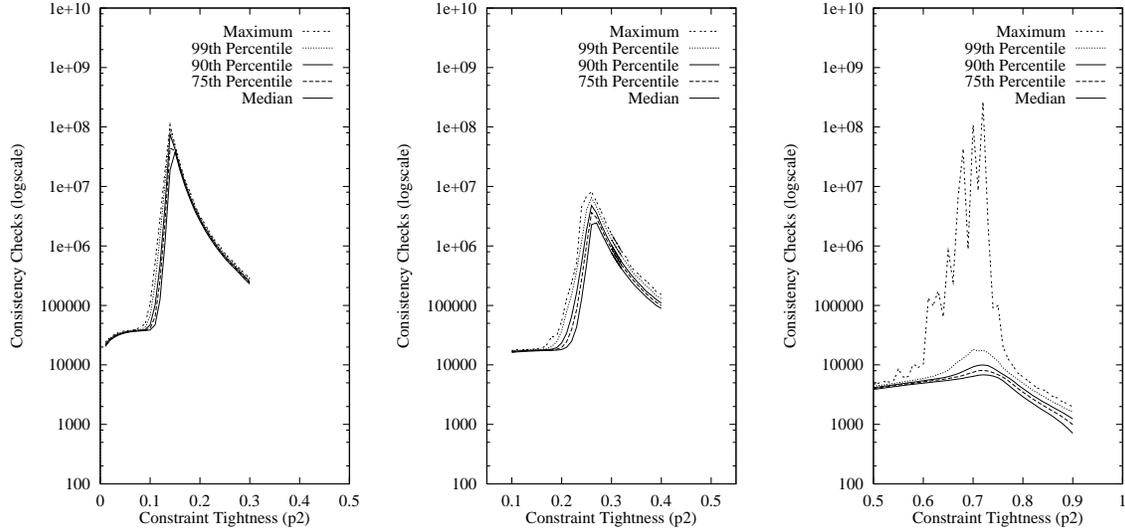


Figure 5: Ranges of consistency checking effort for MAC on  $\langle 30, 10, 1.0 \rangle$ ,  $\langle 30, 10, 0.5 \rangle$  and  $\langle 30, 10, 0.1 \rangle$  problems respectively.

search cost, we see a sharp transition as constraint tightness increases, from the region where problems have very many solutions and are easy to solve, through the crossover point, and into the insoluble region where the cost gradually decreases. Looking at the maximum cost, we see a similarly smooth curve, with values not greatly above those of the median. In particular, in the easy-soluble region where ehps may occur, even the most difficult problem at each value of  $p_2$  is still easy, compared with those in the mushy region.

As problems become more sparsely constrained, we see the peaks in median search cost become less sharply defined, although for each problem class there is still a clear phase transition peak. However, the maximum cost becomes highly erratic for the sparse problems. At  $p_1 = 0.1$  we begin to see instances of exceptionally hard problems in the easy-soluble problem regions that are much more difficult (by at least an order of magnitude) than 99% of the other problems occurring in the sample at the same constraint tightness, and much more difficult (again by at least an order of magnitude) than 99% of the sample problems in the phase transition. As in our previous studies [36, 39], we found no ehps in the easy-soluble regions that are insoluble problems: we continue to conjecture that in the case of CSPs such problems must be exceptionally rare<sup>14</sup>, even among ehps.

Figure 6 shows the median behaviour of MAC, in terms of consistency checks, on a selection of the  $\gamma \approx 4.92$  problem classes, while Figure 7 shows the median and higher percentiles of four of these problem classes. As expected, the median consistency checking effort increases steadily as  $n$  in-

<sup>14</sup>This is not necessarily true for other problem classes such as SAT [12].

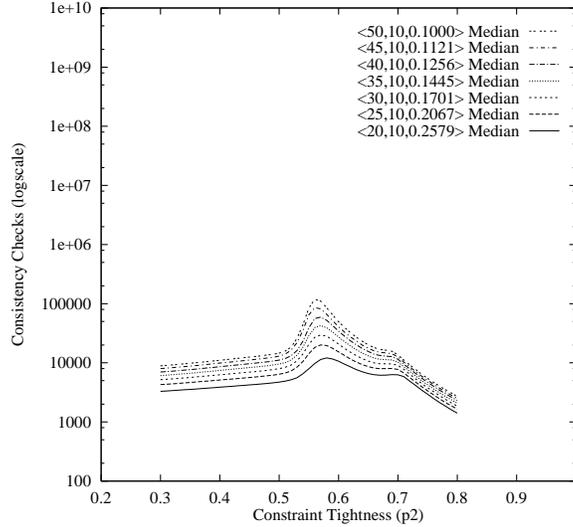


Figure 6: Median performance of MAC on  $\gamma \approx 4.92$  series, in terms of consistency checks performed.

creases, and the phase transition regions are in approximately the same location, as indicated by the satisfiability curves of Figure 2. As  $\gamma$  is relatively low for these problem classes we would expect to see some ehps activity, and indeed clear ehps are visible from the plots of Figure 7. It would also appear from these plots that the incidence and magnitude of ehps increases as  $n$  is increased. The single most difficult problem encountered by MAC in the whole study occurs for a  $\langle 50, 10, 0.1 \rangle$  problem, where  $p_2 = 0.49$ , and is clearly visible in the bottom right hand plot of Figure 7. Solving this problem takes MAC over 1.547 billion consistency checks - over five orders of magnitude greater than the median at that point, and over 100 times more difficult than the hardest phase transition problem. We study this particular ehps, among others, in greater detail in Section 10.

## 7.2 Backtrack-free search

If we look at the performance of MAC in terms of search nodes visited, we can see that there are sets of problems for which no backtracking during search is required on average. When a problem has a solution, a backtrack-free search will visit  $n$  nodes (a single node is required when instantiating each variable). For problems with no solution, we can identify regions where no backtracking<sup>15</sup> is required on average, where either: AC preprocessing wipes

<sup>15</sup>We class the search of an insoluble problem as being “backtrack-free” if only one variable is considered before the search terminates. It should be noted that this definition is not strictly correct, as the algorithm will consider every value in the variable’s domain, and the undoing of each of these trial instantiations is technically a backward search move.

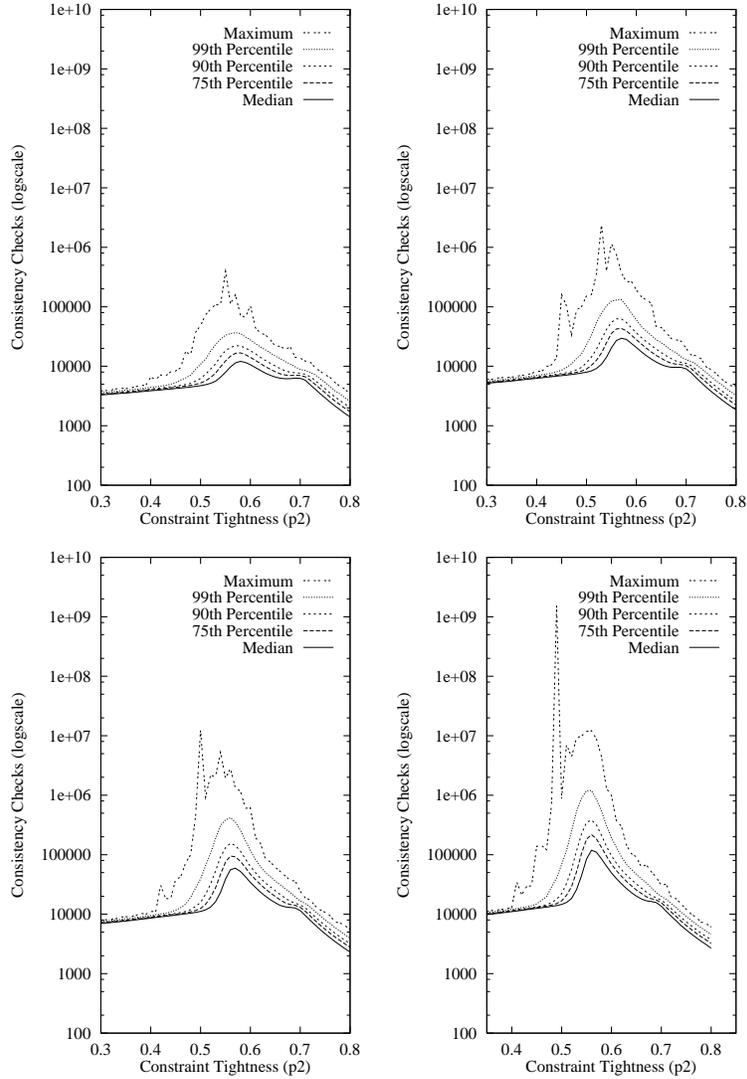


Figure 7: Ranges of consistency checking effort for MAC on (clockwise from top left)  $\langle 20, 10, 0.2579 \rangle$ ,  $\langle 30, 10, 0.1701 \rangle$ ,  $\langle 50, 10, 0.1 \rangle$  and  $\langle 40, 10, 0.1256 \rangle$  problems respectively.

out an entire variable domain, in which case no search is required and so no nodes are visited; or AC preprocessing removes some values in the domain of the first variable the search considers, in which case  $k$  nodes are visited, where  $k$  is the size of the reduced domain (and so  $k < m$ ); or AC preprocessing has no effect, but the MAC look-ahead shows every value of the first variable’s domain to be inconsistent, in which case  $m$  nodes are visited.

Figures 8 and 9 show the median behaviour of MAC in terms of nodes visited, for each of the  $n = 30$  and  $\gamma \approx 4.92$  problem classes to which it was applied. From these plots, it can be seen that for each problem class there

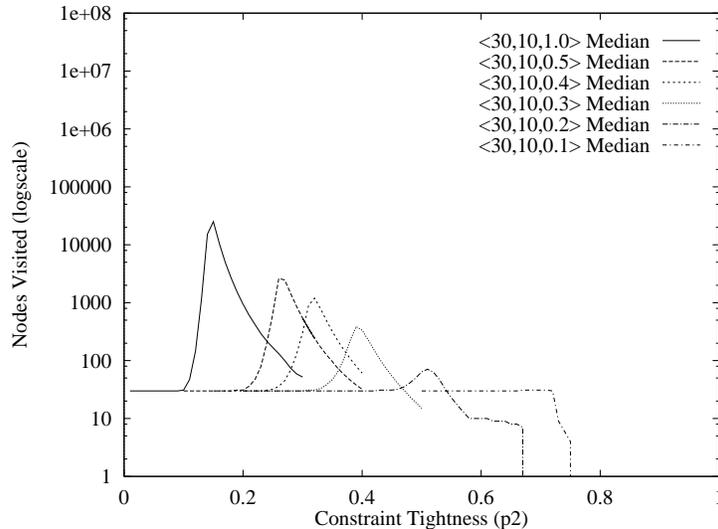


Figure 8: Median (left) and maximum (right) performance of MAC on  $n = 30$  series, in terms of nodes visited.

are parts of the easy-soluble problem regions for which at least half of all searches are backtrack-free. On the other side of the phase transitions the medians fall, and for some of the problem classes with low  $\gamma$  we can see the points in the insoluble regions at which half of all searches are backtrack-free.

An interesting observation from Figures 8 and 9 is that the regions where backtracking search occurs on average are “squeezed” as  $\gamma$  falls. This shows the opposite trend to that of consistency checks, where the peaks coinciding with the phase transitions spread out as  $\gamma$  falls. In fact, the region of backtracking search is all but squeezed out of existence for the  $\langle 30, 10, 0.1 \rangle$  problem class. There is a barely perceptible rise just above  $n$  in the median as the constraint tightness increases, followed by a sharp drop below  $n$  and quickly to zero. This discontinuity in the curve must correspond to the cross-over point, separating the regions where more than half of the problems are soluble and more than half are insoluble.

We compare the regions of backtrack-free search produced by MAC with those of FC in the following subsection.

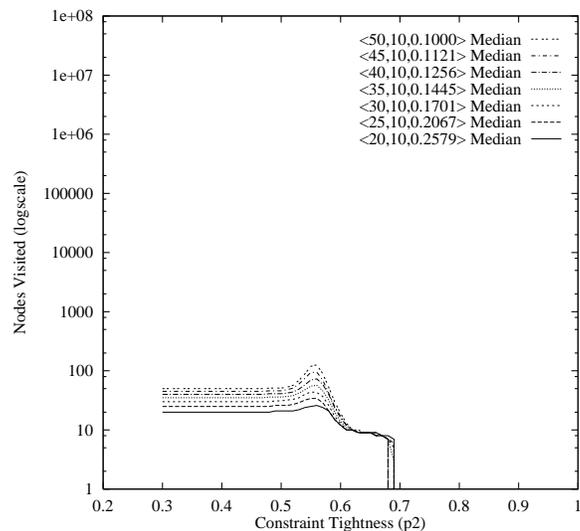


Figure 9: Median performance of MAC on  $\gamma \approx 4.92$  series, in terms of nodes visited.

### 7.3 Comparison with FC

We compare the performance of MAC and FC by plotting the median behaviour in terms of both consistency checks and nodes visited, and by studying the relative incidence of ehps over the same populations of problems. It should be noted that fair comparison of performance between MAC and FC on *individual* problems is not possible as the nature of the fail-first principle means that the algorithms do not necessarily follow the same search paths – thus problems that are hard for one algorithm are not necessarily hard for the other.

Figure 10 shows the relative median performance of the algorithms over each of the  $n = 30$  problem classes studied, in terms of both checks and nodes visited. From these plots, we can see that MAC performs more poorly on average, in terms of consistency checks, for each class. The differences are particularly large over the easy-soluble regions, although this is partly to be expected as the extra look-ahead of MAC (and indeed the lesser look-ahead of FC) is redundant effort on most of these very easy problems.

However, we see a significant improvement on average for MAC over FC in terms of nodes visited over each problem class. Significantly, it can be seen that MAC extends the part of the easy-soluble region in which no backtracking during search is required (for at least half of the problem samples) further towards the crossover point. It is clear from the study of AC preprocessing in Section 6 that it is the re-establishing of arc consistency that extends the region of backtrack-free search, and not the initial preprocessing that is done by MAC before search. MAC also extends the part of the insoluble problem

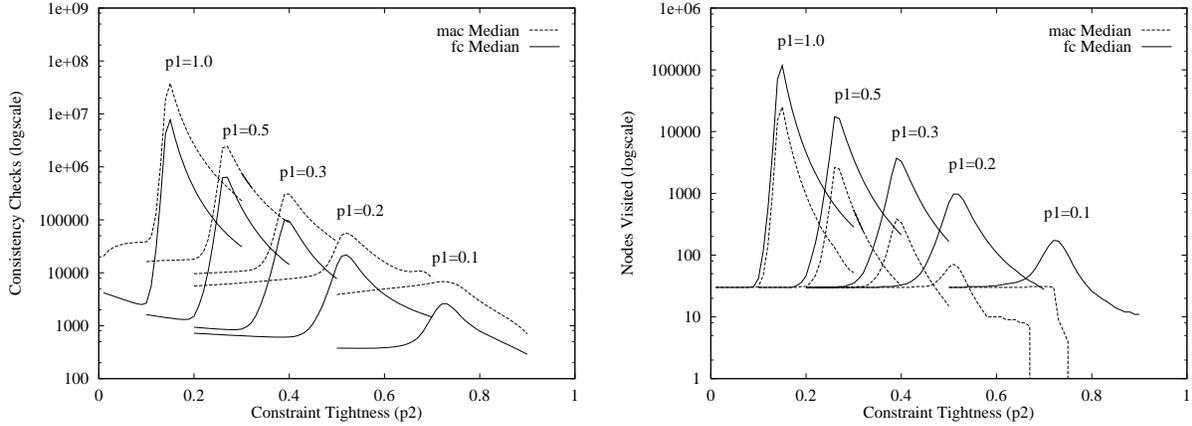


Figure 10: Comparison of median performance of MAC versus FC for  $n = 30$  series, in terms of consistency checks (left) and nodes visited (right).

regions in which no search is required for at least half of the problems further towards the crossover point. This is undoubtedly due in large part, however, to the AC preprocessing becoming effective when the constraints are tight. In short, the plots of nodes visited show that by using MAC, the range of values over which any search is required for at least half of the problems is squeezed towards the phase transition.

Figure 11 shows a similar comparison to Figure 10, this time for three of the  $\gamma \approx 4.92$  problem classes. Interestingly, we see that although on average FC always outperforms MAC in terms of consistency checks at low  $n$ , the gap reduces as  $n$  is increased, and at  $n = 50$  MAC actually outperforms FC on average on the hardest phase transition problems around the crossover point. If this trend continues, then MAC must considerably outperform FC in terms of checks as  $n$  is further increased (holding  $\gamma$  constant). The plots of relative nodes visited also show MAC outperforming FC to a greater extent as  $n$  increases: at  $n = 20$  MAC is around an order of magnitude better than FC in the phase transition, rising to two orders of magnitude at  $n = 50$ .

Figure 12 shows the median and higher percentiles of consistency checking for FC on four of the  $\gamma \approx 4.92$  problem classes. These plots can be directly compared with those showing the performance of MAC over the same problems, in Figure 7. Making this comparison, it can clearly be seen that, although MAC has been shown to still be susceptible to ehps, their incidence is greatly reduced from that of FC over the same populations of problems. Whether the use of MAC also reduces the magnitude of the exceptionally hard searches that it does encounter is not entirely clear, however: although the heights of the ehps “peaks” that can be seen in the plots for MAC are generally lower than those for FC, some MAC ehps such as that at  $\langle 50, 10, 0.1, 0.49 \rangle$  are more extreme than nearly all FC ehps in the same

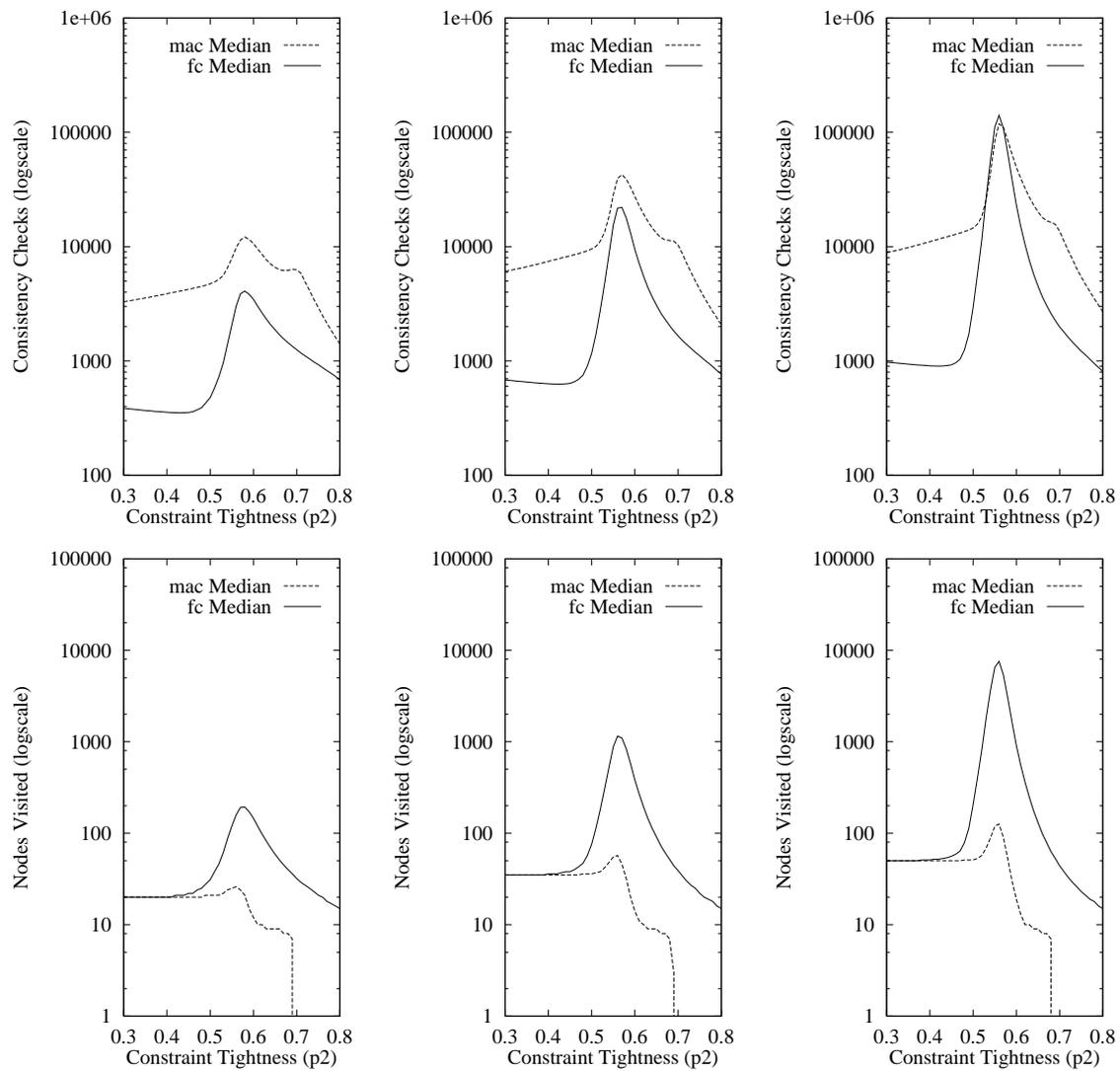


Figure 11: Comparison of median performance of MAC versus FC for (left to right)  $\langle 20, 10, 0.2579 \rangle$ ,  $\langle 35, 10, 0.1445 \rangle$  and  $\langle 50, 10, 0.1 \rangle$ , showing consistency checks (top) and nodes visited (bottom).

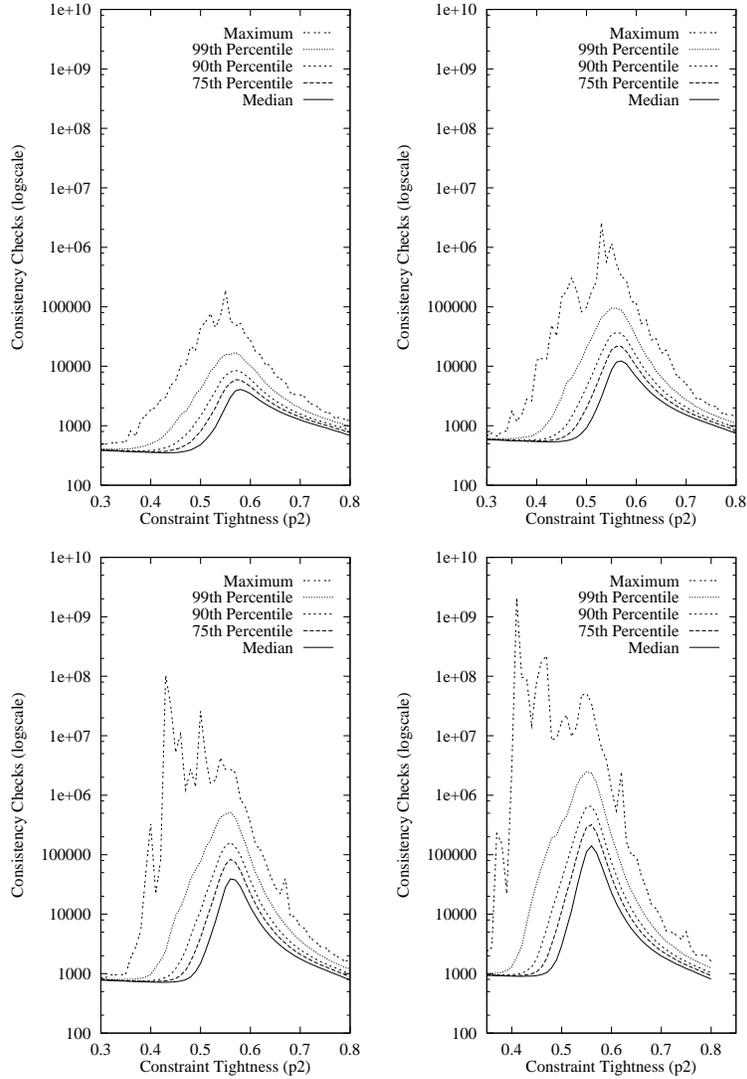


Figure 12: Ranges of consistency checking effort for FC on (clockwise from top left)  $\langle 20, 10, 0.2579 \rangle$ ,  $\langle 30, 10, 0.1701 \rangle$ ,  $\langle 50, 10, 0.1 \rangle$  and  $\langle 40, 10, 0.1256 \rangle$  problems respectively.

problem class. This suggests that the ehp criteria for MAC may be somewhat more complex than that for FC. We investigate this in Section 10.

The results that have been presented in this section show that the number of consistency checks performed by MAC searches compared to other algorithms is less informative than the number of nodes visited, since it depends greatly on how efficiently arc consistency is re-established and whether or not this effort can be properly reflected in this measure. While we have used an AC-3 based implementation of MAC, the study by Sabin & Freuder used an AC-4 based version of MAC, and this implementation sets up all of the required support sets and counters that are required by AC-4 at the outset of search; thereafter, there need be no reference to the constraints and so search effort cannot be measured in consistency checks. Sabin & Freuder noted significant gains in term of cpu time by their MAC over FC for problems where we see MAC perform fairly poorly in terms of consistency checks, although MAC appears to get better as we increase the problem sizes. This may be due in part to the arguably more efficient AC-4, and the uncertainties associated with reporting cpu times. However, when we compare MAC and FC in terms of nodes visited – an implementation independent measure – we see MAC perform much better than FC on problems of all constraint densities. These results suggest that the efficiency of the MAC implementation will be a a very important issue in practical situations. The worth of using the increased looking-ahead capabilities of MAC is also demonstrated in its ability to greatly reduce the incidence of exceptionally hard problems from that of FC.

## 8 The extra look-ahead of MAC

In Section 6 we considered the question of whether we can expect the pruning performed by MAC during search to occur only when searching problems for which AC preprocessing has been shown to be useful (i.e. problems that are generally to the right of the phase transition and are unlikely to be soluble), or whether the extra look-ahead of MAC can be effective on problems over a greater range of constraint tightnesses. An analogy with the behaviour of the forward checking algorithm can be drawn to support the latter: FC maintains node consistency during search, and although our binary CSPs are always node consistent initially, FC is known to perform useful work over a large range of control parameter values.

In order to confirm the work done by MAC, for each search performed in the empirical studies we recorded the total number of “temporary nogoods” (as defined in Section 5) found by MAC during search. This measure of the amount of domain pruning is a superset of that which would be done by

forward checking, and although this is a fairly crude measure, it will clearly indicate the regions of constraint tightness where the look-ahead effort has the greatest effect.

Figure 13 shows the median amount of pruning performed by MAC on three of the  $n = 30$  problem classes studied, shown on a linear scale and with the p-sat curves for each problem class superimposed to indicate the location of the phase transition region. These plots show that the peak in

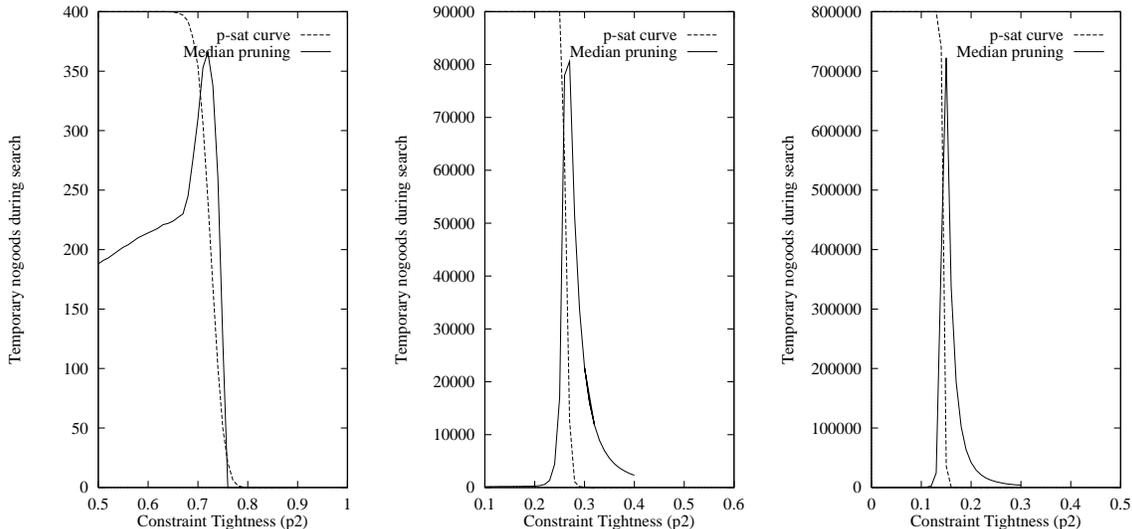


Figure 13: The median amount of temporary pruning from variable domains for (left to right)  $\langle 30, 10, 0.1 \rangle$ ,  $\langle 30, 10, 0.5 \rangle$  and  $\langle 30, 10, 1.0 \rangle$  problems.

pruning performed by MAC coincides exactly with the phase transition over a range of problem densities, and that MAC performs useful work across the spectrum of constraint tightnesses, including the easy-soluble regions where the ehps occur. Similar plots for all other problem classes studied show that this behaviour is maintained over the range of  $n$  and  $\gamma$  values.

These plots clearly show that MAC can be much more effective on all types of problems than simple AC preprocessing, and would appear to confirm the suggestion made in Section 6 that the subproblems created during search can be very different in nature to a problem in its entirety. Incidentally, we can observe from these plots the points at which AC preprocessing eliminates the need for search for at least half of the problems, where the median amount of pruning falls to zero.

## 9 Macroscopic performance of MAC-CBJ

We now present an analysis of the performance of MAC-CBJ at the population level, in a similar fashion to that of MAC presented in Section 7. The

performance of MAC-CBJ in isolation is observed, in terms of its average and extremes of behaviour, followed by a comparison with FC-CBJ.

## 9.1 General and extreme behaviour

Figure 14 shows the median behaviour of MAC-CBJ, in terms of consistency checks, on a selection of the  $n = 30$  problem classes, while Figure 15 shows the median and higher percentiles for three of these problem classes, and Figure 16 shows a similar analysis for a selection of the  $\gamma \approx 4.92$  problem classes. These figures can be directly compared with those showing the

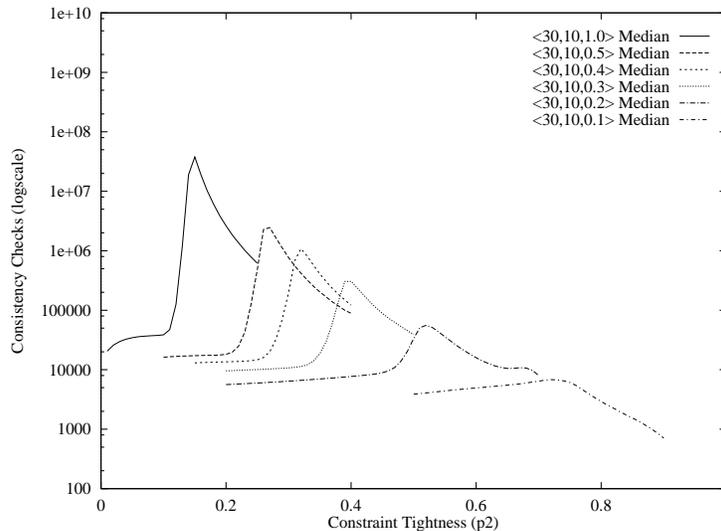


Figure 14: Median performance of MAC-CBJ on  $n = 30$  series, in terms of consistency checks performed.

equivalent data for MAC, shown in Figures 4, 5 and 7 respectively.

By comparing the sets of plots, it is evident that the behaviour of MAC and MAC-CBJ is very similar at the median and higher percentile levels apart from the maximum, for all problem classes. This suggests that CBJ’s biggest effect is on the most difficult problems, and that its performance is otherwise similar to chronological backtracking, when “fail-first” dynamic variable ordering is used. We can also see from the sets of plots that the addition of CBJ does significantly reduce the difficulty of the ehps that MAC finds in these populations of problems. We can see from the maximum curves for the MAC-CBJ plots that backjumping greatly moderates the extreme problem behaviour for all of the sparsely constrained problem classes, even for the extremely sparse  $\langle 30, 10, 0.1 \rangle$  populations of problems, for which we still see highly erratic maximum behaviour with MAC. It should be remembered that MAC and MAC-CBJ have been implemented so that they both follow

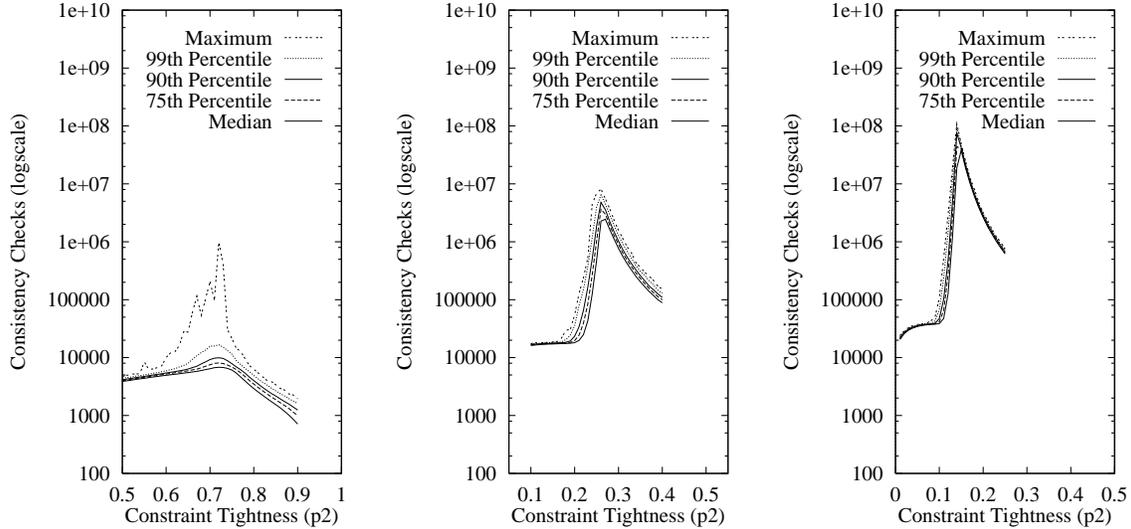


Figure 15: Ranges of consistency checking effort for MAC-CBJ on (left to right)  $\langle 30, 10, 0.1 \rangle$ ,  $\langle 30, 10, 0.5 \rangle$  and  $\langle 30, 10, 1.0 \rangle$  problems.

the same search paths, thus MAC-CBJ must encounter the same subproblems which lead MAC into exceptionally hard search, but clearly deals with them much more quickly than the chronological backtracker can.

These results are similar to those reported in [40], where the effects of adding CBJ to FC were studied. However, the difference in performance between MAC and MAC-CBJ on the non-exceptional problems is less than that between FC and FC-CBJ: in [40] we observed a noticeable improvement with FC-CBJ at the 99% level of behaviour, and at lower levels in some cases, for sparsely constrained classes of problems; we see no noticeable differences in performance at the 99% level when CBJ is added to MAC for the many sparsely constrained problem classes we study here. A likely explanation for this can be observed from the plots of nodes visited by MAC on the sparse problem classes shown in Section 7. We see that the extra look-ahead of MAC eliminates the need for any backtracking during search for a far greater amount of problems than FC, and so backjumping is rendered redundant on many problems. MAC also reduces the amount of backtracking necessary on the remaining problems, and so CBJ still has limited opportunity to become effective.

## 9.2 Comparison with FC-CBJ

Figure 17 shows the median and higher percentiles of consistency checking effort for FC-CBJ on the same  $\gamma \approx 4.92$  problem classes shown for MAC-CBJ in Figure 16, with which it can be directly compared. Once again, it should be noted that fair comparison of performance between MAC-CBJ and FC-

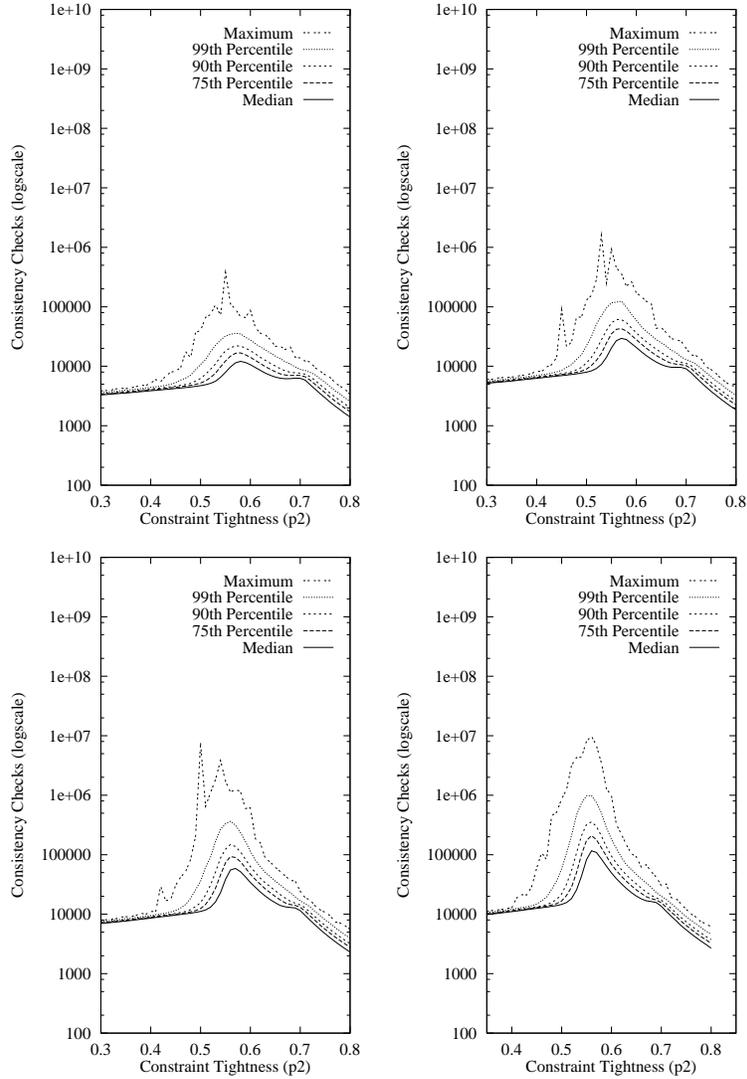


Figure 16: Ranges of consistency checking for MAC-CBJ on (clockwise from top left)  $\langle 20, 10, 0.2579 \rangle$ ,  $\langle 30, 10, 0.1701 \rangle$ ,  $\langle 50, 10, 0.1 \rangle$  and  $\langle 40, 10, 0.1256 \rangle$  problems respectively.

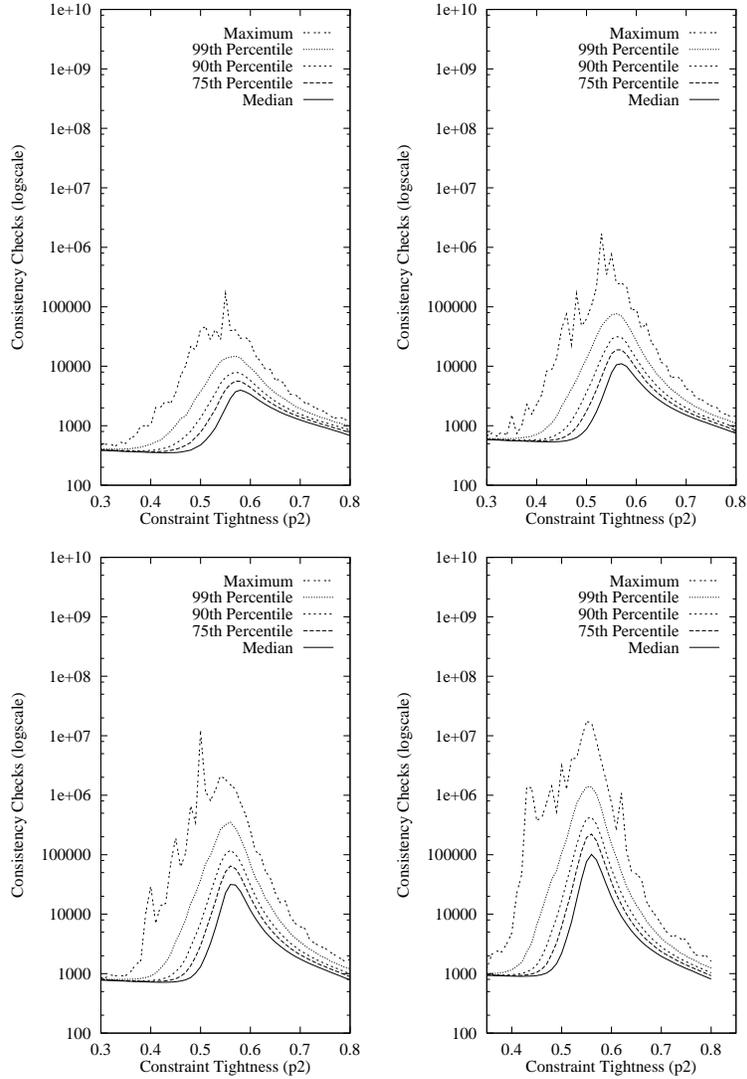


Figure 17: Ranges of consistency checking for FC-CBJ on (clockwise from top left)  $\langle 20, 10, 0.2579 \rangle$ ,  $\langle 30, 10, 0.1701 \rangle$ ,  $\langle 50, 10, 0.1 \rangle$  and  $\langle 40, 10, 0.1256 \rangle$  problems respectively.

CBJ on individual problems is not possible as the algorithms will generally not follow the same search paths, as explained in Section 7.

We see a similar improvement in terms of maximum and median consistency checks by MAC-CBJ over FC-CBJ to that of MAC over FC. In particular, for the maximum curves we see that while FC-CBJ clearly suffers from instances of exceptionally hard problems, the extra look-ahead of MAC combined with the backjumping of CBJ results in clear ehp behaviour being almost eliminated from our populations of sparse problems. However, a small amount of ehp behaviour can still be observed for MAC-CBJ: the spike in the maximum curve for the  $\langle 40, 10, 0.1256 \rangle$  problem class at  $p_2 = 0.50$ , which can be seen in Figure 16, clearly fits the ehp criteria given in Section 3.

These results indicate that while MAC-CBJ, as conjectured in Section 3, shows the most stable performance in respect of the occurrence of exceptionally hard problems, a very few instances can clearly still arise in sparsely constrained problem classes.

## 10 A microscopic study of some ehps

In [40] we studied the behaviour of two problems that forward checking found exceptionally hard, and first reported the insoluble subproblem behaviour mentioned in Section 3. We revisit these problems here, studying the behaviour of MAC when applied to them, and also analyse a problem which MAC itself finds exceptionally hard.

As mentioned earlier, fair comparison of individual FC and MAC searches is difficult as the algorithms do not necessarily follow the same search paths. Clearly this is important when trying to compare the performance of MAC on ehps found by FC. However, a way in which we can “force” MAC to search the same insoluble subproblems as FC does in ehps is to suppress the extra look-ahead of MAC above the necessary search depth, and so reduce MAC to FC until the subproblem is entered. Thus, for example, where FC discovers an extremely hard subproblem after the first four variable instantiations, suppressing MAC look-ahead until search at depths at and below four will force MAC to consider the same subproblem. Each of the ehps that we study here are  $\langle 50, 10, 0.1 \rangle$  CSPs: this is not significant, but as this is the largest class of problems studied, we tend to see the most extreme individual ehp behaviour here.

**Problem 358** at  $p_2 = 0.48$  is an ehp for FC, and it was shown in [40] that the first four instantiations made by the algorithm are  $v_5 = 1^{16}$ ,  $v_{16} = 4$ ,  $v_{22} = 9$  and  $v_{37} = 4$ . It (eventually) becomes clear that this set of assignments

---

<sup>16</sup>i.e. variable 5 is assigned the value 1.

leads to an insoluble subproblem. However, proving insolubility takes more than 79 million consistency checks and 8 million nodes visited; the algorithm frequently finds partial solutions with 38 or more variables instantiated before detecting an infeasibility and backtracking. Once it has been proved that there is no solution to the subproblem, the alternative assignment of  $v_{37} = 10$  is tried and leads almost immediately to a solution, and since only one possible instantiation of the first variable,  $v_5$ , has been tried, it is very likely that this problem has many solutions.

We searched this problem again using MAC from search depth zero (i.e. normal MAC with AC preprocessing), and from depth four (as described above). MAC from depth zero prunes five values from domains during AC preprocessing, and finds a solution after around 177,000 checks, while MAC from depth four finds a solution after around 173,000 checks. It should be noted that MAC from depth zero enters the same subproblem as FC, but deals with it fairly quickly in the same manner as MAC from depth four does. MAC from depth four takes fewer consistency checks than from depth zero only because it is spared the MAC look-ahead effort early in the search, and in fact visits one more node than the full MAC search.

**Problem 898** at  $p_2 = 0.47$  is an ehp for FC, and was also studied in [40]. It was shown that the first four instantiations also lead to an exceptionally difficult subproblem that FC takes over 190 million consistency checks and 56 million nodes visited to prove insoluble. Once again, a solution is found almost immediately once this is established. The behaviour of this ehp appears more subtle on closer examination, however: although it is the fourth assignment that must be re-instantiated to find a solution, almost the entire search is spent searching the subproblem created by the first *nine* instantiations; once this “inner” subproblem is resolved, the search backtracks very quickly to the fourth variable and proceeds to find a solution. This behaviour suggests some sort of “ehp within an ehp” situation, and is not fully understood at present.

This problem was re-researched using MAC from search depths of zero, four and nine. MAC from depth zero finds a solution after 12,268 checks and 50 nodes, MAC from depth four finds a solution after 6917 checks and 51 nodes, while MAC from depth nine finds a solution after 6093 checks and 60 nodes. The first MAC search avoids the ehp conditions completely and the search is backtrack-free. The second search clearly discovers that the “outer” subproblem is arc-inconsistent and performs a single backtrack, while the third search finds the “inner” subproblem and all intermediate subproblems encountered on backtracking to depth three arc-inconsistent, giving a total of ten extra nodes visited. The second and third searches clearly show that MAC can in some cases immediately detect the insoluble

subproblem conditions that FC cannot.

**Problem 4150 at  $p_2 = 0.49$**  is an ehp for MAC, which was mentioned in Section 7 as being the only ehp found for MAC in the  $\langle 50, 10, 0.1 \rangle$  problem class. MAC takes over 1.547 billion consistency checks and 38 million nodes to find a solution, and nearly all of this effort is expended in searching the insoluble subproblem created by the first seven assignments. However, there is a vestige of the behaviour shown by FC on problem 898: MAC appears to find a series of other hard insoluble subproblems within the initial subproblem. This suggests that although the increased look-ahead of MAC appears to be capable of dealing with the “standard” ehp situations that FC gets trapped in, more complex situations can arise for which MAC can also be trapped. This would appear to be what is happening here, although more study is required.

We searched this problem again using MAC-CBJ and FC. The addition of backjumping leads to a solution being found after 158,044 checks, which again demonstrates its benefits on the hardest problems, while FC does not encounter the same subproblem and finds a solution in under 1000 checks.

The results of this fairly brief study of a handful of exceptionally hard problems appear to show that many of the ehps which FC encounters are susceptible to the extra look-ahead of MAC. It seems likely from studying problems 358 and 898, as well as other FC ehps, that many of the insoluble subproblems that FC finds exceptionally hard are simply arc-inconsistent, in which case MAC does not search them at all. There are clearly also cases where the cause of insolubility is more subtle (for instance path-inconsistency), and MAC has to perform some search of the subproblem. Subproblems which MAC itself finds exceptionally hard must have still more complex reasons for inconsistency, although in the example we present here, adding backjumping appears to deal with the problem fairly easily. Thus the value of backjumping in dealing with ehps can be as high for MAC as for FC.

Our understanding of the exceptionally hard problems is still limited, and further extensive study of their “microscopic” behaviour is required.

## 11 Discussion

We have attempted to take two recently presented algorithms and position their performance in terms of the average and extremes of behaviour, by means of rigorous empirical experimentation over a broad range of problem sizes and topologies in a manner that we believe has not been attempted before. In doing so, the findings that we have reported throw up a number of subsidiary issues, which warrant major studies in their own right and so

lie beyond the scope of this paper, but which we briefly discuss a little later in this section.

In studying the general behaviour of the MAC and MAC-CBJ algorithms, and comparing them with the FC and FC-CBJ algorithms, we have placed an emphasis on the number of search nodes visited as a good measure with which to compare performance. We have observed from studying nodes visited that the look-ahead of MAC produces backtrack-free searches on average over a greater range of values of the control parameter than the lesser look-ahead of FC can, for all of the CSP problem classes studied. As might be expected, as the searches become harder the extra look-ahead of MAC also allows the algorithm to visit far fewer search nodes on average than FC, again over all observed problem classes. Looking at consistency checking effort, we see MAC perform poorly compared to FC on problems of all constraint densities on relatively small problems, due to the high arc consistency overheads. However, by studying the independent effects of altering problem size and topology we observe that as problem size increases, the FC consistency checking effort on the hard phase transition problems grows at a greater rate than that for MAC, which becomes the cheaper algorithm on larger problems over these regions. Study of the effects of combining MAC with CBJ show that little benefit is gained, except for the very hardest MAC searches, in a similar fashion to that of combining FC and CBJ. 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. Algorithms should clearly be chosen to suit the problem characteristics, based on the knowledge gained from empirical studies such as those presented here and those presented in [42].

An area for future study is a more detailed investigation of exactly how algorithm performance scales as problem size increases. We have been able to show that MAC performance scales at a better rate than FC as problems become larger, but at present the rates of increase cannot be specified exactly. The application of techniques such as finite size scaling [11] may make this possible in future, and this would constitute a major advance in the development of an “empirical science of algorithms”.

Throughout the main empirical studies that have been conducted, the MAC and MAC-CBJ algorithms have employed “fail-first” dynamic variable ordering, and have maintained arc consistency from search depth zero (i.e. as a preprocessing step, and at every search stage). It is clear that many alternative choices may be made for these aspects of the algorithm specifications, and these may affect their behaviour. A number of alternative static and dynamic variable ordering heuristics have been combined with forward checking algorithms, and study of their use with MAC may be worthwhile. Varying the depth from which we choose to maintain arc consistency may have the most interesting effects on performance: for instance it has been

shown in Section 6 and elsewhere that AC preprocessing has no effect on problems with very loose constraints, and so it would clearly be sensible to only enforce arc consistency from search depth 1, or perhaps lower, on such problems. It should also be remembered that all of our experiments are based on random problems generated according to the model described in Section 2. Problems with more structured constraint graphs, varying domain sizes and/or individual constraint tightnesses may well behave differently. Quite how such changes might affect the performance of the algorithms studied, or the incidence of ehps remains to be studied.

It has been shown in the general studies presented in Section 7 that MAC greatly reduces the incidence of ehps in populations of sparsely constrained problems from that of FC. It is clear from this that employing a more sophisticated form of forward search move is a valid alternative, in terms of dealing with ehps, to the use of a more sophisticated backward search move such as backjumping. We have observed that MAC is capable of immediately recognising the insolubility of the subproblems that FC becomes trapped in, for at least some of the ehps, or can generally prove them to be insoluble more quickly. A very brief study of some individual ehps has been presented in Section 10 in order to demonstrate how the extra look-ahead capability of MAC can deal easily with many situations where the more conservative FC becomes engaged in exceptionally difficult search, and illustrate that despite this, MAC can still be susceptible to ehps. The analysis presented in Section 9 also shows us that even combining sophisticated forward and backward search moves when using MAC-CBJ, small quantities of ehps still exist.

It is clear from these results that the study of exceptionally hard problems requires a great deal more work than that presented here. That ehps should still persist with a complex algorithm such as MAC-CBJ leaves us with two possible approaches to tackling the problem. The first would simply be to employ still more sophisticated forward and backward search moves for algorithms: path consistency or some higher level of consistency could be maintained during search, while more elaborate backjumping algorithms have been described. Although maintaining a higher level of consistency would very likely further reduce ehp incidence, the probable overheads involved would be prohibitively expensive. Baker [1] suggests that *all* exceptionally hard problems can be eliminated by a search strategy employing a sufficiently intelligent backtracker, and presents experiments using dependency-directed backtracking [18] which records nogoods during search. However, Baker admits that this algorithm ‘has an increasing [spatial] overhead as problems get harder’ which once again may be prohibitively expensive.

The second approach to tackling ehps would be to understand precisely the circumstances under which they occur for known search algorithms, and to incorporate knowledge into the algorithms that would enable them to detect ehp situations they entered. Algorithms would then be able to take

very simple measures to change the nature of the search. Such an approach seems eminently sensible: increasing the sophistication of the algorithms in a “sledgehammer” approach may merely be pushing the incidence of ehps beyond the horizon of feasible empirical investigation; whereas by accepting that such anomalies may occur for any complete search algorithm, and learning to recognise and deal with their occurrence, we can become far more confident about dealing with the problem once and for all.

Another area for further research, which would follow the work on AC preprocessing presented in Section 6, and may be highly relevant to the study of ehps, would be the search for a CSP “constraint gap”. This notion of a region of control parameter values for sparsely constrained problems, where the propagation of neither goods or nogoods is effective, and where it is suggested that ehps can occur as a result, is described in Section 6. Such a study would initially involve the investigation of CSP reduction operators, which have been presented by Rossi [33], as a method of propagating definite goods in CSPs. This data could be compared with the nogood propagation of AC preprocessing to establish the location in the control parameter range of any such gaps.

The series of empirical studies that are described in Section 5 represent a major investment of computing power, and only a subset of the results of these studies has been presented here, with an emphasis on showing the behaviour of the MAC and MAC-CBJ algorithms and their comparison with the FC and FC-CBJ algorithms. However, the large body of empirical data that now exists on these algorithms over the broad test bed of random CSP problem classes has many other potential uses: for example, the data has been used with that for some other algorithms on the same problem classes to establish an ehps “hierarchy”, presented in [41].

## Acknowledgments

Stuart Grant is partly supported by a studentship from British Telecom plc. We are extremely grateful to Patrick Prosser and Ian Gent at the University of Strathclyde for their invaluable help and comments.

## References

- [1] A. B. Baker. Intelligent backtracking on the hardest constraint problems. Technical report, CIRL and DCIS, University of Oregon, United States, 1995.
- [2] C. Bessiere and J.-C. Regin. An arc consistency algorithm optimal in the number of constraint checks. In M. Meyer, editor, *Proceedings of the ECAI-94 Workshop on Constraint Processing*, pages 9–16, Aug. 1994.

- [3] B. Bollobas. *Random Graphs*. Academic Press, 1985.
- [4] J. E. Borrett and E. P. K. Tsang. Observations on the usefulness of arc consistency preprocessing. Technical Report CSM-236, Department of Computer Science, University of Essex, UK, Feb. 1995.
- [5] P. Cheeseman, B. Kanefsky, and W. Taylor. Where the Really Hard Problems are. In *Proceedings IJCAI-91*, volume 1, pages 331–337, 1991.
- [6] J. M. Crawford and L. D. Auton. Experimental Results on the Crossover Point in Satisfiability Problems. In *Proceedings of AAAI93*, pages 21–27, 1993.
- [7] A. Davenport and E. P. K. Tsang. An empirical investigation into the exceptionally hard problems. Technical Report CSM-239, Department of Computer Science, University of Essex, U.K., 1995.
- [8] R. Dechter and J. Pearl. Network-based heuristics for constraint-satisfaction problems. *Artificial Intelligence*, 34:1–38, 1988.
- [9] J. Gaschnig. A constraint satisfaction method for inference making. In *Proceedings of the 12th Annual Allerton Conference on Circuit and System Theory*, University of Illinois, Urbana-Champaign, USA, Oct. 1974.
- [10] J. Gaschnig. Performance measurement and analysis of certain search algorithms. Technical Report CMU-CS-79-124, Carnegie-Mellon University, Pittsburgh USA, 1979.
- [11] I. P. Gent, E. MacIntyre, P. Prosser, and T. Walsh. Scaling Effects in the CSP Phase Transition. In U. Montanari and F. Rossi, editors, *Proceedings CP-95*, pages 70–87. Springer-Verlag, Sept. 1995.
- [12] I. P. Gent and T. Walsh. Easy Problems are Sometimes Hard. *Artificial Intelligence*, 70:335–345, 1994.
- [13] I. P. Gent and T. Walsh. The Hardest Random SAT Problems. In B. Nebel and L. Dreschler-Fischer, editors, *Proceedings KI-94: Advances in Artificial Intelligence. 18th German Annual Conference on Artificial Intelligence*, pages 355–366. Springer-Verlag, 1994.
- [14] I. P. Gent and T. Walsh. The SAT phase transition. In *Proceedings ECAI-94*, pages 105–109, 1994.
- [15] I. P. Gent and T. Walsh. The Satisfiability Constraint Gap. Research Paper 702, Department of A.I., University of Edinburgh, 1994. To appear in the AI Journal special issue on Phase Transitions in Problem Spaces.

- [16] I. P. Gent and T. Walsh. Computational Phase Transitions from Real Problems. In *Proceedings ISAI-95, Mexico (to appear)*, Oct. 1995.
- [17] I. P. Gent and T. Walsh. The TSP Phase Transition. Technical Report 178-95, Department of Computer Science, University of Strathclyde, UK, 1995.
- [18] M. Ginsberg. Dynamic Backtracking. *Journal of A.I. Research*, 1:25–46, 1993.
- [19] S. W. Golomb and L. D. Baumert. Backtrack Programming. *Journal of the ACM*, 12:516–524, 1965.
- [20] R. Haralick and G. Elliott. Increasing tree search efficiency for constraint satisfaction problems. *Artificial Intelligence*, 14:263–313, 1980.
- [21] T. Hogg and C. P. Williams. The Hardest Constraint Problems: A Double Phase Transition. *Artificial Intelligence*, 69:359–377, 1994.
- [22] S. Kirkpatrick and B. Selman. Critical Behaviour in the Satisfiability of Random Boolean Expressions. *Science*, 264:1297–1301, 1994.
- [23] G. Kondrak and P. van Beek. A Theoretical Evaluation of Selected Backtracking Algorithms. In C. S. Mellish, editor, *Proceedings IJCAI-95*, volume 1, pages 541–547. Morgan Kaufmann, Aug. 1995.
- [24] A. Mackworth. Consistency in networks of relations. *Artificial Intelligence*, 8:99–118, 1977.
- [25] D. Mitchell, B. Selman, and H. Levesque. Hard and Easy Distributions of SAT Problems. In *Proceedings AAAI-92*, pages 459–465, 1992.
- [26] R. Mohr and T. Henderson. Arc and path consistency revisited. *Artificial Intelligence*, 28:225–233, 1986.
- [27] B. Nadel. Constraint satisfaction algorithms. *Comput. Intell.*, 5:188–224, 1989.
- [28] P. Prosser. An empirical study of phase transitions in binary constraint satisfaction problems. Technical Report AISL-49-93, Department of Computer Science, University of Strathclyde, 1993. To appear in the AI Journal special issue on Phase Transitions in Problem Spaces.
- [29] P. Prosser. Hybrid Algorithms for the Constraint Satisfaction Problem. *Computational Intelligence*, 9(3):268–299, 1993.

- [30] P. Prosser. Binary constraint satisfaction problems: some are harder than others. In A. Cohn, editor, *Proceedings of ECAI-94*, pages 95–99. Wiley, 1994.
- [31] P. Prosser. MAC-CBJ: maintaining arc consistency with conflict-directed backjumping. Technical Report 95-177, Department of Computer Science, University of Strathclyde, UK, May 1995.
- [32] J.-F. Puget. A C++ Implementation of CLP. In *Proceedings of SPICIS94 (Singapore International Conference on Intelligent Systems)*, 1994.
- [33] F. Rossi. Redundant Hidden Variables in Finite Domain Constraint Problems. In M. Meyer, editor, *Proceedings of the ECAI94 Workshop on Constraint Processing*, pages 17–25, 1994.
- [34] D. Sabin and E. Freuder. Contradicting Conventional Wisdom in Constraint Satisfaction. In A. Cohn, editor, *Proceedings ECAI94*, pages 125–129, 1994.
- [35] B. M. Smith. In Search of Exceptionally Difficult Constraint Satisfaction Problems. In M. Meyer, editor, *Proceedings of the ECAI'94 Workshop on Constraint Processing*, pages 79–86, Aug. 1994.
- [36] B. M. Smith. In Search of Exceptionally Difficult Constraint Satisfaction Problems. Research Report 94.2, School of Computer Studies, University of Leeds, Jan. 1994.
- [37] B. M. Smith. Phase Transition and the Mushy Region in Constraint Satisfaction Problems. In A.G.Cohn, editor, *Proceedings ECAI-94*, pages 100–104. Wiley, 1994.
- [38] B. M. Smith and M. E. Dyer. Locating the Phase Transition in Constraint Satisfaction Problems. *To appear in Artificial Intelligence*, 1995.
- [39] B. M. Smith and S. A. Grant. Sparse Constraint Graphs and Exceptionally Hard Problems. Research Report 94.36, School of Computer Studies, University of Leeds, Dec. 1994.
- [40] B. M. Smith and S. A. Grant. Sparse Constraint Graphs and Exceptionally Hard Problems. In C. S. Mellish, editor, *Proceedings IJCAI-95*, volume 1, pages 646–651. Morgan Kaufmann, Aug. 1995.
- [41] B. M. Smith and S. A. Grant. Where the Exceptionally Hard Problems Are. In *Proceedings of the CP-95 Workshop on Studying and Solving Really Hard Problems*, pages 172–182. Laboratoire d'Informatique de Marseille, France, Sept. 1995.

- [42] E. P. K. Tsang, J. Borrett, and A. C. M. Kwan. An attempt to map the performance of a range of algorithm and heuristic combinations. In J. Hallam, editor, *Proceedings AISB-95*, pages 203–216. IOS Press, Amsterdam, 1995.
- [43] P. van Beek. On the Inherent Level of Local Consistency in Constraint Networks. In *Proceedings AAAI-94*, 1994.
- [44] R. J. Wallace. Why AC-3 is Almost Always Better than AC-4 for Establishing Arc Consistency in CSPs. In *Proceedings IJCAI93*, volume 1, pages 239–245, 1993.
- [45] C. Williams and T. Hogg. Using Deep Structure to Locate Hard Problems. In *Proceedings AAAI-92*, pages 472–477, 1992.
- [46] C. Williams and T. Hogg. Extending Deep Structure. In *Proceedings of AAAI-93*, pages 152–158, 1993.
- [47] C. Williams and T. Hogg. Exploiting the Deep Structure of Constraint Problems. *Artificial Intelligence*, 70:73–117, 1994.