

An Attempt to Map the Performance of a Range of Algorithm and Heuristic Combinations

Edward P. K. Tsang, James E. Borrett, & Alvin C. M. Kwan
Department of Computer Science
University of Essex
Wivenhoe Park
Colchester CO4 3SQ
United Kingdom
email: {edward, jborrett, alvin}@essex.ac.uk

Abstract. Constraint satisfaction is the core of many AI and real life problems and much research has been done in this field in recent years. Work has been done in the past on comparing the performance of different algorithms and heuristics. Much of such work has focused on finding "the best" algorithm and heuristic combination for all problems.

The objective of this paper is to prove that there is no universally best algorithm and heuristic for all problems -- different problems can be solved most efficiently by different algorithm and heuristic combinations. The implication of this is important because it means that instead of trying to find "the best" algorithms and heuristics, future research should try to identify the application domain of each algorithm and heuristic (i.e. when they are most effective). Furthermore our results point to future research which focuses on how to retrieve the most efficient algorithm for a given problem. The results in this paper provide a first step towards achieving such goals.

1. Introduction

The constraint satisfaction problem (CSP) is central to many real-life problems and to many fields of artificial intelligence. The main task in solving a CSP is the assignment of values to a set of variables given a set of constraints on those variables (see [1] for a formal definition of the CSP). The importance of finding effective methods for solving CSPs has led to a significant increase in the amount of research effort in the field in recent years.

One of the products of this research has been the development of many new algorithms and heuristics for improving the efficiency of CSP solving. These algorithms and heuristics have been compared and contrasted by many researchers, with a view to analysing their relative performance as in [2], [3], [4], [5].

Occasionally work has focused on attempting to identify the best performing algorithm and heuristic combination for a specific set of problems. Some of this work has then gone on to claim that they have identified the best algorithm and heuristic combination but qualifying this with the caveat that it is only the best combination for "the problem set used". For

example in [5] Prosser identifies the forward checking with conflict-directed backjumping as the best algorithm for solving the Zebra problem, although he goes on to give examples of other possible CSPs where this may not be the case.

Many researchers now choose to use forward checking based combinations in their experiments because it seems that they are the best overall, especially in the *hard* region [6]. For example in [7] Smith did much work on the hard region using forward checking with the fail-first principle and Prosser did work on the hard region using a forward checking hybrid in [8].

Despite the work mentioned above, no significant work has been done on actually testing the conjecture being hinted at in many publications – i.e. either:

- 1 - the algorithm identified as being the best for the chosen problem set *is* the best overall
- or
- 2 - the algorithm identified as being the best for the chosen problem set *is not* always the best and that other algorithms would be better suited to other problem sets

Either way, this is an important point that needs to be considered. Studying the relative performance of algorithms and heuristics is of limited value unless we can identify certain characteristics for that problem set. If algorithms and heuristics are compared over problem sets which can be characterised then it should be possible to go on and claim that a particular combination is the best choice, for example, when tackling “hard problems”.

There has also been some recent work on the pitfalls of particular algorithms and on adverse interactions between some algorithms and heuristics [9], [10], [11]. This work has given more evidence to the notion that there is much that we do not yet fully understand the interactions between different algorithms and heuristics, where they perform best, and which one is the best, over the infinite range of possible CSPs. These new findings, however, should be treated with caution. For example, Prosser [9] has identified potential problems with the combination of intelligent back-trackers and consistency techniques in the form of directional k-consistency (DKC) whereby the phenomena of the “Long Jump” and the “Bridge” manifest themselves. The problem with this finding is how significant is it over the range of CSPs, and how can someone make an informed decision on the use of an intelligent backtracker with consistency techniques without full knowledge of the prevalence and significance of it?

In this paper we describe a series of experiments which were carried out in an attempt to map out the performances of different algorithms and heuristic combinations over a wide range of problem specifications for randomly generated binary CSPs. We believe that it is misleading and indeed wrong to claim that there is a universally “best” algorithm and it was our intention with these experiments to demonstrate this.

The range of algorithms used include chronological backtracking (BT) [1], backjumping (BJ) [12], backmarking (BM) [13], conflict-directed backjumping (CBJ) [5], BM with CBJ (BM_CBJ) [5], forward checking (FC) [2], directional arc-consistency lookahead (DAC) [19], [20] and FC with CBJ (FC_CBJ) [5]. The range of heuristics used was natural lexical ordering (NAT), minimum width ordering (MWO)[14], m-ordering (MO) [15] and the fail first principle (FFP) [2]. Some work was also done with pre-processing, where the AC6 [16] algorithm was used.

After running our experiments a map was constructed, showing the best performing combinations at various points in the binary CSP problem space. This map confirmed our

belief that there is no universally best combination for the set of algorithms, heuristics and problems considered.

The implication of these results is important because it means that instead of trying to find "the best" algorithms and heuristics, future research should try to identify the application domain of each algorithm and heuristic (i.e. when they are most effective). Furthermore our results point to future research which focuses on how to retrieve the most efficient algorithm for a given problem.

The structure of the rest of this paper will be as follows. Section 2 discusses the different algorithms and heuristics used in our experiments. Section 3 outlines the experiments that we undertook. In section 4 we present our results and in section 5 these results are analysed. Finally our conclusions are given in section 6 and plans for further work in section 7.

2. Combinations of Algorithm and Heuristic

As outlined in the previous section, there has been a large selection of algorithms and heuristics developed in recent years. For the purposes of our experiments, we wanted to use the combination of a wide range of well known algorithms and heuristics. We also decided to focus on complete algorithms, whereby they will always find a solution if it exists.

The algorithms we chose were as follows;

BT	- Chronological Backtracking
BJ	- Backjumping
CBJ	- Conflict-Directed Backjumping
BM	- Backmarking
BM_CBJ	- Backmarking with Conflict-Directed Backjumping
FC	- Forward Checking
FC_CBJ	- Forward Checking with Conflict-Directed Backjumping
DAC	- Directional Arc Consistency Lookahead

The ordering heuristics we used were;

NAT	- Natural Lexical Ordering
MWO	- Minimum Width Ordering
MO	- M-Ordering
FFP	- Fail First Principle

We experimented with these algorithm-heuristic combinations both with and without preprocessing. The preprocessing algorithm used was the AC6 algorithm.

We acknowledge that this selection of algorithms is by no means exhaustive, for example solution synthesis algorithms have not been included, however, the aim of our experiments is to determine whether or not there is a "best" combination of algorithm and heuristic from this given selection as a starting point. This is a reasonable selection since it includes the FC hybrids that have been seen to be effective on the past [2], [5].

3. Experimental Details

Most research done on the comparison of algorithms and heuristics has concentrated on binary CSPs. One of the reasons for this is because using random binary CSPs makes is

possible to control certain characteristics of the problems and hence the design of controlled experiments is simplified.

We have focused on randomly generated binary CSPs (BCSPs) for our experiments. This does not detract from the aims of our experiments, of showing that there is no one best algorithm for all CSPs, since if we can show that there is no one best algorithm for the class of BCSPs then it follows that the same must be true for all CSPs since this is a small fraction of the problem space.

3.1 Problem Generation

When experimenting with random BCSPs it has become almost a standard to generate problem instances using the characterisation by the tuple $\langle n, m, p1, p2 \rangle$ where n is the number of variables in the problem, m is the domain size for all variables in the problem, $p1$ is a measure of constraint density within the CSP and $p2$ is a measure of the tightness of the individual constraints. The precise definitions of $p1$ and $p2$ have varied slightly for different researchers in particular for $p2$. For our experiments we have used the same definition for $p2$ as in [2] and [17]. A summary of the parameters we used for our particular generator are given below;

n	the number of variables
m	the uniform domain size for all variables
$p1$	the probability that any two variables are constrained
$p2$	the probability of any possible 2-compound label (assignment of values to variables) being legal in a binary constraint

Our problem generator also ensured that the constraint graph it generated was a connected graph, and that no more than one constraint is permitted between any two variables¹.

To represent constraints we used the binary matrix relation where the rows and columns represent the values in the domains of the two variables involved in the constraint. The feasibility of a particular 2-compound label between these variables was denoted by a “1” for compatible and a “0” for incompatible, as in [17].

Using the above system for problem generation, a problem involving n variables has $n \cdot (n-1)/2$ possible unique constraints. Our generator selects $p1 \cdot n \cdot (n-1)/2$ of these constraints at random, with the proviso that $n-1$ are initially selected to ensure that the graph was connected. The compatibility of values with the constraint matrices was achieved by randomly selecting $p2 \cdot m \cdot m$ labels to be “1”, whilst making the remaining incompatible labels “0”. As can be seen from our definition of $p2$, the higher $p2$, the higher the number of “1” entries in the constraint matrices.

3.2. Range of Problems Used

One of the aims of this series of experiments was to investigate the performances of a wide range of algorithms and heuristics over a wide range of problem specifications. An effective way to cover a wide range of problems is to vary the values of the $p1$ and $p2$ characteristics, whilst fixing the number of variables used and their domain sizes.

¹ An unconnected constraint graph can be more efficiently tackled by using divide and conquer[1].

The domain size, m , was fixed at 10 for all our experiments and the number of variables used, n , was either 10 or 20. This meant that we could carry out experiments on a wide range of algorithms which included some which we suspected would take very long execution times (for example BT combinations). Having chosen these values we then varied the values of p_1 and p_2 over the ranges 0.1, 0.2, 0.3, ... 1.0 for p_1^2 and 0.05, 0.10, 0.15, ... 0.95 for p_2 . This gave us a total of 323 problems specifications spread out evenly throughout the p_1 - p_2 space. In fact by doing this we were, as will be seen later, able to chart the map of p_1 - p_2 space for randomly generated binary CSPs for both 10 and 20 variables and with a fixed m of 10.

For each of the cells in the p_1 - p_2 -space, 50 CSPs were generated for that specification. This number of samples is large enough for conclusions to be made by using mean values of given parameters with a reasonable level of confidence. This led to a total of more than 1000000 runs of problem solving.

It should be noted that both soluble and insoluble problems were considered. The results for these were not separated and the measures quoted here are a mixture of runs where a solution was found or where it was found that no solution exists. This is a valid approach since we wish to identify which combinations are best on average for the type of problem presented to it.

3.3 Algorithm and Heuristic Combinations Used

The complete set of algorithms used were as shown in table 1. In this table, the numbers in the cells indicate the experiments where the combinations were used (i.e. 10 shows used for the 10 variable case and 20 shows for the 20 variable case).

	NAT	MO	MWO	FFP
BT	10	10	10	
BM	10	10, 20	10, 20	
BJ	10	10, 20	10, 20	
CBJ	10	10, 20	10, 20	
BM_CBJ	10	10, 20	10, 20	
FC	10	10, 20	10, 20	10, 20
FC_CBJ	10	10, 20	10, 20	10, 20
DAC	10	10, 20	10, 20	10, 20

Table 1. - Algorithms and Heuristics Used

There are some omissions from table 1. These are for combinations which were unlikely to give us more information, or shown to be inefficient. For example, the omissions from the FFP column are there since for non-lookahead algorithms the FFP heuristic is equivalent to the NAT ordering³. Where combinations only occur for the 10 variable experiments, these combinations were shown to be ineffective after the 10 variable experiments had been evaluated and hence they were not run for the 20 variable case to save experimentation time.

² In fact $p_1=0.1$ was not used for the case of 10 variables since this would not guarantee a connected graph.

³ The scenario where FFP would be useful for static orderings is where domain sizes are not equal. In this case FFP might give us a useful initial ordering. All of our experiments used CSPs where the initial domain sizes are all the same, except after preprocessing.

4. Results

There are several ways in which the performance of algorithms can be measured. The most popular in recent research has been the measure of compatibility checks. This is an implementation independent measure of search effort. The other measure commonly used is that of the cpu time used in solving the problem, which can depend on the efficiency of the particular implementation. The choice of measure can affect the validity of any claim that algorithm and heuristic combination A is better than combination B.

We have looked at the measure of compatibility checks required to find the first solution for our analysis mainly because the cpu times for many of the experiments, especially very easy problems, were too small to draw many conclusions. However one feature that was apparent when considering cpu time was a suggestion that a slightly different map can be expected.

An immense amount of data was generated by our experiments. This presented us with the problem of how to present it in a useful and informative way such that the essence of our findings could be easily seen. We decided to opt for two ways of achieving this which are outlined in the following sections.

4.1 Mapping the Algorithms Over p1-p2 Space

One simple way to summarise the vast amount of data collected is to plot the outcome of the experiments as a map. This map would consist of a plot of the best performing algorithm for each of the p1-p2 cells, based on a ranking of each algorithm heuristic combination using the mean number of compatibility checks for finding the first solution, or for proving that no solution exists.

Two maps were generated and these are shown in the figure 1 and figure 2. They show the best performing combination(s) for each cell. In order to identify the best combination for any given cell we had to use a threshold by which the next ranked combination was outperformed. We chose an arbitrary threshold of 2.5% and up to three top ranking combinations are identified for each cell. Where this cannot be achieved a * is displayed.

The key to the map is as follows;

ALGORITHM + ORDERING

where the algorithms indicated on the map are as follows;

A = BM_CBJ	D = FC
B = BJ	E = FC_CBJ
C = CBJ	

and the heuristics are as defined in section 2. Note that only a selection of the total range of combinations appear as the best ranked algorithm for a particular p1-p2 value.

4.2 Sections Across the p1-p2 Terrain

The maps shown in figures 1 and 2 are useful in demonstrating an overall picture of the results of our experiments, however it is also interesting to look at some of the detail of what is happening to the relative ranking of the algorithms as we move across the p1-p2 terrain.

P1 / P2	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
0.05	A + MO C + MO	A + MWO	A + MWO	A + MWO A + MO	A + MWO A + MO	A + MWO A + MO	A + MWO A + MO	A + MO A + MWO	A + MO A + MWO A + NAT
0.10	A + MO C + MO	A + MWO	A + MWO	A + MWO	A + MWO A + MO	A + MO	A + MWO A + MO	A + MO A + MWO	A + MO A + MWO A + NAT
0.15	A + MO C + MO	A + MWO	A + MWO	A + MWO	A + MWO A + MO	A + MWO	A + MWO	A + MO A + MWO	A + MO A + MWO A + NAT
0.20	A + MWO C + MWO	A + MWO	A + MWO	A + MWO	A + MWO	A + MWO	A + MO A + MWO	A + MO A + MWO	A + MO A + MWO A + NAT
0.25	A + MWO C + MWO B + MWO	A + MWO	A + MWO	A + MWO	A + MWO	A + MWO	A + MWO	A + MO	A + MO A + MWO A + NAT
0.30	A + MO C + MO B + MO	A + MWO	A + MWO	A + MWO	A + MWO	A + MWO	A + MWO A + MO	A + MO A + MWO	A + MO A + MWO A + NAT
0.35	*	A + MWO	A + MWO	A + MWO	A + MWO	A + MWO	A + MWO	A + MO A + MWO	A + MWO A + NAT
0.40	*	A + MWO	A + MWO	A + MWO	A + MWO	A + MWO	A + MWO	A + MWO A + MO	A + MO
0.45	*	A + MO C + MO B + MO	A + MWO	A + MWO	A + MWO A + MO	A + MWO	A + MWO	A + MO A + MWO	A + MO A + MWO A + NAT
0.50	*	A + MWO C + MWO B + MWO	A + MWO	A + MWO	A + MWO	A + MWO	A + MWO	*	E + FFP D + FFP
0.55	*	*	A + MWO	A + MO	A + MWO	A + MWO	E + FFP D + FFP	E + FFP D + FFP	E + FFP D + FFP
0.60	*	*	A + MO C + MO	A + MO	A + MO	A + MWO	E + FFP D + FFP	E + FFP D + FFP	E + FFP D + FFP
0.65	*	*	*	A + MO	A + MWO	A + MWO	A + MO	E + FFP D + FFP	E + FFP D + FFP
0.70	*	*	*	*	A + MO	A + MWO	A + MWO A + MO	A + MO	E + FFP D + FFP
0.75	*	*	*	*	*	*	A + MWO	A + MO A + MWO	A + MO
0.80	*	*	*	*	*	*	A + MWO	A + MO	A + MWO A + NAT
0.85	*	*	*	*	*	*	*	A + MWO	A + MO A + MWO A + NAT
0.90	*	*	*	*	*	*	*	*	*
0.95	*	*	*	*	*	*	*	*	*

Figure 1 - Map of best performing algorithm-heuristic-preprocessing combination based on the mean compatibility checks for first solution with $n=10$ and $m=10$. Shaded cells show the approximate position of the *hard* region.

P1 / P2	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
0.05	A+MO C+MO	A+MWO	A+MWO	A+MWO A+MO	A+MO A+MWO	A+MWO A+MO	A+MWO A+MO	A+MO A+MWO	A+MO A+MWO	A+MO A+MWO
0.10	*	A+MWO	A+MWO	A+MWO	A+MO A+MWO	A+MWO A+MO	A+MO A+MWO	A+MO A+MWO	A+MWO A+MO	*
0.15	A+MO C+MO	A+MWO	A+MWO	A+MWO A+MO	A+MWO	A+MWO	A+MWO A+MO	A+MWO A+MO	A+MWO	A+MO A+MWO
0.20	A+MWO C+MWO C+MO	A+MWO	A+MWO	A+MWO	A+MWO	A+MWO A+MO	A+MO A+MWO	A+MO A+MWO	A+MO	*
0.25	A+MWO C+MWO	A+MWO	A+MWO	A+MWO	A+MWO	A+MWO	A+MO A+MWO	A+MWO	A+MWO A+MO	A+MO
0.30	A+MWO C+MO C+MWO	A+MWO	A+MWO	A+MWO	A+MWO	A+MWO A+MO	A+MWO A+MO	A+MO A+MWO	A+MO A+MWO	*
0.35	A+MO C+MO B+MO	A+MWO	A+MWO	A+MWO	A+MWO	A+MWO	A+MWO	A+MO	A+MO A+MWO	A+MWO
0.40	*	A+MWO	A+MWO	A+MWO	A+MWO	A+MWO	A+MWO	A+MO	A+MO A+MWO	*
0.45	*	A+MWO	A+MWO	A+MWO	A+MWO	A+MWO	A+MWO	A+MWO A+MO	A+MWO A+MO	A+MO A+MWO
0.50	*	A+MWO	A+MWO	A+MWO	A+MWO	A+MWO	A+MWO	A+MWO	A+MWO A+MO	*
0.55	*	A+MO A+MWO	E+FFP	A+MWO	A+MO	A+MWO	A+MWO	A+MWO	A+MWO	A+MO
0.60	*	A+MO	A+MO	E+FFP D+FFP	A+MWO	A+MWO	A+MWO	E+FFP D+FFP	E+FFP D+FFP	E+FFP D+FFP
0.65	*	*	A+MWO	E+FFP D+FFP	E+FFP D+FFP	E+FFP D+FFP	E+FFP D+FFP	E+FFP D+FFP	E+FFP D+FFP	E+FFP D+FFP
0.70	*	*	A+MWO A+MO	A+MO	E+FFP D+FFP	E+FFP D+FFP	E+FFP D+FFP	E+FFP D+FFP	E+FFP D+FFP	E+FFP D+FFP
0.75	*	*	A+MO C+MO B+MO	A+MO	A+MWO	E+FFP D+FFP	E+FFP D+FFP	E+FFP D+FFP	E+FFP D+FFP	E+FFP D+FFP
0.80	*	*	*	A+MWO	A+MO	A+MWO	E+FFP D+FFP	E+FFP D+FFP	E+FFP D+FFP	E+FFP D+FFP
0.85	*	*	*	*	*	A+MO	A+MO	A+MWO	E+FFP D+FFP	E+FFP D+FFP
0.90	*	*	*	*	*	*	A+MWO	A+MWO	A+MO	A+MWO
0.95	*	*	*	*	*	*	*	*	*	*

Figure 2 - Map of best performing algorithm-heuristic-preprocessing combination based on the mean compatibility checks for first solution with $n=20$ and $m=10$. Shaded cells show the approximate position of the *hard* region.

Mean Compatibility Checks for P1 = 0.3

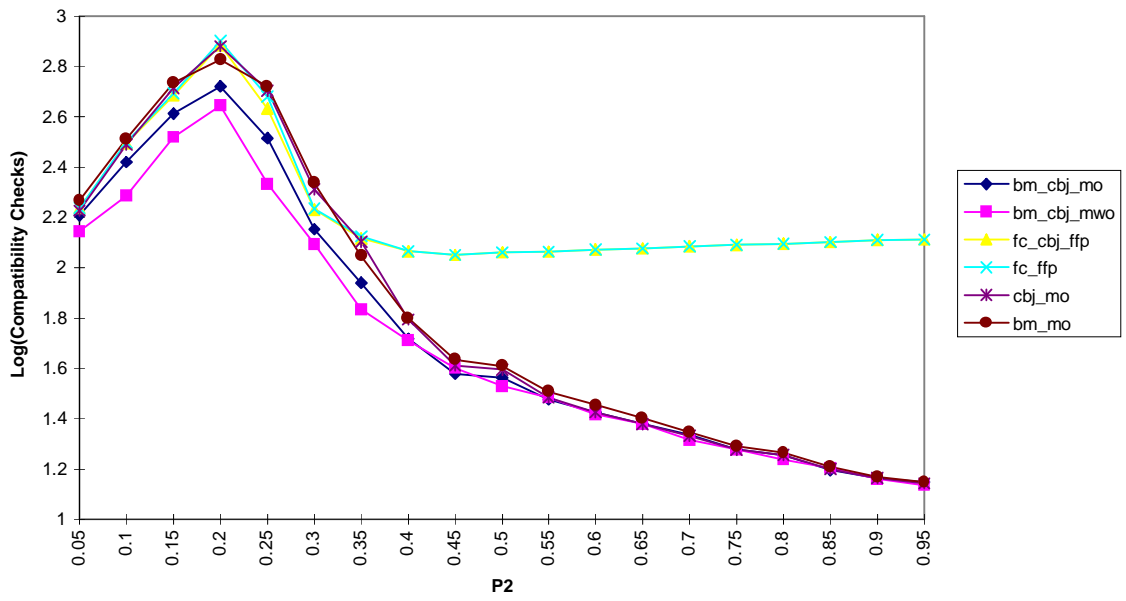


Figure 3 - Cross section for P1 = 0.3, n = 10 and m = 10

Mean Compatibility Checks for P1 = 0.7

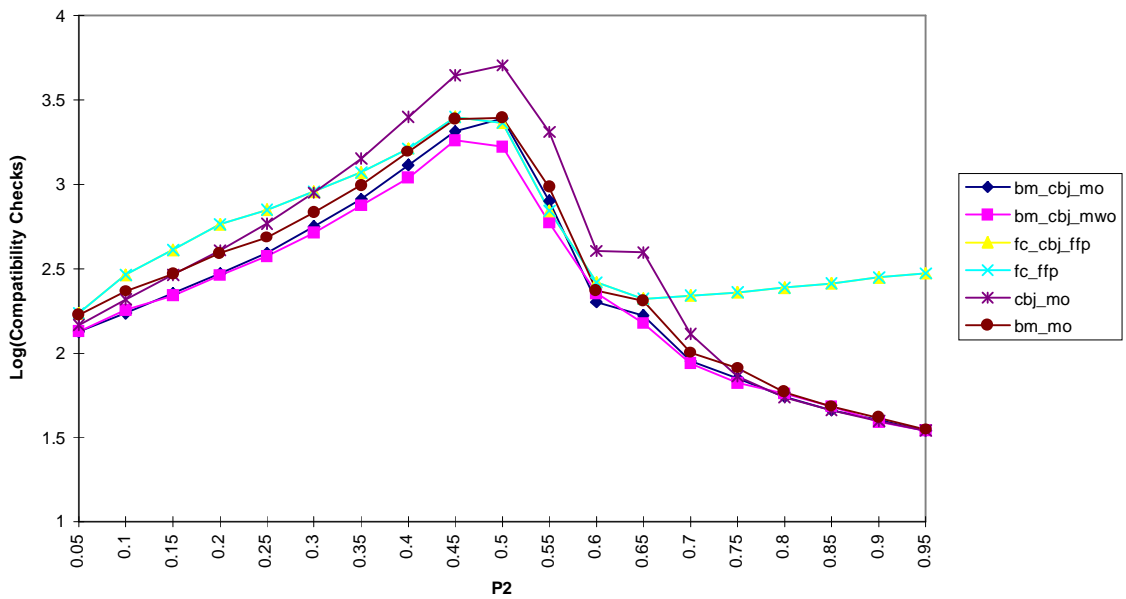


Figure 4 - Cross section for P1 = 0.7, n = 10 and m = 10

Mean Compatibility Checks for P1 = 0.3

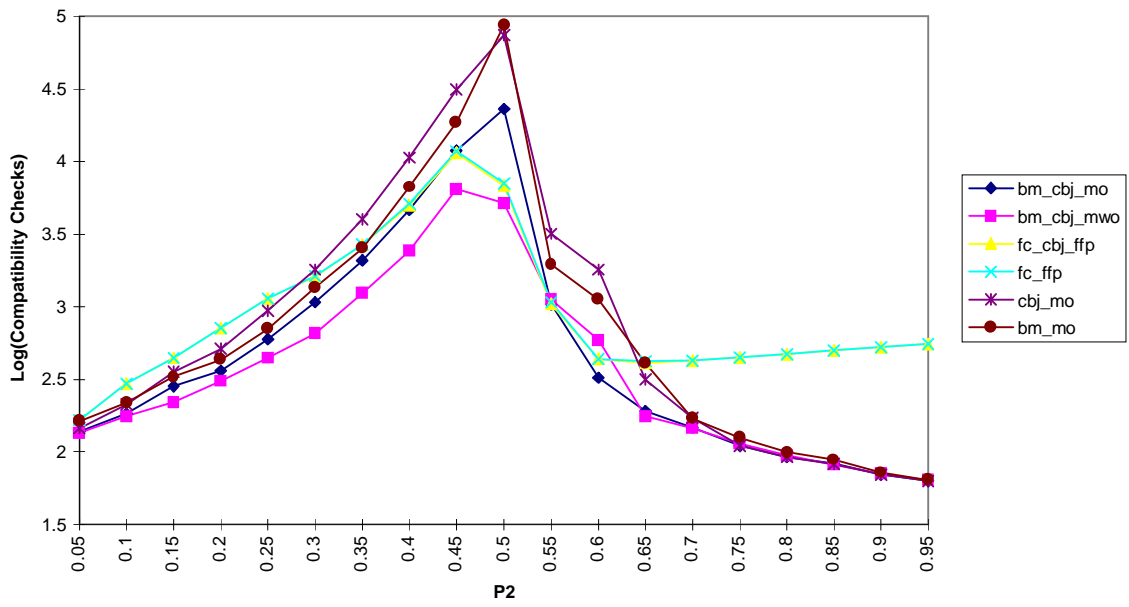


Figure 5 - Cross section for P1 = 0.3, n = 20 and m = 10

Mean Compatibility Checks for P1 = 0.7

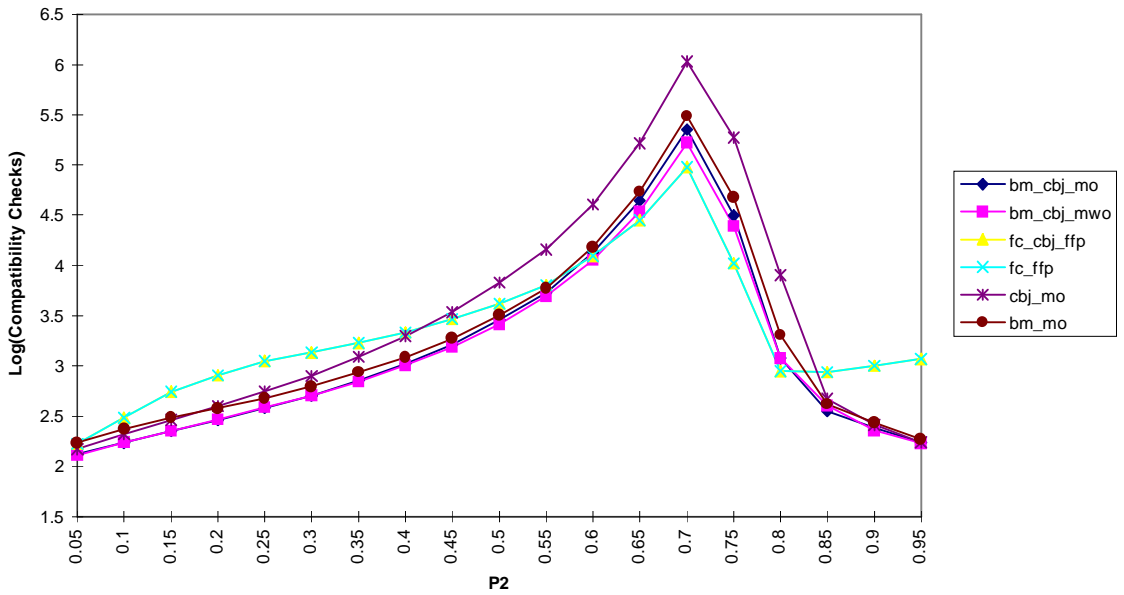


Figure 6 - Cross section for P1 = 0.7, n = 20 and m = 10

Figures 3 - 6 show cross sections through p_1 - p_2 space, where p_1 is held constant and p_2 is varied. For each of the cross sections, the top performing combinations are shown and also a combination for each of the basic non-lookahead algorithms BM and CBJ. This gives us a total of 6 combinations.

5. Analysis

The results presented provide us with some interesting and significant findings. The most obvious of these is that regions of dominance of a particular algorithm and heuristic combination do exist. We can see from both of the maps that there is a wide area of domination for the BM_CBJ algorithm with either the MO or MWO ordering heuristic. This is evident for both the 10 and 20 variable case. Similar regions appear where FC_CBJ + FFP or FC + FFP are dominant and there is a major region where there appears to be no outright best combination.

The fact that FC hybrids do not dominate the whole of the hard region might be considered to be somewhat surprising by many researchers since, as mentioned in section 1, it has become common practice to use FC based combinations when investigating the hard region on the basis that this is an efficient choice. This may be the case for many hard problems but our results suggest that BM_CBJ with MWO or MO is also a good candidate in some hard problems. Prosser identifies BM_CBJ to be a high ranking algorithm in his work on hybrid algorithms [5] but he effectively only uses a random variable ordering and only with the Zebra problem. This shows the importance of assessing the performance of a combination of algorithms and ordering heuristics, and over a wider range of problems.

Prosser also identifies possible scenarios where the combination of BM and CBJ can result in a worse performance than BM. However in our experiments, when considering the average case, the performance of BM_CBJ combination is found to be consistently better than corresponding BM combinations⁴⁵. It could be argued that the average performance is more significant, when considering the choice of algorithm for a particular problem type.

Another interesting feature of the maps in figures 1 and 2 is that the size of the regions varies with the number of variables in the CSP. This is evident if we look at the region where FC_CBJ + FFP dominates the map. It appears that when 20 variables are used, this region extends along the hard region and at the same time it broadens from the equivalent 10 variable region. This is significant since it suggests that FC based algorithms may become increasingly applicable as the number of variables increases, in terms of compatibility checks.

A notable feature of the two p_1 - p_2 maps is that combinations using preprocessing (in this case AC6) do not appear. This phenomena seems to occur because despite the fact that AC6 did improve the combined performance in some instances, these did not occur when in regions favourable to that particular combination. This is significant since it suggests that AC6 (and possibly arc-consistency preprocessing in general) may not be useful for solving random BCSPs when a large selection of alternative algorithms and heuristics exists.

When we look at the cross sections of the p_1 - p_2 -space (figures 3-6) we can see other interesting features. For example, it appears that for the average case the hard region is found in the same place for all the algorithms tested. This gives more support to the notion that the phase transition for solving CSPs is associated with the problems themselves and not with the different problem solving methods used [7], [8]. We acknowledge that we have only

⁴ Prosser also finds that BM_CBJ performs well for the average case.

⁵ More recently in [18] a new version of BM_CBJ has been proposed which avoids the disadvantages of using BM together with CBJ.

used a granularity of 0.05 for the variation in p_2 but there does appear to be a broad agreement between the results for the different algorithms.

The sections also demonstrate how the dominance of a particular algorithm changes as p_2 is varied. A good example of this is the case where p_1 is 0.7 for the 20 variable case, figure 6. Here we can see how the FC based combinations vary from being significantly worse performers for low and high p_2 values but significantly better for the p_2 values near the hard region.

Another phenomenon clearly visible from figures 3-6 is a divergence between the FC based combinations and the non-lookahead combinations as p_2 reaches its maximum (and where p_2 is beyond p_{2crit}^6). This may not be too surprising since, as p_2 reaches its maximum, there is an abundance of solutions in the search space and a lot of forward checking effort is wasted. However this clearly demonstrates how the use of an FC combination could be a wrong decision and when scaled up to very large problems, this phenomenon may result in significant savings in search effort by choosing an alternative algorithm, when the problems are easy.

6. Conclusions

One of the aims of our work was to clearly demonstrate that, given a reasonable selection search algorithms and heuristics, there is no universally best choice of algorithm and heuristic combination. We have carried out a series of experiments on binary CSPs over a wide range of p_1 and p_2 values for a selection of well known complete search algorithms in order to achieve this aim.

We believe that our results do demonstrate that there are regions over the space of possible binary CSPs where different algorithm and heuristic combinations could be considered the best.

FC based algorithms have proved to be effective over the hard region for high P_1 values. This may be considered to be confirmation of frequently reported results. However, we have also found that the BM_CBJ algorithm combined with MWO or MO has its own part to play. BM_CBJ with either the MWO or MO heuristic is very effective with low P_1 values across the range of P_2 and we have also found it an effective algorithm for higher P_1 values although not in the cases where the problems are hard. In addition we have identified a region where there is no clear best combination.

We have shown that FC hybrids are not universally the best over the whole problem space covered by these experiments which has implications for solving easy problems as well as hard problems. The choice of an algorithm is just as important for easy constraint satisfaction problems as it is for hard ones, especially when the size of the problem becomes large. The differences in relative performance of algorithms on easy problems may be quite small when compared with the relative differences of the performance of different algorithms on hard problems, but these small differences may translate into many hours of search effort when the problem size is increased, as is often the case in real world problems.

Since, as we believe we have shown, there are no universally best algorithm-heuristic combinations, future research should try to identify the application domains of each combination - i.e. where they are most efficient. Furthermore, research should focus on how

⁶ p_{2crit} is the p_2 value where one solution can be expected - i.e. the predicted peak in the hard region. Our definition is somewhat different from that in [7] due to our definition of p_2 being different. For us;

$$p_{2crit} = m^{-2/p_1(n-1)}$$

to retrieve the most efficient combinations for a given problem. Results in this paper provide us with a first step towards such goals.

7. Future Work

As we have pointed out in this paper, there are several algorithms that have not been included in our experiments. There have been many new algorithms developed recently such as MAC [11] and FC_CBJ with directional k-consistency [9]. We plan to further the scope of our set of algorithm and heuristics by evaluating these algorithms and, as noted in the conclusion, we intend to characterise the application domains of them.

8. Acknowledgements

This research was supported by the University of Essex research promotion fund ref. R7027 and by the EPSRC research grant ref. GR/J/42878. The authors would also like to thank the reviewers for their useful comments, some of which have been incorporated into this paper.

References

- [1] Tsang E. P. K., Foundations of Constraint Satisfaction, Academic Press, 1993
- [2] Haralick, R. M. & Elliott, G. L., Increasing Tree Search Efficiency for Constraint Satisfaction Problems, *Artificial Intelligence* **14** (1980), 263-314
- [3] Nadel, B. A., Constraint Satisfaction Algorithms, *Computational Intelligence* **5** (1989), 188-224
- [4] Dechter, R. & Meiri, I., Experimental evaluation of preprocessing algorithms for constraint satisfaction problems, *Artificial Intelligence* **68** (1994), 211-241
- [5] Prosser, P., Hybrid Algorithms for the Constraint Satisfaction Problems, *Computational Intelligence* **9** (1993), 268-299
- [6] Cheeseman, P., Kanefsky, B. & Taylor, W., Where the Really Hard Problems are, *Proceedings 12th International Joint Conference on Artificial Intelligence*, 1991, 331-337
- [7] Smith M. B., Phase Transition and the Mushy Region in Constraint Satisfaction Problems, *Proceedings 11th European Conference on Artificial Intelligence*, 1994, 100-104
- [8] Prosser, P., Binary Constraint Satisfaction Problems: Some are Harder than Others, *Proceedings 11th European Conference on Artificial Intelligence*, 1994, 95-99
- [9] Prosser, P., Domain Filtering can Degrade Intelligent Backjumping Search, *Proceedings 13th International Joint Conference on Artificial Intelligence*, 1993, 263-267
- [10] Baker, A. B., The Hazards of Fancy Backtracking, *Proceedings 12th National Conference on Artificial Intelligence*, 1994, 288-293
- [11] Sabin, D. & Freuder, E. C., Contradicting Conventional Wisdom in Constraint Satisfaction, *Proceedings 11th European Conference on Artificial Intelligence*, 1994, 125-129
- [12] Gaschnig, J., Performance measurement and analysis of certain search algorithms, CMU-CS-79-124 Technical Report, Carnegie-Mellon University, Pittsburg, 1979

- [13] Gaschnig, J., A General Backtrack Algorithm that Eliminates Most Redundant Tests, *Proceedings 5th International Joint Conference on Artificial Intelligence*, 1977, 457
- [14] Freuder, E. C., A sufficient condition for backtrack-free search, *Journal of ACM* **29** (1982), 24-32
- [15] Dechter, R. & Pearl, J., Tree Clustering for Constraint Networks, *Artificial Intelligence* **38** (1989), 353-366
- [16] Bessière, C. Arc-Consistency and Arc-Consistency Again, *Artificial Intelligence* **65** (1994), 179-190
- [17] Nudel, B., Consistent-Labeling Problems and their Algorithms: Expected-Complexities and Theory-Based Heuristics, *Artificial Intelligence* **21** (1983), 135-178
- [18] Kondrak, G., A Theoretical Evaluation of Selected Backtracking Algorithms, MSc. Thesis, Department of Computer Science, University of Alberta, Canada, 1994
- [19] Dechter, R., & Pearl, J., Network-based Heuristics for constraint-satisfaction problems, *Artificial Intelligence* **34** (1988), 1-38
- [20] Mackworth, A. K., Consistency in Networks of Relations, *Artificial Intelligence* **8** (1977), 99-118