Understanding the Empirical Hardness of Random Optimisation Problems [⋆]

Ciaran McCreesh$^{1[0000-0002-6106-4871]}$, William Pettersson$^{1[0000-0003-0040-2088]}$, and Patrick Prosser$^{1[0000-0003-4460-6912]}$

University of Glasgow, Glasgow, Scotland
{ciaran.mccreesh,william.pettersson,patrick.prosser}@glasgow.ac.uk

Abstract. We look at the empirical complexity of the maximum clique problem, the graph colouring problem, and the maximum satisfiability problem, in randomly generated instances. Although each is NP-hard, we encounter exponential behaviour only with certain choices of instance generation parameters. To explain this, we link the difficulty of optimisation to the difficulty of a small number of decision problems, which are already better-understood through phenomena like phase transitions with associated complexity peaks. However, our results show that individual decision problems can interact in very different ways, leading to different behaviour for each optimisation problem. Finally, we uncover a conflict between anytime and overall behaviour in algorithm design, and discuss the implications for the design of experiments and of search strategies such as variable- and value-ordering heuristics.

1 Introduction

The gap between the best theoretical understanding we have of what makes problems hard and the behaviour witnessed in practice from modern solvers remains vast. For many decision problems in random instances, we have a good general understanding of what happens: as a key parameter is altered, there is often sharp phase transition from satisfiable to unsatisfiable instances, and associated with this is a complexity peak, where instances near the transition are much harder to solve than those far from it on either side [6, 16]. (However, this behaviour is not universal—for example, problems involving more than one kind of constraint can exhibit much more complicated behaviour [7, 15]).

This paper looks at three optimisation problems: maximum clique, graph colouring, and maximum satisfiability. One view of an optimisation problem is as a sequence of decision problems—but is that all that is needed to understand their behaviour? Previous small-scale experiments [17, 12] have only been able to provide an incomplete picture. In this paper we perform experiments on tens

[⋆] This work was supported by the Engineering and Physical Sciences Research Council [grant numbers EP/P026842/1 and EP/P028306/1]. This work used the Cirrus UK National Tier-2 HPC Service at EPCC (http://www.cirrus.ac.uk) funded by the University of Edinburgh and EPSRC (EP/P020267/1).

of billions of problem instances, which is finally sufficient to comprehensively answer the question: yes, there is a link between individual decision and optimisation problems, but these decision problems can interact in many different ways, leading to complex emergent behaviour. Along the way, we uncover interesting implications for the design of search algorithms, and provide lessons for future experimenters. Most interestingly, we identify a trade-off between anytime behaviour and overall behaviour, which could ultimately encourage a rethink of the entire branch and bound paradigm.

1.1 Experimental Setup

Our experiments are performed on the EPCC Cirrus HPC facility, on systems with dual Intel Xeon E5-2695 v4 CPUs and 256GBytes RAM, running Centos 7.3.1611, with GCC 7.2.0 as the compiler. These machines are optimised for providing throughput rather than consistent timing measurements, so we avoid measuring runtimes, and instead use whichever natural measure of work each solver provides. Our results therefore do not allow for comparisons between different solvers.

In Section 2, we use the Glasgow Subgraph Solver implementation[1] of Prosser's MCSa1 [17]. This is a bit-parallel branch and bound algorithm, which uses a greedy colouring as its bound [22, 20]; it can easily be modified to solve the decision problem, rather than the optimisation problem. We measure instance difficulty by counting the number of recursive calls carried out by the algorithm. Later in the section, we also use the MoMC solver [11][2]. MoMC is a more modern branch and bound solver, which incorporates a number of search and inference strategies which are chosen dynamically.

In Section 3 we use Trick's implementation[3] of the classic DSATUR branch and bound algorithm [4], and Zhou et al.'s state of the art Color6 solver[4] [24] (which solves only the decision problem). For the Trick solver we measure the number of recursive calls made, whilst for Color6 we measure the number of backtracks. In Section 4 we use the Clasp solver[5] version 3.3.4 [8], and we measure the number of decisions made.

2 Maximum Clique

We begin by looking at the maximum clique problem. A clique in a graph is a subset of vertices, each of which is adjacent to every other within the subset, and a maximum clique is one with as many vertices as possible. For random graphs,

[1] https://github.com/ciaranm/glasgow-subgraph-solver

[2] https://home.mis.u-picardie.fr/~cli/EnglishPage.html. Our experiments uncovered bugs in the published version of this solver—thanks to its authors, our final results use a fixed version of this solver that is not currently publicly available.

[3] https://mat.gsia.cmu.edu/COLOR/color.html

[4] https://home.mis.u-picardie.fr/~cli/EnglishPage.html

[5] https://potassco.org/clasp/

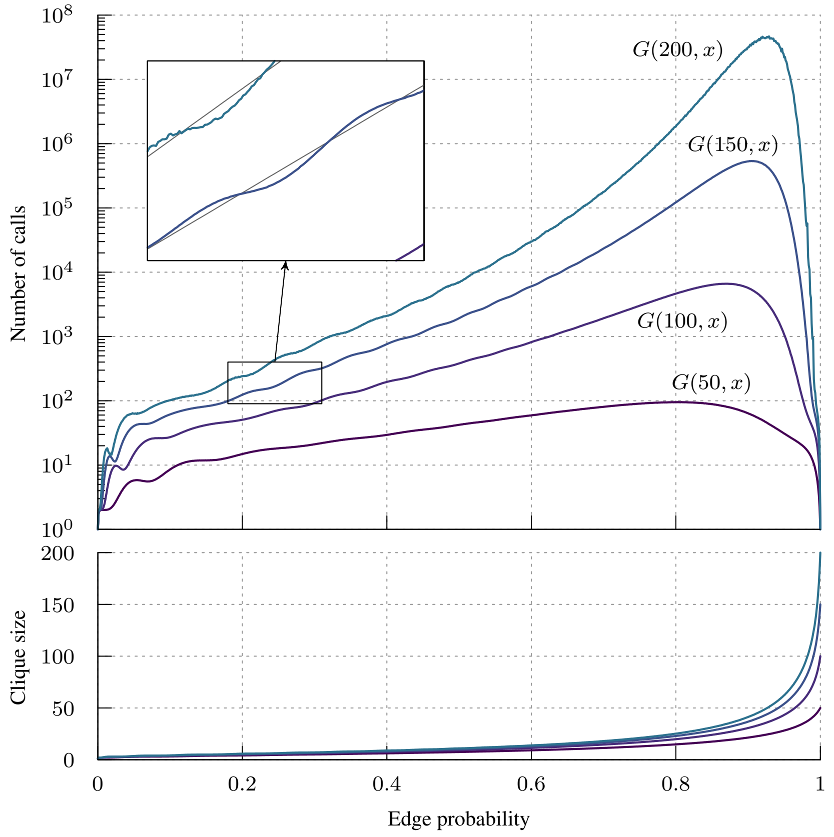


Fig. 1. On top, the difficulty of solving the maximum clique problem in random graphs $G(50, x)$, $G(100, x)$, $G(150, x)$, and $G(200, x)$. Underneath, the mean size of an optimal solution. Density is increased in steps of 0.001 with 100,000 samples per step for the three smaller families, and 1,000 per step for the largest.

we use the Erdős-Rényi model: by $G(n, p)$ we mean a graph with n vertices, and an edge between every distinct pair of vertices with probability p. Clique-finding in Erdős-Rényi graphs is known to be exponentially difficult for current clique algorithms [3].

2.1 Maximum Cliques in Random Graphs

In Figure 1 we show the difficulty of solving the maximum clique problem as we vary the edge probability in Erdős-Rényi graphs with a fixed number of vertices, as well the mean size of an optimal solution. In extremely sparse and extremely dense graphs, the algorithm finds all instances extremely easy, whilst at around densities of 0.8 to 0.96, instances are particularly hard—and unsurprisingly, as the number of vertices increases, all densities get exponentially harder.

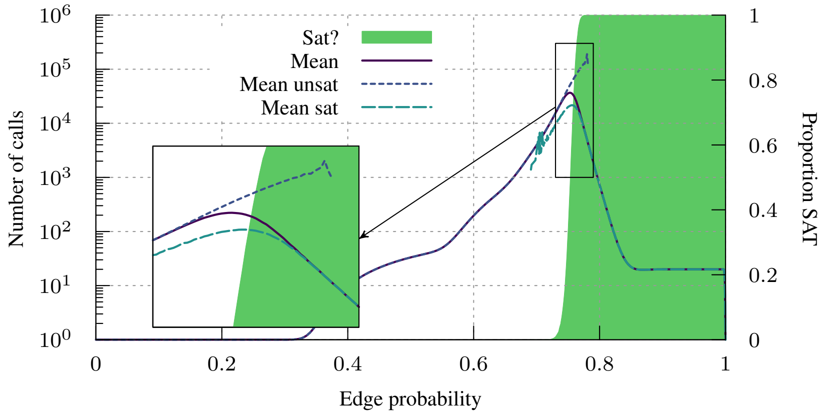


Fig. 2. Does $G(150, x)$ contains a clique of 20 vertices? Mean search effort for satisfiable and unsatisfiable instances are also shown separately. Density is increased in steps of 0.001, with 100,000 samples per step.

These rough trends match up with those presented by Prosser [17]. However, we are using a *much* larger number of instances: we increase density in steps of 0.001, and take 100,000 samples per density step. This scale of experiments reveals a new interesting feature of the plots: the lines are, for lack of a better term, wiggly. This is most readily apparent towards the left of the graph, where several slight peaks and troughs are easily visible by eye, but in fact the wiggles are present throughout the entire plot, with a decreasing "wavelength" as density increases. The remainder of this section shows that these wiggles are not an experimental artifact or sampling error, but instead illustrate an important aspect of the algorithm's behaviour.

2.2 The Clique Decision Problem

To understand what is going on, we first revert to the clique decision problem. In Figure 2 we ask whether $G(150, x)$ contains a clique of twenty vertices. For very sparse graphs, the answer is obviously no, and the solver can establish this with no search effort. For very dense graphs, the answer is obviously yes, and the solver similarly finds all instances easy. For densities in between 0.691 and 0.782, there is a mix of satisfiable and unsatisfiable instances, but these instances are hard for the solver. For unsatisfiable instances, the higher the density the harder the instance, and the hardest density is 0.780, where all but one of the 100,000 instances sampled are satisfiable. Unexpectedly, for satisfiable instances, we do not get a hard—easy curve, but rather a medium—hard—easy peak, with the hardest density being 0.756 where 62,587 instances were satisfiable. Instances in the "medium" region are extremely rare, however.

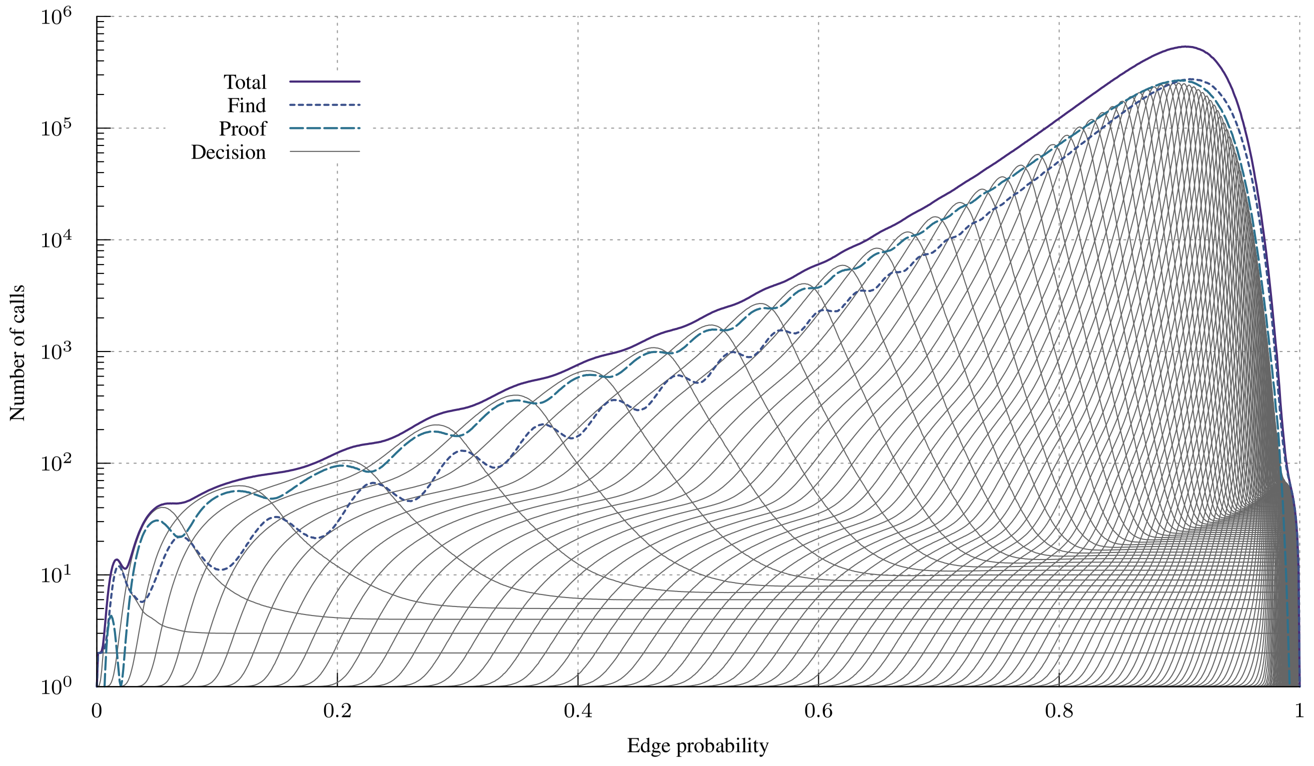


Fig. 3. A more detailed picture of difficulty of solving the clique optimisation problem for $G(150, x)$. Also plotted is the mean search effort to find the optimal solution but not prove its optimality, and the mean search effort needed to prove optimality after the optimal solution is found. Finally, each light line shows the mean search effort for a single decision problem. For each line, density is increased in steps of 0.001, with 100,000 samples per step.

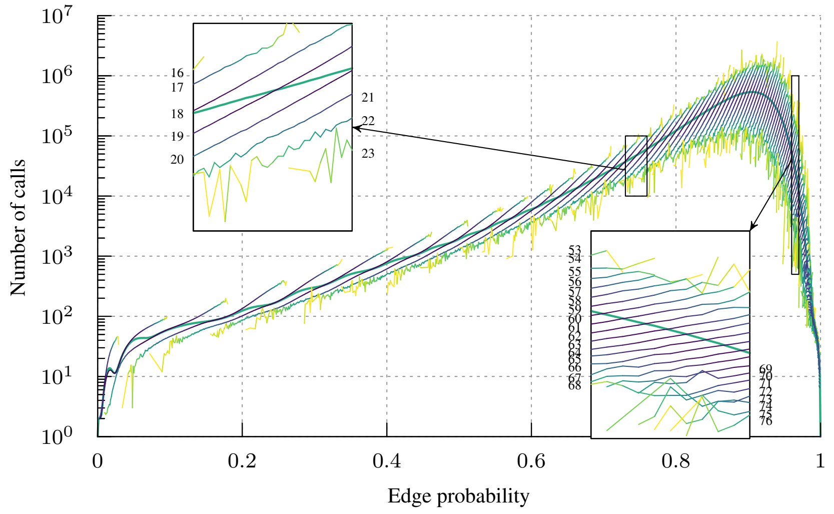


Fig. 4. The mean difficulty of solving the clique optimisation problem for $G(150, x)$, also showing the search effort for each actual optimal size. On each of the individual value lines, darker colours represent exponentially larger sample sizes. Density is increased in steps of 0.001, with 100,000 samples per step.

2.3 Decision and Optimisation

In Figure 3 we simultaneously plot the difficulty of every decision problem, and show how this correlates with the total search effort seen in Figure 1. The "total" line is usually only slightly above whichever decision line is the hardest at a particular density—even at the hardest density of 0.905, the mean gap between the hardest decision problem and the overall cost of solving is only a factor of 2.2. This explains the wiggly lines: they are the result of the gaps between the complexity peaks of different decision problems.

Figure 3 also breaks down the runtimes to show the mean time to find an optimal solution but not prove its optimality, and the time to prove optimality once an optimal solution has already been found. These two lines are perfectly out of phase with each other: densities where finding a solution is relatively easy are the hardest for proving optimality, and vice-versa.

2.4 Difficulty by Actual Solution Size

Another way of grouping results is presented in Figure 4. Alongside a plot of mean search effort, we also show mean search effort only considering instances where the maximum clique has ω vertices, for each value of ω. The darkness of each line indicates the relative sample size. The plot shows that at any given density, there are several common solution sizes, and the difficulty varies considerably depending upon what the optimal solution size actually is. It also shows

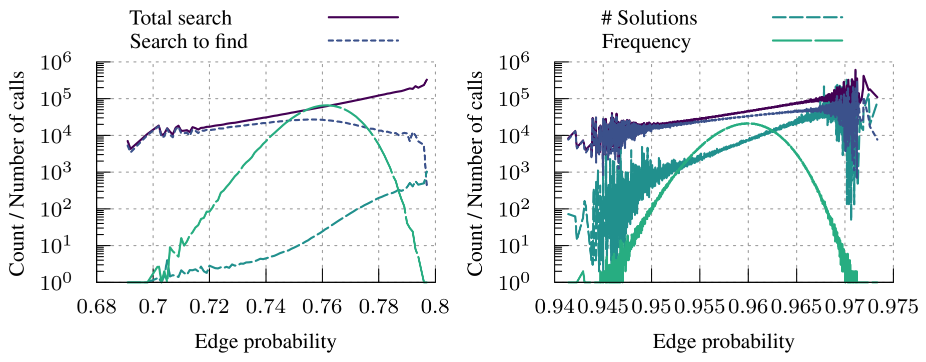


Fig. 5. On the left, looking at only instances where the maximum clique has twenty vertices in $G(150, x)$, and showing the mean search effort, mean time to find but not prove optimality, the frequency of such instances, and the mean number of times a clique of that size occurs in any selected instance. On the right, the same, for a maximum clique of sixty vertices. Density is increased in steps of 0.001 (twenty) or 0.00001 (sixty), with 100,000 samples per step.

that, for any particular maximum clique size ω, there are unusually low densities where occasionally this is the optimum, and these instances are very easy. There are also rare unusually high densities where this is the optimum, and these instances are very hard. Finally, for densities in the middle, instances with solution size ω are common, and are of moderate difficulty. Alternatively, for a given instance, if the maximum clique size is unexpectedly large, the instance will be relatively easy, whilst if it is unexpectedly small, it will be unusually hard.

2.5 How Common are Optimal Solutions?

Recall that typically, proving optimality is many times harder than finding an optimal solution. If an instance has an unusually large optimal solution, this should make the proof of optimality much easier. But what about *finding* this unusually large optimum? We might expect that there will only be one optimal solution, if the optimum is unusually large, whilst if the optimum is unusually small, perhaps there are many witnesses to choose from?

In the left-hand plot of Figure 5 we show that this is the case, looking only at instances where twenty is the optimal solution. We plot the frequency of optimal solutions (how common they are, by density), as well as the effort required to find a first optimal solution but not prove its optimality, and the effort to both find and prove optimality. Finally, we also solve the *maximum* clique enumeration problem, and count how many such optimal solutions exist.

Towards the left of this plot, with densities up to 0.72, instances with a maximum clique size of twenty are rare. Furthermore, the total number of optimal solutions (witnesses) in any given instance is very low, often being one or only a few—and nearly all of our search effort is spent finding the optimal, with op-

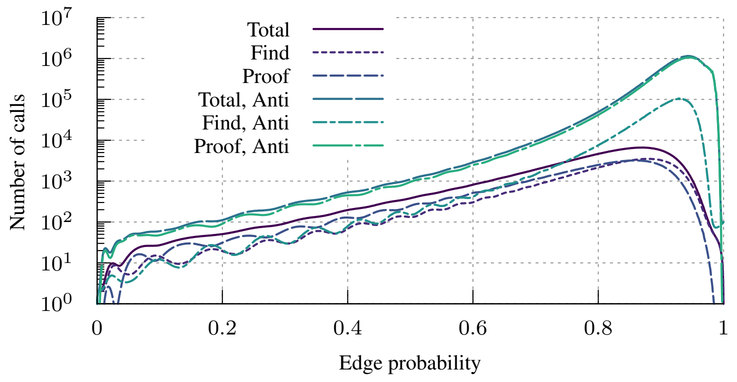


Fig. 6. The mean difficulty of solving the clique optimisation problem for $G(150, x)$, using both the standard search heuristic order for the algorithm, and the opposite search order. For each line, density is increased in steps of 0.001, with 100,000 samples per step.

timality proofs being easy. As the density rises, the typical number of optimal solutions per instance also rises, and the time to find but not prove optimality makes up smaller and smaller portions of the overall runtime.

Interestingly, there is not a straightforward inverse relationship between the number of optimal solutions and the amount of time required to find an optimal solution. Rather, finding the unique optimal solution in a lower density graph is somewhat easier than finding any one of several optimal solutions in a medium density graph, and it is not until much higher densities that finding becomes easier again. This is similar to the "medium–hard–easy" complexity peak seen in Figure 2. One could conjecture that this is because higher densities are harder overall than lower densities. However, the right-hand plot of Figure 5 looks at instances where sixty is the optimal solution, with densities between 0.94 and 0.975. At this stage, higher densities are easier overall—but the same pattern occurs.

2.6 Anytime Behaviour

To explain this behaviour, we now demonstrate that the algorithm is in fact not optimised for anytime behaviour, but rather aims to make the proof of optimality as short as possible. McCreesh and Prosser [14] observe that the branching strategy used by this algorithm approximates "smallest domain first" [10], and that (contrary to the claims of the algorithm's designers) it is not good at finding a strong incumbent quickly. So what if we reverse the branching strategy used by the algorithm? Figure 6 compares the behaviour of the heuristic and the anti-heuristic, showing that the anti-heuristic performs much worse except on the easiest of instances. However, in Figure 7, we plot the mean size of the first solution found by both heuristics, as a proportion of the optimal: despite

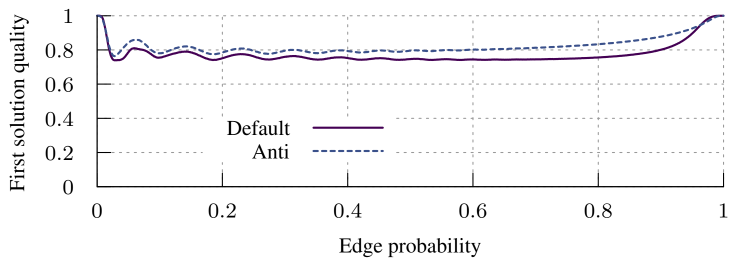


Fig. 7. The mean size of the first solution found expressed as a proportion of the optimal, for $G(100, x)$, using both the standard search heuristic order for the algorithm, and the opposite search order. For each line, density is increased in steps of 0.001, with 100,000 samples per step.

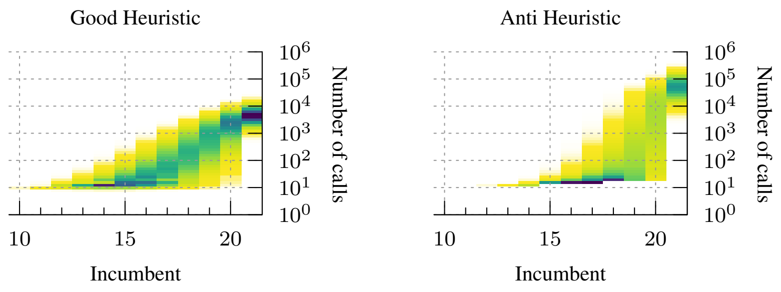


Fig. 8. Comparing the solution quality over time for the two different search orders. We look at instances of $G(100, x)$ where the maximum clique has twenty vertices. Each time a new incumbent is found, we record the search effort so far; each grid point shows the relative frequency of new incumbents of that size during that time window, with darker colours being exponentially more common. We also show termination time, in the final column. Instances drawn from a run with density increased in steps of 0.001, with 100,000 samples per step.

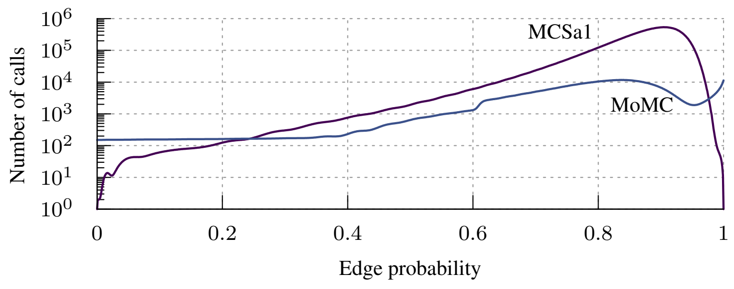


Fig. 9. The difficulty of solving the maximum clique problem in $G(150, x)$ using two different solvers. (This plot does not compare runtimes—the rate of recursive calls per second is much lower in MoMC.)

being much worse overall, the anti-heuristic finds a better first solution in nearly all cases.

To understand this seemingly contradictory behaviour, we compare the size of the incumbent as a function of time for the two algorithms. In Figure 8 we select all the instances of $G(150, x)$ where the optimal solution had twenty vertices, and for both heuristics, record a timepoint for each time the incumbent is improved. We also record when the algorithm terminates, representing this as an incumbent of twenty-one. We then convert this to a heatmap by bucketing, using darker colours to represent exponentially larger buckets. The plot shows us that with the good heuristic, the initial solution size is lower (most commonly fourteen or fifteen) compared to the anti-heuristic (most commonly sixteen to eighteen), but that the anti-heuristic then becomes slower to advance, and slower still to finally prove optimality. This suggests that the anti-heuristic's branching choices cause it to become trapped in larger subproblems before it can advance to a better region of the search space. In contrast, the good heuristic tries to eliminate as many subproblems as possible, even at the expense of much less favourable anytime behaviour.

This observation also explains Figure 5: the algorithm does not spend nearly all of its time attempting to find an unusually large optimal solution in a sparser instance because this solution is rare, but rather because it is instead spending all of its time eliminating the remaining portions of the search space. As density increases, the remaining portion of the search space increases, explaining the increase in difficulty despite the higher solution counts.

2.7 Solver Independence

What about other solvers? Is what we are seeing merely a quirk of the MCSa1 algorithm, or is it more widespread? In Figure 9 we repeat parts of Figure 1, showing the difficulty of solving $G(150, x)$ using the MoMC solver [11].[6] Again, we see wiggles in the curve rather than a smooth straight line, but we also see three other odd features that are not present in the MCSa1 curve. Firstly, MoMC will always require at least 150 recursive calls (and more generally, it requires at least one recursive call per vertex in the input graph). Secondly, there is a sharp change in behaviour around density 0.60—this is because MoMC switches search strategy based upon the density of the input graph, and has this critical density as a hard-coded parameter. And thirdly, MoMC struggles with extremely dense graphs. (Further experiments could have uncovered a fourth oddity: MoMC also switches search strategy when the input graph has more than a thousand vertices.) Despite this, the general dependency between optimisation and underlying decision problems remains.

[6] We stress that comparing the number of recursive calls between two different algorithms is not a measure of which algorithm is faster—indeed, MoMC performs much more work per recursive call, and on our hardware and on these random instances, is the slower algorithm outwith densities 0.89 to 0.95, despite the lower number of recursive calls.

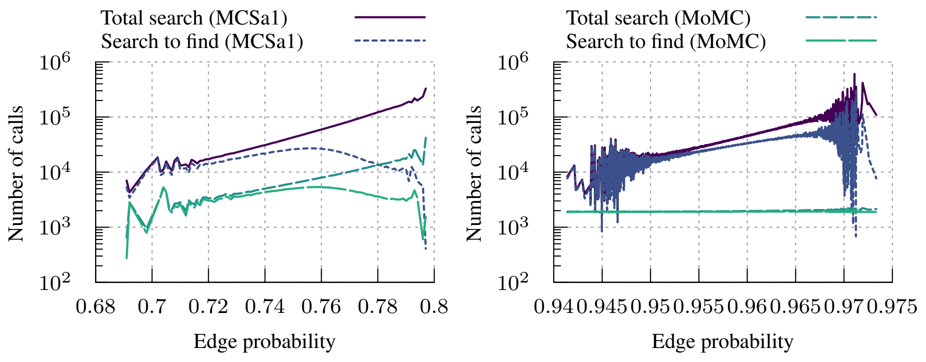


Fig. 10. Repeating Figure 5 using two solvers. On the left, instances where the optimal solution for $G(150, x)$ has twenty vertices, and on the right, sixty vertices.

In Figure 10 we repeat parts of Figure 5, now showing the time to find and the total time for both MCSa1 and MoMC. For solution size twenty instances, the behaviours are remarkably close, despite MoMC using a very different set of search heuristics—in particular, as the number of optimal solutions increases, MoMC initially takes longer to find a witness. For solution size sixty instances, the scaling factor is different, but still MoMC spends nearly all of its time during search having not found an optimal solution, even when witnesses are extremely common.

2.8 Algorithm Design Implications

These results show that clique algorithms have been optimised for proofs of optimality, at the expense of worse anytime behaviour—and also that, if the algorithms were better at finding strong solutions quickly, then their performance would improve considerably on certain instances. It is therefore worth considering whether it is possible to modify these algorithm for both good anytime behaviour, and good overall performance. However, adapting search order heuristics does not appear to help: although doing so can help an algorithm find stronger solutions faster, it then quickly becomes stuck in a subproblem that is hard to eliminate.

Other alternatives may be possible. For example, Maslov et al. [13] apply an iterated local search (ILS) heuristic to generate an initial solution, rather than starting from zero. This technique was also adopted by Tomita et al. [23], who use a different form of local search to prime the incumbent. Both papers describe this as assistance, rather than recognising that their exact algorithms are not optimised for finding strong solutions quickly; both papers also have difficulties selecting a principled amount of time to spend running local search before starting the exact algorithm. Both papers also claim large successes (sometimes being thousands or millions of times faster), particularly on certain families from the standard DIMACS benchmark suite. However, a close inspection of the instances where this happens shows that all come from crafted families that are designed to have unusually large hidden optimal solutions [5, 18, 19], rather than from application instances.

3 Graph Colouring

Having looked in detail at the maximum clique problem, we now repeat some of our experiments using solvers for the graph colouring problem: we must give a colour to each vertex in a graph, giving adjacent vertices different colours, and using as few colours as possible.

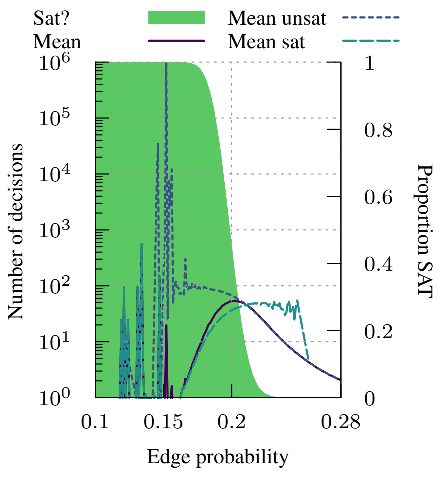
3.1 A Phase Transition, and Outliers

In the left of Figure 11 we show the difficulty of five-colouring $G(60, x)$, for varying values of x, using the Color6 solver. For densities in between 0.16 and 0.23, we encounter a mix of satisfiable and unsatisfiable instances, and the solver finds the instances more difficult than those outside of this density range. However, at much lower densities, comfortably inside the "satisfiable" region, the mean search effort is extremely variable, and is sometimes far higher than at the complexity peak. Looking more closely at the data shows that for densities between 0.11 and 0.15, between one in ten thousand and one in a hundred thousand instances that we generate are tens of millions of times harder than typical (and this rarity explains why Mann's [12] experiments did not uncover them). Furthermore, rather than being entirely satisfiable, these instances are a mix of satisfiable and unsatisfiable. Such instances also occur for other values of the decision problem, although it appears to be even less common as the objective value increases. A similar phenomenon occurs with other random satisfaction problems [21, 1], and it could potentially be alleviated by the use of restarts and randomisation [9].

3.2 Branch and Bound

In the right-hand plot of Figure 11 we show the difficulty of each decision problem together, but exclude these outliers from calculating the means. As for the clique problem, we observe wiggles, with the problem getting easier then harder then easier then harder and so on as we pass successive complexity peaks. The Color6 solver only supports the decision problem. Thus, we also plot the classic DSATUR branch and bound algorithm (whose performance is somewhat worse overall). As with the maximum clique algorithm, the mean complexity line goes from easy to hard to easy over the full range of densities, but this peak has wiggles that line up with the objective values changing.

We also break down the behaviour of the DSATUR solver by optimal solution size: we show this in Figure 12, in the same style as Figure 4. Because we are dealing with a minimisation problem, instances that are relatively sparse for their solution size are now found to be harder, rather than easier. And, as with Color6, DSATUR also occasionally finds very sparse instances very hard.



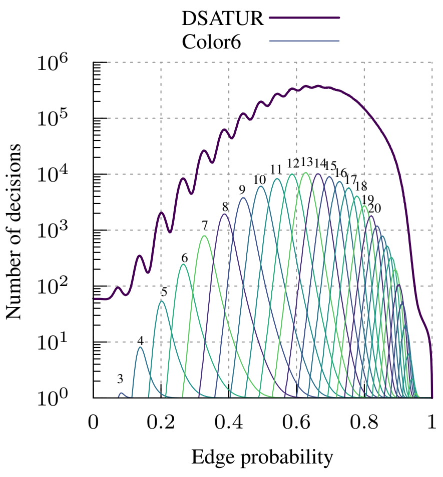


Fig. 11. On the left, the five-colouring phase transition in $G(60, x)$, using the Color6 solver. On the right, the difficulty of each colouring decision problem in $G(60, x)$, using Color6, with outliers removed; the top line is the minimisation problem, using Trick's DSATUR. For each line, density is increased in steps of 0.001, with 100,000 samples per step.

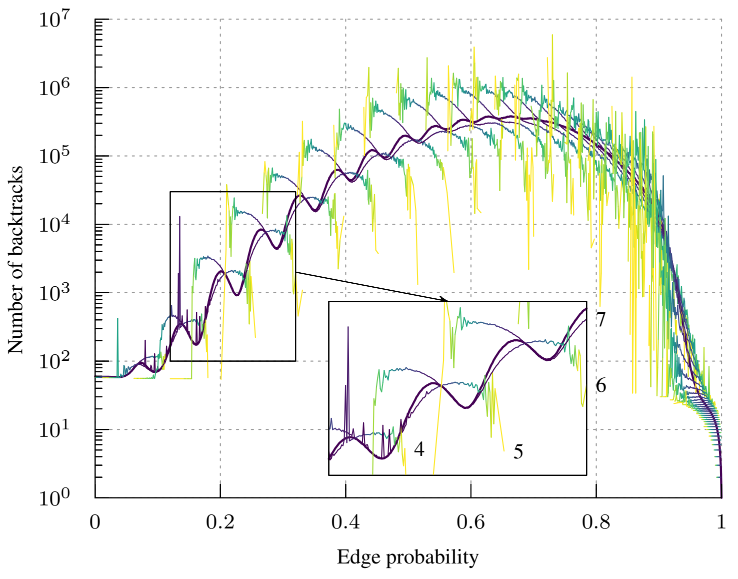


Fig. 12. The difficulty of the minimum colouring problem in $G(60, x)$, using the Trick solver, also showing the means only for instances with each individual optimal solution. Density is increased in steps of 0.001, with 100,000 samples per step.

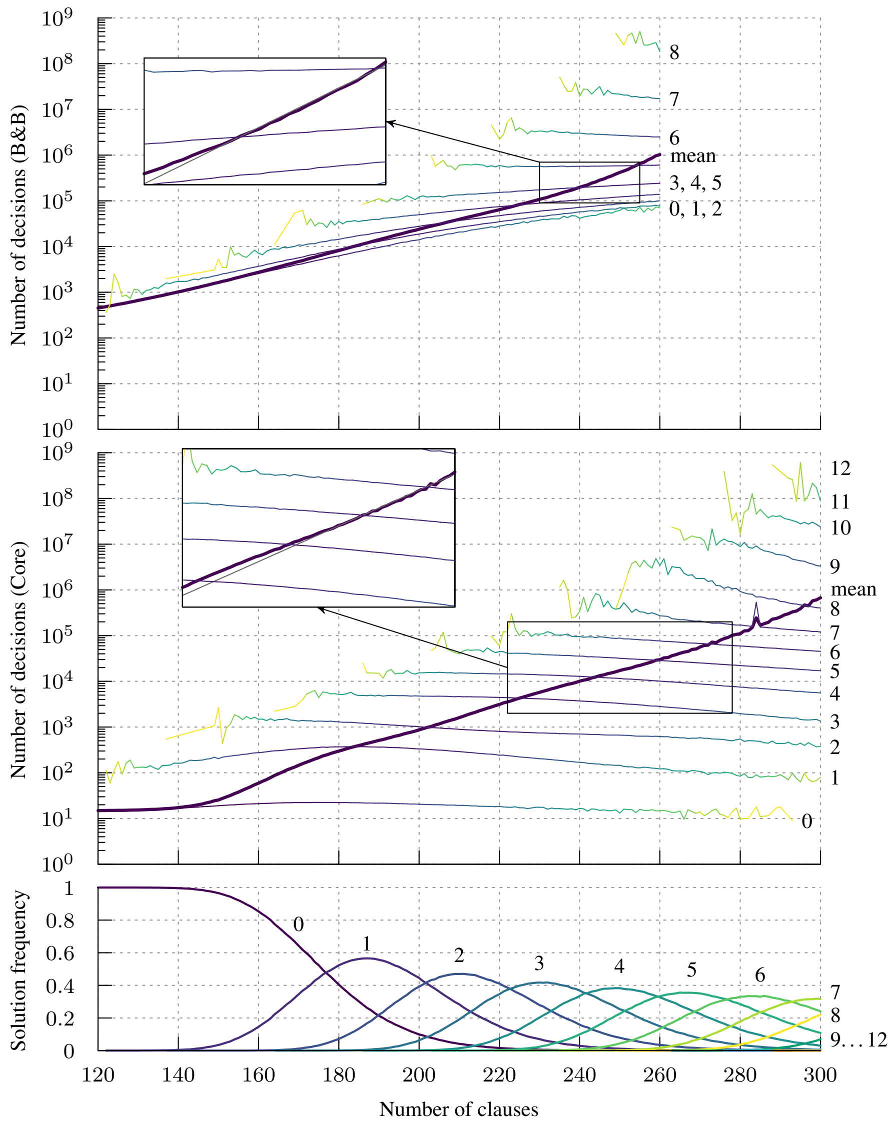


Fig. 13. The difficulty of the maximum 3-satisfiability problem in random instances with 40 variables using Clasp in branch and bound mode (top) and in core-guided mode (middle). The number of clauses is increased in steps of one, and there are 100,000 samples per step; the core-guided plot omits six instances with 290 or more clauses that timed out after one day. Results only for each particular objective value (i.e. the number of unsatisfiable clauses) are also shown as smaller lines in both plots. On the bottom, we show how common each objective value is.

4 Maximum Satisfiability

We finish with a brief look at random maximum satisfiability, or MaxSAT. To generate random MaxSAT instances, we use 40 variables, and a varying number of clauses. Each clause contains three distinct variables chosen uniformly at random, and the polarity of each variable in each clause is also set uniformly at random; all clauses are soft with equal weight. We plot our results in Figure 13, showing both Clasp's default branch and bound mode, and core-guided optimisation [2], which performs better. Although harder to see, the mean search effort lines for both configuration do exhibit wiggles. Interestingly, the relative difficulty of different instances depends upon the search strategy used—we believe this warrants further experimentation.

5 Discussion and Conclusion

By using very large sample sizes, we have demonstrated that the behaviour of solvers on hard optimisation problems is indeed influenced by the behaviour on individual decision problems—but that these decision problems can interact in many different ways. We also uncovered several interesting phenomena that happened only for one in every ten thousand instances (or even fewer). We therefore encourage future experiments to use similarly large sample sizes if possible, and to consider running many relatively easy experiments instead of a small number of experiments on instances that are as large as possible.

A further advantage of this approach is in uncovering bugs. Indeed, during our experiments, we found that the published version of the MoMC solver produced incorrect results for approximately one in every hundred thousand instances. In fact it was relying upon incorrect reasoning much more frequently than this, but would usually produce the correct answer anyway—the bug only became evident in instances with one or a very small number of witnesses for the optimal clique size, and only if a large combination of events caused the subtree containing these witnesses to be eliminated prematurely and without the witness being found by other means.

Our experiments also uncovered a conflict between designing search order heuristics for anytime behaviour or for overall performance in branch and bound algorithms, which explains why recent exact clique algorithms are using priming with local search algorithms, and which has implications for the design of future solvers. This conflict should also be recognised by experimenters when comparing algorithms in the future—in particular, we would be wary of tables of results that present both "number of instances solved" and "average solution size found" for a single arbitrary choice of timeout.

Acknowledgments

The authors would like to thank Gary Curran, Sonja Kraiczy, David Manlove, Kitty Meeks, and James Trimble for their comments and inspiration.

References

1. Achlioptas, D., Beame, P., Molloy, M.S.O.: A sharp threshold in proof complexity. In: Proceedings on 33rd Annual ACM Symposium on Theory of Computing, July 6-8, 2001, Heraklion, Crete, Greece. pp. 337–346 (2001). https://doi.org/10.1145/380752.380820
2. Andres, B., Kaufmann, B., Matheis, O., Schaub, T.: Unsatisfiability-based optimization in clasp. In: Technical Communications of the 28th International Conference on Logic Programming, ICLP 2012, September 4-8, 2012, Budapest, Hungary. pp. 211–221 (2012). https://doi.org/10.4230/LIPIcs.ICLP.2012.211
3. Atserias, A., Bonacina, I., de Rezende, S.F., Lauria, M., Nordström, J., Razborov, A.A.: Clique is hard on average for regular resolution. In: Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2018, Los Angeles, CA, USA, June 25-29, 2018. pp. 866–877 (2018). https://doi.org/10.1145/3188745.3188856
4. Brélaz, D.: New methods to color vertices of a graph. Commun. ACM **22**(4), 251–256 (1979). https://doi.org/10.1145/359094.359101
5. Brockington, M., Culberson, J.C.: Camouflaging independent sets in quasi-random graphs. In: Cliques, Coloring, and Satisfiability, Proceedings of a DIMACS Workshop, New Brunswick, New Jersey, USA, October 11-13, 1993. pp. 75–88 (1993)
6. Cheeseman, P.C., Kanefsky, B., Taylor, W.M.: Where the really hard problems are. In: Proceedings of the 12th International Joint Conference on Artificial Intelligence. Sydney, Australia, August 24-30, 1991. pp. 331–340 (1991)
7. Dudek, J.M., Meel, K.S., Vardi, M.Y.: The hard problems are almost everywhere for random CNF-XOR formulas. In: Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI 2017, Melbourne, Australia, August 19-25, 2017. pp. 600–606 (2017). https://doi.org/10.24963/ijcai.2017/84
8. Gebser, M., Kaufmann, B., Schaub, T.: Conflict-driven answer set solving: From theory to practice. Artif. Intell. **187**, 52–89 (2012). https://doi.org/10.1016/j.artint.2012.04.001
9. Gomes, C.P., Selman, B., Kautz, H.A.: Boosting combinatorial search through randomization. In: Proceedings of the Fifteenth National Conference on Artificial Intelligence and Tenth Innovative Applications of Artificial Intelligence Conference, AAAI 98, IAAI 98, Madison, WI., USA. pp. 431–437 (1998)
10. Haralick, R.M., Elliott, G.L.: Increasing tree search efficiency for constraint satisfaction problems. Artif. Intell. **14**(3), 263–313 (1980). https://doi.org/10.1016/0004-3702(80)90051-X
11. Li, C., Jiang, H., Manyà, F.: On minimization of the number of branches in branch-and-bound algorithms for the maximum clique problem. Computers & OR **84**, 1–15 (2017). https://doi.org/10.1016/j.cor.2017.02.017
12. Mann, Z.Á.: Complexity of coloring random graphs: An experimental study of the hardest region. ACM J. of Experimental Algorithmics **23** (2018)
13. Maslov, E., Batsyn, M., Pardalos, P.M.: Speeding up branch and bound algorithms for solving the maximum clique problem. J. Global Optimization **59**(1), 1–21 (2014). https://doi.org/10.1007/s10898-013-0075-9
14. McCreesh, C., Prosser, P.: Reducing the branching in a branch and bound algorithm for the maximum clique problem. In: Principles and Practice of Constraint Programming - 20th International Conference, CP 2014, Lyon, France, September 8-12, 2014. Proceedings. pp. 549–563 (2014). https://doi.org/10.1007/978-3-319-10428-7_40

15. McCreesh, C., Prosser, P., Solnon, C., Trimble, J.: When subgraph isomorphism is really hard, and why this matters for graph databases. J. Artif. Intell. Res. **61**, 723–759 (2018). https://doi.org/10.1613/jair.5768, https://doi.org/10.1613/jair.5768
16. Mitchell, D.G., Selman, B., Levesque, H.J.: Hard and easy distributions of SAT problems. In: Proceedings of the 10th National Conference on Artificial Intelligence, San Jose, CA, USA, July 12-16, 1992. pp. 459–465 (1992)
17. Prosser, P.: Exact algorithms for maximum clique: A computational study. Algorithms **5**(4), 545–587 (2012). https://doi.org/10.3390/a5040545
18. Sanchis, L.A.: Test case construction for the vertex cover problem. In: Computational Support for Discrete Mathematics, Proceedings of a DIMACS Workshop, Piscataway, New Jersey, USA, March 12-14, 1992. pp. 315–326 (1992)
19. Sanchis, L.A.: Generating hard and diverse test sets for NP-hard graph problems. Discrete Applied Mathematics **58**(1), 35–66 (1995). https://doi.org/10.1016/0166-218X(93)E0140-T
20. Segundo, P.S., Matía, F., Rodríguez-Losada, D., Hernando, M.: An improved bit parallel exact maximum clique algorithm. Optimization Letters **7**(3), 467–479 (2013). https://doi.org/10.1007/s11590-011-0431-y
21. Smith, B.M., Grant, S.A.: Modelling exceptionally hard constraint satisfaction problems. In: Principles and Practice of Constraint Programming - CP97, Third International Conference, Linz, Austria, October 29 - November 1, 1997, Proceedings. pp. 182–195 (1997). https://doi.org/10.1007/BFb0017439
22. Tomita, E., Sutani, Y., Higashi, T., Takahashi, S., Wakatsuki, M.: A simple and faster branch-and-bound algorithm for finding a maximum clique. In: WALCOM: Algorithms and Computation, 4th International Workshop. Proceedings. pp. 191–203 (2010). https://doi.org/10.1007/978-3-642-11440-3_18
23. Tomita, E., Yoshida, K., Hatta, T., Nagao, A., Ito, H., Wakatsuki, M.: A much faster branch-and-bound algorithm for finding a maximum clique. In: Frontiers in Algorithmics, 10th International Workshop, FAW 2016, Qingdao, China, June 30-July 2, 2016, Proceedings. pp. 215–226 (2016). https://doi.org/10.1007/978-3-319-39817-4_21
24. Zhou, Z., Li, C.M., Huang, C., Xu, R.: An exact algorithm with learning for the graph coloring problem. Computers & OR **51**, 282–301 (2014). https://doi.org/10.1016/j.cor.2014.05.017