A comparison of complete and incomplete algorithms in the easy and hard regions

Andrew Davenport
Department of Computer Science, University of Essex,
Wivenhoe Park, Colchester,
Essex CO4 3SQ, United Kingdom.
daveat@essex.ac.uk

Abstract

A number of non-systematic, incomplete constraint satisfaction search algorithms have been developed in recent years, however there has been little work on characterising when they should be preferred over systematic, complete search algorithms. In this paper we report on an empirical investigation with the aim of determining whether one type of algorithm is to be preferred over another in a particular region of the graph colouring problem space with respect to the phase transistion.

1 Introduction

Cheeseman et al [Cheeseman *et al.*, 1991] have shown that for some NP-complete problems [Garey and Johnson, 1979] (*e.g.*, graph colouring, hamiltonian circuit) the hard instances occur around a critical value of some problem parameter (*e.g.*, the average degree of the nodes for graph colouring). A phase transistion is said to occur as this parameter is varied, from a region of under-constrained, mostly soluble problems to a region of over-constrained, mostly insoluble problems. The under-constrained problems tend to be relatively easy to solve and thus form an "easy" region of the problem space. Similiarly the over-constrained problems tend to be relatively easy to prove insoluble. However the peak in median search cost is associated with the phase transistion, where we find both soluble and insoluble problems. These problems have been typically found to be hard to solve by all algorithms and thus can be characterised as being inherently hard.

Cheeseman et al [Cheeseman *et al.*, 1991] have conjectured that this phase transistion phenomena occurs for all NP-complete problems. Since then the phase transistion phenomenon has been observed in randomly generated binary constraint satisfaction problems [Prosser, 1994a; Smith, 1994b], in propositional satisfiability problems [Mitchell *et al.*, 1992; Gent and Walsh, 1994b] and more recently in real-life problems such as the travelling salesman decision problem, boolean circuit synthesis, boolean induction and exam timetabling [Gent and Walsh, 1994a].

In recent years a number of non-systematic, incomplete algorithms have been developed for constraint satisfaction *e.g.*, heuristic repair [Minton *et al.*, 1992], GSAT [Selman *et al.*, 1992] and GENET [Davenport *et al.*, 1994]. These new algorithms can outpeform systematic, complete algorithms on certains kinds of problems *e.g.*, the 1,000,000 queens problem [Minton *et al.*, 1992], randomly generated 3-SAT problems [Selman and Kautz, 1993]. However these non-systematic algorithms can also be worse than systematic algorithms on other kinds of "structured" problems [Konolige, 1994]. One important issue which has not, to our knowledge, been adequately addressed by previous research is how do systematic, complete and non-systematic, incomplete algorithms compare in different regions of the problem space with respect to the phase transistion ? Is one type of algorithm clearly better than another in a particular region ? In the following sections we investigate these issues in more depth and go on to present the methodology and results of an empirical investigation into a number of complete and incomplete algorithms on a particular type of NP-complete constraint satisfaction problem, namely graph colouring, in the easy and hard regions of the problem space.

2 The mostly easy, soluble region

Although the peak in *median* search cost has been found to occur in the phase transistion region of the problem space, recent research indicates that, at least for some algorithms, the peak in *mean* search cost may occur in the mostly easy, soluble region [Gent and Walsh, 1994c; Hogg and Williams, 1994; Smith, 1994a]. Although most problems in this easy region can be solved very quickly there are occasional, very rare instances that can be so difficult to solve they dominate the mean search cost of solving problems in this region. These *exceptionally hard problems* can be orders of magnitude harder to solve than even the hardest problems in the phase transistion region. Williams and Hogg [Hogg and Williams, 1994] have found that the hardest problems are concentrated below the phase transistion peak in median search cost and identified these problems with a sec-

ond phase transistion, corresponding to the transistion between polynomial and exponential scaling of the median search cost. Gent and Walsh [Gent and Walsh, 1994c] observe that the greatest variability in problem solving cost occurs in this region and it occurs for both soluble and insoluble problems.

Smith and Grant [Smith and Grant, 1995] define a problem as being exceptionally hard if:

- it occurs in the region where almost all problems are soluble, and on average easy to solve;
- (for some search algorithms) it is much more difficult than almost all other problems with the same parameter values;
- it is more difficult than almost all the problems occuring in the hard region.

There is some uncertainty over whether this phenomena of exceptionally hard problems occurs for all algorithms, only occurs for all complete algorithms or is purely algorithm dependent. Smith [Smith, 1994a] concludes that some exceptionally hard problems are "inherently more difficult than other similiar problems, independently of the number of solutions they have". Gent and Walsh [Gent and Walsh, 1994c] make the weaker conjecture that the phenomena of exceptionally hard problems may occur for all complete algorithms. It is interesting to note that nobody has yet reported any exceptionally hard, soluble problems for non-systematic, incomplete search algorithms. Williams and Hogg [Hogg and Williams, 1994] have looked at the performance of heuristic repair on graph colouring problems while Gent and Walsh [Gent and Walsh, 1994d] have examined the performance of GSAT on easy 3-SAT problems, but neither found any exceptionally hard problems for these algorithms. Does this mean that systemacity is the cause of exceptionally hard problems ? If this is the case should we always prefer non-systematic, incomplete search algorithms to complete search algorithms when tackling problems in the easy region, or can we avoid exceptionally hard problems for complete search by using more sophisticated algorithms ? As Prosser [Prosser, 1994b] puts it, are exceptionally hard problems merely existence proofs for exceptionally bad algorithms ?

3 The hard region

Although systemacity may be a disadvantage in the mostly easy, soluble region where solutions are usually numerous, intuitively we would expect systemacity to be more useful in the hard region of the problem space, where solutions, if there are any, are very rare. Non-systematic, incomplete search algorithms may visit a state in the search space more than once, and have no guaranteed performance bounds.

However some studies comparing incomplete (GSAT) and complete (Davis-Puttnam) search algorithms on randomly generated 3-SAT problems discovered that the incomplete search was able to solve problems far larger than any that could be solved by the complete search algorithm [Selman *et al.*, 1992]. Could it be that non-systematic searches are better suited to solving (soluble) problems in the hard region for other types of problems as well, such as constraint satisfaction problems in general ? Although complete algorithms may never revisit a state, it may be the case that they search in a much larger search space than that looked at by the incomplete search algorithms. However a large number of algorithms for constraint satisfaction have been developed in recent years, many of which can significantly prune or reduce the size of the search space *e.g.*, k-consistency/k-consistency lookahead algorithms, intelligent backjumping techniques. Such algorithms may be able to compete with or outperform incomplete methods in the hard region.

4 An empirical investigation

In order to address some of the issues raised above we conducted an empirical investigation into a number of complete and incomplete constraint satisfaction search algorithms in easy and hard regions of the problem space.

We chose graph 3-colourability as our problem domain. One reason for this is that we are interested in finding exceptionally hard problems. Recent work [Smith and Grant, 1995] has indicated that these problems are more likely to occur for problems with sparse constraint graphs. Furthermore exceptionally hard graph colouring problems have been found in previous studies [Hogg and Williams, 1994]. Graph colouring is a well-studied NP-complete problem [Garey and Johnson, 1979], and has a number of applications in artificial intelligence, including planning and scheduling.

The graphs we used were guaranteed by construction to be connected by first generating a spanning tree for the graph. We use the average degree of the nodes (which we also refer to as the connectivity) of the graph as a problem parameter to distinguish between easy and hard regions in the space of graph colouring problems [Cheeseman *et al.*, 1991].

For all experiments we generated graphs varying in connectivity in steps of 0.1. The critical point of the phase transistion is expected to occur around a connectivity of 4.6 for graph 3-colourability. At each connectivity we generated 10,000 graphs. For complete algorithms we require the algorithm to either find the first 3-colouring of the graph or show that there is no 3-colouring. We only ran incomplete algorithms on soluble problems, and required them to find a single 3-colouring. We used forward checking [Haralick and Elliot, 1980] as our base complete search algorithm, and experimented with variable ordering and backjumping strategies[1]. We

[1] Forward-checking and other algorithms used in this chapter are described in [Tsang, 1993]. See also [Prosser, 1993]

also experimented with two local search techniques: GENET and GSAT. GENET [Davenport *et al.*, 1994] is an incomplete, local search algorithm based upon the min-conflicts heuristic [Minton *et al.*, 1992] but with the ability to escape local minima[2]. GSAT is a greedy local search procedure for solving propositional satisfiability problems [Selman *et al.*, 1992]. GSAT has been extended in a number of ways since its original incarnation [Selman and Kautz, 1993; Selman *et al.*, 1994]—we chose to use the averaging in strategy (reset after 25 tries) with MAX-FLIPS set to 10× the number of variables. Other GSAT options may be better for the problems we looked at, however these particular settings have been used to successfully solve some very difficult graph colouring problems [Selman and Kautz, 1993].

We present results using the number of consistency checks as the search cost measure for the complete algorithms and number of repairs as the search cost measure for the incomplete algorithms. We also look at CPU time, since this is arguably the only way to compare complete with incomplete algorithms. Although any conclusions we draw will be implementation dependent[3] we hope to be able to identify trends in the CPU time required by different algorithms to solve problems in different parts of the problem space.

To be able to compare CPU times of algorithms we must first say something about their implementations. All the complete search algorithms and GENET were implemented by the author in C++. The complete algorithms were implemented using the iterative style search given in [Prosser, 1993], although many of the heuristics are based on the pseudo-code given in [Tsang, 1993]. We used version 35 of GSAT as implemented by Bart Selman and Henry Kautz in C.

5 Experimental results

First we looked at graphs consisting of 100 nodes. We looked at forward-checking with the Brélaz heuristic (FC-BZ) [Brélaz, 1979; Turner, 1988] to order variables. The Brélaz variable ordering can be regarded as an extension of fail-first—it first selects the unlabelled variables with the smallest domains. In tie situations it selects the variable connected to the largest number of unlabelled variables, breaking further ties randomly. Exceptionally hard problems have been discovered for this algorithm by Hogg and Williams[Hogg and Williams, 1994] for graph 3-colourability problems.

Figures 1 and 2 present results for FC-BZ, showing the 100%, 99%, 90%, 75% and 50% percentiles for the number of consistency checks required to solve all problems and to solve only the soluble problems. Thus the 100% percentile shows the cost of solving the hardest problem, the 99% percentile shows the

[2]The version of GENET we used is a slight modification of that presented in [Davenport *et al.*, 1994], with a form of limited sideways moves described in [Davenport, 1995].

[3]All algorithms were run on a 175MHz DEC 3000 Model 600 AXP.

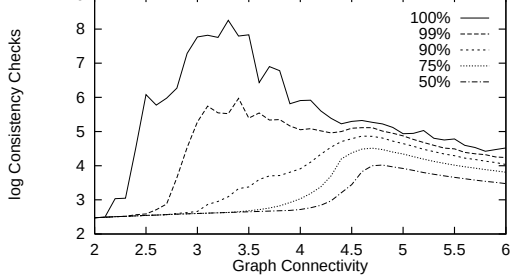

Figure 1: *Forward-checking with Brélaz on 100 node graph 3-colourability problems (soluble and insoluble problems).*

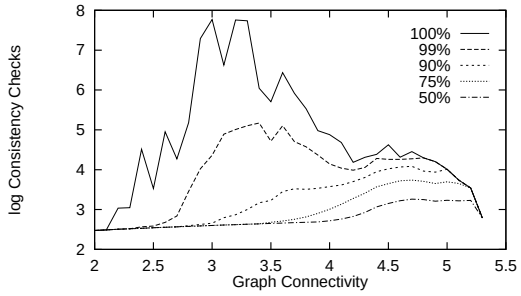

Figure 2: *Forward-checking with Brélaz on 100 node graph 3-colourability problems (soluble problems only).*

cost of solving the 9,900th hardest problem and the 50% percentile shows the median cost of solving problems at each connectivity. Here the hardest problems for FC-BZ, soluble and insoluble, do not occur in the phase transistion but in a region of relatively low connectivity. These results correspond to what has been reported in the literature on exceptionally hard problems [Gent and Walsh, 1994c; Hogg and Williams, 1994].

One problem with chronological backtracking algorithms such as FC-BZ is that they can exhibit a phenomena known as "thrashing" [Mackworth, 1977]. This occurs when the algorithm backtracks from a "dead-end" variable, for which a consistent assignment cannot be found, to a previously labelled variable which is not in conflict with the dead-end variable. Any change of assignment to this previously labelled variable will not prevent the algorithm rediscovering this dead-end when it comes once again to labelling the dead-end variable. This behaviour can be particularly harmful if there are many variables which have been labelled between the dead-end variable and any of the variables whose assignments are actually in conflict with the dead-end variable. One way to avoid this wasted search effort is to use some form of backjumping to backtrack directly to a variable which is causing the conflict [Prosser, 1993].

We added a conflict-directed backjumping capability to FC-BZ to obtain forward-checking with conflict-directed backjumping and Brélaz (FC-CBJ-BZ), and present results for this algorithm in Figures 3 and 4. Here the hardest soluble and insoluble problems

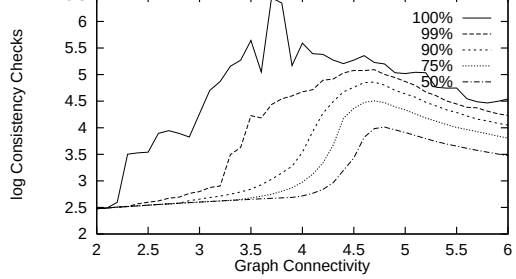

Figure 3: *Forward-checking with conflict-directed backjumping and Brélaz on 100 node graph 3-colourability problems.*

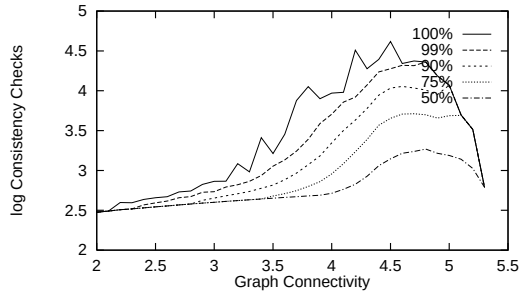

Figure 4: *Forward-checking with conflict-directed backjumping and Brélaz on 100 node* soluble *graph 3-colourability problems.*

are tending to occur at higher connectivities than for FC-BZ. In fact, by adding conflict-directed backjumping we have eliminated most of the soluble problems which were exceptionally hard for FC-BZ at the lower connectivities. These results support the hypothesis that thrashing is a major cause of exceptionally hard problems for FC-BZ.

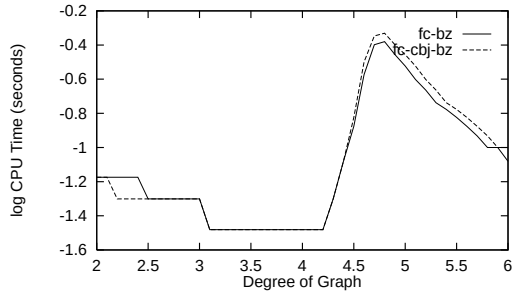

Figure 5: *Comparison of* FC-BZ *and* FC-CBJ-BZ *by median* CPU *time required to solve 100 node graph 3-colourability problems (soluble and insoluble).*

Figures 5 and 6 compare the median and mean CPU times required by FC-BZ and FC-CBJ-BZ. We see that the CPU time for FC-BZ and FC-CBJ-BZ appears to decrease slightly for graphs between connectivity of 2.0 and 4.0. This is caused by forward-checking performing more redundant constraint propagation for graphs of very low connectivity where the problems are so underconstrained that very few values can be pruned. The comparison of median CPU time shows that conflict-directed backjumping appears to be more

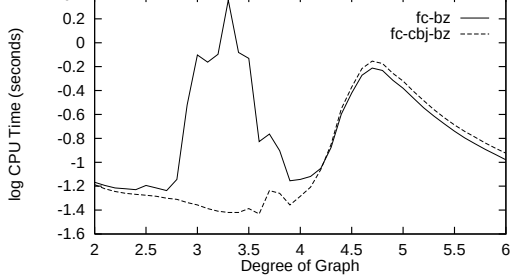

Figure 6: *Comparison of* FC-BZ *and* FC-CBJ-BZ *by mean* CPU *time required to solve 100 node graph 3-colourability problems (soluble and insoluble).*

useful on sparse graphs, being no worse, and even slightly better than FC-BZ at very low connectivities. Overall however the cost of adding conflict-directed backjumping appears to be neglible. The comparison of mean CPU time gives a very different picture at low connectivities. Here the presence of exceptionally hard problems dominates the mean CPU time for FC-BZ at low connectivities.

Brélaz is quite an expensive variable ordering heuristic which, since it is a dynamic variable ordering, can be called an exponential number of times during the course of the search. Since thrashing appears to be a major cause of exceptionally hard problems, can we avoid such problems by using a cheaper variable ordering heuristic and retaining conflict-directed backjumping ?

We substituted the fail-first variable ordering heuristic for the Brélaz heuristic to obtain forward-checking with conflict-directed backjumping and fail-first (FC-CBJ-FF). We ran FC-CBJ-FF on the 100 node graphs and present the results in Figures 7 and 8. The median number of consistency checks required by FC-CBJ-FF to solve problems in the hard region is higher than for FC-BZ. Furthermore the occurence of exceptionally hard problems is much higher for FC-CBJ-FF than for FC-CBJ-BZ, and is comparable to FC-BZ. This is somewhat surprising since FC-BZ is a chronological backtracking algorithm.

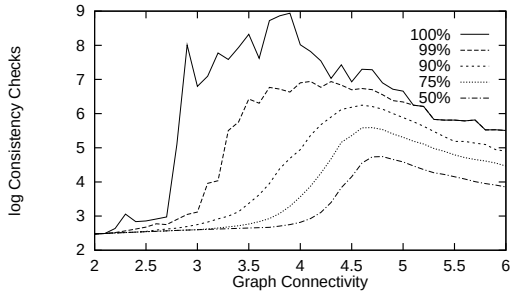

Figure 7: *Forward-checking with conflict-directed backjumping and fail-first on 100 node graph 3-colourability problems.*

Figures 9 and 10 compare CPU time for FC-CBJ-FF and FC-CBJ-BZ. The extra computation performed by Brélaz is very significant at low connectivities, where

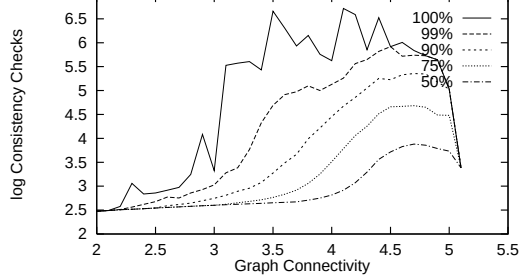

Figure 8: *Forward-checking with conflict-directed backjumping and fail-first on 100 node* soluble *graph 3-colourability problems.*

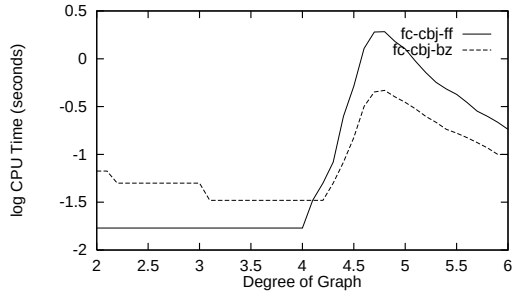

Figure 9: *Comparison of* FC-CBJ-FF *and* FC-CBJ-BZ *by median* CPU *time required to solve 100 node graph 3-colourability problems (soluble and insoluble).*

fail-first results in a much faster CPU time. However in the hard region Brélaz outperforms fail-first in terms of median and mean CPU time. At intermediate connectivities the mean and median CPU times give a different picture—this is caused by the higher occurence of exceptionally hard problems for FC-CBJ-FF, which can dominate the mean search cost of solving problems in this region. Thus an interesting point to arise out of this is that for small problem samples it may appear better to use relatively cheap heuristics such as the fail-first variable ordering rather than more expensive ones such as Brélaz, since for problems in the easy region the median CPU time to find a solution may be lower. However, the presence of rare, exceptionally hard problems in large samples of problems can result in the mean CPU time for algorithms such

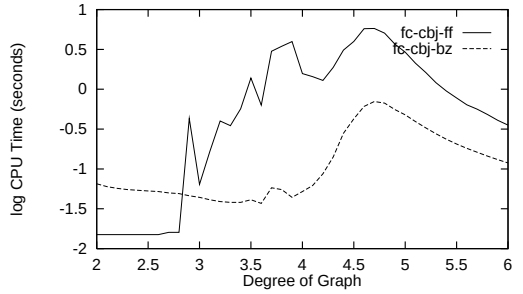

Figure 10: *Comparison of* FC-CBJ-FF *and* FC-CBJ-BZ *by mean* CPU *time required to solve 100 node graph 3-colourability problems (soluble and insoluble).*

as FC-CBJ-BZ being lower than that for FC-CBJ-FF in parts of the easy region. Thus investing more time in variable ordering and retaining backjumping information may pay off in the long run.

The final complete search algorithm we looked at was forward-checking with minimal width ordering (FC-MWO) [Freuder, 1982]. The idea behind minimal width ordering is to label the most constrained variables first by minimising the width[4] of the variable ordering. The minimal width ordering is a static variable ordering heuristic with worst case time complexity $O(n^2)$ to order n variables.

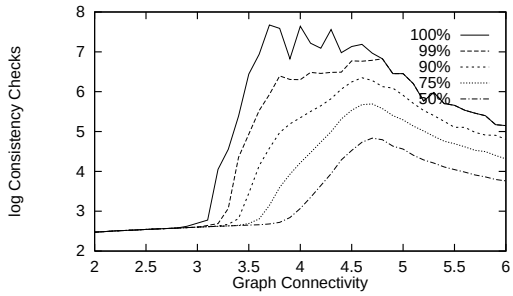

Figure 11: *Forward-checking with minimal width ordering on 100 node graph 3-colourability problems.*

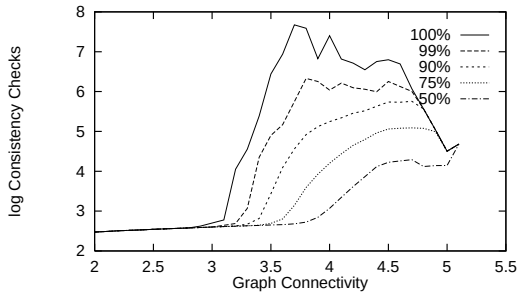

Figure 12: *Forward-checking with minimal width ordering on* soluble *100 node graph 3-colourability problems.*

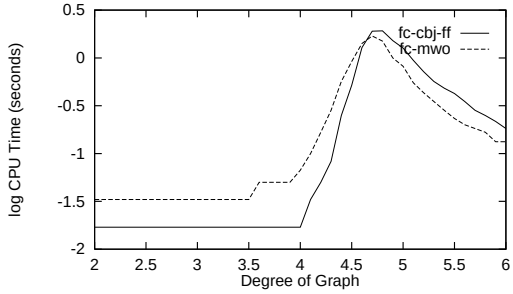

Figure 13: *Comparison of* FC-CBJ-FF *and* FC-MWO *by median* CPU *time required to solve 100 node graph 3-colourability problems (soluble and insoluble).*

[4] The *width* of a node in an ordered constraint graph is the number of arcs between that node and the previous nodes in the ordering. The width of an ordering is the maximum width of all the nodes in the ordering. The width of a constraint graph is the minimal width of all the possible orderings of the graph.

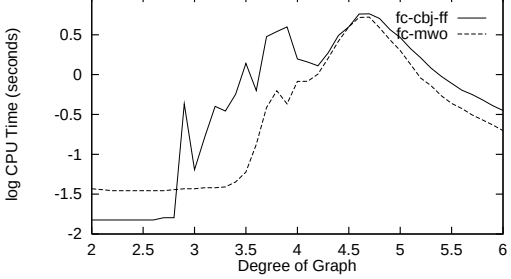

Figure 14: *Comparison of* FC-CBJ-FF *and* FC-MWO *by mean* CPU *time required to solve 100 node graph 3-colourability problems (soluble and insoluble).*

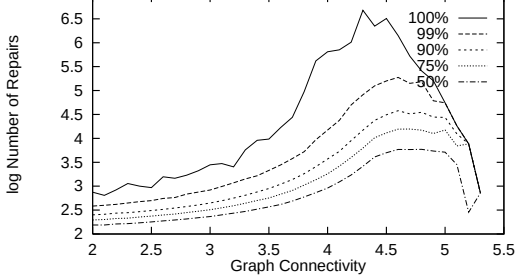

Figure 15: *Incomplete search method* GENET *on soluble 100 node graph 3-colourability problems.*

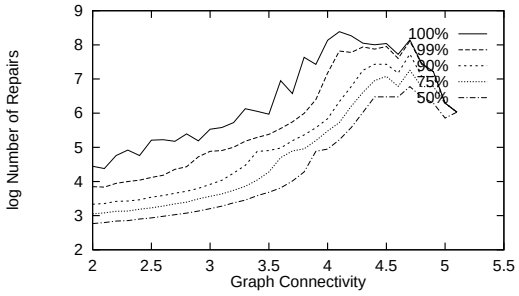

Figure 16: *Incomplete search methods* GSAT *on soluble 100 node graph 3-colourability problems.*

Figures 11 and 12 present results for FC-MWO on the 100-node graphs. At very low connectivities FC-MWO has the best performance of all the complete algorithms we looked at, finding no soluble or insoluble problems which could be called exceptionally hard. However at higher connectivities in the easy region FC-MWO has the worst performance out of all the complete algorithms. Why is this the case ? Freuder has shown [Freuder, 1982] that there exists a backtrack-free variable ordering for a CSP if the level of strong consistency in the CSP is greater than the width of the constraint graph. Graph colouring with k colours is always at least strongly k-consistent—thus graph 3-colourability is at least strongly path-consistent[5]. This means that any graph of width 2 can be solved with a backtrack-free search given a variable ordering of width 2—this will be found using the minimal width ordering. Sparse graphs at low connectivites are likely to have a small width, and it is these graphs that FC-MWO can solve without backtracking. These results indicate that at very low connectivities there were no *inherently* exceptionally hard problems for graph 3-colourability, since they could all be solved with backtrack-free search given a minimal width ordering.

Figures 13 and 14 compare the CPU time used by FC-MWO and FC-CBJ-FF in solving these problems. As we would expect, FC-MWO outperforms FC-CBJ-FF in parts of the easy region since it can solve without backtracking many of the problems which are exceptionally hard for FC-CBJ-FF. At very low connectivities the cost of finding a minimal width does not seem to be worthwhile. We were surprised to find that in parts of the hard region FC-MWO is slightly faster than FC-CBJ-FF, given the reputation of fail-first [Haralick and Elliot, 1980; Bacchus and Van Run, 1995]. It is also worthwhile noting that at some connecitivites FC-CBJ-FF used less consistency checks than FC-MWO, it still used more CPU time since it is using an $O(n)$ dynamic variable ordering heuristic.

The results for the incomplete search algorithms, GENET and GSAT, are given in Figures 15 and 16 for the soluble instances. We do not find any exceptionally hard, soluble problems for GENET or GSAT. In terms of CPU time GENET significantly outperforms GSAT (Figure 17), although again this may be entirely a result of the implementations. Figure 18 compares the mean CPU time required to solve the soluble problems for FC-CBJ-BZ and GENET. In the easy region GENET is faster than FC-CBJ-BZ, however FC-CBJ-BZ is more than an order of magnitude faster than GENET in the hard region.

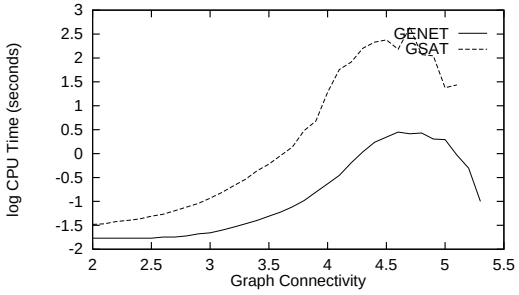

Figure 17: *Comparison of mean* CPU *time for* GENET *and* GSAT *to solve soluble 100 node graph 3-colourability problems.*

These results for 100 node problems are interesting since they show that the occurence of exceptionally hard problems is, at least to some extent, dependent upon the algorithm being used to solve them. This is in contrast to problems in the phase transistion, which appear to be hard for all algorithms. It is also interesting to observe that the incomplete search algorithms were clearly inferior to the best complete search algorithms on soluble problems in the hard region. Al-

[5]Given a consistent partial assignment for any two variables in the CSP we can always consistently extend this partial assignment to any other third variable.

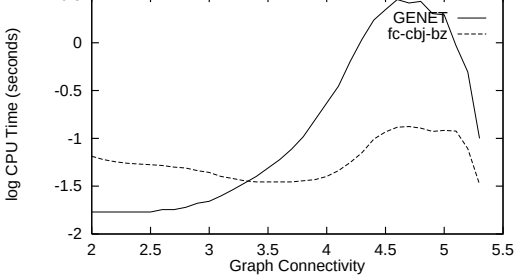

Figure 18: *Comparison of mean* CPU *time required by* FC-CBJ-BZ *and* GENET *to solve soluble 100 node graph 3-colourability problems.*

though differences in CPU time could be solely due to implementation details we do note that here the difference in CPU time between the best complete (FC-CBJ-BZ) and the best incomplete (GENET) search algorithm is well over an order of magnitude. To confirm that these results were not just due to the algorithm implementations we compared these two algorithms on larger graph colouring problems.

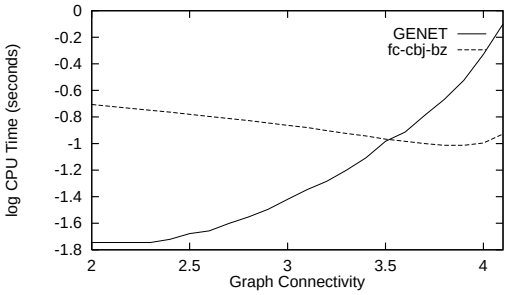

Figure 19: *Comparison of mean* CPU *time required by* FC-CBJ-BZ *and* GENET *to solve soluble 150 node graph 3-colourability problems.*

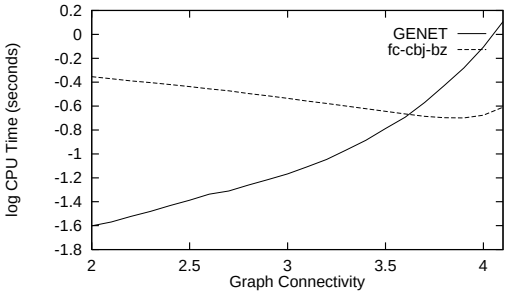

Figure 20: *Comparison of mean* CPU *time required by* FC-CBJ-BZ *and* GENET *to solve soluble 200 node graph 3-colourability problems.*

Figures 19, 20 and 21 compare the CPU time required by FC-CBJ-BZ and GENET in solving the soluble problems for 150, 200 and 250 node graphs. In order to get these results we had to restrict the graph connectivities to range from 2.0 to 4.1, which is mostly in the easy region of the graph colouring problem space. Despite this the results we did obtain do reinforce our

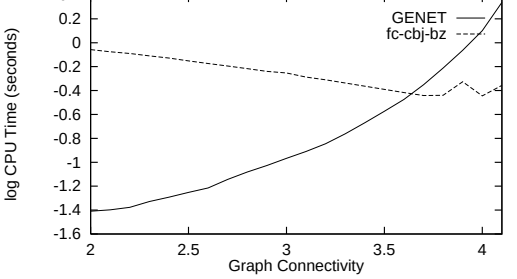

Figure 21: *Comparison of mean* CPU *time required by* FC-CBJ-BZ *and* GENET *to solve soluble 250 node graph 3-colourability problems.*

earlier findings for 100 node graphs. We see the following trends:

- the non-systematic search (GENET) significantly outperforms the systematic search (FC-CBJ-BZ) in the easy region. By significant we mean the difference in CPU time is over an order of magnitude;
- FC-CBJ-BZ significantly outperforms GENET in the hard region;
- there is a point where the CPU time required by FC-CBJ-BZ and GENET to solve these problems crossover;
- this crossover point appears to move to higher connectivities, towards the hard region, as we increase problem size—we would expect this behaviour to be asymptotic;
- the difference in CPU time in the easy region between FC-CBJ-BZ and GENET increases as we increase problem size.

6 Discussion

Smith [Smith, 1994a] identifies a number of possible reasons for the occurence of exceptionally hard problems:

"The problem has no solutions when most problems in the same region of the problem space have many solutions."

We have found insoluble problems in the easy region which can be exceptionally hard to prove insoluble for certain algorithms, but not for others. However we did find exceptionally hard insoluble problems for all the complete search algorithms we looked at.

"The problem has very few solutions."

In this case we would expect these problems to be hard for all algorithms since they must, on average, explore a larger portion of the search space before finding a solution. We did not find any such problems for the incomplete algorithms.

"The problem may have many solutions but they are clustered together in a limited region of the search space, and, because of the variable and value ordering, the algorithm will not reach any solutions until it has traversed most of the search space."

Smith [Smith, 1994a] cites this as the reason why some problems are inherently more difficult than others, independently of the number of solutions they may have. The cause of the difficulty is that the algorithm may select a wrong value for a variable early in the search and will not change this value until it has traversed the entire search tree below this variable. Although we did find soluble problems where FC-BZ and FC-CBJ-FF explored almost the whole search space before finding a solution, we did not find any such problems for FC-CBJ-BZ (although these problems may appear for FC-CBJ-BZ if we run it on larger problem samples). Thus it appears that by using good heuristics we can significantly reduce the impact of exceptioanlly hard problems of this kind on a complete algorithm. Non-systematic search algorithms such as GSAT and GENET do not have this problem since they do not need to be systematic in this way. Thus we would not expect these problems to be hard for this kind of search.

"The search space induced by the algorithm is exceptionally large, so that, whenever solutions occur, it will take a long time to reach the first one."

Smith backs up this argument with experimental results showing that the search space size can vary quite significantly depending on the constraint graph. However the algorithm used in these experiments was a chronological backtracking algorithm, forward-checking with fail-first variable ordering (FC-FF), which is susceptible to thrashing. This is going to result in a much larger search space, and explains the difference in the number of exceptionally hard problems found for FC-BZ which were not hard for FC-CBJ-BZ. We believe that by using backjumping rather than chronological backtracking we can significantly improve the performance of an algorithm on these kinds of exceptionally hard problems.

7 Conclusions

We have compared a number of well-known complete and incomplete constraint satisfaction search algorithms on graph colouring problems with the aim of determining whether one type of algorithm dominates in a particular region of the problem space with respect to the phase transistion.

One issue which affects this study is the phenomena of exceptionally hard problems. We found exceptionally hard problems for all the complete algorithms we looked at. However we also found that we can significantly reduce the impact of exceptionally hard problems on an algorithm by incorporating better heuristics and some form of intelligent backjumping. The best complete algorithm in our study for avoiding exceptionally hard problems was FC-CBJ-BZ. This was the only complete algorithm for which we found very little evidence of soluble exceptionally hard problems. We also found very little evidence of soluble exceptionally hard problems for the incomplete algorithms we looked at: GSAT and GENET. Our results indicate that the phenomena of exceptionally hard problems in the easy region of problem spaces is mainly an algorithm dependent one. This is in contrast to the phase transistion peak in median search cost, which is caused by problems that appear to be hard for all algorithms.

In terms of CPU time FC-CBJ-BZ was more than an order of magnitude slower on soluble problems than GENET in some parts of the easy region, and more than an order of magnitude faster in the hard region. We can close this gap in CPU time between complete and incomplete algorithms in the easy region by using cheaper heuristics in the complete search. However our results show that this will increase the likelihood of encountering exceptionally hard problems.

These results suggest that by sacrificing completeness we can gain significantly in performance in the easy region. However in the hard region we find that complete search is to be preferred, at least for graph colouring. It would be interesting to see whether similar results can be obtained for problems other than graph colouring, since the findings presented here are in contrast to that of work performed on 3-SAT, where incomplete GSAT is significantly better at solving soluble problems in the hard region [Selman and Kautz, 1993].

Acknowledgements

We would like to thank Edward Tsang, James Borrett and Alvin Kwan for useful discussions and comments on this paper. Andrew Davenport is supported by an Engineering and Physical Sciences Research Council Ph.D studentship award. This research has also been supported by an Engineering and Physical Sciences Research Council grant (ref. GR/H75275).

References

[Bacchus and Van Run, 1995] F. Bacchus and P. Van Run. Dynamic variable ordering in CSPs. In *Proceedings of Principles and Practice of Constraint Programming*, 1995. To appear.

[Brélaz, 1979] D. Brélaz. New methods to color the vertices of a graph. *Communications of the ACM*, 22(4):251–256, 1979.

[Cheeseman *et al.*, 1991] P. Cheeseman, B. Kanefsky, and W. Taylor. Where the really hard problems are. In *Proceedings of the 12th International Joint Conference on Artificial Intelligence*, volume 1, pages 331–337, 1991.

[Davenport *et al.*, 1994] A. J. Davenport, E. P. K. Tsang, C. J. Wang, and K. Zhu. GENET: A connectionist architecture for solving constraint satisfaction problems by iterative improvement. In *Proc, 12th National Conference on Artificial Intelligence*, volume 1, pages 325–330, 1994.

[Davenport, 1995] A. J. Davenport. *Iterative repair methods for solving constraint satisfaction prob-*

lems. PhD thesis, University of Essex, 1995. In preparation.

[Freuder and Hubbe, 1993] E. C. Freuder and P. D. Hubbe. Using inferred disjunctive constraints to decompose constraint satisfaction problems. In *Proceedings of IJCAI-93*, pages 254–260, 1993.

[Freuder and Wallace, 1992] E. C. Freuder and R. J. Wallace. Partial constraint satisfaction. *Artificial Intelligence Journal*, 58:21–70, 1992.

[Freuder, 1982] E. C. Freuder. A sufficient condition for backtrack-free search. *Journal of the Association for Computing Machinery*, 29(1):24–32, January 1982.

[Garey and Johnson, 1979] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W.H. Freeman and Company, New York, 1979.

[Gent and Walsh, 1994a] I. P. Gent and T. Walsh. Computational phase transistions in real problems. Technical Report 724, Department of Artificial Intelligence, Universirty of Edinburgh, 1994.

[Gent and Walsh, 1994b] I. P. Gent and T. Walsh. The SAT phase transistion. In *11th European Conference on Artificial Intelligence, Workshop on Constraint Processing*, pages 105–109, 1994.

[Gent and Walsh, 1994c] I.P. Gent and T. Walsh. Easy problems are sometimes hard. *Artificial Intelligence*, 70:335–345, 1994.

[Gent and Walsh, 1994d] I.P. Gent and T. Walsh. The hardest random SAT problems. In B. Nebel and L. Dreschler-Fischer, editors, *KI-94: Advances in Artificial Intelligence. 18th German Annual Conference on Artificial Intelligence*, pages 355–366. Springer-Verlag, 1994.

[Haralick and Elliot, 1980] R. M. Haralick and G. L. Elliot. Increasing tree search efficiency for constraint satisfaction problems. *Artificial Intelligence Journal*, 14:263–314, 1980.

[Hogg and Williams, 1994] T. Hogg and C.P. Williams. The hardest constraint problems: A double phase transistion. *Artificial Intelligence*, 69:359–377, 1994.

[Konolige, 1994] K. Konolige. Easy to be hard: Difficult problems for greedy algorithms. In J. Doyle, E. Sandewall, and P. Torasso, editors, *Principles of Knowledge Representation and Reasoning*, pages 374–378. Morgan-Kaufmann, 1994.

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

[Minton *et al.*, 1992] S. Minton, M. Johnston, A.B. Philips, and P. Laird. Minimizing conflicts: a heuristic repair method for constraint satisfaction and scheduling problems. *Artificial Intelligence*, 58:161–205, 1992.

[Mitchell *et al.*, 1992] D. Mitchell, B. Selman, and H. Levesque. Hard and easy distributions of SAT problems. In *Proc, 10th National Conference on Artificial Intelligence*, pages 459–465, 1992.

[Prosser, 1993] P. Prosser. Hybrid algorithms for the constraint satisfaction problem. *Computational Intelligence*, 9(3):268–299, 1993.

[Prosser, 1994a] P. Prosser. Binary constraint satisfaction problems: Some are harder than others. In *11th European Conference on Artificial Intelligence*, pages 95–99, 1994.

[Prosser, 1994b] P. Prosser. Exceptionally hard problems. USENET Article 3473dh$27c@todd-06.cs.strath.ac.uk, comp.constraints, September 1994.

[Selman and Kautz, 1993] B. Selman and H.A. Kautz. Domain independent extensions to GSAT: Solving large structured satisfiability problems. In *Proceedings of IJCAI-93*, 1993.

[Selman *et al.*, 1992] B. Selman, H. Levesque, and H.A. Kautz. A new method for solving hard satisfiability problems. In *Proc, 10th National Conference on Artificial Intelligence*. The American Association for Artificial Intelligence, AAAI Press/MIT Press, 1992.

[Selman *et al.*, 1994] B. Selman, H. A. Kautz, and B. Cohen. Noise strategies for improving local search. In *Proc, 12th National Conference on Artificial Intelligence*, volume 1, pages 337–343. The American Association for Artificial Intelligence, AAAI Press/The MIT Press, 1994.

[Smith and Grant, 1995] B. M. Smith and S. A. Grant. Sparse constraint graphs and exceptionally hard problems. In *Proc. IJCAI-95*, 1995.

[Smith, 1994a] B. M. Smith. In search of exceptionally difficult constraint satisfaction problems. In *11th European Conference on Artificial Intelligence, Workshop on Constraint Processing*, pages 79–86, 1994.

[Smith, 1994b] B. M. Smith. Phase transistion and the mushy region in constraint satisfaction problems. In *11th European Conference on Artificial Intelligence*, pages 100–104, 1994.

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

[Turner, 1988] J. S. Turner. Almost all k-colorable graphs are easy to color. *Journal of Algorithms*, 9:63–82, 1988.