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

Ciaran Mccreesh, Patrick Prosser, Christine Solnon, James Trimble

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 1: On the left, an induced subgraph isomorphism. On the right, a non-induced subgraph isomorphism: the extra dashed edge is not present in the pattern graph.

(Conte, Foggia, Sansone, & Vento, 2004). A labelled version of subgraph isomorphism is also one of the key components in supporting complex queries in graph databases, which we discuss in detail in Section 6.

However, subgraph isomorphism algorithms cannot handle *arbitrary* instances involving hundreds or thousands of vertices. Experimental evaluations of subgraph isomorphism algorithms are usually performed using randomly generated pairs of pattern and target graphs, sometimes together with a mix of real-world graphs, for example biochemistry and computer vision problems. Using random instances to evaluate algorithm behaviour can be beneficial, because it provides a way of generating many instances cheaply, and reduces the risk of over-fitting when tuning design parameters. It also allows experimenters to control graph parameters such as order, density, or degree distribution, and to study scaling properties of algorithms with respect to these parameters. The random instances used in previous evaluations come from common datasets (Santo, Foggia, Sansone, & Vento, 2003; Zampelli, Deville, & Solnon, 2010), which were generated by taking a random subgraph of a random (Erdős-Rényi, scale-free, bounded degree, or mesh) graph and permuting the vertices. Such instances are guaranteed to be satisfiable—Anton and Olson (2009) exploited this property to create large sets of random satisfiable boolean satisfiability instances. This is the most common approach to generating random subgraph isomorphism instances, meaning existing benchmark suites contain relatively few non-trivial unsatisfiable instances (although a few of the patterns in the instances by Zampelli et al. have had extra edges added, making them unsatisfiable). Also, the satisfiable instances tend to be computationally fairly easy, with most of the difficulty being in dealing with the size of the model. This has led to bias in algorithm design, to the extent that some proposed techniques, such as those of Battiti and Mascia (2007), will *only* work on satisfiable instances.

The first contribution of this paper is to present and evaluate new methods for creating random pattern / target pairs. The method we introduce in Section 2 generates both satisfiable and unsatisfiable instances, and can produce computationally challenging instances with only a few tens of vertices in the pattern, and 150 vertices in the target. Note that the lack of unsatisfiable instances for testing purposes cannot be addressed simply by taking a pattern graph from one of the existing random suites with the "wrong" target graph, as this tends to give either a trivially unsatisfiable instance, or a satisfiable instance. (In particular, it is *not* the case that a relatively small random graph is unlikely to appear in a larger random graph.)

This work builds upon the phase transition phenomena observed in satisfiability and graph colouring problems first described by Cheeseman, Kanefsky, and Taylor (1991) and Mitchell, Selman, and Levesque (1992). For subgraph isomorphism in Erdős-Rényi random

graphs, we identify three relevant control parameters: we can independently alter the edge probability of the pattern graph, the edge probability of the target graph, and the relative orders (number of vertices) of the pattern and target graphs. For non-induced isomorphisms, with the correct choice of parameters we see results very similar to those observed with boolean satisfiability problems: there is a phase transition (whose location we can predict) from satisfiable to unsatisfiable, and we see a solver-independent complexity peak near this phase transition. Additionally, understanding this behaviour helps us to design better search strategies, by improving variable- and value-ordering heuristics (that is, by improving the order in which pattern and target vertices are selected for guessed assignments). This is the second contribution of this paper.

In Section 3 we look at the induced subgraph isomorphism problem. For certain choices of parameters, there are two phase transitions, going from satisfiable to unsatisfiable, and then from unsatisfiable back to satisfiable. Again, when going from satisfiable to unsatisfiable (from either direction), instances go from being trivial to really hard to solve. However, each of the three solvers we test also finds the central unsatisfiable region to be hard, despite it not being near a phase transition. To show that this is not a simple weakness of current subgraph isomorphism algorithms, we verify that this region is also hard for boolean satisfiability, pseudo-boolean, and mixed integer solvers, and under reduction to the clique problem. Interestingly, the constrainedness measure proposed by Gent, MacIntyre, Prosser, and Walsh (1996) *does* predict this difficult region—the third contribution of this paper is to use these instances to provide evidence in favour of constrainedness, rather than proximity to a phase transition, being an accurate predictor of difficulty, and to show that constrainedness is not simply a refinement of a phase transition prediction.

What about other random graph models? In Section 4 we see that graphs where every vertex has the same degree (known as k-regular) exhibit very similar characteristics. However, when labels on vertices are introduced (as is commonly seen in real-world applications and in graph database systems), richer behaviour emerges, particularly when moving away from a uniform labelling scheme, and the popular VF2 algorithm (Cordella, Foggia, Sansone, & Vento, 2004) behaves much worse than two more recent constraint-based algorithms (Solnon, 2010; McCreesh & Prosser, 2015) in certain cases. In Section 5 we take a close look at these cases, and argue that they *should* be easy to solve.

This is not simply a theoretical curiosity. A labelled version of subgraph isomorphism is one of the key components in supporting complex queries in graph databases—typically, these systems store a fixed set of target graphs, and for a sequence of pattern queries, they must return every target graph which contains that pattern. One common approach to this problem is to combine a subgraph isomorphism algorithm with an indexing system in a so-called "filter / verify" model, where invariants are used to attempt to pre-exclude unsatisfiable instances to avoid the cost of a subgraph isomorphism call. In this context, the terms *matching* and *verification* are often used for the subgraph isomorphism step (or a slightly broader problem, for example permitting wildcards).

In Section 6 we look at some of the datasets commonly used to test graph database systems, which the literature suggests are hard to solve. Simple experiments show that the entire perceived difficulty of each of these datasets comes down to the widespread use of particular subgraph isomorphism algorithms, rather than inherent hardness. Our fourth contribution is to show that the simplest constraint programming approach, even without

the more sophisticated recent advances in subgraph algorithms, makes the entire graph database filter / verify paradigm unnecessary. Finally, we explain why filter / verify and other recently proposed techniques cannot be beneficial even on more challenging instances, unless the chosen subgraph isomorphism algorithm exhibits unnecessarily poor behavior on certain classes of instance.

1.1 Definitions

Throughout, our graphs are undirected, and do not have any loops. In some cases vertices have labels, which are treated as integers; we write $\ell(v)$ for the label of vertex v. Two vertices are *adjacent* if there is an edge between them. The *neighbourhood* of a vertex is the set of vertices to which it is adjacent, and the *degree* of a vertex is the cardinality of its neighbourhood. The *order* of a graph is the cardinality of its vertex set. We write $\mathrm{V}(G)$ for the vertex set of a graph G. The *complement* of a graph G, denoted $\overline{G}$, is the graph with the same vertex set as G, and with an edge between distinct vertices v and w if and only if v and w are not adjacent in G.

An *Erdős-Rényi random graph*, written $G(n, p)$, is a graph with n vertices, and an edge between each distinct unordered pair of vertices with independent probability p (Gilbert, 1959; Erdős & Rényi, 1959).[1]

A *non-induced subgraph isomorphism* from a graph P (called the *pattern*) to a graph T (called the *target*) is an injective mapping i from $\mathrm{V}(P)$ to $\mathrm{V}(T)$ which preserves adjacency—that is, for every adjacent vertices v and w in $\mathrm{V}(P)$, the vertices $i(v)$ and $i(w)$ are adjacent in T. Where labels are present, the mapping must also preserve labels, so $\ell(v) = \ell(i(v))$ for any v. An *induced subgraph isomorphism* additionally preserves non-adjacency—that is, if v and w are not adjacent in P, then $i(v)$ and $i(w)$ must not be adjacent in T. We use the notation $i : P \rightarrowtail T$ for a non-induced isomorphism, and $i : P \hookrightarrow T$ for an induced isomorphism. Observe that an induced isomorphism $i : P \hookrightarrow T$ is a non-induced isomorphism $i : P \rightarrowtail T$ which is also a non-induced isomorphism $i : \overline{P} \rightarrowtail \overline{T}$. Deciding whether a subisomorphism of either form exists between two given graphs exists is NP-complete.

1.2 Experimental Setup

The experiments in this paper are performed on systems with Intel Xeon E5-4650 v2 CPUs and 768GBytes RAM, running Scientific Linux release 6.7. We will be working with four subgraph isomorphism solvers: the Glasgow solver (McCreesh & Prosser, 2015; Kotthoff et al., 2016); LAD (Solnon, 2010); VF2 (Cordella et al., 2004); and VF3 (Carletti, Foggia, Saggese, & Vento, 2017). Each was compiled using GCC 4.9 with the "-O3" option. The Glasgow, LAD, and VF2 solvers support both non-induced and induced isomorphisms (although neither Glasgow nor LAD were designed with the induced version in mind), whilst VF3 supports only the induced problem.

1. Note that elsewhere, this same name sometimes instead refers to a graph $G(n, E)$ chosen uniformly from among all graphs with n vertices and exactly E edges.

1.3 Subgraph Isomorphism Solvers

The Glasgow and LAD solvers are constraint programming inspired algorithms (although they use dedicated data structures, and do not use a constraint programming toolkit). Constraint programming is a field concerned with solving *constraint satisfaction* problems. In such a problem, we have a set of *variables*, each of which has a *domain* of *values*. We also have a set of *constraints*. The objective is to assign each variable a value from its domain, whilst respecting every constraint. This is done using a combination of *inference*, which eliminates infeasible values from domains, and backtracking search. This search is driven by *variable-ordering* heuristics, which select which unassigned variable to branch on, and *value-ordering* heuristics, which select which value to try giving that variable.

In the case of subgraph isomorphism, the basic constraint model is to have a variable for each pattern vertex, with each variable's domain being the set of target vertices. An *all-different* constraint ensures injectivity, whilst the adjacency rules can be expressed as extensional constraints by listing allowed tuples. With this model, the backtracking search builds up partial assignments of pattern vertices to target vertices, until either a complete assignment is reached (and the instance is *satisfiable*), or search establishes that no such assignment can exist (and it is *unsatisfiable*). Hence, the role of the variable-ordering heuristic for subgraph isomorphism is to select a pattern vertex, whilst value-ordering tells us the order in which to try different candidate target vertices for that pattern vertex. If at any point during search a *domain wipeout* occurs (that is, if every value is removed from any domain), the search will backtrack immediately.

For example, in Figure 2, we would have three variables A, B and C, each of which has a domain $\{1, 2, 3, 4, 5, 6\}$. Suppose initially we guessed the assignment $A = 1$: inference would tell us that vertices adjacent to A must be mapped to vertices adjacent to 1, and so the value 6 would be eliminated from the domains of both B and C, whilst injectivity would eliminate 1 from B and C. Now suppose we then guessed $B = 2$: this would leave no suitable values for C, and so we would backtrack and try a new assignment for B. Guessing $B = 3$ will similarly give a wipeout on C, but guessing $B = 4$ will leave the value 5 in the domain of C, and we could then guess $C = 5$ to obtain a complete assignment.

This model can be extended with additional constraints. These allow more inference to be performed, and so a solution may be found (or the lack of solution may be proven) with less search. The simplest example is that a pattern vertex of degree d may only be mapped to a target vertex of degree at least d, and so constraints can be posted which eliminate values corresponding to target vertices of low degree from domains corresponding to pattern vertices of high degree. In the example in Figure 2, this rule would eliminate the values 2, 3

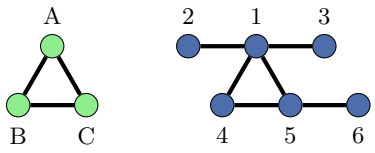

Figure 2: An example used to illustrate the simple constraint programming model for the problem: the pattern graph is on the left, and the target graph on the right.

and 6 from every domain at the top of search. Richer constraints involving neighbourhood degree sequences (Zampelli et al., 2010), neighbourhood difference constraints (Solnon, 2010), and distances and paths (Audemard et al., 2014; McCreesh & Prosser, 2015; Kotthoff et al., 2016) are also possible.

The Glasgow and LAD solvers both use extensions of this basic model, but differ in terms of the additional constraints used (and hence the inference which is performed), their choices of underlying data structures and algorithms, and in the details of their variable- and value-ordering heuristics.

The approach used by VF2 and VF3 is different: domains of variables are not stored (so the algorithm operates in the style of conventional backtracking), and wipeouts are not detected until an assignment is made. Instead, an assignment is built up by repeatedly branching only on vertices which are adjacent to a previously-assigned vertex.

Our experiments will primarily measure the number of *search nodes* (that is, the number of recursive calls made, or the number of guessed assignments), not runtimes. We are not aiming to give a simple comparison of absolute performance between solvers; rather, we are looking for solver-independent patterns of difficulty. However, roughly speaking, on the small graphs used in the following sections, Glasgow manages around 300,000 recursive calls per second on our hardware, whilst LAD and VF2 perform around 1,000 and 70,000 recursive calls per second respectively. All experiments use a timeout of 1,000 seconds, which is enough for the Glasgow solver to solve nearly all instances (whose orders were selected with this timeout in mind), although we may slightly overestimate the proportion of unsatisfiable instances for extremely sparse or dense pattern graphs. The LAD, VF2, and VF3 solvers experienced many more failures with this timeout, so our picture of just how hard the hardest instances are with these solvers is less detailed.

2. Non-induced Subgraph Isomorphisms

We begin by introducing a way of randomly generating a mix of satisfiable and unsatisfiable instances for the non-induced problem, and by measuring the empirical difficulty of such instances.

2.1 A Phase Transition when Varying Pattern Edge Probability

Suppose we arbitrarily decide upon a pattern graph order of 20, a target graph order of 150, a fixed target edge probability of 0.40, and no vertex labels. As we vary the pattern edge probability from 0 to 1, we would expect to see a shift from entirely satisfiable instances (with no edges in the pattern, we can always find a match) to entirely unsatisfiable instances (a maximum clique in this order and edge probability of target graph will usually have between 9 and 12 vertices). The move from green circles (satisfiable) to blue crosses (unsatisfiable) in Figure 3 shows that this is the case. For densities of 0.67 or greater, no instance is satisfiable; with densities of 0.44 or less, every instance is satisfiable; and with a density of 0.55, roughly half the instances are satisfiable.

The line plots arithmetic mean search effort using the Glasgow solver: for sparse patterns, the problem is trivial; for dense patterns, proving unsatisfiability is not particularly difficult; and we see a complexity peak around the point where half the instances are satisfiable. We also plot the search cost of individual instances, as points. The behaviour we

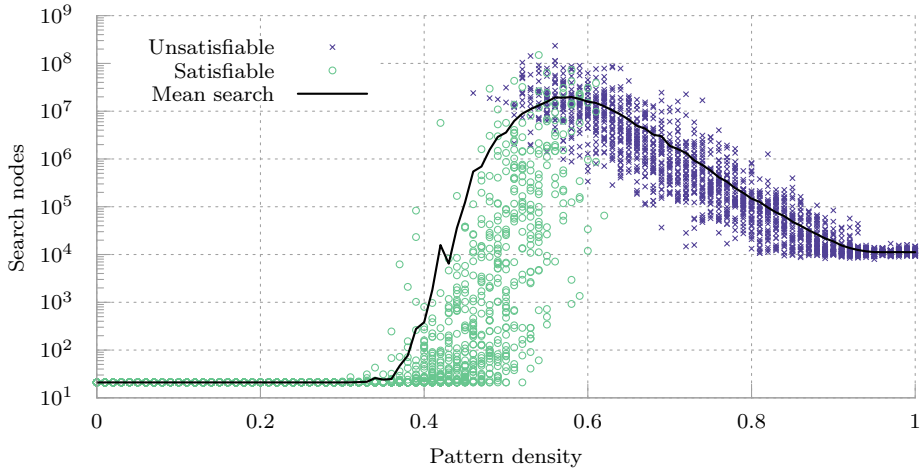

Figure 3: With a fixed pattern graph order of 20, a target graph order of 150, a target edge probability of 0.40, and varying pattern edge probability, we observe a phase transition and complexity peak with the Glasgow solver in the non-induced variant. Each point (x, y) represents one instance i, where 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 0 to 1 in steps of 0.01. For each of these points, we generate ten random instances. The colour denotes the proportion of these instances which were found to be satisfiable. Inside the red region, at the bottom right of each plot, every instance is unsatisfiable—here we are trying to find a dense pattern in a sparse target. In the green region, at the top left, every instance is satisfiable—we are looking for a sparse pattern in a dense target (which is easy, since we only have to preserve adjacency, not non-adjacency). The white band between the regions shows the location of the phase transition: here, roughly half the instances are satisfiable. (We discuss the black line below.)

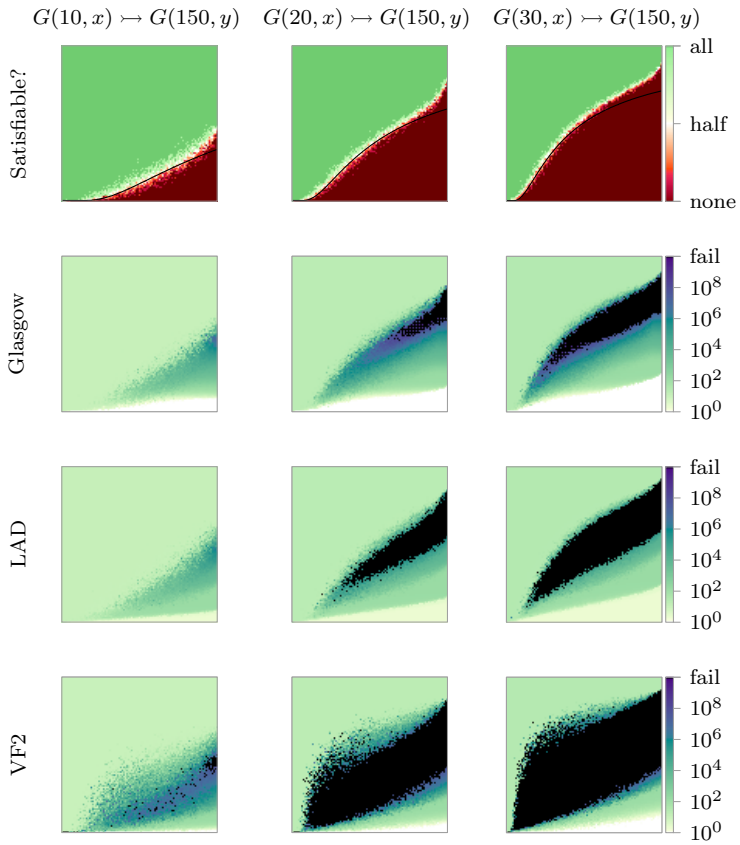

Figure 4: Behaviour of algorithms on the non-induced variant, using target graphs of 150 vertices, and pattern graphs of 10, 20, or 30 vertices. For each plot, the x-axis is the pattern edge probability and the y-axis is the target edge probability, both from 0 to 1. Along the top row, we show the proportion of instances which are satisfiable; the white bands show the phase transitions, and the black lines are the predictions using equation (1) of where the phase transition will occur. On the subsequent three rows, we show the average number of search nodes used by the Glasgow, LAD and VF2 solvers over ten instances; the dark regions indicate "really hard" instances, and black points indicate that at least one timeout occurred.

On subsequent rows, we show the average number of search nodes used by the different algorithms. Darker regions indicate harder difficulties, and black is used to indicate that at least one timeout occurred. In general, satisfiable instances are easy, until very close to the phase transition. As we hit the phase transition and move into the unsatisfiable region, we see complexity increase. Finally, as we pass through the phase transition and move deeper into the unsatisfiable region, instances become easier again. This behaviour is largely solver-independent, although VF2 has a larger hard region than Glasgow or LAD.

Thus, although we have moved away from a single control parameter, we still observe the easy-hard-easy pattern seen in many NP-complete problems.

Finally, a close look at the very bottom left of the plots shows that VF2 will sometimes give a timeout on instances where the target is empty but the pattern is not—this turns out to be extremely important, and we return to it in Sections 5 and 6.

2.3 Locating the Phase Transition

We can approximately predict the location of the phase transition by calculating (with simplifications regarding rounding and independence) the expected number of solutions for given parameters. Since we are trying to find an *injective* mapping from a pattern $P = G(p, d_p)$ to a target $T = G(t, d_t)$, there are

$$t^{\underline{p}} = t \cdot (t-1) \cdot \ldots \cdot (t-p+1)$$

possible injective assignments of target vertices to pattern vertices. If we assume the pattern has exactly $d_p \cdot \binom{p}{2}$ edges, we obtain the probability of all of these edges being mapped to edges in the target by raising d_t to this power, giving an expected number of solutions of

$$\langle Sol \rangle = t^{\underline{p}} \cdot {d_t}^{d_p \cdot \binom{p}{2}}. \tag{1}$$

This formula predicts a very sharp phase transition from $\langle Sol \rangle \ll 1$ to $\langle Sol \rangle \gg 1$, which may easily be located numerically. We plot where this occurs using black lines in the first row of Figure 4.

This prediction is generally reasonably accurate, except that for very low and very high pattern densities, we overestimate the satisfiable region. This is due to variance: although an expected number of solutions much below one implies a high likelihood of unsatisfiability, it is not true that a high expected number of solutions implies that any particular instance is likely to be satisfiable. (Consider, for example, a sparse graph which has several isolated vertices. If one solution exists, other symmetric solutions can be obtained by permuting the isolated vertices. Thus although the expected number of solutions may be one, there cannot be exactly one solution.) A similar behaviour is seen with random constraint satisfaction problems (Smith & Dyer, 1996).

2.4 Variable- and Value-Ordering Heuristics

The Glasgow and LAD solvers are driven by ordering heuristics: variable-ordering tells us which pattern vertex to branch on, whilst value-ordering tells the order in which we should try target vertices for that pattern vertex. Various general principles have been considered when designing variable- and value-ordering heuristics for backtracking search algorithms—one of these is to try to maximise the expected number of solutions inside any subproblem considered during search (Gent, MacIntyre, Prosser, Smith, & Walsh, 1996). This is usually done by cheaper surrogates, rather than direct calculation. When branching, both LAD and Glasgow pick a variable with fewest remaining values in its domain: doing this will generally reduce the first part of the $\langle Sol \rangle$ equation (i.e. $t^{\underline{p}}$) by as little as possible. When two or more domains are of equal size, LAD breaks ties lexicographically, whereas Glasgow will pick a variable corresponding to a pattern vertex of highest degree. This strategy was

determined empirically, but could have been derived from the $\langle Sol \rangle$ formula: picking a pattern vertex of high degree will make the remaining pattern subgraph sparser, which will decrease the exponent in the second half of the formula, maximising the overall value. LAD does not apply a value-ordering heuristic, but Glasgow does: it prefers target vertices of highest degree. Again, this was determined empirically, but it has the effect of increasing $\langle Sol \rangle$ by leaving as many vertices as possible available for future use. The search strategy used by VF2, in contrast, is based around preserving connectivity, which gives very little discrimination except on the sparsest of inputs.

3. Induced Subgraph Isomorphisms

So far we have looked at the non-induced problem, where extra edges in the target graph are permitted. In the first five rows of Figure 5 we repeat our experiments, now searching for induced isomorphisms. With a pattern of order 10, we get two independent phase transitions: the bottom right halves of the plots resemble the non-induced results, and the top left halves are close to a mirror image. This second phase transition comes from the fact that an empty pattern is not an induced subgraph of a complete target: an empty pattern of order k is an induced subgraph only if there exist k vertices in the target that are not pairwise adjacent. The central satisfiable region, which is away from either phase transition, is computationally easy, but instances near the phase transition are hard.

For larger patterns of order 20 and 30, we have a large unsatisfiable region in the middle. Despite not being near either phase transition, instances in the centre remain computationally challenging for every solver. We also plot patterns of orders 14, 15 and 16, to show the change between the two behaviours.

We might expect these complexity plots to be symmetric along the diagonal, since for the induced problem, if we replace both inputs with their complements, the solutions remain the same. For the Glasgow solver, this is almost the case. This should be expected, because this complement property is precisely how the Glasgow solver handles the induced variant (although the heuristics may differ between the two, which we discuss below). For LAD, some of the very dense patterns are slightly harder than their diagonal opposites (LAD reasons about degrees, but not about complement-degrees).

It is interesting to note that for larger target graphs, VF2 finds *all* dense pattern graphs difficult. Meanwhile, VF3 behaves better than VF2 in some regions, but worse than VF2 in others, and has a larger hard region than either domain-based algorithm.[2] VF3 also occasionally finds some instances with sparse target graphs (along the bottom of the plot) extremely difficult, whilst other solvers do not.

3.1 Predictions and Heuristics

To predict the location of the induced phase transition, we repeat the argument for locating the non-induced phase transition and additionally consider non-edges, to get an expected

2. The version of VF3 supplied by its inventors will crash if given an empty pattern graph. We have chosen to treat these results as successes taking zero search, even though empty pattern graphs are not trivial for the induced problem. If the reader prefers to interpret a crash as a timeout, the VF3 plots would contain an additional area of black points down the left-hand side.

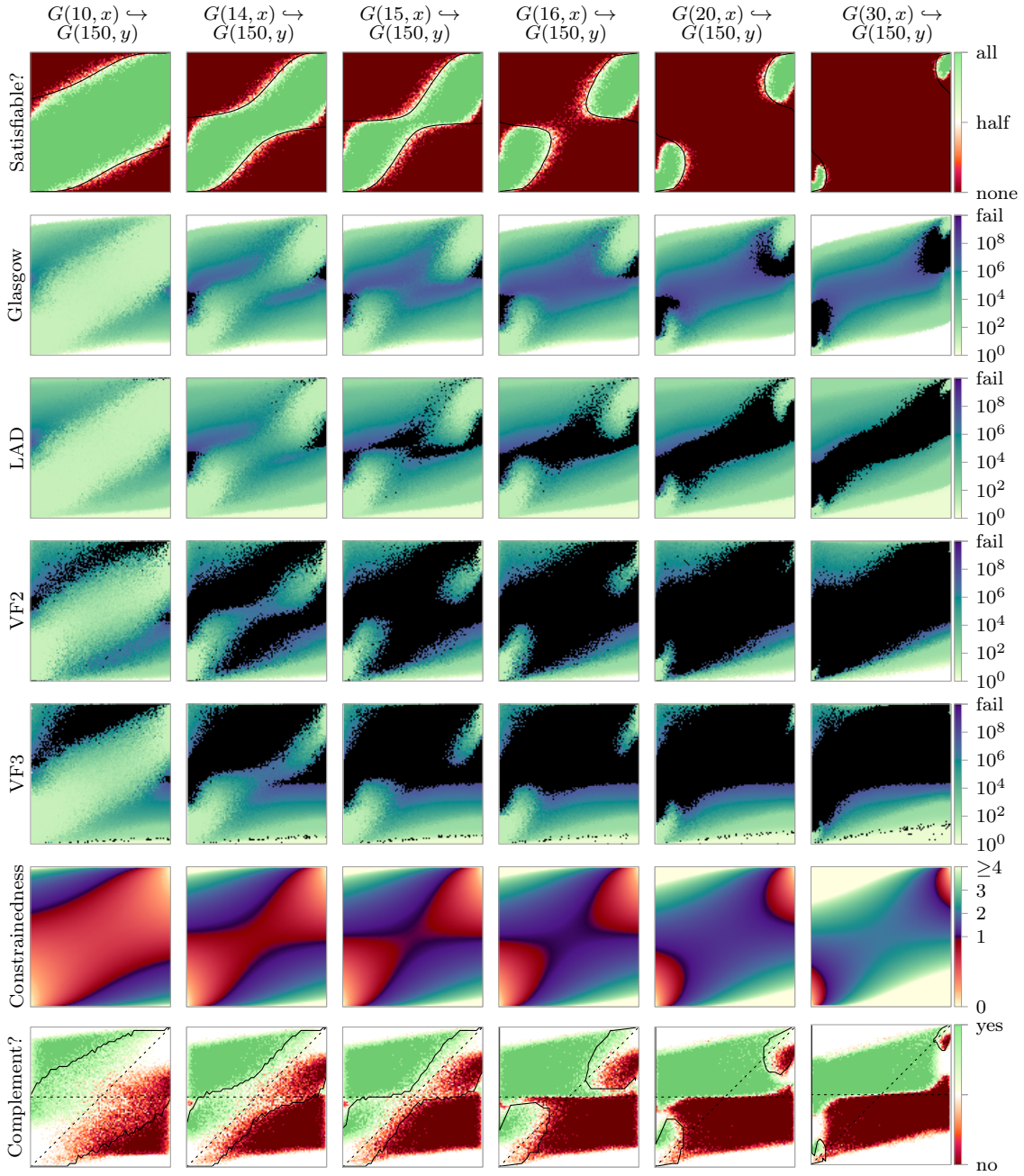

Figure 5: Behaviour of algorithms on the induced variant with target graphs of 150 vertices, shown in the style of Figure 4. The sixth row plots constrainedness using equation (3): the darkest region is where $\kappa = 1$, and the lighter regions show where the problem is either over- or under-constrained. 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.

number of solutions of

$$\langle Sol \rangle = t^{p} \cdot {d_t}^{d_p \cdot \binom{p}{2}} \cdot (1 - d_t)^{(1-d_p) \cdot \binom{p}{2}}. \quad (2)$$

We plot this using black lines on the top row of Figure 5—again, our prediction is accurate except for very sparse or very dense patterns.

We might guess that degree-based heuristics would just not work for the induced problem: for any claim about the degree, the opposite will hold for the complement constraints. Should we therefore resort to just using "smallest domain first", abandoning degree-based tiebreaking and the value-ordering heuristic? Empirically, this is not the case: on the final row of Figure 5, we show whether it is better to use the original pattern and target as the input to the Glasgow algorithm, or to take the complements. (The only steps performed by the Glagsow algorithm which differ under taking the complements are the degree-based heuristics. LAD, VF2, and VF3 are not symmetric in this way: LAD performs a filtering step using degree information, but does not consider the complement degree, and VF2 and VF3 use connectivity.)

For patterns of order 10, it is always better to try to move towards the satisfiable region: if we are in the bottom right diagonal half, we are best retaining the original heuristics (which move us towards the top left), and if we are in the top left we should use the complement instead. This goes against a suggestion by Walsh (1998) that switching heuristics based upon an estimate of the solubility of the problem may offer good performance.

For larger patterns, more complex behaviour emerges. If we are in the intersection of the bottom half and the bottom right diagonal of the search space, we should always retain the original heuristic, and if we are in the intersection of the top half and the top left diagonal, we should always use the complements. This behaviour can be predicted by taking the partial derivatives of $\langle Sol \rangle$ in the $-d_p$ and d_t directions. However, when inside the remaining two eighths of the parameter space, the partial derivatives of $\langle Sol \rangle$ disagree on which heuristic to use, and using directional derivatives is not enough to resolve the problem. A close observation of the data suggests that the actual location of the phase transition may be involved—and perhaps Walsh's (1998) suggestion applies only in these conditions. In any case, $\langle Sol \rangle$ is insufficient to explain the observed behaviour in these two eighths of the parameter space.

In practice, this is unlikely to be a problem: most of the real-world instances we have seen tend to be relatively sparse. In this situation, these experiments justify reusing the non-induced heuristics on induced problems.

3.2 Is the Central Region Genuinely Hard?

The region in the parameter space where both pattern and target have medium density is far from a phase transition, but nevertheless contains instances that are hard for all four solvers. We would like to know whether this is due to a weakness in current solvers (perhaps our solvers cannot reason about adjacency and non-adjacency simultaneously?), or whether instances in this region are inherently difficult to solve. Thus we repeat the experiments on smaller pattern and target graphs, using different solving techniques. Although these techniques are not competitive in absolute terms, we wish to see if the same pattern of behaviour occurs. The results are plotted in Figure 6.

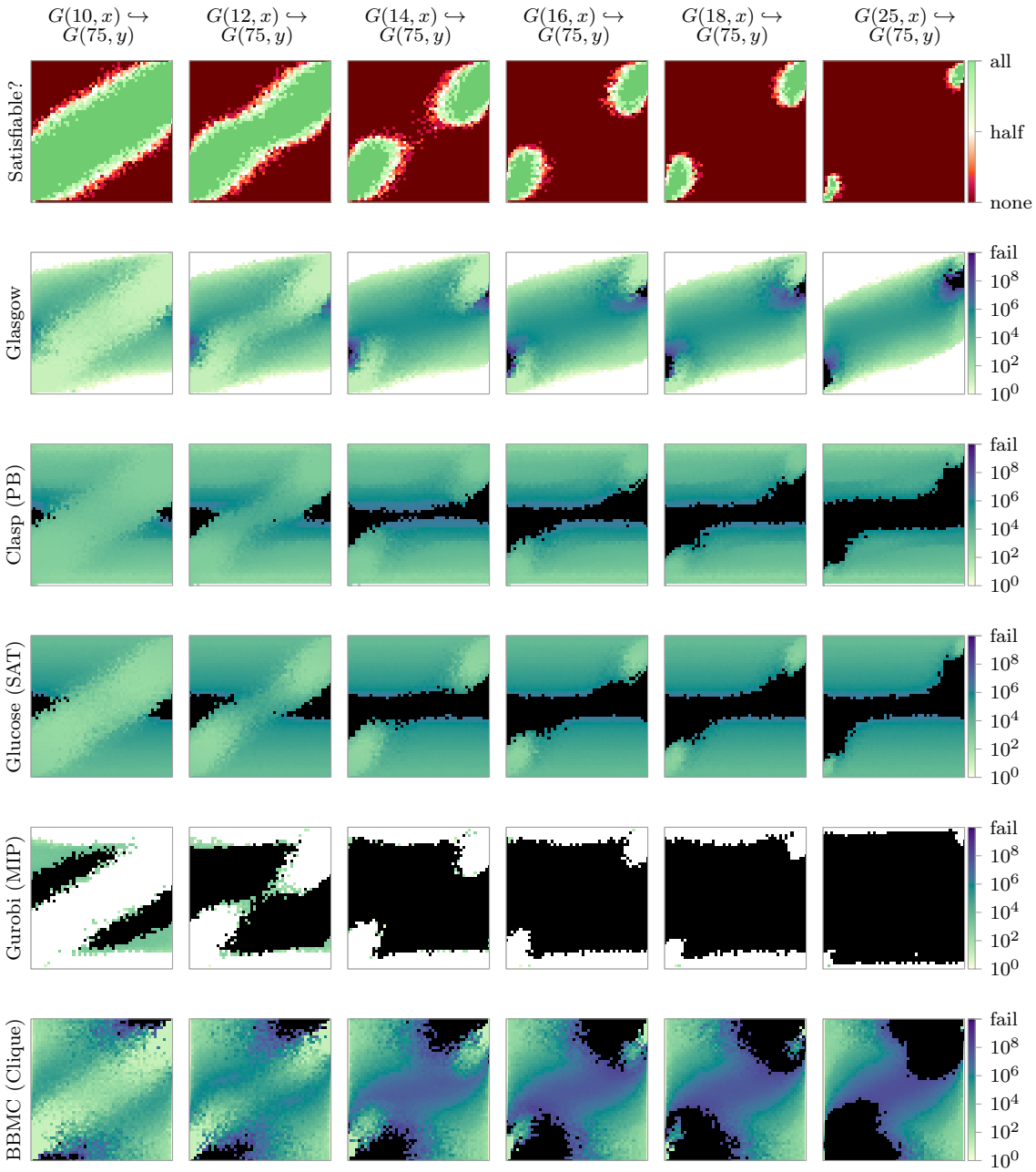

Figure 6: Behaviour of other solvers on the induced variant using smaller target graphs with 75 vertices, shown in the style of Figure 4. The second row shows the number of search nodes used by the Glasgow algorithm, the third and fourth rows show the number of decisions made by the pseudo-boolean and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

The pseudo-boolean (PB) encoding is as follows. For each pattern vertex v and each target vertex w, we have a binary variable which takes the value 1 if and only if v is mapped to w. Constraints are added to ensure that each pattern vertex maps to exactly one target vertex, that each target vertex is mapped to by at most one pattern vertex, that adjacent vertices are mapped to adjacent vertices, and that non-adjacent vertices are mapped to non-adjacent vertices. We use the Clasp solver (Gebser, Kaufmann, Kaminski, Ostrowski, Schaub, & Schneider, 2011) version 3.1.3 to solve the pseudo-boolean instances. The instances that are hard for the Glasgow solver remain hard for the PB solver, including instances inside the central region, and the easy satisfiable instances remain easy. Similar results are seen with the Glucose SAT solver (Audemard & Simon, 2014) using a direct encoding. We also show an integer program encoding: the Gurobi solver is only able to solve some of the trivial satisfiable instances, and was almost never able to prove unsatisfiability within the time limit.

The *association graph encoding* of a subgraph isomorphism problem is a reduction to the clique decision problem. McCreesh, Ndiaye, Prosser, and Solnon (2016) describe and study this approach in more detail. Briefly, the association graph is constructed by creating a new graph with a vertex for each pair (p, t) of vertices from the pattern and target graphs respectively. There is an edge between vertex (p_1, t_1) and vertex (p_2, t_2) if mapping p_1 to t_1 and p_2 to t_2 simultaneously is permitted, i.e. p_1 is adjacent to p_2 if and only if t_1 is adjacent to t_2. A clique of size equal to the order of the pattern graph exists in the association graph if and only if the problem is satisfiable (Levi, 1973). We used this encoding with an implementation of San Segundo, Rodríguez-Losada, and Jiménez's (2011) bit-parallel maximum clique algorithm BBMC, modified to solve the decision problem rather than the optimisation problem. Again, our results show that the instances in the central region remain hard, and additionally, some of the easy unsatisfiable instances become hard.

Together, these experiments suggest that the central region may be genuinely hard, despite not being near a phase transition. The clique results in particular rule out the hypothesis that subgraph isomorphism solvers only find this region hard due to not reasoning simultaneously about adjacency and non-adjacency, since the constraints in the association graph encoding consider compatibility rather than adjacency and non-adjacency.

3.3 Constrainedness

Constrainedness, denoted κ, is an alternative measure of difficulty designed to refine the phase transition concept, and to generalise hardness parameters across different combinatorial problems (Gent et al., 1996). An ensemble of problem instances with $\kappa < 1$ is said to be underconstrained, and is likely to be made up of satisfiable instances; an ensemble with $\kappa > 1$ is overconstrained, and is likely to be made of unsatisfiable instances. Empirically, problem instances from an ensemble with κ close to 1 are hard, and problem instances drawn from an ensemble where κ is very small or very large are usually easy.

By handling injectivity as a restriction on the size of the state space rather than as a constraint, we derive

$$\kappa = 1 - \frac{\log\left(\underline{t^p} \cdot {d_t}^{d_p \cdot \binom{p}{2}} \cdot (1 - d_t)^{(1-d_p)\cdot\binom{p}{2}}\right)}{\log \underline{t^p}} \tag{3}$$

for induced isomorphisms, which we plot on the sixth row of Figure 5. We see that constrainedness predicts that the central region will still be relatively difficult for larger pattern graphs: although a family of instances generated with these parameters is overconstrained, it is less overconstrained than in the regions the Glasgow and LAD solvers found easy. Thus it seems that rather than just being a unification of existing generalised heuristic techniques, constrainedness also gives a better predictor of difficulty than proximity to a phase transition—our method generates collections of instances where constrainedness and "close to a phase transition" give very different predictions, and constrainedness closely matches the empirical results.

Unfortunately, constrainedness does not give additional heuristic information: minimising constrainedness gives the same predictions as maximising the expected number of solutions.

4. k-Regular Graphs

What about richer graph structures? In this section we briefly look at what happens with the non-induced problem if we use k-regular graphs rather than the Erdős-Rényi model, and then in the following section we see the effects of introducing vertex labels. Both models still exhibit phase transition behaviour, although with sufficiently many labels the "satisfiable" region now contains a mix of satisfiable and unsatisfiable instances.

By $R(n, k)$ we mean a random graph with n vertices, each of which has degree k. In Figure 7 we recreate Figure 4, using the NetworkX implementation (Hagberg, Schult, & Swart, 2008) of the Steger and Wormald (1999) algorithm to generate regular graphs. The x-axes range from degree 0 to degree $n-1$ where $n \in \{10, 20, 30\}$ is the number of pattern vertices, and the y-axes range from degree 0 to degree 149 (one less than the number of target vertices). The satisfiable / unsatisfiable plots show very similar results to the non-induced problem on Erdős-Rényi random graphs, with a similarly sharp phase transition, and again only instances near the phase transition are difficult.

Regularity means that degree-based heuristics have no information to work with. Additionally, the degree-based filtering techniques used by Glasgow and LAD have no effect on these graphs at the top of search, and so initially every domain is identical. However, once a guessed assignment has been made, selecting from small domains first remains an effective strategy to guide the remainder of the search.

5. Labelled Graphs

So far, we have looked at unlabelled graphs. What happens when labels on vertices are introduced? This is common in real-world applications—for example, when working with graphs representing chemical molecules, mappings are typically expected only to match carbon atoms with carbon atoms, hydrogen atoms with hydrogen atoms, and so on. We will look at the non-induced variant, as this seems to be more common in the literature.

5.1 Predictions and Empirical Hardness

Suppose our labels are drawn randomly from a set $L = \{1, \ldots, k\}$, where k is reasonably small compared to the number of pattern vertices p. Recall that $\ell(v)$ is the label of vertex

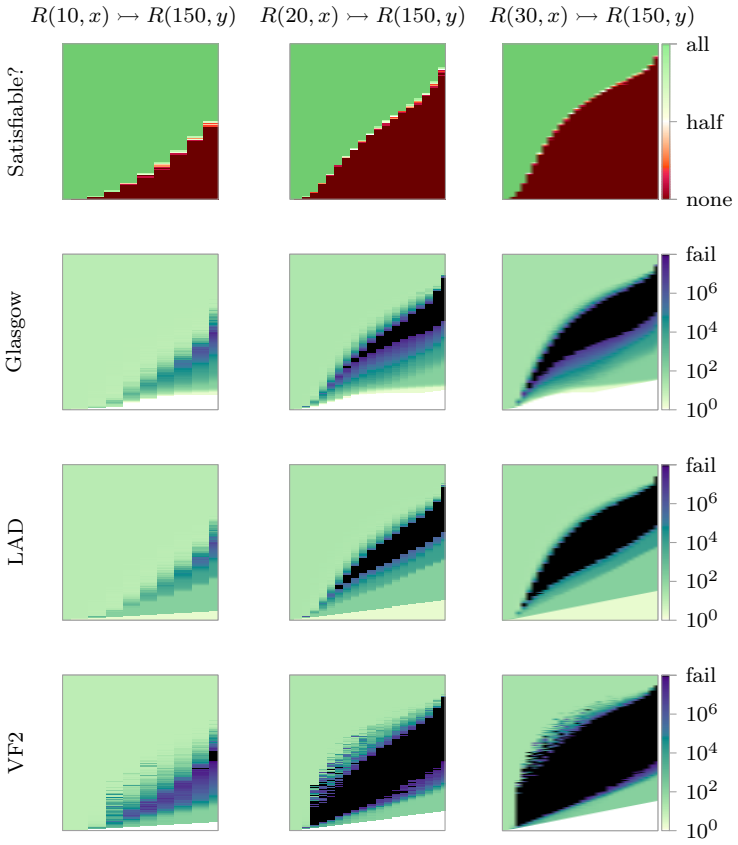

Figure 7: Behaviour of algorithms on the non-induced problem on random k-regular graphs, as the pattern degree (x-axis) and target degree (y-axis) are varied from 0 to $|\mathrm{V}(G)| - 1$, using target graphs of 150 vertices, and pattern graphs of 10, 20, and 30 vertices.

v. By defining

$$\mathrm{V}(P)|_x = \{v \in \mathrm{V}(P) : \ell(v) = x\}$$

to be the set of vertices with label x, we can partition the pattern vertices by label into disjoint sets $\{\mathrm{V}(P)|_1, \ldots, \mathrm{V}(P)|_k\}$, each of which is expected to contain p/k vertices. Similarly, we may partition the target vertices into disjoint sets $\{\mathrm{V}(T)|_1, \ldots, \mathrm{V}(T)|_k\}$.

Without labels, there are $t^{\underline{p}} = t \cdot (t-1) \cdot \ldots \cdot (t-p+1)$ possible injective assignments of target vertices to pattern vertices. With labels, observe that for any label x, vertices in $\mathrm{V}(P)|_x$ may only be mapped to vertices in $\mathrm{V}(T)|_x$. Thus for each label x, we have an expected p/k variables, each of whose domains contain t/k values. We would like to say that the size of the state space is now

$$|S| = \left((t/k)^{\underline{p/k}}\right)^k,$$

but to do this we must state what $a^{\underline{b}}$ means when b is fractional. The gamma function $\Gamma(n)$ is equal to $(n-1)!$ for integers $n \geq 1$ (Davis, 1959), but is also defined for positive

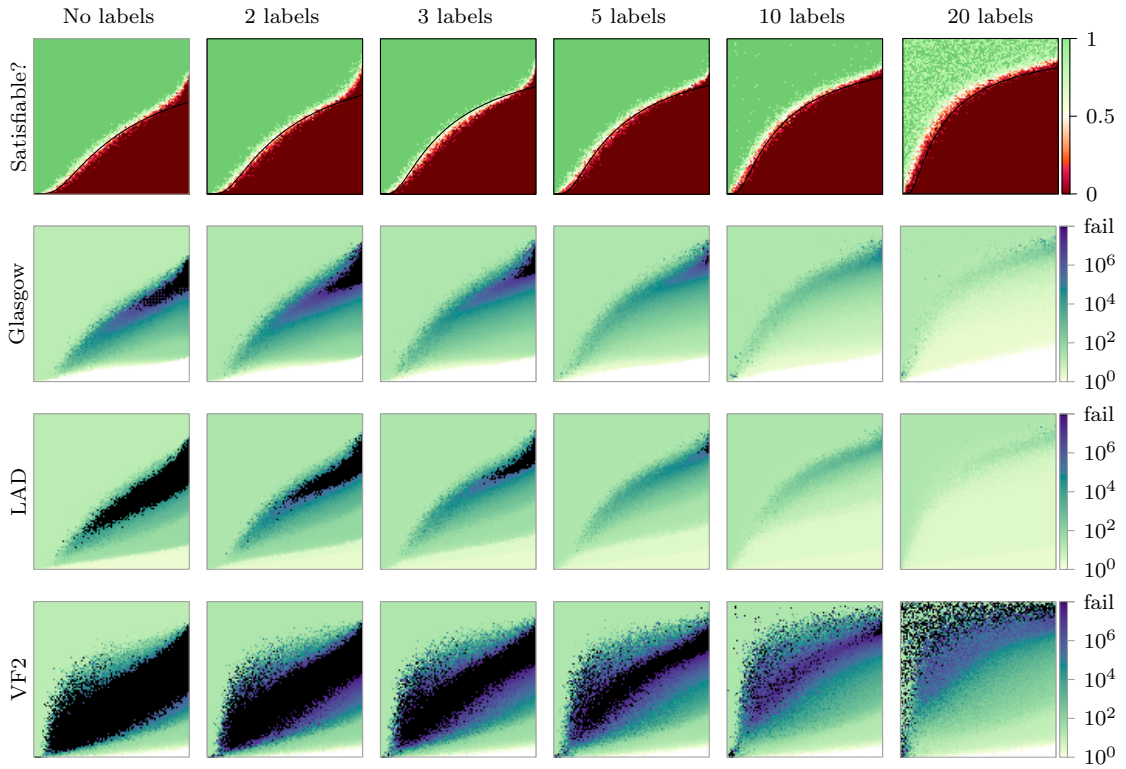

Figure 8: On the top row, predicted and actual location of the phase transition for labelled non-induced random subgraph isomorphism, with a pattern order of 20, a target order of 150, varying pattern (x-axis) and target (y-axis) density, and varying numbers of labels. On subsequent rows, the average number of search nodes needed to solve an instance, for three different solvers

real numbers, obeying the identify $\Gamma(x+1) = x\Gamma(x)$. By noting that $t^{\underline{p}} = {}^{t!}/_{(t-p)!}$, we may obtain a reasonable continuous extension by taking

$$|S| = \left(\frac{\Gamma\left({}^{t}/_{k} + 1\right)}{\Gamma\left({}^{t}/_{k} - {}^{p}/_{k} + 1\right)} \right)^{k}.$$

As before, we expect the pattern to have $d_p \cdot \binom{p}{2}$ edges, and so if we simplify by assuming the pattern will have *exactly* this many edges, we obtain the probability of all of these edges being mapped to edges in the target by raising d_t to this power, giving an estimate of

$$\langle Sol \rangle = \left(\frac{\Gamma\left({}^{t}/_{k} + 1\right)}{\Gamma\left({}^{t}/_{k} - {}^{p}/_{k} + 1\right)} \right)^{k} \cdot {d_t}^{d_p \cdot \binom{p}{2}}. \tag{4}$$

So how good are these predictions? The black lines on the first row of heatmaps in Figure 8 plot where we calculate $\langle Sol \rangle = 1$ will occur. For small numbers of labels, our predictions are slightly better than in the unlabelled case: there seems to be less of a variance problem for very dense patterns. Even as the number of labels becomes relatively

large, the prediction of the phase transition still occurs in the right place, but with ten labels we start to see sporadic unsatisfiable instances deep inside the satisfiable region. With twenty labels, we instead get a kind of phase transition from "all unsatisfiable" to "mixed satisfiable and unsatisfiable". We can understand this intuitively: with sufficiently many labels, we might generate, say, a red vertex adjacent to a blue vertex in the pattern, but not in the target. With twenty labels, we even sometimes generate no vertices with a particular label in the target at all. This in many ways resembles "flaws" generated by certain random constraint satisfaction problem instance generators (Achlioptas, Molloy, Kirousis, Stamatiou, Kranakis, & Krizanc, 2001; Gent, MacIntyre, Prosser, Smith, & Walsh, 2001).

What about empirical hardness? As before, some instances on the phase transition are hard for all solvers (although as the number of labels increases, the hardest instances become easier). Instances far from the phase transition are easy, except for VF2, which occasionally finds some of the instances with larger numbers of labels very difficult. This behaviour occurs both on satisfiable instances, and on the flawed unsatisfiable instances deep inside the "satisfiable" region. It is interesting to observe that all such unsatisfiable instances that VF2 finds difficult have a very small proof of unsatisfiability, with neither Glasgow nor LAD requiring more than one hundred recursive calls.

5.2 Richer Label Models

Why does VF2 find some of these instances so hard? Unlike Glasgow and LAD, VF2 does not track domains, and so cannot detect that there are no suitable target vertices available for a given pattern vertex until it branches on that vertex. It also cannot detect small domains, and only uses "adjacency to an existing assignment" as a branching heuristic. This can make it very hard for VF2 to detect that it is in an obviously failed state, or that an instance or subproblem is trivially unsatisfiable.

To illustrate this point further, we now look at some instances created using a slightly more structured random model. We create a family of one thousand labelled instance pairs, as follows. To create a pattern graph, we create ten vertices with label zero, with edges between these vertices with probability 0.2. We then add another ten vertices, with labels chosen randomly between one and thirty, and add edges between these vertices and the zero-labelled vertices with probability 0.1. To generate a target, we follow a similar process: we have fifty vertices with label zero and edge probability 0.2, fifty vertices with labels randomly between one and thirty, and edges from the first set of vertices to the second set with probability 0.3. This model was selected and the parameters tuned to provide a demonstration of particular behaviours of VF2, not because of any natural property (although inspiration came from seeking a very crude approximation of chemical graphs, which often contain a lot of carbon in the centre, and other atoms around the outside). From our set of one thousand such instances, sixty six are satisfiable.

We plot the cumulative number of instances solved over time for these instances in Figure 9: for a given runtime choice along the x-axis, the y value for a given algorithm shows how many instances, individually, took at most x milliseconds to solve with that algorithm. Both Glasgow and LAD find all of these instances trivial, with no instance

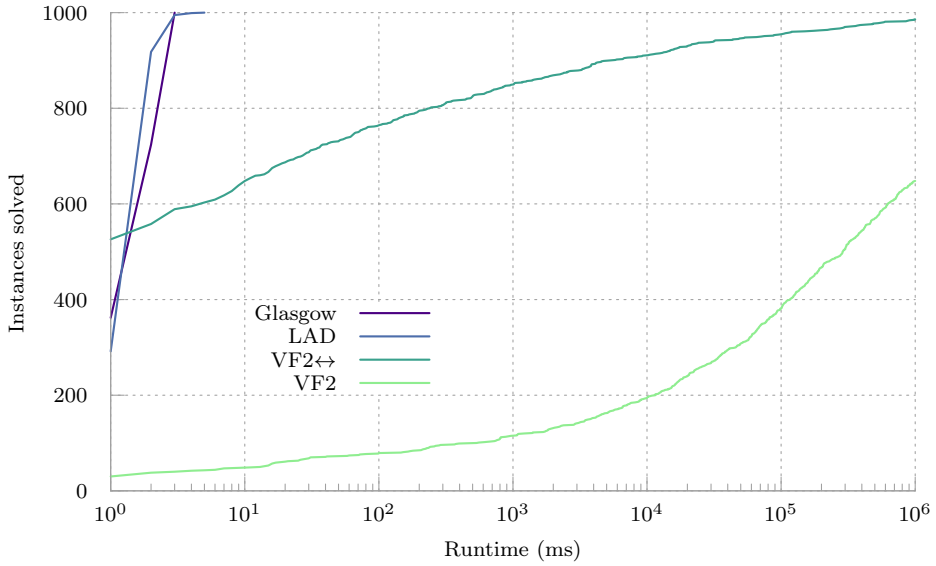

Figure 9: The cumulative number of instances solved over time, using the richer model of randomness described in Section 5.2.

requiring even ten milliseconds to solve. VF2, in contrast, finds many of them extremely difficult, and cannot solve 352 of the 1,000 instances with a one thousand second timeout.

The "VF2↔" line shows what happens with VF2 if the graphs are permuted, so that the non-zero labelled vertices are given lower vertex numbers rather than higher vertex numbers. In other words, we permute pattern vertices in such a way that the first ten vertices have non-zero labels, and the ten last vertices have zero labels (and similarly for the target graphs). This makes VF2 behave better, but still very poorly compared to the other two solvers; the motivation behind this modification will become clear later in this paper. It will also be important to remember that both VF2 variations find some satisfiable *and* some unsatisfiable instances extremely hard.

An even more extreme example of VF2's misbehaviour can be seen in Figure 10. Here we have a pattern graph and a target graph, both of which are cliques plus one isolated vertex, and the isolated vertices have different unique labels. This is trivially unsatisfiable (and Glasgow and LAD detect this without search), but unless the labelled vertex is given the lowest vertex number (so it is branched on first), VF2 takes nearly a million recursive calls to detect this: VF2 will try to map every adjacent vertex in the clique before considering the unmatchable isolated vertex. This example can be extended slightly to fool any simple static heuristic, or any simple label counting mechanism (for example, by attaching an additional vertex with a unique label to the clique). Note also the resemblance to the cases seen in the bottom left of the VF2 plots in Figure 4, where a pattern graph which has a few edges combined with a target graph with no edges gives a timeout.

These experiments further highlight that VF2 occasionally finds some instances which should be easy hard. But does this cause problems in practice? The literature suggests that

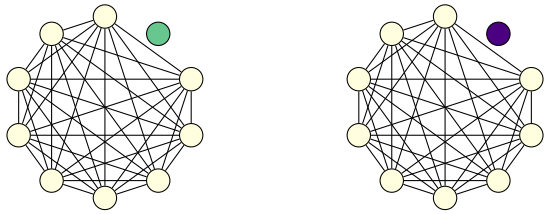

Figure 10: A trivially unsatisfiable subgraph isomorphism instance which VF2 finds exponentially difficult. The pattern (left) and target (right) both consist of a clique, plus one extra vertex which has a different label in each graph.

it does. For example, Grömping (2014) uses VF2 in a package for the R statistics language, and states

> "There are (not so many) instances for which creation of a clear design is prohibitively slow in the current implementation that evaluates subgraph isomorphism with the VF2 algorithm ... Recent experiences with a few of these showed that the LAD algorithm was very fast in ruling out impossible matches, where VF2 took a long time." (p. 44)

Similarly, Murray (2012) uses VF2 inside a compiler, and observes extremely variable (and prohibitively high) compile times in some cases due to the expense of subgraph isomorphism calls—given the heavily labelled nature of these graphs, we conjecture that any domain-based algorithm would eliminate this cost.

However, by far the biggest problem is in graph databases. The following section shows how widespread use of certain non-constraint-based subgraph isomorphism algorithms has not just led to overly pessimistic conclusions regarding performance, but has misdirected the design of larger systems.

6. Querying Graph Databases

A particularly common use of subgraph isomorphism algorithms is inside graph databases. The general problem these systems solve is, for a set of target graphs, to process a pattern query and return every target graph which is subgraph-isomorphic to that pattern. The set of target graphs is usually seen as fixed, or at least rarely-changing, whilst the patterns arrive dynamically. This has led to the development of systems which perform extensive computations on the target graphs beforehand, in the hope of reducing the response times for individual pattern queries. The most popular of these strategies is a form of indexing which is often named *filter / verify*.

6.1 The Filter / Verify Paradigm

The filter / verify approach has an interesting history, of which we now give an abridged overview. Our description is biased by a general modern understanding of the empirical hardness of NP-complete problems, which was not widely known at the time of the earlier papers we discuss. The common theme of all of the following papers is that pre-computed

information is used to eliminate certain unsatisfiable instances from consideration, without performing a subgraph isomorphism test. For example, an index might contain a bit of information expressing whether a target graph contains at least two red vertices. When a pattern graph with two red vertices is used as a query, any target whose feature set does not have this bit set would not be considered, and so a subgraph isomorphism call would not be made for that pattern / target pair. (In practice, feature hashing is often used, which can lead to false positives, but this is not relevant to our discussion.)

An early graph database system by Shasha, Wang, and Giugno (2002) uses a filtering heuristic to eliminate unsatisfiable instances based upon simple structural elements. It is not clear whether the aim is to reduce I/O costs, or to reduce the number of queries which must be tested, and the experiments do not answer whether the indexing is effective. However, the work was influenced by a commercial graph database system, whose documentation (Daylight Chemical Information Systems, Inc., 2011) states that indexing is used to minimise disk accesses.

Subsequently, in a widely cited[3] paper, Yan, Yu, and Han (2004) introduce an indexing system called gIndex. Again, this system handles queries by first producing a set of candidates by eliminating certain unsatisfiable instances, this time by using substructures. They argue that the query response time, which is to be minimised, is governed by the equation

$$T = T_{search} + |C_q| \cdot T_{iso\_test},$$

where T_{search} is the time taken to search for a candidate set of potential solutions of size $|C_q|$, and T_{iso_test} is the cost of a subgraph isomorphism test. They reason that since subisomorphism testing is NP-complete, by making $|C_q|$ as small as possible, the query response time will be reduced. (Importantly, it is *not* the time taken to load graphs from disk which contributes to the per-candidate cost, but rather the time to perform a subgraph isomorphism call.) They conclude that "graph indexing plays a critical role at efficient query processing in graph databases" (p. 345).

This equation is repeated and expanded upon by Yan, Yu, and Han (2005) to explicitly separate I/O and subisomorphism testing costs into T_{io} and T_{iso_test} respectively, obtaining a query response time of

$$T_{search} + |C_q| \cdot (T_{io} + T_{iso\_test}).$$

The authors explicitly state that "the value of T_{iso_test} does not change much for a given query" (p. 996). The argument presented is that

> "Sequential scan is very costly because one has to not only access the whole graph database but also check subgraph isomorphism. It is known that subgraph isomorphism is an NP-complete problem. Clearly, it is necessary to build graph indices in order to help processing graph queries." (p. 961)

We reassess these claims in light of what we now know about the behaviour of modern subgraph isomorphism algorithms and the nature of solving NP-complete problems. Contrary to the authors' claims, we do not expect T_{iso_test} to be anything like a constant, even if the orders of the input graphs are similar. In particular, any instance which can be excluded based upon filtering must have a very small proof of unsatisfiability. These instances

3. 685 citations according to Google Scholar in February 2018.

should be trivial with any domain-based subgraph isomorphism algorithm. Thus, all filtering *should* be doing is eliminating the startup costs of a trivial subgraph isomorphism call. The fact that filtering was successful empirically should make us wonder whether the subgraph isomorphism algorithm used was substantially weaker than the state of the art. Indeed, the algorithm is described only as "the simplest approach" in the paper. Additionally, the experiments focus on reducing the size of the candidate set, without considering the time taken to verify different candidate set instances. The claim that T_{iso_test} does not change much is not justified experimentally, and no consideration is given as to whether this would hold true for other subgraph isomorphism algorithms.

Moving forwards, the (simpler form of the) query response time equation is repeated by Zhao, Yu, and Yu (2007). Again, the work has a focus on reducing the candidate set size through indexing. The authors appear to believe that the cost of the subisomorphism test is not a major factor in influencing the result, and focus on the remaining terms in the equation. They use the "average cost" of a subgraph isomorphism test as a constant, without considering that the average cost could be influenced by the character of the candidate set.

The equation is also used by Jiang, Wang, Yu, and Zhou (2007), who claim that "usually the verification time dominates the Query Response Time [s]ince the computational complexity of T_{iso_test} is NP-Complete". They note that

> "Approximately, the value of T_{iso_test} does not change too much with the difference of query. Thus, the key to reducing query response time is to minimize the size of the candidate answer set". (p. 568)

This claim becomes understandable when one examines the subgraph isomorphism algorithm chosen for the verification step (Ullmann, 1976): as the algorithm predates techniques like generalised arc-consistent all-different propagation (Régin, 1994), it cannot immediately detect unsatisfiability in simple cases like there being two red vertices in the pattern but only one in the target.[4]

Without using the equation, Cheng, Ke, Ng, and Lu (2007) state that since subgraph isomorphism is NP-complete, processing by a sequential scan is infeasible. They introduce new filtering techniques to try to avoid the subgraph isomorphism step. A similar claim is made by Zhang, Hu, and Yang (2007) in an introduction of another indexing technique: "obviously it is inefficient to perform a sequential scan on every graph in the database, because the subgraph isomorphism test is expensive" (p. 966).

Muddying the waters slightly, a survey by Han, Cheng, Xin, and Yan (2007) states that "large volumes of data" (not NP-completeness) is the reason for using indexing in these systems. However, in a description of a system tailored to biological networks, Zhang, Li, and Yang (2009) state that

> "Since the size of the raw database graph is small, it can be easily fit in the main memory. However, the query (matching) time will be very long due to the NP-hard complexity." (p. 193)

They suggest that indexing is a way of avoiding this. Similarly, Zhao and Han (2010) argue that "the graph query problem is hard in that ... it requires subgraph isomorphism checking

4. Although interestingly, this algorithm effectively maintains (binary) arc consistency, and has a variable-ordering heuristic, before these concepts appeared in the constraints literature.

... which has proven to be NP-complete" and that working with large networks is hard or impossible "due to the lack of scalable graph indexing mechanisms" (p. 340).

Cao, Yang, Wang, Ren, and Lou (2011) propose a privacy-preserving cloud graph database system using filter / verify, stating that "checking subgraph isomorphism is NP-complete, and therefore it is infeasible to employ such a costly solution" (p. 393) which simply checks every graph for a match. Wang, Wang, Yang, and Yu (2012) look at indexing large sparse graphs. They state that "because subgraph isomorphism is an NP-complete problem, a filter-and-verification method is usually employed to speed up the search efficiency of graph similarity matching over a graph set" (p. 441). Similarly, after reviewing the literature, Yuan and Mitra (2013) argue that subgraph querying is costly because it is NP-complete, and that indices can improve the performance of graph database queries. Again, new indexing techniques are introduced. Subsequently, Katsarou, Ntarmos, and Triantafillou (2015) perform a comprehensive comparison of graph database filtering techniques. They state that performing a query against each graph in the dataset "obviously does not scale, as subgraph isomorphism is NP-complete" (p. 1566). In describing a system which returns a special subset of graphs which match a query, Zheng, Lian, Zou, Hong, and Zhao (2016) suggest that it is "NP-hard to check the graph isomorphism" (meaning subgraph isomorphism), and "in order to improve the time efficiency" they use an indexing system to "avoid as many costly subgraph isomorphism checkings as possible" (p. 1806). Their index takes tens of thousands of seconds to build, and they suggest that Ullmann's algorithm and VF2 are state of the art for verification. Peng, Zou, Chen, Lin, and Zhao (2016) state that "obviously, it is impossible to employ some subgraph isomorphism algorithm, such as Ullmann or VF2", and argue that "in order to speed up query processing", they need to create indices (p. 418). And Azaouzi and Romdhane (2016) argue that "since the subgraph isomorphism test is expensive, checking all graphs of a large database can be unfeasible", saying that "naturally, the verification step is computationally more expensive since it requires a subgraph isomorphism" (p. 251); their experiments look at candidate set reduction sizes, and their choice of subgraph isomorphism algorithm is not mentioned.

The filter / verify paradigm also influences other research. For example, it is used by Tian and Patel (2008) in an approximate subgraph searching system: they state that Ullmann's algorithm "is prohibitively expensive for querying against [a] database with a large number of graphs", and that indices are used "to filter out graphs that do not match the query" (p. 964). More recently, Yuan, Mitra, and Giles (2013) continue a line of supergraph search work, again using a filtering step to avoid subgraph isomorphism calls (in the opposite direction, so queries are now target graphs). Hong, Zou, Lian, and Yu (2015) look at graph database subgraph isomorphism with an additional set-similarity constraint, and state that Ullmann's algorithm and VF2 are "usually costly for large graphs" because they "do not utilize any index structure" (p. 2509). They propose an indexing structure, which takes over 2,000 seconds to construct, and uses nearly 2GBytes of space. There is also research into maintaining indices when the set of target graph changes: for example Yuan, Mitra, Yu, and Giles (2015) look at algorithms for updating graph indices. And describing a system for reusing results of queries which are sub- or super-graphs of previous queries, Wang, Ntarmos, and Triantafillou (2016) state that querying is a "very costly operation as it entails the NP-complete problem of subgraph isomorphism" (p. 41), and place "an emphasis on the number of unnecessary subgraph isomorphism tests" (p. 42).

After some early ambiguity, then, it becomes clear that the intent behind filter / verify systems is to reduce the number of subgraph isomorphism calls, and that the cost of loading graphs from disk is not considered to be problematic. It is worth noting that the entire test datasets from most of these papers will comfortably fit in RAM on a modern desktop machine, even when an adjacency matrix representation is used.

Thus we can see that there are two critical beliefs underlying all of this work—firstly, that subgraph isomorphism is necessarily hard because it is NP-complete, and secondly, that there are ways of identifying unsatisfiable instances using short proofs that a subgraph isomorphism algorithm will not detect, but that an indexing system can. Throughout, the cost models used assume that the time for a subgraph isomorphism query does not particularly depend upon the instance, and nowhere is it considered that a good subgraph isomorphism algorithm should be able to eliminate obviously-unsatisfiable instances with a similar time requirement to an indexing system.

These beliefs are not entirely unfounded: none of the subisomorphism algorithms considered in these papers will immediately detect if a pattern contains two red vertices, whilst the target graph contains only one. This kind of flaw *should* be picked up at the top of search by an all-different propagator (Régin, 1994). However, as Katsarou et al. (2015) note, VF2 (Cordella et al., 2004) or a similar algorithm is the usual subgraph isomorphism algorithm of choice for graph database systems, although Ullmann's algorithm is sometimes chosen. Other approaches have been considered, albeit not with algorithms strong enough to establish all-difference. For example, Shang, Zhang, Lin, and Yu (2008) propose an algorithm which makes use of the frequency of various features to guide search; Lee, Han, Kasperovics, and Lee (2012) determine experimentally that this technique tends to be very effective, even on families of graphs for which it was not designed.

We therefore believe it would be unlikely to cause too much astonishment if we suggested that a better subgraph isomorphism algorithm could be dropped in as a black box replacement in graph databases systems to improve their performance. This is not our claim. Instead, this paper shows that better algorithms invalidate the premise underlying the entire filter / verify approach. We will now demonstrate empirically that filter / verify is simply a limited workaround for a subset of the weaknesses in non-constraint-based subgraph isomorphism algorithms that we demonstrated towards the end of the previous section.

6.2 Is Filtering Necessary?

To show that a pure subgraph isomorphism approach is feasible, with no indexing or supporting preprocessing, we look at four datasets commonly used in graph indexing evaluations. We do not claim that these are high-quality datasets or that the associated queries are sensible, merely that following the example of Giugno et al. (2013), they are widely used. Each dataset has labels on vertices (but not on edges), and we will look at the non-induced problem.

- The AIDS dataset contains graphs representing 40,000 chemical molecules. These graphs are labelled, and are fairly small (mean 45 vertices) and sparse. Following Giugno et al., the queries are compounds with 8, 16 or 32 edges.

- The PDBS dataset (He, Lin, Chipman, Bator, Baker, Shoham, Kuhn, Medof, & Rossmann, 2002) contains 30 labelled graphs representing DNA, RNA, and proteins, each duplicated twenty times to give a dataset of 600 graphs. These can be relatively large, having up to tens of thousands of vertices, but are extremely sparse. The queries are randomly selected connected subgraphs and do not have a real-world meaning.
- The PCM dataset (Vehlow, Stehr, Winkelmann, Duarte, Petzold, Dinse, & Lappe, 2011) contains 50 protein contact maps, each duplicated four times to give a dataset of 200 graphs. These graphs have under a thousand vertices and below twenty thousand edges; they are slightly less sparse than the previous two datasets. As for PDBS, the queries are randomly generated and do not have a particular meaning.
- The PPI dataset contains 20 protein interaction networks, with up to ten thousand vertices. The queries have either four or eight vertices.

Rather than the modern techniques used by Glasgow and LAD, we deliberately select a very simple starting point: a constraint programming model implemented in Gecode 5.1.0 using only toolkit-provided constraints and heuristics. The model uses a "smallest domain first" variable ordering heuristic with tie-breaking on degree, an all-different constraint, extensional constraints for adjacency, and labelled-degree filtering at the top of search. Even this simple approach finds nearly every instance in each of the four datasets trivial, and no exponential behaviour is observed. As we show in Figure 11, the *hardest* instance for the AIDS dataset requires 287 recursive calls, and 149,268 of the instances can be solved without search; furthermore, no instance individually requires more than 12ms. For both PCM and PPI, the hardest problem requires 23 recursive calls, and no instance takes more than 12ms and 282ms respectively. For PDBS, the hardest problem requires 3,161 recursive calls (which occurs 20 times, due to duplicated queries); however, Gecode struggles on these instances due to its general purpose data structures not scaling well to the large target graph sizes, and some of these instances can take as long as 30s to solve.

In other words, none of these instances are in any way computationally hard even for a simple domain-based algorithm, even before introducing stronger filtering, inference, or heuristics, and the only difficulties arise from Gecode requiring the use of extensional constraints rather than the specialised adjacency data structures used by Glasgow and LAD. Furthermore it is certainly not the case that a sequential scan is infeasible as so many filter / verify papers claim. In contrast, with VF2, the hardest instance in PDBS requires 31,189 recursive calls; in AIDS, 168,569 calls; in PCMS, 6,476,072 calls; and in PPI, 2,913. In other words, any exponential behaviour seen in these instances is purely down to the choice of subgraph isomorphism algorithm, and these instances are not inherently "really hard".

6.3 Benefits of Constraint Programming over Filter / Verify

There are at least four benefits beyond simplicity towards abandoning filter / verify in favour of a purely constraint programming inspired approach:

- Domain filtering is useful on both satisfiable and unsatisfiable instances, whilst filtering can only eliminate trivially unsatisfiable instances, and is entirely wasted on satisfiable instances. Domain filtering is also useful even on relatively hard instances,

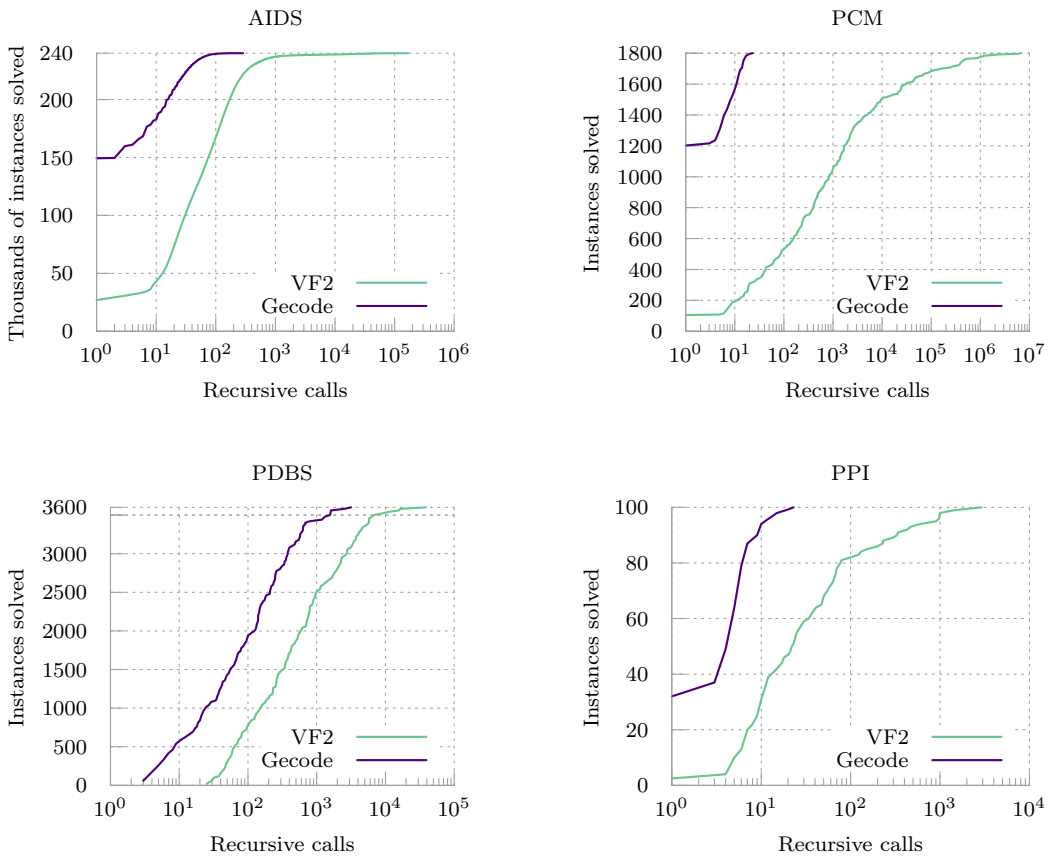

Figure 11: The cumulative number of instances solved as a function of search space size, using four graph database datasets, and either VF2 or a simple constraint programming subgraph isomorphism algorithm implemented in Gecode. The x-axis shows recursive calls, using a log scale, and each line ends at the x value where every instance has been solved.

since the information cuts down the search space rather than providing a simple "yes" or "no".

- Domain-based heuristics are widely studied and well-understood (Gent et al., 1996), and unlike VF2's adjacency branching rules, provide discrimination even on dense and low-diameter graphs. Picking from small domains dynamically allows search to focus on the hardest part of the problem, and can exploit conditional structure which is not visible at the top of search.

- Domain-based algorithms automatically combine features. For example, a pattern / target pair may have matching label features, and matching degree features, but if the only red pattern vertex has degree three whilst no red target vertex has degree

more than two, domain filtering will detect this immediately. When combined with all-different propagation and maintained during search, this effect is even stronger.

- Finally, as indexing systems get more and more complex in an attempt to filter out a few more instances where VF2 performs poorly, the cost of index construction and maintenance is considerable. Indices proposed in the literature often take many hours to build, and can consume much more space than the original graphs (Hong et al., 2015).

In other words, using domain-based algorithms would not simply be a viable alternative to filter / verify with VF2, but rather would be a much better solution.

6.4 Other Implications

Indexing is not the only incorrect design choice being made in systems built around subgraph isomorphism algorithms, and lessons from constraint programming can be applied to other subgraph-related research. We illustrate this by looking at several recent papers on subgraph isomorphism, discussing how a deeper understanding of the empirical hardness of subgraph isomorphism could lead the reader to different conclusions than the ones presented.

Katsarou, Ntarmos, and Triantafillou (2017) present approaches to improve the run times for what they call "straggler queries"—the small subset of queries that they observe taking much longer than others to solve. They observe that permuting graphs (for example, by using degree or label frequency information) before running the subgraph isomorphism algorithm can improve run times by orders of magnitude. However, the connection between this approach and variable- and value-ordering heuristics is not noted, and no consideration is given to dynamic ordering heuristics such as "smallest domain first" (Haralick & Elliott, 1980). Of the orderings proposed, those involving placing rare labels first are effectively approximations to a static "smallest domain at top of search first" ordering, whilst the remainder use degree as we did earlier in this paper. Katsarou et al. do not investigate any algorithm which employs strong variable- or value-ordering heuristics, all-different propagation, or even domains, and do not consider that the apparent successes of their approach could be due to VF2's weaknesses rather than an inherent property of NP-completeness.

We saw in the previous sections that although permuting input graphs could improve VF2's behaviour on some instances, doing so does not make its performance come close to that of domain-based algorithms. Katsarou et al. (2017) say they "hope that our findings will open up new research directions, striving to find appropriate, per-query, isomorphic rewritings, in combination with alternate per-query sub-iso algorithms that can yield large improvements" (p. 36). We believe it is important instead to emphasise previous research directions that have already solved most of this problem, and to help ensure that these techniques become more widely known.

Katsarou et al.'s (2017) second claim is that "different algorithms find different queries hard" (p. 30). To address this, they run many subgraph isomorphism algorithms and input permutations in parallel; the extensive literature on parallel portfolios in general (Gomes & Selman, 2001), and the approach by Battiti and Mascia (2007) for subgraph isomorphism in particular, is not noted. (Nor is the portfolios literature referenced when they say that "using machine learning models to predict which version ... to employ per query is of high

interest", p. 36). The evidence so far in this paper suggests that we should carefully examine the reasons behind these speedups. It is certainly true that there are some instances that all algorithms find hard, and there are good theoretical reasons to believe that these instances genuinely are really hard. Furthermore, algorithm portfolios are also well-known to be a successful technique, and are beneficial even with modern subgraph isomorphism algorithms (Kotthoff et al., 2016). However, in Section 5 we saw that there are many instances that VF2 finds hard that should not be hard, and that are not hard for other algorithms. If permuting graphs sometimes addresses the difficulty of some of these instances, then it is likely that they are not genuinely hard instances at all. Perhaps then, it would be better simply to use an algorithm with domain tracking and domain-based ordering heuristics.

To test this suggestion, we return briefly to the experiments on the datasets discussed earlier in this section. What if we modify our solver so that it does not use "smallest domain first", and does not detect domain wipeouts until attempting to branch on an empty variable? In this case, three of the four datasets become extremely hard, with many instances now timing out after making hundreds of millions of recursive calls, and the PPI dataset now requires hundreds of thousands of calls for some queries. Furthermore, slight changes to the input vertex ordering can now make some of these "hard" instances easy again. This verifies our intuition: the apparent success of Katsarou et al.'s (2017) technique is due to the ease of sometimes getting better results out of a poor algorithm. Obtaining huge improvements from an algorithm portfolio consisting only of variations of poor algorithms is not sufficient to demonstrate a genuine improvement over the state of the art.

Improved search orderings for VF2-style algorithms also appear in recent work by Carletti et al. (2015) and Carletti (2016), cumulating in VF3 (Carletti et al., 2017; Carletti, Foggia, Saggese, & Vento, 2017), and by Shen and Zou (2017). We have seen in Section 5 that such an approach could give improvements, but will not bring an algorithm that does not use domains close to the performance of one that does on harder instances. We thus believe that a significant amount of effort in designing algorithms for and systems involving subgraph isomorphism has been misdirected due to a lack of familiarity with the constraint programming literature.

Another such lesson is the importance of choosing a good set of benchmark instances. As Gent (1998) observes:

> "Benchmark problems should be hard. I report on the solution of the five open benchmark problems introduced ... for testing bin packing problems. Since the solutions were found either by hand or by using very simple heuristic methods, these problems would appear to be easy. In four cases I give improved packings to refute conjectures that previously reported packings were optimal, and I give a proof that the fifth conjecture was correct. ... Future experimenters should be careful to perform tests on problems that can reasonably be regarded as hard." (p. 299)

We hope that this paper provides some much-needed help in this direction, as some recent evaluations work only with extremely easy instances. For example, Bonnici and Giugno (2017) compared various non-constraint-based approaches to LAD using benchmark sets whose difficulties are measured between microseconds and milliseconds. Their results

suggest that LAD is an order of magnitude slower than other approaches. This should not surprise us: the initialisation costs of LAD are non-trivial due to it constructing domains, calculating neighbourhood degree sequences, and performing all-different filtering. However, the risks of occasionally encountering exponential performance from weaker algorithms on easy instances should now be clear.

Carletti et al. (2017) correctly observe that "a performance improvement on [hard instances] has a far greater practical impact than it would have on easier graphs, since the saved time can be of several hours or even days" (p. 805). To evaluate VF3, they introduce a new dataset consisting entirely of satisfiable instances created by taking a connected subgraph of a randomly generated graph and permuting the nodes. They conclude that VF3 is the best algorithm based upon its performance solving the enumeration problem on these new instances. We urge a more cautious interpretation of these results: as we saw in Section 3, being genuinely hard (rather than simply larger or denser) is a subtle property, and care must be taken when empirically evaluating algorithms. In particular, if one only considers large, easy instances, then the simpler the algorithm, the better it scales. Indeed, we saw in Figure 5 that VF3 performs *worse* than VF2 on genuinely harder instances, and additionally behaves erratically in parts of the parameter space that every other solver finds easy; finally, an induced analogue of Figure 8 confirms that VF3 also still struggles with many obviously unsatisfiable instances.

Of course, hard random instances of the kind we describe in Sections 2 and 3 are also atypical, and should not form the sole basis for benchmarking. In particular, distance-based filtering (Audemard et al., 2014; McCreesh & Prosser, 2015; Kotthoff et al., 2016) has little to no effect on these instances, but is extremely beneficial on other kinds of graph, including those from real-world applications. These graphs also have a low degree spread and very little structure to exploit, which could mislead experimenters as to the value of certain kinds of inference. Gent and Walsh (1995) give an example of an exam timetabling graph which contains an unexpected ten-vertex clique, which would be extremely unlikely in a randomly generated graph with a similar order and density. We therefore emphasise the importance of working with a diverse set of benchmark instances, and of performing a careful family-by-family analysis of the results.

7. Conclusion

We have shown how to generate small but hard instances for the non-induced and induced subgraph isomorphism problems, which will help offset the bias in existing datasets. For non-induced isomorphisms, behaviour was as in many other hard problems, but for induced isomorphisms we uncovered several interesting phenomena: there are hard instances far from a phase transition, constrainedness predicts this, and existing general techniques for designing heuristics do not work in certain portions of the parameter space.

We note that these "really hard" instances have a very particular structure, which is unlikely to arise naturally. Indeed, the successes seen with constraint programming approaches on the instances in Section 6, and in the wider range of application instances used by Kotthoff et al. (2016), suggest that the real-world instances encountered so far are generally fairly easy for a good algorithm. Nonetheless, carefully designed synthetic

instances provide a good starting point for scientific experiments, and can help improve the behaviour of algorithms on real-world instances too.

For example, when labels were introduced, we saw that VF2 found some instances hard which other algorithms found easy. We looked at this kind of instance in more detail, and argued that this was due to shortcomings in VF2's design, rather than an interesting property of NP-completeness. Although inevitably there will be instances that every algorithm finds hard, we see it as a serious weakness for an algorithm to exhibit exponential behaviour in such trivial cases as when there are two red vertices in the pattern but only one in the target.

Unfortunately, we saw that this aspect of VF2's performance has had practical consequences, with the most notable being the misdesign of graph database systems. By not trying even the simplest constraint programming techniques from the literature, and disregarding all that is known about the empirical hardness of NP-complete problems, members of the graph databases community have reached the mistaken conclusion that extensive research into supporting techniques is important. With this new understanding of what does and does not make subgraph isomorphism hard, it is time for a radical rethink of how graph database systems work.

We do not claim that the ultimate "big data" subgraph matching algorithm already exists: on the contrary, there is likely to be plenty of room for future improvement now that we understand the importance of getting algorithm design right. For example, a major disadvantage of using domains is the relatively expensive initialisation costs, which quickly add up when dealing with large numbers of trivial instances. Employing a presolver is an obvious approach—and VF2 is in fact good in this role (Kotthoff et al., 2016)—but there are other possibilities. For example, minimal or lazy forward-checking (Dent & Mercer, 1994; Bacchus & Grove, 1995; Dent, 1996; Larrosa & Meseguer, 1998) avoids constructing every domain upfront, although adopting this may require alternatives to "smallest domain first" and to all-different. Such an approach may also be beneficial for huge target graphs, where having domains range over the entire target is impractical. Indeed, we saw that a raw constraint programming approach using Gecode suffered from relatively long runtimes on the very large but sparse PDBS instances, despite finding the instances very easy in terms of number of recursive calls. An algorithm which is capable of performing most of the reasoning of the Glasgow and LAD solvers, but without needing to store domains and adjacency matrices, could represent a major breakthrough.

There is also scope for precalculating supporting information about target graphs. For example, neighbourhood degree sequences and supplemental graphs could both be precomputed and stored. The aim here is to reduce the initialisation costs of a good subgraph isomorphism algorithm, and not to provide indexing (although additionally using this information as an index may not hurt, if initialisation is still costly).

We must stress that we are not simply concluding that graph database systems should use a better subgraph isomorphism algorithm. Instead, this paper has shown that such systems need to be designed in light of a modern understanding of the empirical hardness of NP-complete problems, that subgraph isomorphism algorithms should not be treated as a black box, and that lessons learned in constraint programming and artificial intelligence should not go unheeded in other domains.

Acknowledgments

The authors would like to thank Kitty Meeks and Craig Reilly for their comments. We are grateful to Iva Babukova for her help with the datasets and for sharing the results of her experiments. The heatmap schemes used in this paper were created with the assistance of the "Chroma.js Color Scale Helper" by Gregor Aisch. Parts of this paper were presented at IJCAI (McCreesh, Prosser, & Trimble, 2016); the first three sections have been expanded, whilst Sections 4 to 6 are new material. Most of this work also appeared in a PhD thesis (McCreesh, 2017).

This work was supported by the Engineering and Physical Sciences Research Council [grant numbers EP/K503058/1, EP/M508056/1, and EP/P026842/1], and by the ANR project SoLStiCe (ANR-13-BS02-0002-01).

References

Achlioptas, D., Molloy, M. S. O., Kirousis, L. M., Stamatiou, Y. C., Kranakis, E., & Krizanc, D. (2001). Random constraint satisfaction: A more accurate picture. *Constraints*, *6*(4), 329–344.

Anton, C., & Olson, L. (2009). Generating satisfiable sat instances using random subgraph isomorphism. In Gao, Y., & Japkowicz, N. (Eds.), *Advances in Artificial Intelligence*, Vol. 5549 of *Lecture Notes in Computer Science*, pp. 16–26. Springer Berlin Heidelberg.

Audemard, G., Lecoutre, C., Modeliar, M. S., Goncalves, G., & Porumbel, D. C. (2014). Scoring-based neighborhood dominance for the subgraph isomorphism problem. In O'Sullivan, B. (Ed.), *Principles and Practice of Constraint Programming - 20th International Conference, CP 2014, Lyon, France, September 8-12, 2014. Proceedings*, Vol. 8656 of *Lecture Notes in Computer Science*, pp. 125–141. Springer.

Audemard, G., & Simon, L. (2014). The Glucose SAT solver..

Azaouzi, M., & Romdhane, L. B. (2016). A minimal rare substructures-based model for graph database indexing. In Madureira, A. M., Abraham, A., Gamboa, D., & Novais, P. (Eds.), *Intelligent Systems Design and Applications - 16th International Conference on Intelligent Systems Design and Applications (ISDA 2016) held in Porto, Portugal, December 16-18, 2016*, Vol. 557 of *Advances in Intelligent Systems and Computing*, pp. 250–259. Springer.

Bacchus, F., & Grove, A. J. (1995). On the forward checking algorithm. In Montanari, U., & Rossi, F. (Eds.), *Principles and Practice of Constraint Programming - CP'95, First International Conference, CP'95, Cassis, France, September 19-22, 1995, Proceedings*, Vol. 976 of *Lecture Notes in Computer Science*, pp. 292–308. Springer.

Battiti, R., & Mascia, F. (2007). An algorithm portfolio for the sub-graph isomorphism problem. In Stützle, T., Birattari, M., & Hoos, H. H. (Eds.), *Engineering Stochastic Local Search Algorithms. Designing, Implementing and Analyzing Effective Heuristics, International Workshop, SLS 2007, Brussels, Belgium, September 6-8, 2007, Proceedings*, Vol. 4638 of *Lecture Notes in Computer Science*, pp. 106–120. Springer.

Bonnici, V., & Giugno, R. (2017). On the variable ordering in subgraph isomorphism algorithms. *IEEE/ACM Trans. Comput. Biology Bioinform.*, *14*(1), 193–203.

Cao, N., Yang, Z., Wang, C., Ren, K., & Lou, W. (2011). Privacy-preserving query over encrypted graph-structured data in cloud computing. In *2011 International Conference on Distributed Computing Systems, ICDCS 2011, Minneapolis, Minnesota, USA, June 20-24, 2011*, pp. 393–402. IEEE Computer Society.

Carletti, V., Foggia, P., Saggese, A., & Vento, M. (2017). Challenging the time complexity of exact subgraph isomorphism for huge and dense graphs with VF3. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *PP*(99), 1–1.

Carletti, V. (2016). *Exact and Inexact Methods for Graph Similarity in Structural Pattern Recognition*. Ph.D. thesis, Université de Caen; Universita degli studi di Salerno.

Carletti, V., Foggia, P., Saggese, A., & Vento, M. (2017). Introducing VF3: A new algorithm for subgraph isomorphism. In Foggia, P., Liu, C., & Vento, M. (Eds.), *Graph-Based Representations in Pattern Recognition - 11th IAPR-TC-15 International Workshop, GbRPR 2017, Anacapri, Italy, May 16-18, 2017, Proceedings*, Vol. 10310 of *Lecture Notes in Computer Science*, pp. 128–139.

Carletti, V., Foggia, P., & Vento, M. (2015). VF2 plus: An improved version of VF2 for biological graphs. In Liu, C., Luo, B., Kropatsch, W. G., & Cheng, J. (Eds.), *Graph-Based Representations in Pattern Recognition - 10th IAPR-TC-15 International Workshop, GbRPR 2015, Beijing, China, May 13-15, 2015. Proceedings*, Vol. 9069 of *Lecture Notes in Computer Science*, pp. 168–177. Springer.

Cheeseman, P., Kanefsky, B., & Taylor, W. M. (1991). Where the really hard problems are. In Mylopoulos, J., & Reiter, R. (Eds.), *Proceedings of the 12th International Joint Conference on Artificial Intelligence. Sydney, Australia, August 24-30, 1991*, pp. 331–340. Morgan Kaufmann.

Cheng, J., Ke, Y., Ng, W., & Lu, A. (2007). Fg-index: towards verification-free query processing on graph databases. In Chan, C. Y., Ooi, B. C., & Zhou, A. (Eds.), *Proceedings of the ACM SIGMOD International Conference on Management of Data, Beijing, China, June 12-14, 2007*, pp. 857–872. ACM.

Chirkova, R., Dogac, A., Özsu, M. T., & Sellis, T. K. (Eds.). (2007). *Proceedings of the 23rd International Conference on Data Engineering, ICDE 2007, The Marmara Hotel, Istanbul, Turkey, April 15-20, 2007*. IEEE Computer Society.

Conte, D., Foggia, P., Sansone, C., & Vento, M. (2004). Thirty years of graph matching in pattern recognition. *IJPRAI*, *18*(3), 265–298.

Cordella, L. P., Foggia, P., Sansone, C., & Vento, M. (2004). A (sub)graph isomorphism algorithm for matching large graphs. *IEEE Trans. Pattern Anal. Mach. Intell.*, *26*(10), 1367–1372.

Damiand, G., Solnon, C., de la Higuera, C., Janodet, J., & Samuel, É. (2011). Polynomial algorithms for subisomorphism of nd open combinatorial maps. *Computer Vision and Image Understanding*, *115*(7), 996–1010.

Davis, P. J. (1959). Leonhard Euler's integral: A historical profile of the Gamma function. *The American Mathematical Monthly*, *66*(10), 849–869.

Daylight Chemical Information Systems, Inc. (2011). Daylight theory manual, version 4.9. http://www.daylight.com/dayhtml/doc/theory/theory.thor.html.

Dent, M. J. (1996). *Minimal Forward Checking–a Lazy Constraint Satisfaction Search Algorithm: Experimental And Theoretical Results*. Ph.D. thesis, The University of Western Ontario.

Dent, M. J., & Mercer, R. E. (1994). Minimal forward checking. In *Sixth International Conference on Tools with Artificial Intelligence, ICTAI '94, New Orleans, Louisiana, USA, November 6-9, 1994*, pp. 432–438. IEEE Computer Society.

Erdős, P., & Rényi, A. (1959). On random graphs I.. *Publicationes Mathematicae (Debrecen)*, *6*, 290–297.

Garey, M. R., & Johnson, D. S. (1979). *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman.

Gebser, M., Kaufmann, B., Kaminski, R., Ostrowski, M., Schaub, T., & Schneider, M. T. (2011). Potassco: The potsdam answer set solving collection. *AI Commun.*, *24*(2), 107–124.

Gent, I. P. (1998). Heuristic solution of open bin packing problems. *J. Heuristics*, *3*(4), 299–304.

Gent, I. P., MacIntyre, E., Prosser, P., Smith, B. M., & Walsh, T. (1996). An empirical study of dynamic variable ordering heuristics for the constraint satisfaction problem. In Freuder, E. C. (Ed.), *Proceedings of the Second International Conference on Principles and Practice of Constraint Programming, Cambridge, Massachusetts, USA, August 19-22, 1996*, Vol. 1118 of *Lecture Notes in Computer Science*, pp. 179–193. Springer.

Gent, I. P., MacIntyre, E., Prosser, P., Smith, B. M., & Walsh, T. (2001). Random constraint satisfaction: Flaws and structure. *Constraints*, *6*(4), 345–372.

Gent, I. P., MacIntyre, E., Prosser, P., & Walsh, T. (1996). The constrainedness of search. In Clancey, W. J., & Weld, D. S. (Eds.), *Proceedings of the Thirteenth National Conference on Artificial Intelligence and Eighth Innovative Applications of Artificial Intelligence Conference, AAAI 96, IAAI 96, Portland, Oregon, August 4-8, 1996, Volume 1.*, pp. 246–252. AAAI Press / The MIT Press.

Gent, I. P., & Walsh, T. (1995). Phase transitions from real computational problems. In *Proceedings of the 8th International Symposium on Artificial Intelligence*, pp. 356–364.

Gilbert, E. N. (1959). Random graphs. *Ann. Math. Statist.*, *30*(4), 1141–1144.

Giugno, R., Bonnici, V., Bombieri, N., Pulvirenti, A., Ferro, A., & Shasha, D. (2013). Grapes: A software for parallel searching on biological graphs targeting multi-core architectures. *PLOS ONE*, *8*(10), 1–11.

Gomes, C. P., & Selman, B. (2001). Algorithm portfolios. *Artif. Intell.*, *126*(1-2), 43–62.

Grömping, U. (2014). R package FrF2 for creating and analyzing fractional factorial 2-level designs. *Journal of Statistical Software*, *56*(1), 1–56.

Hagberg, A. A., Schult, D. A., & Swart, P. J. (2008). Exploring network structure, dynamics, and function using NetworkX. In *Proceedings of the 7th Python in Science Conference (SciPy2008)*, pp. 11–15, Pasadena, CA USA.

Han, J., Cheng, H., Xin, D., & Yan, X. (2007). Frequent pattern mining: current status and future directions. *Data Min. Knowl. Discov.*, *15*(1), 55–86.

Haralick, R. M., & Elliott, G. L. (1980). Increasing tree search efficiency for constraint satisfaction problems. *Artif. Intell.*, *14*(3), 263–313.

He, Y., Lin, F., Chipman, P. R., Bator, C. M., Baker, T. S., Shoham, M., Kuhn, R. J., Medof, M. E., & Rossmann, M. G. (2002). Structure of decay-accelerating factor bound to echovirus 7: A virus-receptor complex. *Proc Natl Acad Sci U S A*, *99*(16), 10325–10329.

Hong, L., Zou, L., Lian, X., & Yu, P. S. (2015). Subgraph matching with set similarity in a large graph database. *IEEE Trans. Knowl. Data Eng.*, *27*(9), 2507–2521.

Jiang, H., Wang, H., Yu, P. S., & Zhou, S. (2007). Gstring: A novel approach for efficient search in graph databases.. In Chirkova et al. (Chirkova, Dogac, Özsu, & Sellis, 2007), pp. 566–575.

Katsarou, F., Ntarmos, N., & Triantafillou, P. (2015). Performance and scalability of indexed subgraph query processing methods. *PVLDB*, *8*(12), 1566–1577.

Katsarou, F., Ntarmos, N., & Triantafillou, P. (2017). Subgraph querying with parallel use of query rewritings and alternative algorithms. In Markl, V., Orlando, S., Mitschang, B., Andritsos, P., Sattler, K., & Breß, S. (Eds.), *Proceedings of the 20th International Conference on Extending Database Technology, EDBT 2017, Venice, Italy, March 21-24, 2017.*, pp. 25–36. OpenProceedings.org.

Kotthoff, L., McCreesh, C., & Solnon, C. (2016). Portfolios of subgraph isomorphism algorithms. In Festa, P., Sellmann, M., & Vanschoren, J. (Eds.), *Learning and Intelligent Optimization - 10th International Conference, LION 10, Ischia, Italy, May 29 - June 1, 2016, Revised Selected Papers*, Vol. 10079 of *Lecture Notes in Computer Science*, pp. 107–122. Springer.

Larrosa, J., & Meseguer, P. (1998). Partial lazy forward checking for MAX-CSP. In *ECAI*, pp. 229–233.

Lee, J., Han, W., Kasperovics, R., & Lee, J. (2012). An in-depth comparison of subgraph isomorphism algorithms in graph databases. *PVLDB*, *6*(2), 133–144.

Levi, G. (1973). A note on the derivation of maximal common subgraphs of two directed or undirected graphs. *CALCOLO*, *9*(4), 341–352.

Leyton-Brown, K., Hoos, H. H., Hutter, F., & Xu, L. (2014). Understanding the empirical hardness of *NP*-complete problems. *Commun. ACM*, *57*(5), 98–107.

McCreesh, C. (2017). *Solving Hard Subgraph Problems in Parallel*. Ph.D. thesis, University of Glasgow.

McCreesh, C., Ndiaye, S. N., Prosser, P., & Solnon, C. (2016). Clique and constraint models for maximum common (connected) subgraph problems. In Rueher, M. (Ed.), *Principles and Practice of Constraint Programming - 22nd International Conference, CP 2016, Toulouse, France, September 5-9, 2016, Proceedings*, Vol. 9892 of *Lecture Notes in Computer Science*, pp. 350–368. Springer.

McCreesh, C., & Prosser, P. (2015). A parallel, backjumping subgraph isomorphism algorithm using supplemental graphs. In Pesant, G. (Ed.), *Principles and Practice of Constraint Programming - 21st International Conference, CP 2015, Cork, Ireland, August 31 - September 4, 2015, Proceedings*, Vol. 9255 of *Lecture Notes in Computer Science*, pp. 295–312. Springer.

McCreesh, C., Prosser, P., & Trimble, J. (2016). Heuristics and really hard instances for subgraph isomorphism problems. In Kambhampati, S. (Ed.), *Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence, IJCAI 2016, New York, NY, USA, 9-15 July 2016*, pp. 631–638. IJCAI/AAAI Press.

Mitchell, D. G., Selman, B., & Levesque, H. J. (1992). Hard and easy distributions of SAT problems. In Swartout, W. R. (Ed.), *Proceedings of the 10th National Conference on Artificial Intelligence. San Jose, CA, July 12-16, 1992.*, pp. 459–465. AAAI Press / The MIT Press.

Murray, A. C. (2012). *Customising compilers for customisable processors*. Ph.D. thesis, The University of Edinburgh.

Peng, P., Zou, L., Chen, L., Lin, X., & Zhao, D. (2016). Answering subgraph queries over massive disk resident graphs. *World Wide Web*, *19*(3), 417–448.

Régin, J. (1994). A filtering algorithm for constraints of difference in csps. In Hayes-Roth, B., & Korf, R. E. (Eds.), *Proceedings of the 12th National Conference on Artificial Intelligence, Seattle, WA, USA, July 31 - August 4, 1994, Volume 1.*, pp. 362–367. AAAI Press / The MIT Press.

San Segundo, P., Rodríguez-Losada, D., & Jiménez, A. (2011). An exact bit-parallel algorithm for the maximum clique problem. *Computers & OR*, *38*(2), 571–581.

Santo, M. D., Foggia, P., Sansone, C., & Vento, M. (2003). A large database of graphs and its use for benchmarking graph isomorphism algorithms. *Pattern Recognition Letters*, *24*(8), 1067–1079.

Shang, H., Zhang, Y., Lin, X., & Yu, J. X. (2008). Taming verification hardness: an efficient algorithm for testing subgraph isomorphism. *PVLDB*, *1*(1), 364–375.

Shasha, D. E., Wang, J. T., & Giugno, R. (2002). Algorithmics and applications of tree and graph searching. In Popa, L., Abiteboul, S., & Kolaitis, P. G. (Eds.), *Proceedings of the Twenty-first ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, June 3-5, Madison, Wisconsin, USA*, pp. 39–52. ACM.

Shen, Y., & Zou, Z. (2017). Efficient subgraph matching on non-volatile memory. In Bouguettaya, A., Gao, Y., Klimenko, A., Chen, L., Zhang, X., Dzerzhinskiy, F., Jia, W., Klimenko, S. V., & Li, Q. (Eds.), *Web Information Systems Engineering - WISE 2017 - 18th International Conference, Puschino, Russia, October 7-11, 2017, Proceedings, Part I*, Vol. 10569 of *Lecture Notes in Computer Science*, pp. 457–471. Springer.

Smith, B. M., & Dyer, M. E. (1996). Locating the phase transition in binary constraint satisfaction problems. *Artif. Intell.*, *81*(1-2), 155–181.

Smith, B. M., & Grant, S. A. (1997). Modelling exceptionally hard constraint satisfaction problems. In Smolka, G. (Ed.), *Principles and Practice of Constraint Programming -*

CP97, Third International Conference, Linz, Austria, October 29 - November 1, 1997, Proceedings, Vol. 1330 of *Lecture Notes in Computer Science*, pp. 182–195. Springer.

Solnon, C. (2010). Alldifferent-based filtering for subgraph isomorphism. *Artif. Intell.*, *174*(12-13), 850–864.

Solnon, C., Damiand, G., de la Higuera, C., & Janodet, J. (2015). On the complexity of submap isomorphism and maximum common submap problems. *Pattern Recognition*, *48*(2), 302–316.

Steger, A., & Wormald, N. C. (1999). Generating random regular graphs quickly. *Combinatorics, Probability & Computing*, *8*(4), 377–396.

Tian, Y., & Patel, J. M. (2008). TALE: A tool for approximate large graph matching. In Alonso, G., Blakeley, J. A., & Chen, A. L. P. (Eds.), *Proceedings of the 24th International Conference on Data Engineering, ICDE 2008, April 7-12, 2008, Cancún, México*, pp. 963–972. IEEE Computer Society.

Ullmann, J. R. (1976). An algorithm for subgraph isomorphism. *J. ACM*, *23*(1), 31–42.

Vehlow, C., Stehr, H., Winkelmann, M., Duarte, J. M., Petzold, L., Dinse, J., & Lappe, M. (2011). Cmview: Interactive contact map visualization and analysis. *Bioinformatics*, *27*(11), 1573.

Walsh, T. (1998). The constrainedness knife-edge. In Mostow, J., & Rich, C. (Eds.), *Proceedings of the Fifteenth National Conference on Artificial Intelligence and Tenth Innovative Applications of Artificial Intelligence Conference, AAAI 98, IAAI 98, July 26-30, 1998, Madison, Wisconsin, USA.*, pp. 406–411. AAAI Press / The MIT Press.

Wang, G., Wang, B., Yang, X., & Yu, G. (2012). Efficiently indexing large sparse graphs for similarity search. *IEEE Trans. Knowl. Data Eng.*, *24*(3), 440–451.

Wang, J., Ntarmos, N., & Triantafillou, P. (2016). Indexing query graphs to speedup graph query processing. In Pitoura, E., Maabout, S., Koutrika, G., Marian, A., Tanca, L., Manolescu, I., & Stefanidis, K. (Eds.), *Proceedings of the 19th International Conference on Extending Database Technology, EDBT 2016, Bordeaux, France, March 15-16, 2016, Bordeaux, France, March 15-16, 2016.*, pp. 41–52. OpenProceedings.org.

Yan, X., Yu, P. S., & Han, J. (2004). Graph indexing: A frequent structure-based approach. In Weikum, G., König, A. C., & Deßloch, S. (Eds.), *Proceedings of the ACM SIGMOD International Conference on Management of Data, Paris, France, June 13-18, 2004*, pp. 335–346. ACM.

Yan, X., Yu, P. S., & Han, J. (2005). Graph indexing based on discriminative frequent structure analysis. *ACM Trans. Database Syst.*, *30*(4), 960–993.

Yuan, D., & Mitra, P. (2013). Lindex: a lattice-based index for graph databases. *VLDB J.*, *22*(2), 229–252.

Yuan, D., Mitra, P., & Giles, C. L. (2013). Mining and indexing graphs for supergraph search. *PVLDB*, *6*(10), 829–840.

Yuan, D., Mitra, P., Yu, H., & Giles, C. L. (2015). Updating graph indices with a one-pass algorithm. In Sellis, T. K., Davidson, S. B., & Ives, Z. G. (Eds.), *Proceedings of the*

2015 ACM SIGMOD International Conference on Management of Data, Melbourne, Victoria, Australia, May 31 - June 4, 2015, pp. 1903–1916. ACM.

Zampelli, S., Deville, Y., & Solnon, C. (2010). Solving subgraph isomorphism problems with constraint programming. *Constraints*, *15*(3), 327–353.

Zhang, S., Hu, M., & Yang, J. (2007). Treepi: A novel graph indexing method.. In Chirkova et al. (Chirkova et al., 2007), pp. 966–975.

Zhang, S., Li, S., & Yang, J. (2009). GADDI: distance index based subgraph matching in biological networks. In Kersten, M. L., Novikov, B., Teubner, J., Polutin, V., & Manegold, S. (Eds.), *EDBT 2009, 12th International Conference on Extending Database Technology, Saint Petersburg, Russia, March 24-26, 2009, Proceedings*, Vol. 360 of *ACM International Conference Proceeding Series*, pp. 192–203. ACM.

Zhao, P., & Han, J. (2010). On graph query optimization in large networks. *PVLDB*, *3*(1), 340–351.

Zhao, P., Yu, J. X., & Yu, P. S. (2007). Graph indexing: Tree + delta >= graph. In Koch, C., Gehrke, J., Garofalakis, M. N., Srivastava, D., Aberer, K., Deshpande, A., Florescu, D., Chan, C. Y., Ganti, V., Kanne, C., Klas, W., & Neuhold, E. J. (Eds.), *Proceedings of the 33rd International Conference on Very Large Data Bases, University of Vienna, Austria, September 23-27, 2007*, pp. 938–949. ACM.

Zheng, W., Lian, X., Zou, L., Hong, L., & Zhao, D. (2016). Online subgraph skyline analysis over knowledge graphs. *IEEE Trans. Knowl. Data Eng.*, *28*(7), 1805–1819.