Where are the hard problems?
2019

An Empirical Study of the Stable Marriage Problem with Ties and Incomplete Lists [1]

Ian Philip Gent [2] and **Patrick Prosser** [3]

Abstract. We present the first complete algorithm for the SMTI problem, the stable marriage problem with ties and incomplete lists. We do this in the form of a constraint programming encoding of the problem. With this we are able to carry out the first empirical study of the complete solution of SMTI instances. In the stable marriage problem (SM) [?] we have n men and n women. Each man ranks the n women, giving himself a preference list. Similarly each woman ranks the men, giving herself a preference list. The problem is then to marry men and women such that they are *stable* i.e. such that there is no incentive for individuals to divorce and elope. This problem is polynomial time solvable. However, when preference lists contain ties and are incomplete (SMTI) the problem of determining if there is a stable matching of size n is then NP-complete, as is the optimisation problem of finding the largest or smallest stable matching [?, ?]. In this paper we present constraint programming solutions for the SMTI decision and optimisation problems, a problem generator for random instances of SMTI, and an empirical study of this problem.

1 Introduction

In the stable marriage problem [?] we have n men and n women. Each man ranks the n women, giving himself a preference list. Similarly each woman ranks the men, giving herself a preference list. The problem is then to marry men and women such that they are *stable*. By stable we mean that there is no incentive for individuals to divorce and elope. For example, a matching would be unstable if it contained the marriages Romeo to Isobel and John to Juliet, where Romeo prefers Juliet to Isobel, and Juliet prefers Romeo to John, i.e. Romeo and Juliet would elope. This problem has a long history, and an optimal algorithm was proposed by Gale and Shapley almost 40 years ago [?]. The algorithm's complexity is $O(n^2)$, and is linear in the size of the problem, where size is measured in terms of the n people each with a preference list of size n.

If men or women find some members of the opposite sex unacceptable, preference lists become incomplete. These problems are classified as stable marriage problems with incomplete lists (SMI) and are again solvable in polynomial time. We might also have ties in the preference lists. That is, a man (or a woman) might be indifferent between a number of his (or her) choices. For example John might have a preference such that he prefers Isobel to Jane, but Jane

the stable marriage problem with ties and incomplete lists (SMTI). We restrict ourselves in this paper to weak stability: under this definition a marriage is unstable if there is a man m_i and a woman w_j, each of whom *strictly* prefers the other to his/her current partner [?], i.e. m_i and w_j will elope.

No complete algorithm has previously been proposed for SMTI and no empirical study has been carried out. The question "Is there a weakly stable matching?" is uninteresting as there always is [?]. So in this paper we present a constraint programming encoding for the SMTI decision problem "Is there a stable matching of size n?" and the optimisation problem, to find the largest or smallest stable matching. These questions are NP-complete [?, ?]. We also propose a problem generator for random instances of SMTI. We then study the SMTI investigating what features of the problem appear to influence the hardness and size of stable matchings.

The paper is organised as follows. In the next section we present a problem generator for SMTI. We then present constraint programming solutions for the SMTI decision problem and optimisation problem. These are the first complete algorithms for these problems. We then discuss the constrainedness of SMTI. The empirical study is then presented and the paper concludes.

2 Random Instance Generation

A class of randomly generated instance of SMTI is represented by a triple $\langle n, p_1, p_2 \rangle$ where n is the number of men and women in the problem, p_1 is the probability of incompleteness and p_2 is the probability of ties. Problems are generated as follows

1. A random preference list of size n is produced for each man and each woman.
2. We iterate over each man's preference list as follows. For a man m_i and for all women w_j in his preference list, we generate a random number $0 \leq p < 1$. If $p \leq p_1$ we delete w_j from m_i's preference list and delete m_i from w_j's preference list.
3. If any man or woman has an empty preference list, discard the problem and go to step 1.
4. We iterate over each person's (men and women's) preference list as follows. For a man m_i and for his choices c_i ranging from his second to his last, we generate a random number $0 \leq p < 1$. If $p \leq p_2$ then the preference for his c_i^{th} choice is the same as his

for those variables.

By representing SMTI as a constraint satisfaction problem we can measure κ for each instance generated. This will give us some indication of what ensemble such an instance most likely belongs to. However, we can make some conjectures as to how the difficulty of SMTI will vary as we vary the problem generation parameters $\langle n, p_1, p_2 \rangle$. When we increase p_2 we should expect each stable marriage constraint to become looser, i.e. the number of infeasible tuples will fall. Therefore κ should be inversely related to p_2. When we increase p_1 this will increase the amount of incompleteness in preference lists. Consequently domain sizes will fall, and we should expect κ to rise. However, as domain sizes fall so too does the number of stable marriage constraints. Therefore, it is not immediately clear if this fall in the number of constraints will win out against the falling domain sizes. Will κ fall or rise with p_1? And what will happen as we vary p_1 and p_2 together? Will these be opposing forces, where p_1 tends to drive problems towards insolubility, whereas p_2 tends to make problems looser? We will investigate these questions in the next section.

5 The Empirical Study

We performed our experiments using the choco constraint programming toolkit [?]. The study is mostly of problems of size 10. Problems were generated with incompleteness p_1 varying from 0.1 to 0.8 in steps of 0.1. When $p_1 \geq 0.9$ problems have empty preference lists, and are rejected from this study. For each value of p_1 we vary ties p_2 from 0.0 to 1.0 in steps of 0.01, with a sample size of either 100 or 50 at each data point. Experiments were run on machines with either 733MHz or 1GHz processors, with between 256MB and 1GB of RAM. The experiments reported here took in excess of 2 months CPU time. We also coded an independent implementation, written by a different author in Eclipse, and obtained consistent results with those presented here. In our experiments we first investigate how parameters p_1 and p_2 influence the decision problem "Is there a stable matching of size n?". We then explore the optimisation problem "What is the size of the largest and the smallest stable matchings?".

5.1 The Decision Problem

In the decision problem we determine if there is a stable matching of size n. This is a feature of the problem, and is algorithm independent. Figure ?? shows for each value of p_1 the proportion of soluble instances as we vary the amount of ties p_2. We see that as the amount of ties p_2 increases the proportion of soluble instances increases. This suggests that as we increase ties the constraints between men and women become looser, consequently we should expect to see a fall in the constrainedness of problems. We also observe that as p_1 increases, i.e. preference lists get shorter, solubility decreases. This might at first appear unsurprising. However, as preference lists get shorter the number of stability constraints fall. This fall is not enough to prevent a fall in solubility due to falling domain size.

In Figure ?? we plot solubility against the average constrainedness of the problem instances, i.e. κ is on the x-axis. We see the familiar phase transition behaviour as observed in [?, ?].

The allDiff constraint makes no difference. The number of search nodes was the same with and without this redundant constraint, and there was no significant difference in run times. Figure ?? shows the average cost of answering the decision problem, measured in terms of search nodes, for $\langle 10, p_1, p_2 \rangle$ plotted against p_2. Search costs increases as we increase ties p_2. This is because constraints get looser as ties increase, consequently the problem is less determined by propagation. Therefore at each instantiation a choice has to be made. Nev-

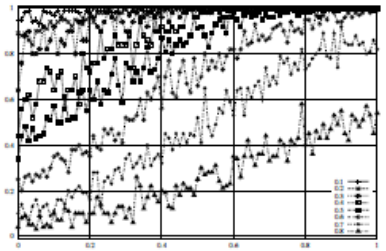


Figure 3. The decision problem: is there a stable matching of size n?

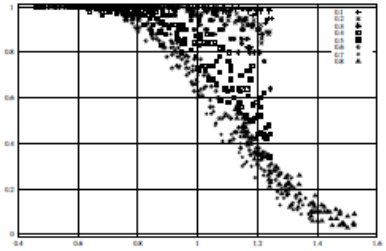


Figure 4. The decision problem: is there a stable matching of size n for a given value of κ?

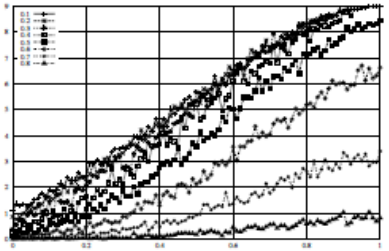
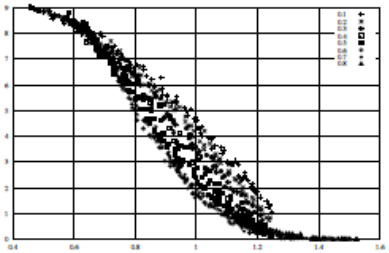


Figure 5. The average cost of the decision problem



A

An Empirical Study of Two Vertex Selection Strategies for the Clique Decision and Maximisation Problems

PATRICK PROSSER, University of Glasgow

Given a graph $G = (V, E)$ a clique is set of vertices $C \subseteq V$ such that all pairs of vertices in C are adjacent. We can construct a clique using an exact algorithm that has a guessing stage, i.e. a stage where it must choose a vertex and add it to C. Intuition suggests that we choose the vertex adjacent to most others. Our study shows that intuition is incorrect and offers an explanation of why that is so. We perform an empirical study of the clique decision problem and demonstrate that it has a phase transition with a corresponding complexity peak. We then characterise this with respect to the constrainedness of the decision problem and show that we can derive a theory-based strategy for vertex selection and predict its behaviour. We show that the vertex selection strategy also works well on the optimisation problem i.e. finding the maximum clique.

Categories and Subject Descriptors: F.2.2 [**Analysis of Algorithms and Problem Complexity**]: Nonnumerical Algorithms and Problems—*Computations on discrete structures*; G.2.1 [**Discrete Mathematics**]: Combinatorics—*Combinatorial algorithms*

General Terms: Algorithms, Heuristics, Experimentation

Additional Key Words and Phrases: Maximum clique, clique decision problem, vertex selection strategy, heuristics, empirical study, phase transition, constrainedness

ACM Reference Format:
ACM J. Exp. Algor. V, N, Article A (January YYYY), 14 pages.
DOI = 10.1145/0000000.0000000 http://doi.acm.org/10.1145/0000000.0000000

1. INTRODUCTION

A simple undirected graph G is a pair (V, E) where V is a set of vertices and E a set of edges. An edge $\{u, v\}$ is in E if and only if $\{u, v\} \subseteq V$ and vertex u is adjacent to vertex v. A *clique* is a set of vertices $C \subseteq V$ such that every pair of vertices in C is adjacent in G. Clique is one of the six basic NP-complete problems given in [Garey and Johnson 1979]. It is posed as a decision problem [GT19]: given a simple undirected graph $G = (V, E)$ and a positive integer $k \leq |V|$, does G contain a clique of size k or more? The optimisation problem is then to find a *maximum clique*, where $\omega(G)$ is the size of a maximum clique.

We can address the decision and optimisation problems with an exact algorithm, such as a backtracking search [Pardalos and Rodgers 1992; Fahle 2002; Régin 2003; Wood 1997; Carraghan and Pardalos 1990; Segundo et al. 2011; Konc and Janezic 2007; Tomita et al. 2010]. Backtracking search incrementally constructs the set C by choosing a vertex from the *candidate set* P (where P is initially V) and adding that vertex to C. The candidate set is then updated, removing vertices that cannot participate in the evolving clique.

Author's address: School of Computing Science, University of Glasgow, Glasgow G12 8QQ, Scotland; email: Patrick.Prosser@glasgow.ac.uk.

DOI 10.1145/0000000.0000000 http://doi.acm.org/10.1145/0000000.0000000

A

An Empirical Study of Two Vertex Selection Strategies for the Clique Decision and Maximisation Problems

PATRICK PROSSER, University of Glasgow

Given a graph $G = (V, E)$ a clique is set of vertices $C \subseteq V$ such that all pairs of vertices in C are adjacent. We can construct a clique using an exact algorithm that has a guessing stage, i.e. a stage where it must choose a vertex and add it to C. Intuition suggests that we choose the vertex adjacent to most others. Our study shows that intuition is incorrect and offers an explanation of why that is so. We perform an empirical study of the clique decision problem and demonstrate that it has a phase transition with a corresponding

2007; Tomita et al. 2010). Backtracking search incrementally constructs the set C by choosing a vertex from the *candidate set* P (where P is initially V) and adding that vertex to C. The candidate set is then updated, removing vertices that cannot participate in the evolving clique.

Author's address: School of Computing Science, University of Glasgow, Glasgow G12 8QQ, Scotland; email: Patrick.Prosser@glasgow.ac.uk.

DOI 10.1145/0000000.0000000 http://doi.acm.org/10.1145/0000000.0000000

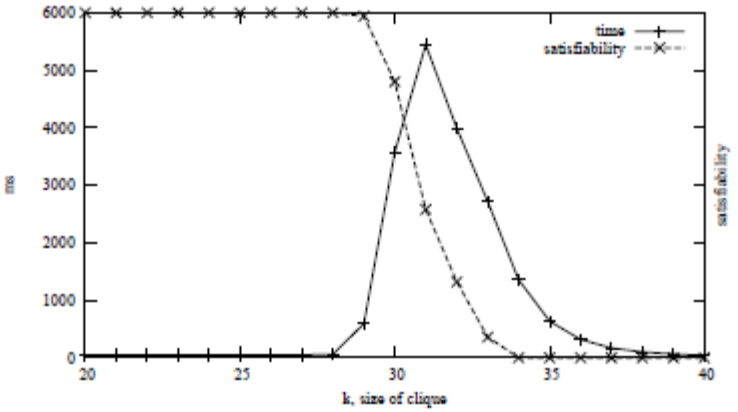


Fig. 1. $G(100, 0.9)$, sample size 100, $20 \leq k \leq 40$, CD_{max}, average run time and satisfiability.

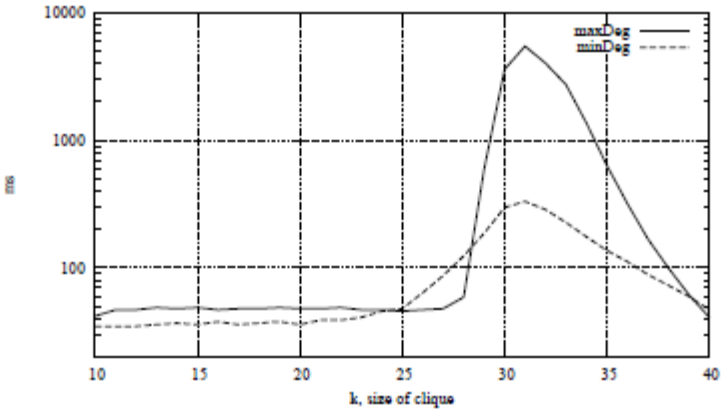


Fig. 2. $G(100, 0.9)$, sample size 100, $20 \leq k \leq 40$, CD_{max} and CD_{min}, log of average run time.

REJECT

$(37 \leq k \leq 40)$ it is easy to determine that there is no clique of that size. However, in the

4. THE CONSTRAINEDNESS OF CLIQUE DECISION

We now define constrainedness κ (kappa) for the clique decision problem and demonstrate that it captures the behaviour seen in Figures 1 and 2. We then use κ to explain the behaviour of the vertex selection heuristics.

4.1. Constrainedness (κ)

The phase transition has been characterised in [Gent et al. 1996] as follows

$$\kappa = 1 - \frac{\log\langle Sol\rangle}{\log|S|}$$

where $\langle Sol\rangle$ is the expected number of solutions and $|S|$ is the size of the state space, i.e. the maximum number of states that could be considered by a simple generate-and-test algorithm. When all states are solutions $\kappa = 0$ and problems are satisfiable and easy, when no state is a solution $\kappa = \infty$ and problems are unsatisfiable and easy, and when there is on average a single solution $\kappa = 1$ and problems are hard. We now define κ for the clique decision problem in $G(n,p)$ for clique size k. The size of the state space $|S|$ is equal to the number of ways we can select k vertices, hence

$$|S| = \binom{n}{k}$$

To compute the expected number of solutions $\langle Sol\rangle$ we first calculate the probability p_{sol} that a state is a solution, i.e. that having chosen k vertices they are all adjacent

$$p_{sol} = p^{\binom{k}{2}}$$

Since the probability holds for any k vertices the expected number of solutions is given by

$$\langle Sol\rangle = |S|.p_{sol}$$

Combining these results we have that for the clique decision problem in random graphs $G(n,p)$ with clique size k

$$\kappa_{clique} = \frac{\binom{k}{2}\log(\frac{1}{p})}{\log(\binom{n}{k})} \quad (1)$$

In Figure 3 we present the same data as in Figure 1 but with constrainedness (κ) on the x-axis, demonstrating that κ is a good measure for this problem.

MC_{min} was applied to $G(n,0.8)$ instances with $n \in \{100,110,120,130,140,150\}$, $10 \leq k \leq 50$ and a sample size of 100. Plotted in Figure 4 is on the left logarithm of average run time in milliseconds and on the right percentage satisfiability, both with κ on the x-axis. Ideally we would like the 50% crossover point to occur at $\kappa \approx 1$ with the complexity peak occuring simultaneously. In fact we see the crossover point and complexity peak occurring in the range $0.82 \leq \kappa \leq 0.90$, i.e. earlier than anticipated but as expected when problem size is small [Gent et al. 1996]. The evidence of Figures 3 and 4 suggest that κ characterises the phase transition phenomena in the clique decision problem for random $G(n,p)$.

4.2. Constrainedness (κ) as a Heuristic

In [Gent et al. 1996] it is proposed that κ be used as a guiding principle when designing variable ordering heuristics, i.e. to make decisions that minimise the constrainedness of the future sub-problem. In particular κ suggests how we should select a vertex to

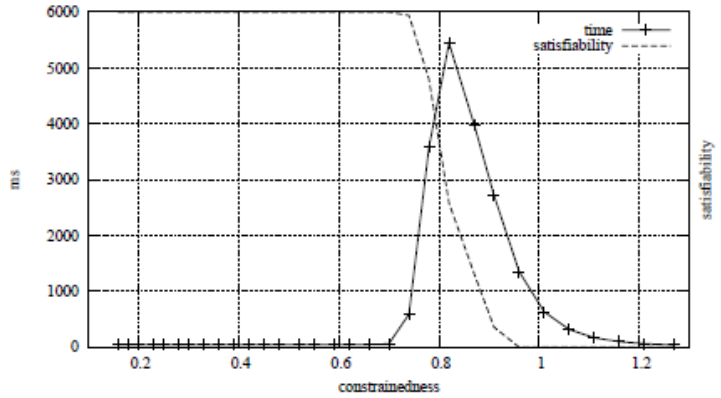


Fig. 3. $G(100, 0.9)$, sample size 100, $20 \leq k \leq 40$, CD_{max}, average run time in milliseconds (ms) and satisfiability against constrainedness (κ).

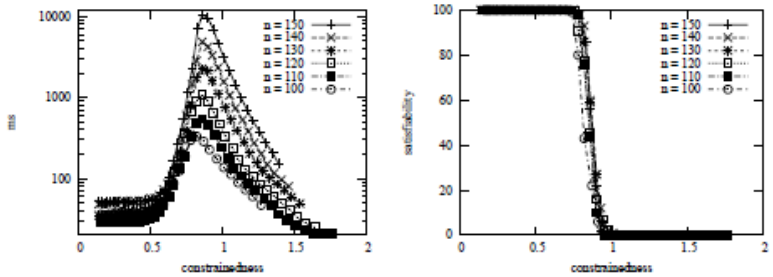


Fig. 4. $G(n, 0.9)$, sample size 100, $n \in \{100, 110, 120, 130, 140, 150\}$, $1 \leq k \leq n$, CD_{max}. On the left, logarithm of run time against κ and on the right satisfiability against κ.

add to the clique. In the setting of $G(n, p)$ with clique size k when a vertex is selected κ changes as follows:

$$\frac{\binom{k}{2}\log(\frac{1}{p})}{\log(\binom{n}{k})} \Longrightarrow \frac{\binom{k}{2}\log(\frac{1}{p'})}{\log(\binom{n-1}{k})} \tag{2}$$

In (2) one vertex is selected so n becomes $n-1$, all the edges emanating from that vertex are removed and edge probability p becomes p', where p' is the number of edges remaining divided by $\binom{n-1}{2}$. Therefore the only differences between vertex selections is the resulting values of p'. If p' decreases then $\log(\frac{1}{p'})$ increases as does κ, and if p' increases then $\log(\frac{1}{p'})$ decreases and so does κ. Consequently we should choose the vertex that makes p' as large as possible and we do this by choosing the vertex that

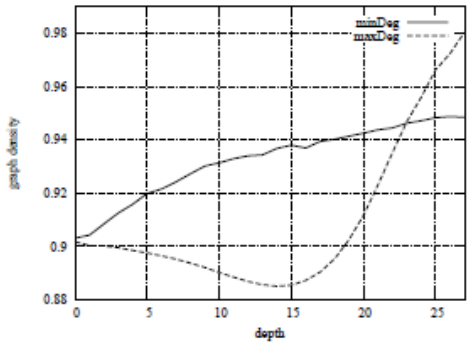


Fig. 7. $G(100, 0.9)$, graph density at depth.

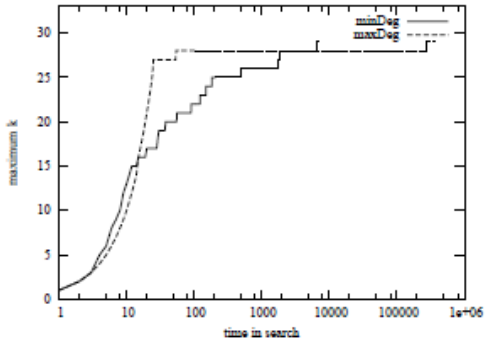


Fig. 8. $G(100, 0.9)$, size of largest clique during search.

Computational results show that greedy was good for (easy) sparse graphs and non-greedy good for (hard) dense graphs.

In [Wood 1997] graph colouring and fractional colouring is used to bound search and vertices are selected in non-increasing degree order.

Patric R. J. Östergård proposed an algorithm that has a dynamic programming flavour [Östergård 2002]. The search process starts by finding the largest clique containing vertices drawn from the set $S_n = \{v_n\}$ and records it size in $c[n]$. Search then proceeds to find the largest clique in the set $S_i = \{v_i, v_{i+1}, ..., v_n\}$ using the value in $c[i+1]$ as a bound. The vertices are ordered at the top of search in colour order, i.e. the vertices are coloured greedily and then ordered in non-decreasing colour order.

[Fahle 2002] presented a simple algorithm (Algorithm 1) (essentially the same as MC in Section 5) with a free selection of vertices. This is then enhanced (Algorithm 2) with forced accept and forced reject steps similar to Rules 4, 5 and 7 of [Pardalos and Rodgers 1992]. Fahle notes that 'The ordering in which nodes are considered during

Journal of Artificial Intelligence Research 61 (2018) 723-759 Submitted 11/17; published 03/18

When Subgraph Isomorphism is Really Hard, and Why This Matters for Graph Databases

Ciaran McCreesh CIARAN.MCCREESH@GLASGOW.AC.UK
Patrick Prosser PATRICK.PROSSER@GLASGOW.AC.UK
University of Glasgow, Glasgow, Scotland

Christine Solnon CHRISTINE.SOLNON@INSA-LYON.FR
INSA-Lyon, LIRIS, UMR5205, F-69621, France

James Trimble J.TRIMBLE.1@RESEARCH.GLA.AC.UK
University of Glasgow, Glasgow, Scotland

Abstract

The subgraph isomorphism problem involves deciding whether a copy of a pattern graph occurs inside a larger target graph. The non-induced version allows extra edges in the target, whilst the induced version does not. Although both variants are NP-complete, algorithms inspired by constraint programming can operate comfortably on many real-world problem instances with thousands of vertices. However, they cannot handle arbitrary instances of this size. We show how to generate "really hard" random instances for subgraph isomorphism problems, which are computationally challenging with a couple of hundred vertices in the target, and only twenty pattern vertices. For the non-induced version of the problem, these instances lie on a satisfiable / unsatisfiable phase transition, whose location we can predict; for the induced variant, much richer behaviour is observed, and constrainedness gives a better measure of difficulty than does proximity to a phase transition. These results have practical consequences: we explain why the widely researched "filter / verify" indexing technique used in graph databases is founded upon a misunderstanding of the empirical hardness of NP-complete problems, and cannot be beneficial when paired with any reasonable subgraph isomorphism algorithm.

1. Introduction

The *non-induced subgraph isomorphism problem* is to find an injective mapping from the vertices of a given pattern graph to the vertices of a given target graph which preserves adjacency—in essence, we are "finding a copy of" the pattern inside the target. The *induced* variant of the problem additionally requires that the mapping preserve non-adjacency, so there are no "extra edges" in the copy of the pattern that we find. We illustrate both variants in Figure 1. Although these problems are NP-complete (Garey & Johnson, 1979), modern subgraph isomorphism algorithms based upon constraint programming techniques can handle problem instances with many hundreds of vertices in the pattern graph, and up to ten thousand vertices in the target graph (Solnon, 2010; Audemard, Lecoutre, Modeliar, Goncalves, & Porumbel, 2014; McCreesh & Prosser, 2015; Kotthoff, McCreesh, & Solnon, 2016), and subgraph isomorphism is used successfully in application areas including computer vision (Damiand, Solnon, de la Higuera, Janodet, & Samuel, 2011; Solnon, Damiand, de la Higuera, & Janodet, 2015), biochemistry (Giugno, Bonnici, Bombieri, Pulvirenti, Ferro, & Shasha, 2013; Carletti, Foggia, & Vento, 2015), and pattern recognition

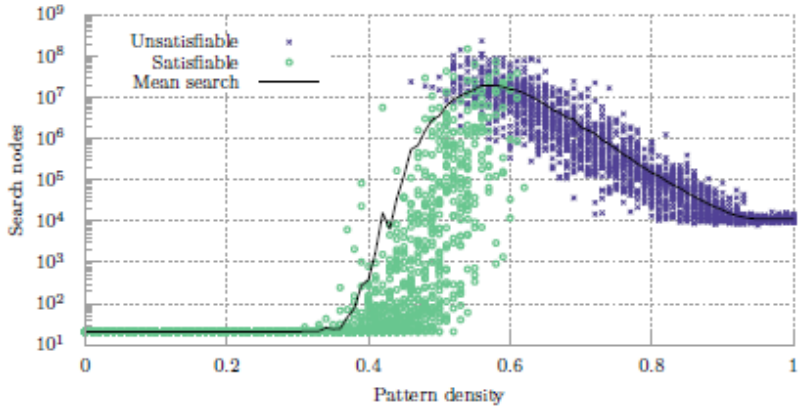


Figure 3: With a fixed pattern graph order of 20, a target graph order of 150, a target edge probability of 0.40, and varying pattern edge probability, we observe a phase transition and complexity peak with the Glasgow solver in the non-induced variant. Each point (x, y) represents one instance i, where x is the pattern edge probability used to generate i, y is the number of search nodes needed by the Glasgow solver to solve i, and the point is drawn as a green circle if i is satisfiable, and a blue cross otherwise. The black line plots the evolution of the arithmetic mean number of search nodes when increasing the pattern edge probability from 0 to 1 in steps of 0.01, using a larger sample size of 1,000.

observe looks remarkably similar to random 3SAT problems—compare, for example, Figure 1 of the work of Leyton-Brown, Hoos, Hutter, and Xu (2014). In particular, satisfiable instances tend to be easier, but show greater variation than unsatisfiable instances, and there are exceptionally hard satisfiable instances (Smith & Grant, 1997).

2.2 Phase Transitions when Varying Pattern and Target Edge Probabilities

What if we alter the edge probabilities for both the pattern graph and the target graph? In the top row of Figure 4 we show the satisfiability phase transition for the non-induced variant, for patterns of order 10, 20 and 30, targets of order 150, and varying pattern (x-axis) and target (y-axis) edge probabilities. Each axis runs over 101 edge probabilities, from

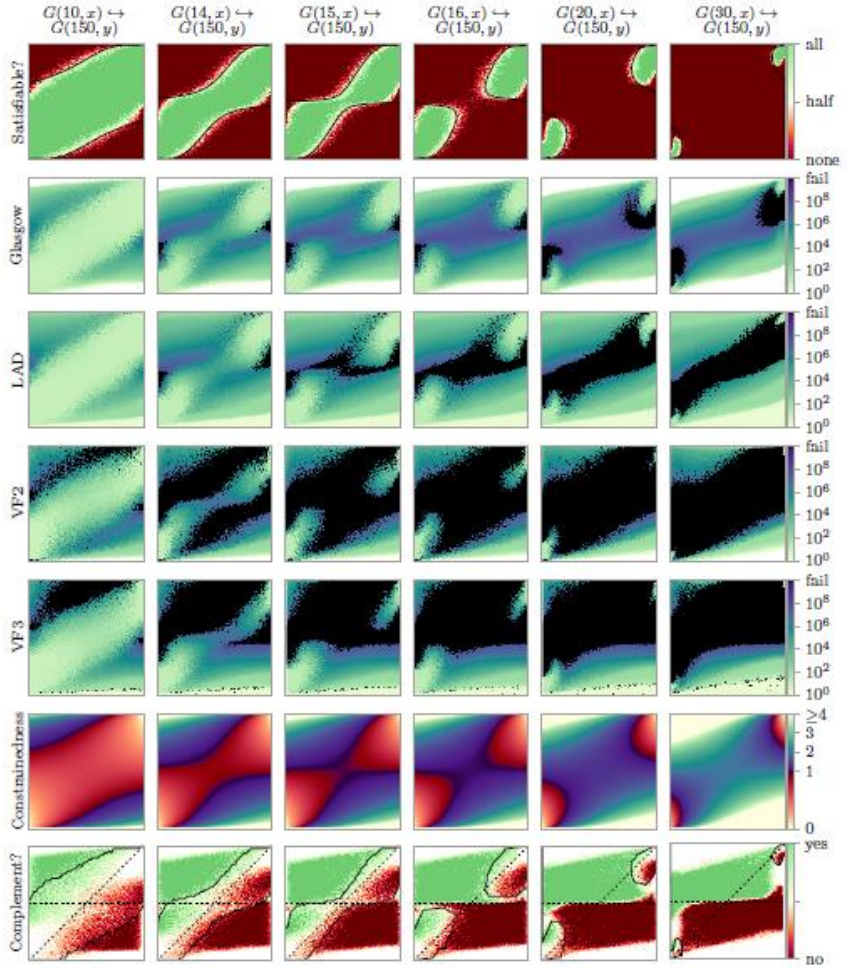


Figure 5: Behaviour of algorithms on the induced variant with target graphs of 150 vertices, shown in the style of Figure 4. The sixth row plots constrainedness using equation (3): the darkest region is where $\kappa = 1$, and the lighter regions show where the problem is either over- or under-constrained. The final row shows when the Glasgow algorithm performs better when given the complements of the pattern and target graphs as inputs—the solid lines show the empirical location of the phase transition, and the dotted lines are $d_t = 0.5$ and the $d_p = d_t$ diagonal.

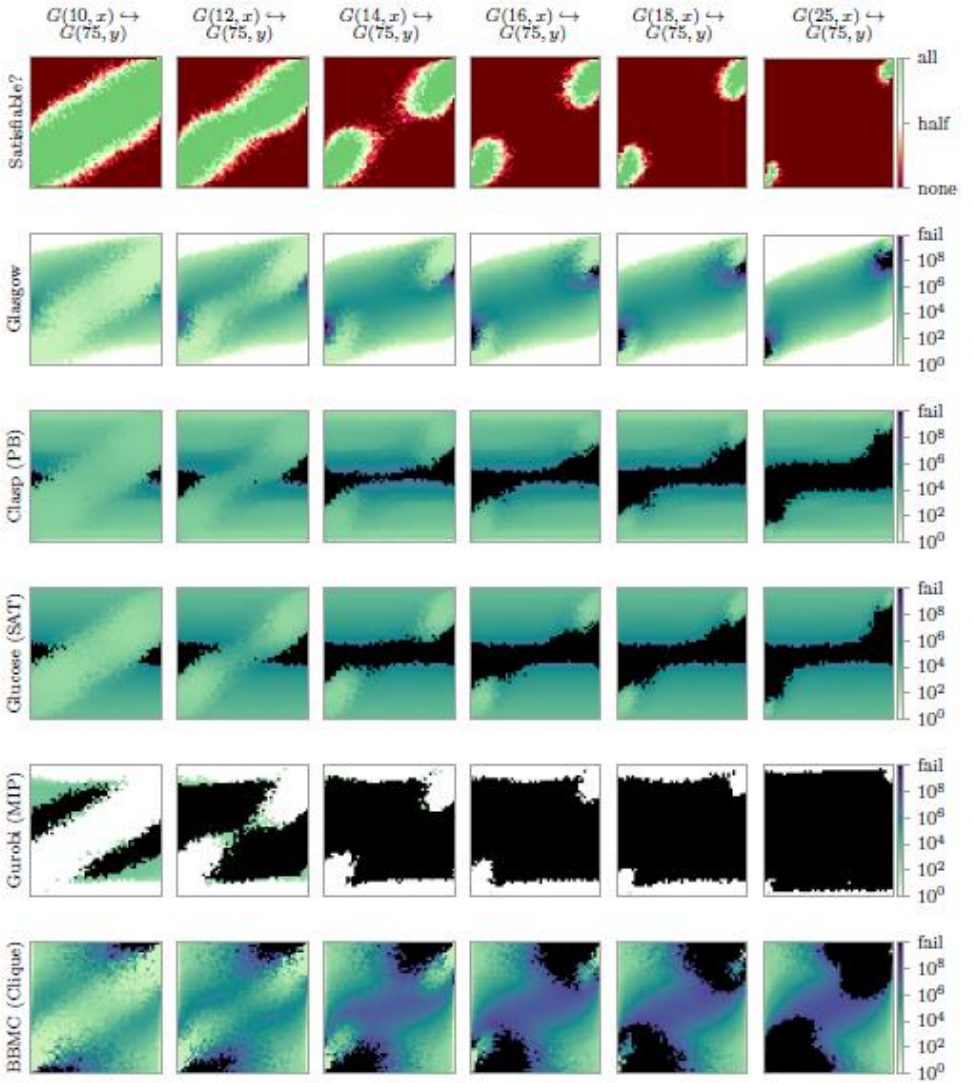


Figure 6: Behaviour of other solvers on the induced variant using smaller target graphs with 75 vertices, shown in the style of Figure 4. The second row shows the number of search nodes used by the Glasgow algorithm, the third and fourth rows show the number of decisions made by the pseudo-boolean and SAT solvers, the fifth shows the number of search nodes used on the MIP encoding, and the final row the clique encoding.

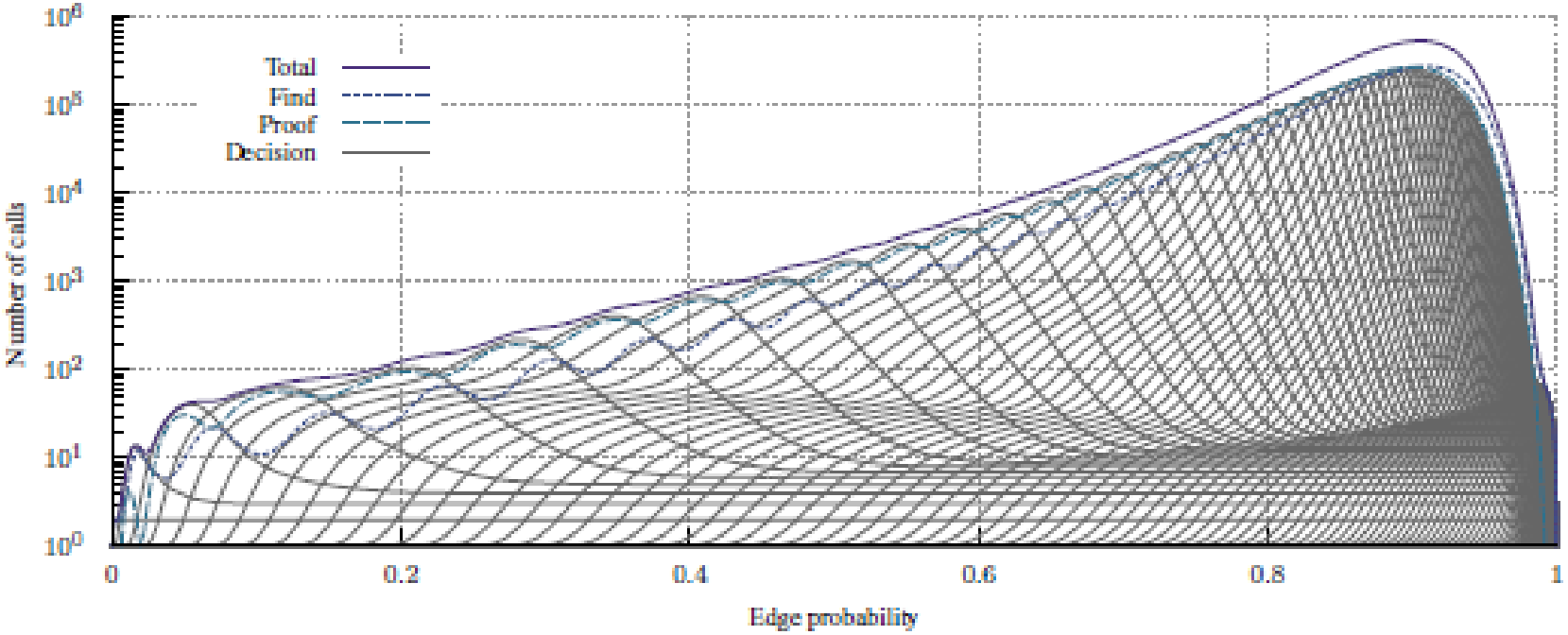


Figure 3: A more detailed picture of difficulty of solving the clique optimisation problem for $G(150, x)$. Also plotted is the mean search effort to find the optimal solution but not prove its optimality, and the mean search effort needed to prove optimality after the optimal solution is found. Finally, 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.

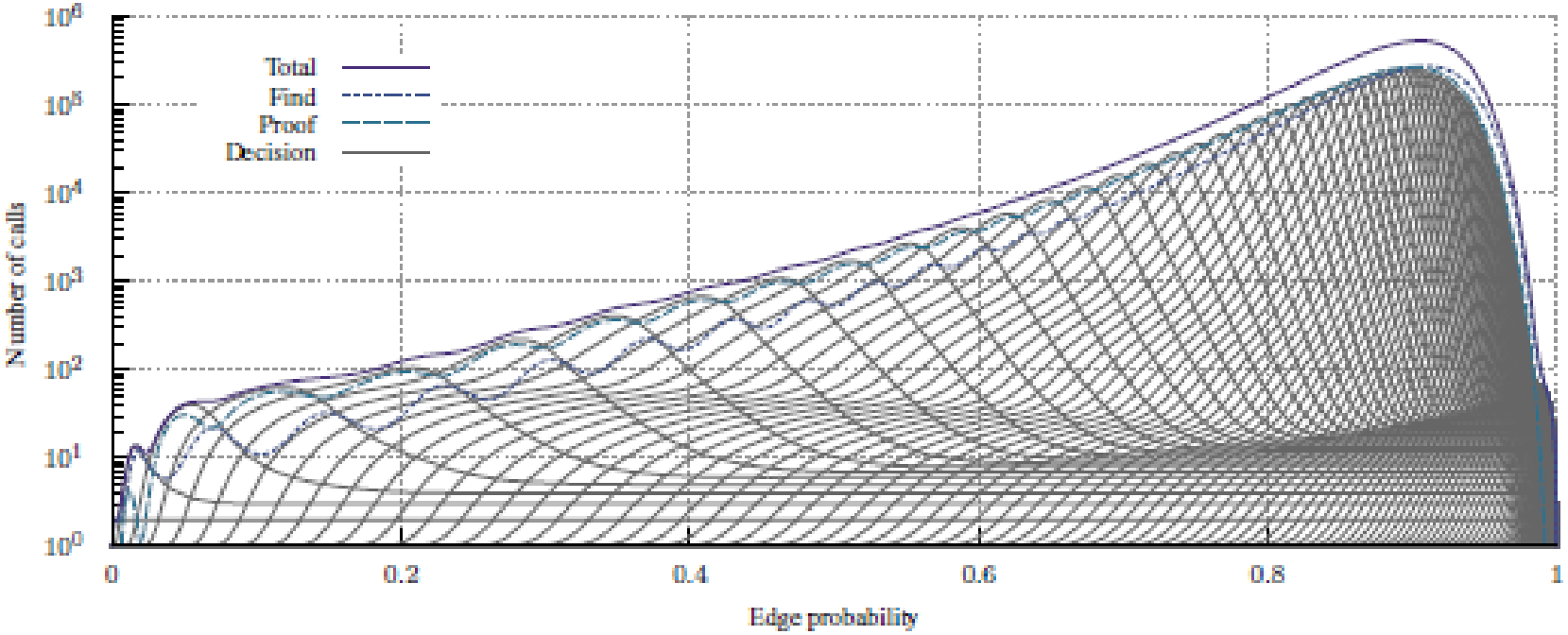
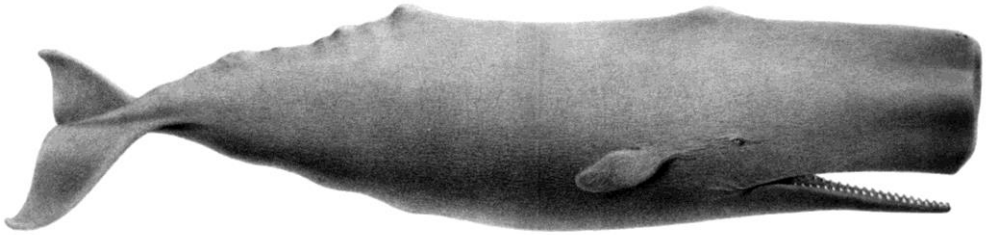


Figure 3: A more detailed picture of difficulty of solving the clique optimisation problem for $G(150, x)$. Also plotted is the mean search effort to find the optimal solution but not prove its optimality, and the mean search effort needed to prove optimality after the optimal solution is found. Finally, 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.



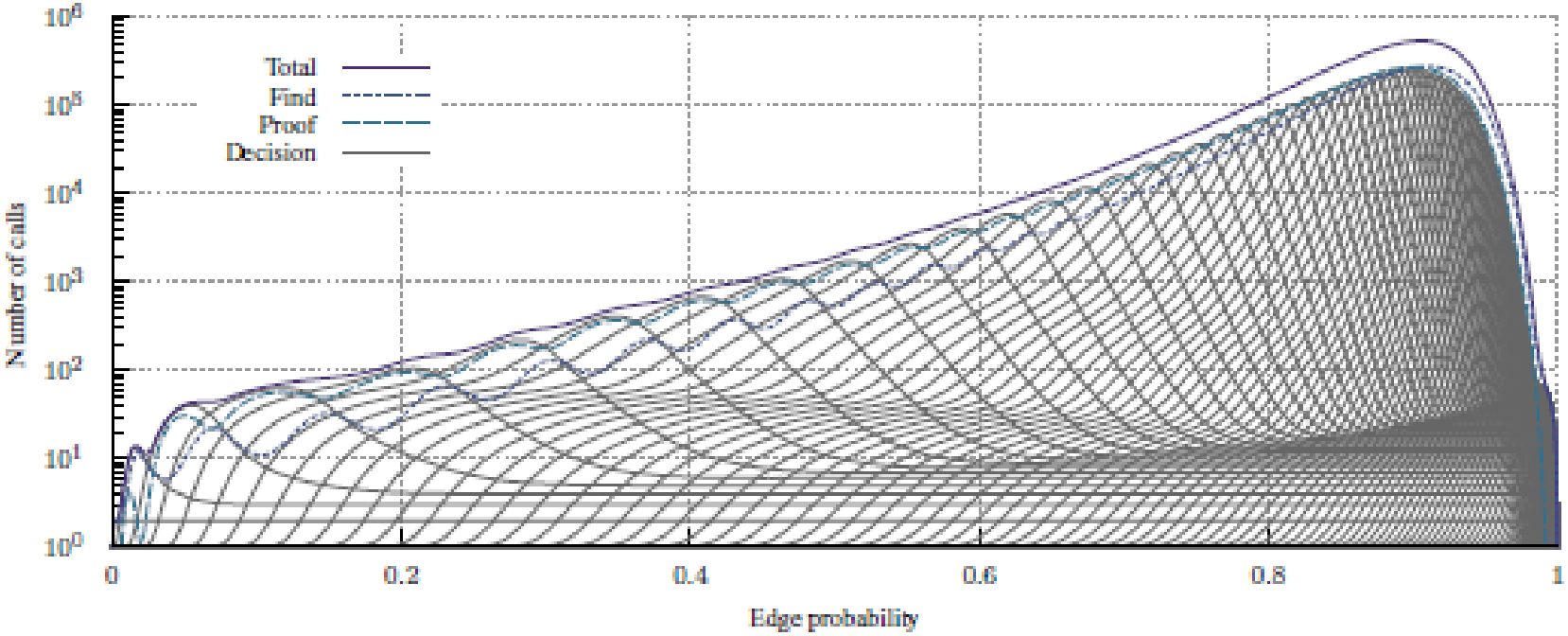


Figure 3: A more detailed picture of difficulty of solving the clique optimisation problem for $G(150, x)$. Also plotted is the mean search effort to find the optimal solution but not prove its optimality, and the mean search effort needed to prove optimality after the optimal solution is found. Finally, 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.

In pre-rejection mode