Comparing CSP Algorithms Without Considering Variable Ordering Heuristics can be Misleading

Alvin C. M. Kwan & Edward P. K. Tsang
Department of Computer Science, University of Essex
Wivenhoe Park, Colchester, Essex CO4 3SQ, United Kingdom

Email: {alvin@essex.ac.uk, edward@essex.ac.uk}

December 1995

Abstract

Many new algorithms for solving constraint satisfaction problems have been proposed in recent years. Their performance is usually compared with some other algorithms by measuring the number of compatibility checks or the cpu time that the algorithms need for solving a predetermined set of problems. There are some studies which compare algorithms without using any variable ordering heuristics. We argue that evaluating algorithms without considering their combinations with variable ordering heuristic could produce misleading results. We advocate that algorithm-heuristic combinations should be compared instead of algorithms alone.

1 Introduction

A constraint satisfaction problem (CSP) can be informally defined as a set of variables, each of them associated with a set of feasible domain values which the variable can be instantiated to, and a set of constraints, each of them imposes some restrictions on the values of its constrained variables can take simultaneously. The aim is to find a solution of the problem such that every variable is assigned with a value without violating any constraints [Tsang 93].

As many real problems can be formulated as CSPs, much research has been done on the area in recent years. This results in a number of new CSP algorithms. Since analysing the average-case behaviour of those techniques is very difficult, most evaluations are done in an empirical way. One of the most commonly adopted performance indicators in evaluating a CSP algorithm is the number of compatibility checks that the algorithm performs in solving a problem. In many studies, CSP algorithms are evaluated by applying each of the algorithms under investigation to solve a set of problems, typically artificial ones. The empirical results are then analysed statistically.

End-users are always interested to use an algorithm in the most effective way. Since it has been established that variable ordering heuristics usually improve the performance of CSP algorithms in problem solving [Haralick & Elliott 80] [Dechter & Meiri 94] [Tsang *et. al.* 95], it is a natural conclusion that the performance of algorithm-heuristic

combinations, instead of the algorithms alone, should be studied from a practical viewpoint. However this issue might have been overlooked. There are studies which compare algorithms in a random variable ordering [Haselböck 93] [Prosser 93] [Schiex & Verfaillie 93]. The risk of doing this is that misleading conclusions could be made. It is the aim of this paper to explore this notion.

The discussion of interactions between heuristics and algorithms is not new. For example, the performance of forward checking with conflict-directed backjumping is found to be marginally better than that of forward checking when both are combined with the first-first heuristic in some research [Smith & Grant 95] [Bacchus & van Run 95]. This finding is substantially different from the results reported in [Prosser 93] in which forward checking with conflict-directed backjumping is found to be significantly better than forward checking when a random variable ordering is used. In fact this phenomenon is not restricted to these two algorithms. Examples of this type are described in §3. An even worse situation is that algorithm *A* could perform better than algorithm *B* when both use a random variable ordering and worse than algorithm *B* when both use variable ordering heuristics. The details are discussed in §4. Conclusions are given in the last section. We first start with the experimental design in the next section.

2 Experimental design

The aim of our experiments was to find out empirically whether variable ordering heuristics can be ignored when comparing CSP algorithms. Our experiments were conducted on a set of randomly generated binary CSPs which has been widely used as benchmark problems for evaluating CSP methods. The binary CSPs are characterised by a 4-tuple <*n*, *m*, *p*1, *p*2> such that *n* is the number of variables, *m* is the uniform domain size for each variable, *p*1, also called constraint density, is the proportion of pairs of variable that a constraint exists between them and *p*2, also known as constraint tightness, is the proportion of incompatible pairs of values for each pair of variables if there exists a constraint between them.[1] No universal constraints (that allows any pair of values) or null constraints (that do not allow any pair of values) were allowed. All the generated problems have connected constraint graphs.

The experiments were focussed on finding the first solutions or exhausting the search space otherwise for CSPs characterised by <10, 10, *p*1, *p*2>[2] which are within the *mushy* regions [Smith 94].[3] The idea behind choosing this particular set of CSPs for experimentation was that the difference of performance of different algorithms appeared to be more obvious in mushy regions. Furthermore, it facilitates us to contrast our results with other studies which often experiment on problems within the mushy regions, typically around the crossover point [Crawford & Auton 93]. *p*1 was varied at a scale of 0.1

[1] We have used *p*2 as constraint looseness, which was given by [Haralick & Elliott 80] and [Nudel 83], in our previous work. However it appears that *p*2 has been used as constraint tightness in more recent research [Prosser 94a] [Smith 94]. Due to the popularity of the more recent interpretation, we have decided to adopt that definition for the sake of uniformity in the field.

[2] Although the problems are small, they are adequate for illustration purpose.

[3] The "mushy region" denotes the range of values of the order parameter over which the phase transition from solubility to insolubility takes place [Smith 94].

starting from 0.2 to 1.0.[4] $p2$ was varied by an increment of 0.01 across the mushy region. At each data point, 50 samples were generated. Note that the boundary of mushy region for each of the chosen $p1$ values was determined empirically. The exact values $p2$ for different $p1$ values, on which the results in §3 and §4 are based, are given in Table 1.

$p1$	$p2$
0.2	0.83 -- 0.93
0.3	0.73 -- 0.85
0.4	0.64 -- 0.77
0.5	0.58 -- 0.70
0.6	0.53 -- 0.64
0.7	0.48 -- 0.57
0.8	0.45 -- 0.51
0.9	0.40 -- 0.48
1.0	0.37 -- 0.44

Table 1. Values of $p1$ and $p2$ used in the experiment ($p2$'s increment = 0.01)

A number of algorithms were tested. These algorithms were backjumping (BJ) [Gaschnig 78], backmarking (BM) [Gaschnig 77], forward checking (FC) [Haralick & Elliott 80], conflict-directed backjumping (CBJ), forward checking with conflict-directed backjumping (FC-CBJ) and backmarking with conflict-directed backjumping (BM-CBJ) [Prosser 93]. Concerning the variable ordering heuristics, minimal width heuristic (MWO) [Freuder 82] and random ordering (RAN) were applied to all the above algorithms. In addition, the fail-first heuristic (FF) [Haralick & Elliott 80] was applied to FC and FC-CBJ. These algorithms and heuristics were chosen mainly because of their popularity. All the algorithm-heuristic combinations were run on each of the generated CSPs. Note that in the rest of the paper, we use the plus sign "+" to stand for "combined with". For example, "BM+MWO" means backmarking combined with minimal width heuristic.

3 How much do heuristics improve algorithm's performance?

A comparative study on the performance of BJ, CBJ, BM, BM-CBJ, FC and FC-CBJ based on a version of zebra problem appears in [Prosser 93]. According to it, BM performed about 14 times more compatibility checks than BM-CBJ, BJ performed about seven times more checks than CBJ whereas FC performed about two and a half times more checks than FC-CBJ on the tested problems on average.[5] However no heuristics were used in these experiments and it is unclear whether the same scale of improvement can be achieved when variable ordering heuristics are used. Attempting to answer this question, we conducted experiments on FC and FC-CBJ with random ordering and FF, and BJ, CBJ, BM and BM-CBJ with random ordering and MWO. The ratio of the average number of compatibility checks performed by each of the algorithms enhanced by

[4] 0.2 is the smallest $p1$ that allows a 10-variable CSP to have a connected constraint graph.

[5] We have conservation on using mean as the sole statistical measure for evaluating CSP algorithms. A discussion on which statistics to use for evaluating CSP methods can be found in [Kwan 95].

CBJ to that of its base algorithm was measured first with no heuristic and then with heuristic.[6] For example, a ratio of "2" means that the base algorithm performs twice the number of compatibility checks as compared to its enhanced version whereas "1" means the two algorithms perform the same number of compatibility checks over the same problem set. Tables 2 and 3 show the results for $p1 = 0.2$ and 0.3 respectively. We chose the low $p1$ values because the zebra problem described in [Prosser 93] also has a low constraint density ($p1 \approx 0.2$). This might allow us to contrast our results with those in a fairer way.

	*p*2										
	0.83	0.84	0.85	0.86	0.87	0.88	0.89	0.90	0.91	0.92	0.93
BM/ BM-CBJ (RAN)	15.68	10.16	10.16	10.33	12.85	13.47	11.21	10.73	13.85	16.88	26.74
BM/ BM-CBJ (MWO)	1.72	1.84	1.75	1.54	1.50	1.43	1.17	1.33	1.25	1.22	1.15
BJ / CBJ (RAN)	5.08	3.79	3.73	3.93	3.65	3.75	3.82	4.19	4.73	5.08	9.64
BJ / CBJ (MWO)	1.22	1.26	1.46	1.31	1.26	1.27	1.17	1.17	1.12	1.10	1.06
FC / FC-CBJ (RAN)	2.97	2.25	2.32	2.36	2.36	2.42	2.12	2.79	3.29	3.38	4.07
FC / FC-CBJ (FF)	1.02	1.09	1.11	1.10	1.08	1.11	1.08	1.05	1.04	1.02	1.02

Table 2. Ratio of the average number of checks done by BM & BM-CBJ, BJ & CBJ, and FC & FC-CBJ, with and without heuristics over 50 CSPs across the mushy region for $p1 = 0.2$

	*p*2											
	0.73	0.74	0.75	0.76	0.77	0.78	0.79	0.80	0.81	0.82	0.83	0.84	0.85
BM/ BM-CBJ (RAN)	5.00	3.87	3.40	4.58	3.36	3.40	3.25	3.37	3.95	4.44	4.40	4.75	4.99
BM/ BM-CBJ (MWO)	1.21	1.18	1.11	1.19	1.13	1.10	1.12	1.12	1.14	1.12	1.12	1.12	1.11
BJ / CBJ (RAN)	4.92	3.84	3.52	2.71	2.36	2.31	2.28	2.23	2.86	2.94	2.99	3.22	3.22
BJ / CBJ (MWO)	1.15	1.13	1.09	1.13	1.13	1.11	1.11	1.09	1.13	1.08	1.05	1.04	1.03
FC / FC-CBJ (RAN)	1.50	1.38	1.34	1.33	1.47	1.46	1.41	1.43	1.57	1.49	1.45	1.47	1.48
FC / FC-CBJ (FF)	1.04	1.10	1.10	1.10	1.06	1.04	1.06	1.04	1.04	1.03	1.02	1.02	1.02

Table 3. Ratio of the average number of checks done by BM & BM-CBJ, BJ & CBJ, and FC & FC-CBJ, with and without heuristics over 50 CSPs across the mushy region for $p1 = 0.3$

[6] CBJ was taken as an enhanced version of BJ.

The results shown in rows 1, 3 and 5 in Tables 2 and 3 demonstrate that the algorithms with CBJ achieve significant saving over the base algorithms when no variable ordering heuristics are considered. These are broadly consistent with the results described in [Prosser 93] despite the zebra problem instead of random binary CSPs was used in that research. However, when heuristics are used, the benefit was greatly reduced, sometimes to marginal (see row 6 in Tables 2 and 3).

From Tables 2 and 3, it is clear that the adoption of the FF or MWO greatly reduces the benefit of CBJ or its hybrids when contrasting with their base algorithms. For example, row 1 in Table 2 shows that BM checked at least ten times more often than BM-CBJ when a random ordering was used for solving CSPs in the mushy region when $p1$ = 0.2. However, the average gain becomes much less, from a maximum of 84 percent to a minimum of fifteen percent when MWO was used as shown in row 2 in the same table. Similarly, CBJ only improved BJ three to fifteen percent when MWO was used for CSPs with $p1$ = 0.3 (row 4 in Table 3). For a random variable ordering, CBJ typically performed about one-third of the compatibility checks done by BJ on the same problem sets (row 3 in Table 3). The results of FC and FC-CBJ based on a random ordering and an ordering produced by FF give a similar picture. From an end-user's viewpoint, an algorithm should be used in the most effective way. Whether algorithm *A* checks several times more often than algorithm *B* in a random ordering is insignificant as they should never be used without a heuristic.

Since enhanced algorithms usually employ more sophisticated data structures which need more computational resources to maintain, their saving in doing less compatibility checks may not be able to balance the extra overheads. For example, in our implementation, for most of the problems we tested, the average cpu time spent by BM with MWO are less than that of BM-CBJ with MWO and that spent by FC with FF is less than that of FC-CBJ with FF. However readers should bear in mind that the CSPs we tested are small. The benefit of using a more sophisticated algorithm may pay off for large CSPs, *e.g.* when the number of variables of a problem is large. Furthermore the cost of performing a compatibility check, compared to the cost in maintaining more sophisticated data structures, is inexpensive in our implementation as binary constraints are represented by binary matrices. Thus the saving in performing less compatibility checks may simply not enough to balance the extra efforts required for maintaining the more sophisticated data structures. For cases where problem size is large or constraint evaluation is expensive, enhanced algorithms like FC-CBJ could outperform their base algorithms both in terms of the number of compatibility checks and the cpu time.

4 How is the ranking of algorithms affected by heuristics?

We have already seen how misleading it could be to evaluate algorithms without considering heuristics. Could the situation be even worse? Indeed it could. Algorithm *A* may work better than algorithm *B* when both use a random variable ordering but worse than algorithm *B* when variable ordering heuristics are used. The evidence is given in Figures 4 and 5.

By comparing the average number of compatibility checks performed by FC-CBJ and BM-CBJ with a random ordering across the mushy region for $p1$ equal to 0.2 (see Figure

4), one may reach the conclusion that BM-CBJ is less efficient than FC-CBJ. An entirely different conclusion can be reached when we consider the algorithms with variable ordering heuristics as a whole. Figure 5 reveals that BM-CBJ with MWO performs better than FC-CBJ with MWO or with FF over the same problem set.[7] Note that FC-CBJ with FF and FC-CBJ with MWO have very similar performance on the tested CSPs and thus the two corresponding lines in Figure 5 overlap.

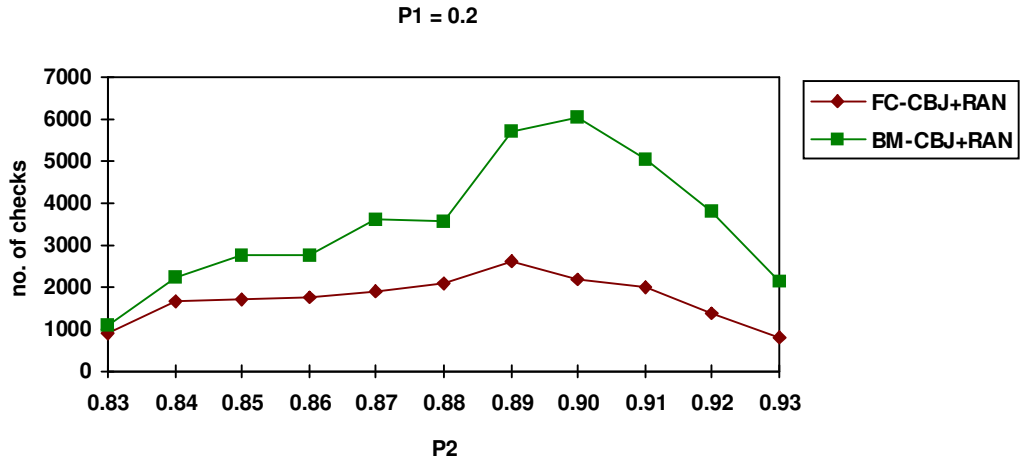

Figure 4. Average number of compatibility checks performed by FC-CBJ and BM-CBJ in random variable ordering over 50 CSPs across the mushy region for $p1 = 0.2$

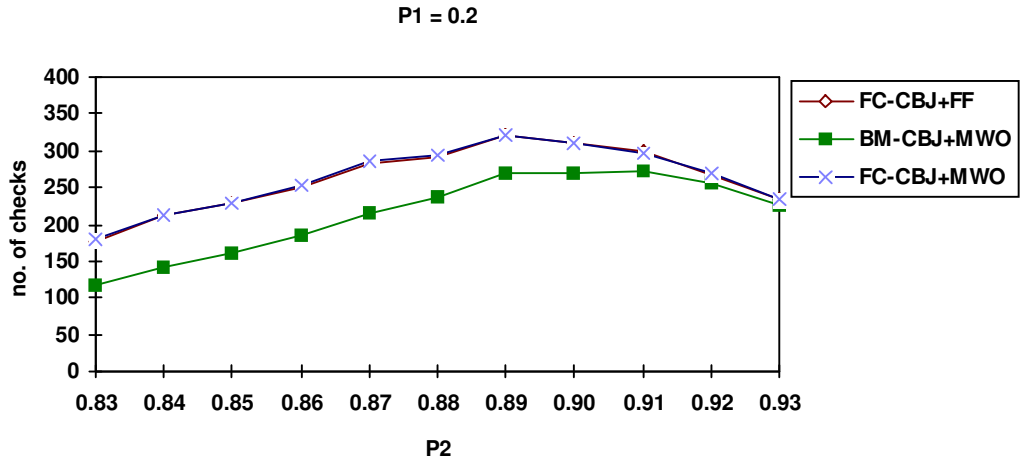

Figure 5. Average number of compatibility checks performed by FC-CBJ (with MWO or FF) and BM-CBJ (with MWO) over 50 CSPs across the mushy region for $p1 = 0.2$

The rankings of algorithms also changes for large $p1$. This is shown in Figure 6. When $p1 = 0.9$, although FC-CBJ performs less efficient than BM-CBJ in random variable

[7] Similar results were obtained over the mushy regions for $p1 = 0.3$ to 0.6. Note that out of the 52 sets of problems, there are twelve exceptions in which the average number of checks performed by FC-CBJ is more than that of BM-CBJ when a random variable ordering was adopted. However, nine of them are within a 10 percent difference.

ordering, FC-CBJ with FF performs better than BM-CBJ with MWO. Similar phenomenon was observed for $p1 = 1$.[8]

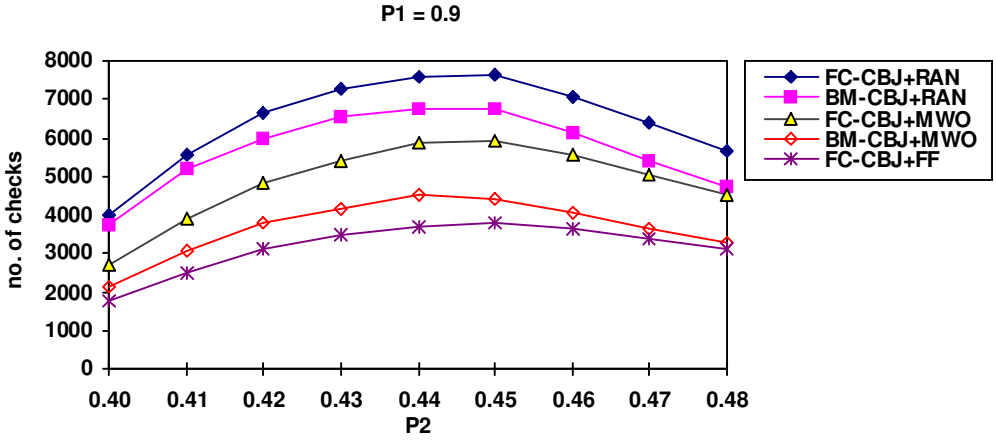

Figure 6. Average number of compatibility checks performed by five algorithm-heuristic combinations over 50 CSPs across the mushy region for $p1 = 0.9$

5 Conclusions

As more people are aware of the importance of CSP research, more efforts have been put into the field than ever before. This results in a number of new algorithms and heuristics which are almost always reported as significantly better than some other existing techniques. However some studies overlook the issue that it is the algorithm-heuristic pairs needed to be compared, not the algorithms themselves. The failure to observe this point results in an inappropriate experimental design. The results produced could lead to misleading conclusions.

We have given evidence that evaluating CSP algorithms without considering variable ordering heuristics could be dangerous. Although our discussion has focussed on the interaction of search algorithms and variable ordering heuristics, the results of a preliminary study show that whether to pre-process a CSP with AC-6 [Bessière 94] before a search could depend on whether variable ordering heuristic is taken into account. This highlights the complicated interaction among pre-processing techniques, search algorithms and variable ordering heuristics. We advocate that preprocessing techniques should be evaluated with search algorithms and heuristics as a whole.

Acknowledgements

Thanks are given to James Borrett who helped to implement the CSP algorithms and heuristics. This research was supported by the University of Essex research promotion fund ref. R7207 and by the EPSRC research grant ref. GR/J/42878.

[8] When $p1$ is set to one, MWO provides no heuristic information and produces a random variable ordering.

References

[Bacchus & van Run 95]
Bacchus, F. & van Run, P. Dynamic Variable Ordering in CSPs. *In Proceedings the First International Conference on Principles and Practice of Constraint Programming* (*CP'95*), 258-275, 1995.

[Bessière 94]
Bessière, C. Arc-Consistency and Arc-Consistency Again. *Artificial Intelligence* 65(1), 179-190, 1994.

[Crawford & Auton 93]
Crawford, J.M. & Auton, L.D. Experimental Results on the Crossover Point in Satisfiability Problems. In *Proceedings AAAI-93*, 21-27, 1993.

[Dechter & Meiri 94]
Dechter, R. & Meiri, I. Experimental Evaluation of Preprocessing Algorithms for Constraint Satisfaction Problems. *Artificial Intelligence* 68, 211-241, 1994.

[Freuder 82]
Freuder, E. A Sufficient Condition for Backtrack-Free Search, *J. ACM* 29, 24-32, 1982.

[Gaschnig 77]
Gaschnig, J. A General Backtrack Algorithm that Eliminates Most Redundant Tests. In *Proceedings IJCAI-77*, 457, 1977.

[Gaschnig 78]
Gaschnig, J. Experimental Case Studies of Backtrack vs. Waltz-Type vs. New Algorithms for Satisficing Assignment Problems. In *Proceedings the 2nd Biennial Conference of the Canadian Society for Computational Studies of Intelligence*, 268-277, 1978.

[Haralick & Elliott 80]
Haralick, R.M. & Elliott, G.L. Increasing Tree Search Efficiency for Constraint Satisfaction Problems. *Artificial Intelligence* 14, 263-314, 1980.

[Haselböck 93]
Haselböck, A. Exploiting Interchangeabilities in Constraint Satisfaction Problems. In *Proceedings IJCAI-93*, 282-287, 1993.

[Kwan 95]
Kwan, A.C.M. The Validity of Normality Assumption in CSP Research. Technical Report CSM-261, Department of Computer Science, University of Essex, U.K., 1995.

[Nudel 83]
Nudel, B. Consistent Labeling Problems and their Algorithms: Expected Complexities and Theory-based Heuristics. *Artificial Intelligence* 21, 135-278, 1983.

[Prosser 93]
Prosser, P. Hybrid Algorithms for the Constraint Satisfaction Problem. *Computational Intelligence* 9(3), 268-299, 1993.

[Schiex & Verfaillie 93]
Schiex, T. & Verfaillie, G. Nogood Recording for Static and Dynamic Constraint Satisfaction Problems. In *Proceedings International Conference on Tools with Artificial Intelligence*, 48-55, 1993.

[Smith 94]
Smith, B.M. Phase Transition and the Mushy Region in Constraint Satisfaction Problems. In *Proceedings ECAI-94*, 100-104, 1994.

[Smith & Grant 95]
Smith, B.M. & Grant, S.A. Sparse Constraint Graphs and Exceptionally Hard Problems. In *Proceedings IJCAI-95*, 646-651, 1995.

[Tsang 93]
Tsang, E. *Foundations of Constraint Satisfaction*. Academic Press, 1993.

[Tsang *et. al*. 95]
Tsang, E.P.K, Borrett, J.E. & Kwan, A.C.M. An Attempt to Map the Performance of a Range of Algorithm and Heuristic Combinations. In *Proceedings AISB-95*, 203-216, 1995.