Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence (AAAI-17)

Between Subgraph Isomorphism and Maximum Common Subgraph

Ruth Hoffmann, Ciaran McCreesh,* Craig Reilly*
University of Glasgow, Glasgow, Scotland
ruth.hoffmann@glasgow.ac.uk and {c.mccreesh.1,c.reilly.2}@research.gla.ac.uk

Abstract

When a small pattern graph does not occur inside a larger target graph, we can ask how to find "as much of the pattern as possible" inside the target graph. In general, this is known as the maximum common subgraph problem, which is much more computationally challenging in practice than subgraph isomorphism. We introduce a restricted alternative, where we ask if all but k vertices from the pattern can be found in the target graph. This allows for the development of slightly weakened forms of certain invariants from subgraph isomorphism which are based upon degree and number of paths. We show that when k is small, weakening the invariants still retains much of their effectiveness. We are then able to solve this problem on the standard problem instances used to benchmark subgraph isomorphism algorithms, despite these instances being too large for current maximum common subgraph algorithms to handle. Finally, by iteratively increasing k, we obtain an algorithm which is also competitive for the maximum common subgraph problem.

1 Introduction

The subgraph isomorphism problem is to find a copy of a small *pattern* graph inside a larger *target* graph. The problem arises in many areas, including bioinformatics (Bonnici et al. 2013), computer vision (Damiand et al. 2011; Solnon et al. 2015), malware detection (Bruschi, Martignoni, and Monga 2006), compilers (Murray and Franke 2012; Blindell et al. 2015), model checking (Sevegnani and Calder 2015), and pattern recognition (Conte et al. 2004), which has lead to substantial research into how best to solve the problem in practice (McGregor 1979; Ullmann 1976; Cordella et al. 2004; Solnon 2010; Audemard et al. 2014; McCreesh and Prosser 2015, and more). The problem comes in two forms: in the non-induced variant, edges must be mapped to edges, but the target may have "extra edges", whilst in the induced variant, non-edges must be mapped to non-edges.

When a pattern cannot be found, we may wish to be given a result which maps as many vertices of the pattern into the target as possible. In the induced case, this is known as the maximum common induced subgraph problem (we discuss the non-induced case in Section 1.1). However, although recent subgraph isomorphism algorithms are comfortable working with graphs with thousands of vertices, the state of the art for the maximum common subgraph problem (McCreesh et al. 2016) becomes computationally infeasible at only 35 vertices when working with unlabelled graphs. This is largely because strong inference, based upon the degrees of vertices (Solnon 2010) and the distances or paths between them (Audemard et al. 2014; McCreesh and Prosser 2015), is possible with subgraph isomorphism, but not maximum common subgraph, and so the state space in the former is much more restricted, whilst filtering during search is vastly stronger.

In this work we discuss an intermediate problem, where we must map all but k vertices of the pattern graph into the target—we show an example in Figure 1. This in some ways resembles the approximate subgraph matching model of Zampelli, Deville, and Dupont (2005), although we allow any vertex to be removed. We show that if k is reasonably small (say, between 1 and 5), then weakened forms of degree and path based filterings are still effective in pruning the initial search space and in providing additional constraints respectively. We then show that combining these techniques leads to a practical algorithm which can scale to work with the families of graphs commonly used to benchmark subgraph isomorphism algorithms: depending upon the benchmark family, we can close a substantial portion of the instances, and in many more cases, we can at least obtain an upper bound. This is a significant improvement over state of the art maximum common subgraph solvers, which cannot even fit many of these instances in 64 GBytes of RAM. Finally, we show that starting with $k = 0$ and iteratively in-

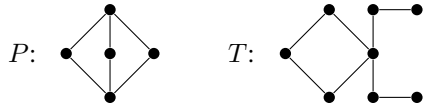

Figure 1: The pattern graph P on the left cannot be found in the target graph T, but if the central vertex in P is removed, then a subgraph isomorphism exists.

*This work was supported by the Engineering and Physical Sciences Research Council [grant numbers EP/K503058/1 and EP/M506539/1]

creasing k gives a competitive algorithm for the maximum common induced subgraph problem.

1.1 Definitions and Notations

Throughout, our graphs are undirected, but may have loops. We write $\mathrm{V}(G)$ for the vertex set of a graph G, and $\mathrm{E}(G)$ for its set of edges. We write $\mathrm{N}_G(v)$ for the *neighbourhood* of a vertex v in G, and $v \sim_G w$ to mean v is adjacent to w. The *degree* of p, $\deg_G(p)$, is the number of vertices to which it is adjacent. Where the graph is clear from the context, the subscripts are omitted.

A *non-induced subgraph isomorphism* from a graph P (called the *pattern*) to a graph T (the *target*) is an injective mapping $\mathrm{V}(P) \rightarrowtail \mathrm{V}(T)$ which maps adjacent vertices to adjacent vertices. An *induced subgraph isomorphism* $P \hookrightarrow T$ additionally maps non-adjacent vertices to non-adjacent vertices. We define a *k-less subgraph isomorphism* from P to T to be a subgraph isomorphism from all but k vertices of P to T; this may be non-induced or induced, written $P \,{}_k\!\!\rightarrowtail T$ and $P \,{}_k\!\!\hookrightarrow T$ respectively. We write $p \,{}_k\!\!\mapsto t$ to mean that the pattern vertex p is mapped to the target vertex v under either kind of mapping.

The *complement* of a (simple) graph P is the graph $\overline{P}$. Adjacent distinct vertices in P are non-adjacent in $\overline{P}$ while non-adjacent vertices in P are adjacent in $\overline{P}$. The *loop complement* of a graph P is the graph $\overline{P}^\circ$. The construction of $\overline{P}^\circ$ is similar to that of $\overline{P}$, however whenever a vertex has a loop in P it does not in $\overline{P}^\circ$ and vice versa. When solving an induced subgraph isomorphism problem we make use of the loop complement; we care about mapping vertices with loops to vertices with loops, and vertices without loops to vertices without loops. The following proposition follows directly from the definitions.

Proposition 1. Let i be an assignment of vertices of T to vertices of P. Then i satisfies the definition of $P \hookrightarrow T$ if and only if i satisfies $P \rightarrowtail T$ and $\overline{P}^\circ \rightarrowtail \overline{T}^\circ$ simultaneously. Similarly, i satisfies the definition of $P \,{}_k\!\!\hookrightarrow T$ if and only if i satisfies both $P \,{}_k\!\!\rightarrowtail T$ and $\overline{P}^\circ \,{}_k\!\!\rightarrowtail \overline{T}^\circ$.

A *common induced subgraph* of graphs G and H is a graph P, together with two induced subgraph isomorphisms to G and H; a *maximum common induced subgraph* is a common induced subgraph with as many vertices as possible. An induced k-less subgraph isomorphism $P \,{}_k\!\!\hookrightarrow T$ is equivalent to a common induced subgraph of P and T with $|\mathrm{V}(P)| - k$ vertices.

If defined similarly, a maximum common non-induced subgraph would allow us to select every vertex in the smaller of the two graphs, and none of the edges. It is therefore traditional to change the objective to maximise the number of edges selected, rather than vertices, when a non-induced common subgraph is sought—this problem is usually called the maximum common *partial* subgraph problem instead (Ndiaye and Solnon 2011). However, this is not what we will be discussing in this paper: maximum common subgraph problems are symmetric in their inputs, but when discussing the non-induced case we are allowing extra edges only in the target graph, not in the pattern.

1.2 Constraint Models and Algorithms

For both subgraph isomorphism and maximum common subgraph, constraint programming is the best known approach[1], although a reduction to the maximum clique problem is better when edge labels are present (Ndiaye and Solnon 2011; McCreesh et al. 2016). For both problems, we create a variable for each vertex in the pattern graph (the smaller graph, in the case of maximum common subgraph), with domains ranging over the vertices in the target graph (the larger graph). For maximum common subgraph, each domain is given an additional $\perp$ value, meaning "unmapped".

Constraints are used to ensure that adjacent pairs of vertices are mapped to adjacenct pairs of vertices. For the maximum common subgraph case, any pair of assignments with $\perp$ as either value is also permitted. To handle the induced case, rather than the usual approach of directly having constraints for non-adjacent pairs of vertices, we will be making use of Proposition 1 by seeking a mapping i which simultaneously satisfies $P \rightarrowtail T$ and $\overline{P}^\circ \rightarrowtail \overline{T}^\circ$. Finally, an "all different except $\perp$ constraint" ensures injectivity.

For subgraph isomorphism, this model can be enhanced with domain filtering at the top of search, to reduce the initial sizes of domains—we discuss this in Section 2. Subgraph isomorphism also allows us to generate additional implied constraints, which we discuss in Section 3. However, neither of these techniques are valid for the maximum common subgraph problem.

There are three easy ways we might try to extend existing algorithms to handle the k-less problem. A non-induced k-less subgraph isomorphism from P to T is equivalent to a subgraph isomorphism between P and T, with k extra universally-adjacent vertices added to T, and so we could try solving subgraph isomorphism with a modified target graph. However, using such an approach is not ideal, because it would introduce symmetries; for induced k-less subgraph isomorphisms, an approach based around adding vertices to T cannot work at all. Another algorithmic approach could be to try each way of removing k vertices from the pattern graph, and solving each subgraph isomorphism problem in turn. This approach might be feasible for $k = 1$, although it would involve a lot of duplication of search effort, but for larger values of k the number of searches which would have to be made would grow very quickly. Finally, for the induced case we could try adapting maximum common subgraph algorithms to solve the decision problem. However, both the maximum common subgraph algorithm and the clique encoding require $O(|\mathrm{V}(P)|^2\,|\mathrm{V}(T)|^2)$ memory (McCreesh et al. 2016), which is extremely problematic on the instances with which we wish to work.

Instead, this paper introduces a new algorithm, inspired by a state of the art subgraph isomorphism algorithm (McCreesh and Prosser 2015). The approach we discuss requires only $O(|\mathrm{V}(P)|^2\,|\mathrm{V}(T)|)$ space (this can be thousands

[1]Note that toolkits are not used in state of the art implementations, to facilitate better memory layouts and propagation queues, although the algorithms are inspired by constraint programming.

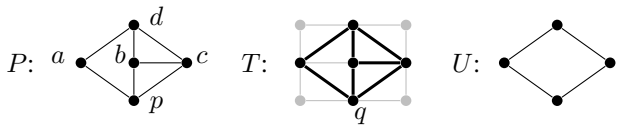

Figure 2: There is a non-induced isomorphism $P \rightarrowtail T$ from the first graph to the second, using the highlighted vertices. No such isomorphism exists between P and U, but there is a 1-less isomorphism $P \,{}_1\!\rightarrowtail U$ which omits vertex b.

of times less than maximum common subgraph approaches, once the constant factors and orders of T in our instances are considered).

1.3 Experimental Setup

We perform our experiments on systems with dual Intel Xeon CPU E5-2640 processors with 64GBytes of RAM, running Ubuntu 14.04. Our algorithm was implemented in C++[2], and source code was compiled using GCC 5.3.0. We used a timeout of 1,000 seconds for each instance. For comparison, we used the implementations of Ndiaye and Solnon (2011) and McCreesh et al. (2016).

For datasets, we use the same 5,725 instances used in a recent work on portfolios of algorithms for subgraph isomorphism (Kotthoff, McCreesh, and Solnon 2016). This collection includes randomly generated scale-free graphs (Zampelli, Deville, and Solnon 2010), an assortment of real-world graphs of varying sizes (Larrosa and Valiente 2002), randomly generated graph pairs (using bounded degree, regular mesh, and uniform models; all are satisfiable), segmented images (Solnon et al. 2015; Damiand et al. 2011), meshes from modelling 3D objects (Damiand et al. 2011), and graphs close to the satisfiable-unsatisfiable phase transition (McCreesh, Prosser, and Trimble 2016). All of these instances are available in a simple text format[3]. Note that many of these instances are *much* larger than were used in the recent comparison of maximum common subgraph algorithms by McCreesh et al. (2016): for unlabelled graphs, they used pairs of graphs with up to 35 vertices in each, whilst our dataset contains graphs with up to 6,671 vertices. For the forward-checking algorithm described by Ndiaye and Solnon (2011), 1,560 of the instances cannot fit in the amount of RAM we have available—we treat these instances as having timed out. The situation for the clique encoding is even worse, and 3,653 of these instances do not fit in 64GBytes of RAM. In contrast, for our algorithm, every instance fits comfortably.

2 Domain Filtering Using Degrees

Let p be a vertex in a graph P. The degree of vertices gives us an invariant, which may be used to eliminate some infeasible values from domains as follows.

[2]https://github.com/ciaranm/aaai17-between-subgraph-isomorphism-and-maximum-common-subgraph-paper

[3]http://liris.cnrs.fr/csolnon/SIP.html

Proposition 2. Let p be a vertex in P and t a vertex in T. For both non-induced and induced k-less subgraph isomorphisms, if $p \,{}_k\!\mapsto t$ then $\deg(p) - k \leq \deg(t)$.

Proof. Let p be a vertex in P, and t a vertex in T, with $p \mapsto t$. Then by the definition of subgraph isomorphisms, $\deg(p) \leq \deg(t)$. Let P' be P less k vertices and p a vertex in P'. Then

$$\deg_P(p) - k \leq \deg_{P'}(p) \leq \deg_P(p) \leq \deg(t). \qquad \square$$

The *neighbourhood degree sequence* (NDS) of a vertex p, $\mathrm{S}(p)$, is the (non-ascending) sequence of degrees of its neighbours. Zampelli, Deville, and Solnon (2010) showed how this may be used for filtering in subgraph isomorphism. We extend this for the k-less setting as follows.

Let $S = (s_1, \ldots, s_n)$ and $T = (t_1, \ldots, t_m)$ be two sequences. We say that $S \preceq T$ if $n \leq m$ and $\forall s_i \in S$ there exists a distinct $t_j \in T$ with $s_i \leq t_j$. When considering a k-less subgraph isomorphism we say that $S \,{}_k\!\preceq T$ if $n - k \leq m$, and if there exists some subsequence S_k of S containing up to k members such that $\forall s_i \in S \setminus S_k$, there exists a distinct $t_j \in T$ with $s_i - k \leq t_j$.

Proposition 3. If $p \,{}_k\!\mapsto t$, then $\mathrm{S}(p) \,{}_k\!\preceq \mathrm{S}(t)$.

Proof. Let $p \,{}_k\!\mapsto t$. Then $\deg(p) - k \leq \deg(q)$, by Proposition 2, which implies that $|\mathrm{S}(p)| - k \leq |\mathrm{S}(t)|$.

Also, $p \,{}_k\!\mapsto t$ implies that for each $q \in N(p) \setminus P_k$, where P_k is some subset of the vertices of P with $|P_k| \leq k$, we have $q \,{}_k\!\mapsto u$, where $u \in N(t)$ and each u is distinct. Then $\deg(q) - k \leq \deg(u)$, by Proposition 2. Hence $\mathrm{S}(p) \,{}_k\!\preceq \mathrm{S}(t)$. $\square$

Example 1. Consider the pattern graph P and the two target graphs T and U shown in Figure 2. The neighbourhood degree sequence of pattern vertex p is $\mathrm{S}(p) = (3, 3, 2)$. We highlight a subgraph in T which is non-induced isomorphic to P. The vertex p can be mapped to q in the target graph T, as $\mathrm{S}(q) = (5, 5, 4, 2, 2)$. There is a non-induced mapping of the pattern graph P into U by removing b of P. This removal changes the neighbourhood degree sequence of p in the k-less version of P to $\mathrm{S}(p) = (2, 2)$.

Figure 2 also illustrates the three cases possible when filtering by neighbourhood degree sequence. Removing vertex d causes each entry in $\mathrm{S}(p)$ to be reduced; removing vertex a removes an entry from $\mathrm{S}(p)$; and removing either of vertex b or c causes both the size of $\mathrm{S}(p)$ to be reduced and an entry in $\mathrm{S}(p)$ to be reduced.

Corollary 1. Since both $\mathrm{S}(p)$ and $\mathrm{S}(t)$ are non-ascending, without loss of generality we can replace $S \setminus S_k$ from the definition of $S \,{}_k\!\preceq T$ with the subsequence consisting of all but the first k members of S.

Figure 3 demonstrates that Proposition 3 is effective in practice: we show the amount of domain reduction we can achieve at the top of search by using invariants and a fixed k, compared to the search space size for the maximum common induced subgraph problem. (The results on the non-induced version show a similar but slightly weaker trend.) We look at the product of the domain sizes, rather than the number of

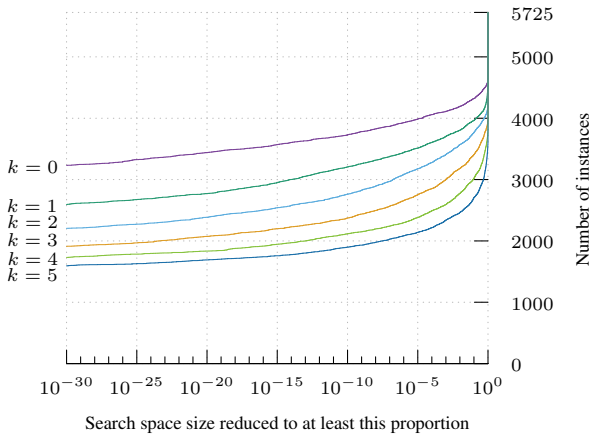

Figure 3: The amount of domain reduction achieved for the induced problem, with increasing k. The y value shows for how many instances we may reduce the initial search space to be at most the x proportion of the size the search space would be for the maximum common subgraph problem.

eliminations, as this better reflects the number of combinations remaining to be considered.

Some other invariants do not translate. For example, another rule which can be effective on regular graphs involves counting the number of neighbours of a vertex which are present in a triangle (McKay and Piperno 2014). Removing a single vertex can alter this count arbitrarily, so we cannot make use of this fact.

3 Filtering During Search Using Paths

As well as reasoning about degrees, we can also reason about paths. A *path* between two vertices p and q is a sequence of edges which can be traversed from p to reach q. We define $\text{paths}(p, q, n)$ to be the *number of paths of length* n between the vertices p and q.

Proposition 4. Let $p, q \in \text{V}(P)$ and $t, u \in \text{V}(T)$. If $p \mathrel{{}_k\!\!\mapsto} t$ and $q \mathrel{{}_k\!\!\mapsto} u$ then $\text{paths}(p, q, 2) - k \leq \text{paths}(t, u, 2)$.

Proof. Let P_2 be the set of all paths of length two between p and q,

$$P_2 = \{((p, x), (x, q)) : (p, x), (x, q) \in \text{E}(P)\}.$$

As we are looking at paths of length 2, the intermediate vertices lie in both neighbour vertex sets of p and q. We can rewrite P_2 as

$$P_2 = \{((p, x), (x, q)) : x \in \text{N}(p) \text{ and } x \in \text{N}(q)\},$$

in other words the intermediate vertices lie in the intersection of the neighbour sets of p and q,

$$P_2 = \{((p, x), (x, q)) : x \in \text{N}(p) \cap \text{N}(q)\}.$$

Therefore $\text{paths}(p, q, 2) = |\text{N}(p) \cap \text{N}(q)| \leq \deg(p)$.

If we remove up to k vertices from the neighbourhood of p, it will impact the intersection of the neighbourhoods of p and q, and by Proposition 2, as $p \mathrel{{}_k\!\!\mapsto} t$,

$$|\text{N}(p) \cap \text{N}(q)| - k \leq \deg(p) - k \leq \deg(t).$$

Figure 4: The pattern graph P on the left cannot be found in the target graph T, but if the vertex p in P is removed, then a subgraph isomorphism exists.

As $|\text{N}(p) \cap \text{N}(q)| \leq \deg(p) \leq |\text{N}(t) \cap \text{N}(u)| \leq \deg(t)$, we have

$$\begin{aligned} |\text{N}(p) \cap \text{N}(q)| - k &\leq \deg(p) - k \\ &\leq |\text{N}(t) \cap \text{N}(u)| \\ &\leq \deg(t) \end{aligned}$$

which tells us

$$\text{paths}(p, q, 2) - k \leq \text{paths}(t, u, 2). \qquad \square$$

Corollary 2. For a graph G, let $G^{n,\ell}$ be the graph with vertex set $\text{V}(G)$, and edges between vertices p and q if there are at least n simple paths of length exactly ℓ between p and q in G. Then any k-less subgraph isomorphism $P \mathrel{{}_k\!\!\rightarrowtail} T$ induces a new k-less subgraph isomorphism $P^{n+k,2} \mathrel{{}_k\!\!\rightarrowtail} T^{n,2}$ using the same vertex assignments.

Unlike in conventional subgraph isomorphism, we cannot extend this filtering to look at paths of length three: as the example in Figure 4 shows, removing a single vertex can delete arbitrarily many such paths. We *could* instead count paths of any length which are vertex disjoint, although calculating this appears to be too expensive to be beneficial in practice.

To allow for fast propagation, rather than calculating paths dynamically like Audemard et al. (2014), we follow the approach of McCreesh and Prosser (2015) and construct *supplemental graphs*, and find a mapping which is simultaneously a non-induced subgraph isomorphism between every supplemental graph pair. We use paths of length 2, looking at whether there are at least 1, 2, and 3 in the target graph (and so whether there are at least $1 + k$ up to $3 + k$ in the pattern). We then investigate whether this leads to new constraints being generated.

By an *assignment*, we mean considering mapping a pattern vertex p to a target vertex t (and not $\perp$) which does not violate any loop constraints. An *assignment pair* is two assignments with distinct p and distinct t, which we say is permitted if it does not violate any adjacency constraint. We define the *permitted assignment pair ratio* to be the proportion of assignment pairs which are permitted. Given this, in Figure 5 we scatter plot the permitted assignment pair ratio with and without supplemental graphs[4].

For $k = 0$, we see many points above the $x - y$ diagonal, which shows that for many instances, we are able to

[4]Because of the large sizes of the domains, we randomly sample one million pairs rather than considering every pair. In some cases, we have nearly a thousand domains, each with nearly ten thousand values—a complete quadratic calculation involving even a trivial arithmetic operation on this would take many hours.

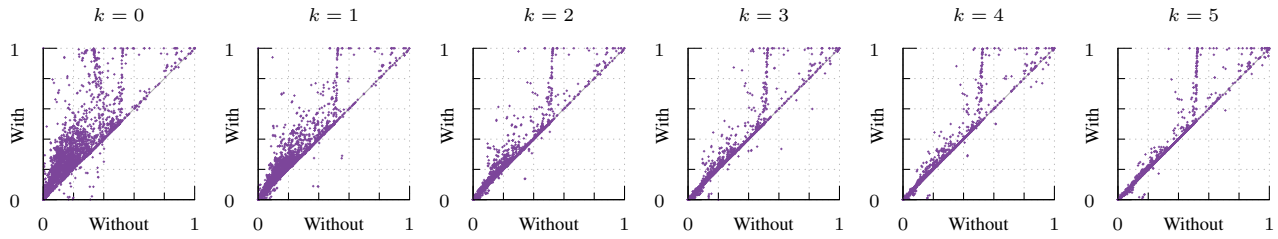

Figure 5: For the induced problem, the proportion of pairs of assignments from the filtered domains which are not permitted simultaneously, without path constraints on the x-axis and with path constraints on the y-axis, for increasing values of k.

create a substantial number of new constraints at the top of search; on the other hand, there are also points on the diagonal, which shows that sometimes this technique provides no benefit. (Occasionally, points fall below the $x - y$ diagonal. This is because the use of neighbourhood degree sequence reasoning on supplemental graphs can also lead to increased domain filtering, which could in turn eliminate a higher proportion of forbidden than permitted assignment pairs.) For $k = 1$ and $k = 2$, the proportion of points above the diagonal diminishes, but we are still able to create new constraints for many instances. By the time $k = 3$, most of the benefit is disappearing—although sometimes we are still able to make a difference, and bear in mind that sometimes adding just one new constrained pair can vastly reduce the search space.

4 A New Algorithm

Algorithm 1 integrates these techniques into a full algorithm. This is derived from an algorithm due to McCreesh and Prosser (2015), which in a recent evaluation of using portfolios of subgraph isomorphism algorithms was the single strongest solver (Kotthoff, McCreesh, and Solnon 2016).

The algorithm performs a constraint-based search. We have a set D containing a variable D_v for each vertex v in the pattern graph. Each variable has a domain containing one value for every vertex in the target graph. The algorithm is bit-parallel: each D_v is stored using bitset, and all graphs are stored as adjacency matrices.

In line 8 we use the reasoning from Section 2 to eliminate infeasible initial values from the domains. We do not use iterated label filtering (Zampelli, Deville, and Solnon 2010) to recalculate neighbourhood degree sequences if any vertices are not present in any target domain after construction: preliminary experiments indicated that this happened very rarely on our instances.

To handle unmapped vertices, on line 9 we include k additional wildcard $\perp$ values in each domain (rather than a single value which may be used k times).

We begin by trying to infer domain deletions. The `propagate` function looks for domains which contain either only a single value, or only wildcards—we call such a domain *effectively-unit*. If such a domain D_v exists, we eliminate its value from every other domain, and then propagate adjacency: for each domain corresponding to a vertex adjacent to v, we eliminate any value from its domain which is not adjacent to v (treating wildcards as being adjacent to all vertices). If a domain wipeout occurs, we return failure.

We deal with the additional constraints discussed in Section 3 by constructing supplemental graphs, as in the McCreesh and Prosser (2015) algorithm. This is done on line 4: the variable L contains a list of pattern / target pairs, and following Corollary 2, we will search for a mapping which is simultaneously a subgraph isomorphism between every pair in this list. We also do degree-based reasoning using each of these graph pairs.

If no effectively-unit domains remain, we attempt stronger propagation for the all-different constraint. Our `allDifferent` function is the bit-parallel propagator taken from McCreesh and Prosser (2015), and is *not* the usual matching-based propagator (Régin 1994) which guarantees generalised arc consistency. Because we use multiple wildcard values, we do not need to modify the algorithm to allow a single wildcard value to be used more than once. This propagation could create new effectively-unit domains; if so, we repeat the process.

If propagation is unable to prove unsatisfiability, we search. We pick the smallest domain (line 16) and try giving it each of its remaining values in turn. We use the value the ordering heuristic from the original algorithm; wildcards are treated as having degree zero, in an attempt to maximising the expected number of solutions remaining during search (McCreesh, Prosser, and Trimble 2016). We introduce a symmetry break (line 18) to try only a single wildcard value for each variable.

To handle the induced case, we make use of Proposition 1 (line 4). We considered using path reasoning on complement graphs, but this is expensive to calculate and provides little benefit in practice on these instances. We also do not strip isolated vertices as the original algorithm did, as this is not a valid simplification in the induced case, and we do not use conflict-directed backjumping as in our experiments it had very little effect for $k > 0$.

5 Empirical Evaluation

We now evaluate our algorithm and show that it is effective in practice, even on the larger subgraph isomorphism instances. In the first two plots of Figure 6 we give cumulative distributions for the induced and non-induced problems, with k ranging from 0 to 5 (we discuss the dotted lines in this plot in Section 5.1). The results are strong: with $k = 0$ we may solve nearly every instance, whilst even at $k = 5$ we

```
1  klessSubgraphIsomorphism(
       Graph 𝒫, Graph 𝒯, Int k) → Bool
2  begin
3      if |V(𝒫)| + k > |V(𝒯)| then return false
4      L ← [(𝒫, 𝒯), (𝒫̄, 𝒯̄) only if we want induced,
         (𝒫^{1+k,2}, 𝒯^{1,2}), (𝒫^{2+k,2}, 𝒯^{2,2}), (𝒫^{3+k,2}, 𝒯^{3,2})]
5      foreach v ∈ V(𝒫) do
6          D_v ← V(𝒯)
7          foreach (P, T) ∈ L do
8              D_v ← {w ∈ D_v :
                       v ∼_P v ⇒ w ∼_T w ∧
                       S_P(v)_k ⪯ S_T(w)}
9          D_v ← D_v ∪ k distinct wildcard values
10     if propagate(L, D) then
11         return search(L, {E ∈ D : |E| > 1}, k)
12     else return false

13 search (GraphPairs L, Domains D, Int k) → Bool
14 begin
15     if D = ∅ then return true
16     D_v ← the smallest domain in D
17     foreach v' ∈ D_v ordered by static degree in 𝒯 do
18         if v' is not the first wildcard we have tried then
19             continue
20         D' ← clone(D)
21         D'_v ← {v'}
22         if propagate(L, D') then
23             if search(L, {E ∈ D' : |E| > 1}, k) then
24                 return true

25 propagate (GraphPairs L, Domains D) → Bool
26 begin
27     while true do
28         if no effectively-unit domains remain (treating all
           wildcard values as a single value) then
29             if not allDifferent(D) then
30                 return false
31         if no effectively-unit domains remain then
32             return true
33         D_v ← an effectively-unit domain from D
34         v' ← the single value in D_v, or an arbitrary wild-
           card value if there is more than one
35         foreach D_w ∈ D \ {D_v} do
36             D_w ← D_w \ {v'}
37             foreach (P, T) ∈ L do
38                 if v ∼_P w then
39                     D_w ← D_w ∩ (N_T(v') ∪ wildcards)
40             if D_w = ∅ then return false

41 allDifferent (Domains D) → Bool
42 begin
43     (H, A, n) ← (∅, ∅, 0)
44     foreach D_v ∈ D from smallest to largest do
45         D_v ← D_v \ H
46         (A, n) ← (A ∪ D_v, n + 1)
47         if D_v = ∅ or |A| < n then return false
48         if |A| = n then (H, A, n) ← (H ∪ A, ∅, 0)
49     return true
```

Algorithm 1: A bit-parallel algorithm for the k-less subgraph isomorphism problem.

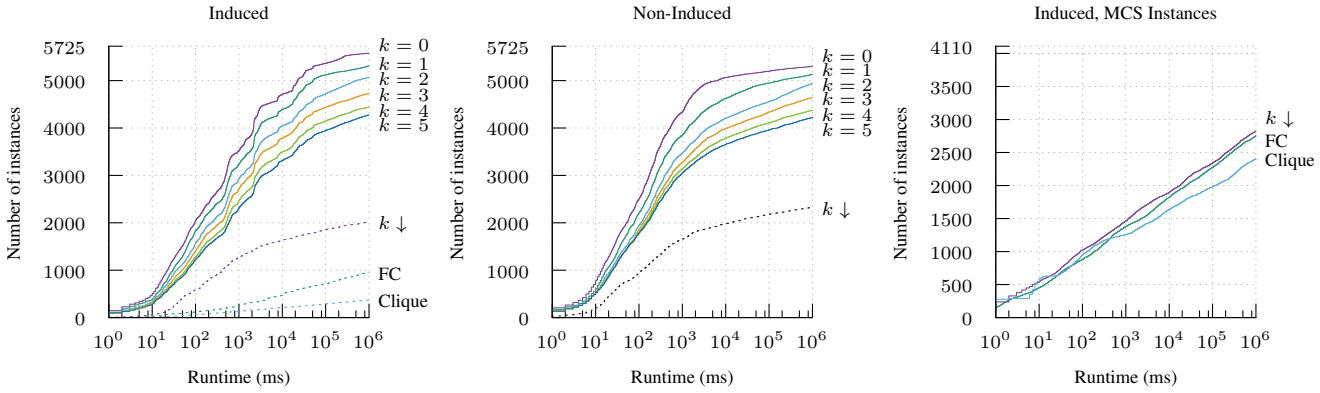

Figure 6: In the first two plots we show the cumulative number of instances solved over time, for the induced and non-induced problems, with different values of k. We also show the results of iteratively increasing k until a solution is found, and in the induced case, the performance of two leading maximum common subgraph algorithms. In the third plot we show results comparing iteratively increasing k with our algorithm to other approaches on maximum common induced subgraph instances.

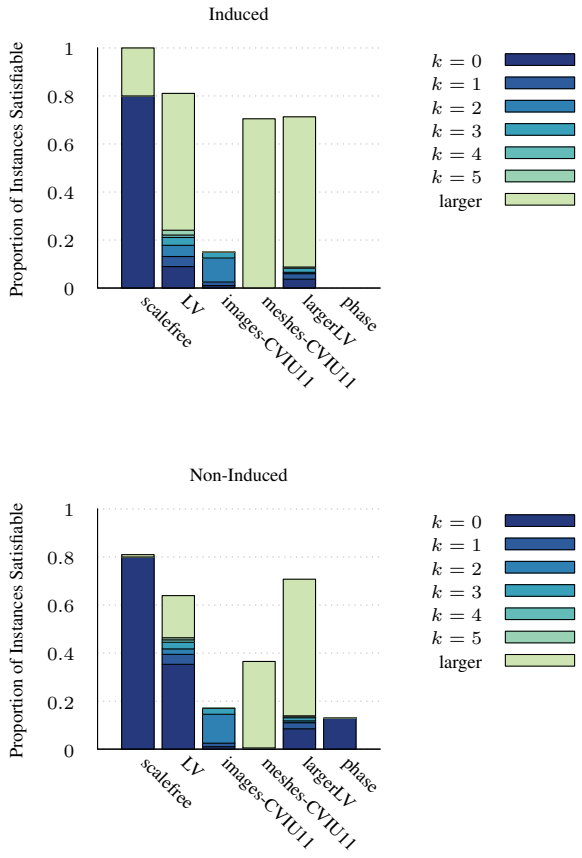

Figure 7: The proportion of instances, in different families, which become satisfiable for increasing values of k. The "larger" instances are those where we can prove unsatisfiability for $k = 5$, whilst the gap between the top of the bar and the top of the graph is the fraction of instances where a timeout was reached for at least one value of k.

can solve over $4,000$ instances in both variants.

But are we learning anything about the results—that is, are there instances for which $k = 0$ is unsatisfiable, but that are satisfiable for small k? For the problem families which do not consist entirely of satisfiable instances, we plot this in Figure 7. In the "phase" family, which consists of instances crafted to be extremely difficult to solve, we are not able to answer this question, and in the "scalefree" family we see no satisfiable instances with low but not zero k (this is simply due to there being many vertices with loops in some pattern graphs, but no loops at all in the targets). However, in several of the remaining families we can more than double the number of instances for which exact solutions are known, and gain upper bounds on many more. This is particularly interesting for the "images-CVIU11" family due to Damiand et al. (2011), where the size of the solution has a direct real-world interpretation in terms of closeness of image matching.

5.1 Solving From the Top Down

What would happen if we used our approach to solve the maximum common induced problem? We could simply start at $k = 0$, and increase k until a solution is found. This would be tackling the problem in the opposite direction from existing approaches, which work by attempting to construct larger and larger solutions. The dotted lines in Figure 6 show this approach: "$k \downarrow$" is our algorithm, whilst "FC" and "clique" are the two algorithms discussed by McCreesh et al. (2016). Recalling the disclaimer in Section 1.3 regarding instances not fitting in the 64GBytes of RAM we have available, we see that our approach is able to close over twice as many of these instances as the previous state of the art. (The same conclusion holds even if every instance which is satisfiable with $k = 0$ is removed from the dataset.)

What about if we use instances designed for the maximum common subgraph problem? In the third plot in Figure 6 we again compare these approaches, but using the first ten instances from each family of a standard benchmark set (Santo et al. 2003; Conte, Foggia, and Vento 2007), selecting undirected, unlabelled graphs with up to 50 vertices—this gives us a total of 4,110 instances. In these instances the number of vertices in the pattern and target graphs is the same, which is not ideal for our algorithm (although our invariants are still effective in many cases). Nonetheless, we are by a small margin the single strongest solver. Interestingly, our algorithm tends to have complementary performance to the clique approach, which suggests that there is scope for an algorithm which runs both an upper bound and a constructive, lower bound approach simultaneously or in parallel, stopping when the two bounds meet in the middle.

6 Conclusion

By considering a restricted variation of the problem, we have been able to go some of the way towards tackling the maximum common subgraph problem on much larger graphs than have previously been possible. We saw that some invariants from the subgraph isomorphism problem can still be used, albeit in a weakened form, with considerable effect when a small number of vertices can be removed. We have by no means cracked the problem completely—our results are still far from where we would like them to be for use in real-world applications, and many instances remain open. However, for many of the open instances we are at least able to get upper bounds on the result for the first time.

In future work, we will combine this approach with existing lower bounding construction methods, to deliver a hybrid algorithm. We also intend to allow restrictions on which vertices may or may not be removed, similar to the approach of Zampelli, Deville, and Dupont (2005), but with much stronger filtering now being possible.

Acknowledgements

We would like to thank Frances Cooper for her comments.

References

Audemard, G.; Lecoutre, C.; Modeliar, M. S.; Goncalves, G.; and Porumbel, D. C. 2014. Scoring-based neigh-

borhood 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*, volume 8656 of *Lecture Notes in Computer Science*, 125–141. Springer.

Blindell, G. H.; Lozano, R. C.; Carlsson, M.; and Schulte, C. 2015. Modeling universal instruction selection. In Pesant (2015), 609–626.

Bonnici, V.; Giugno, R.; Pulvirenti, A.; Shasha, D.; and Ferro, A. 2013. A subgraph isomorphism algorithm and its application to biochemical data. *BMC Bioinformatics* 14(Suppl 7):S13.

Bruschi, D.; Martignoni, L.; and Monga, M. 2006. Detecting self-mutating malware using control-flow graph matching. In Büschkes, R., and Laskov, P., eds., *Detection of Intrusions and Malware & Vulnerability Assessment, Third International Conference, DIMVA 2006, Berlin, Germany, July 13-14, 2006, Proceedings*, volume 4064 of *Lecture Notes in Computer Science*, 129–143. Springer.

Conte, D.; Foggia, P.; Sansone, C.; and Vento, M. 2004. Thirty years of graph matching in pattern recognition. *International Journal of Pattern Recognition and Artificial Intelligence* 18(03):265–298.

Conte, D.; Foggia, P.; and Vento, M. 2007. Challenging complexity of maximum common subgraph detection algorithms: A performance analysis of three algorithms on a wide database of graphs. *J. Graph Algorithms Appl.* 11(1):99–143.

Cordella, L. P.; Foggia, P.; Sansone, C.; and 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.-C.; and Samuel, E. 2011. Polynomial algorithms for subisomorphism of nD open combinatorial maps. *Computer Vision and Image Understanding (CVIU)* 115(7):996–1010.

Kotthoff, L.; McCreesh, C.; and Solnon, C. 2016. Portfolios of subgraph isomorphism algorithms. In *Learning and Intelligent OptimizatioN Conference LION 10 Ischia Island (Napoli), Italy, 29 May - 1 June, 2016, Proceedings*.

Larrosa, J., and Valiente, G. 2002. Constraint satisfaction algorithms for graph pattern matching. *Mathematical. Structures in Comp. Sci.* 12(4):403–422.

McCreesh, C., and Prosser, P. 2015. A parallel, backjumping subgraph isomorphism algorithm using supplemental graphs. In Pesant (2015), 295–312.

McCreesh, C.; Ndiaye, S. N.; Prosser, P.; and 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*, volume 9892 of *Lecture Notes in Computer Science*, 350–368. Springer.

McCreesh, C.; Prosser, P.; and 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*, 631–638. IJCAI/AAAI Press.

McGregor, J. J. 1979. Relational consistency algorithms and their application in finding subgraph and graph isomorphisms. *Inf. Sci.* 19(3):229–250.

McKay, B. D., and Piperno, A. 2014. Practical graph isomorphism, II. *Journal of Symbolic Computation* 60:94–112.

Murray, A. C., and Franke, B. 2012. Compiling for automatically generated instruction set extensions. In Eidt, C.; Holler, A. M.; Srinivasan, U.; and Amarasinghe, S. P., eds., *10th Annual IEEE/ACM International Symposium on Code Generation and Optimization, CGO 2012, San Jose, CA, USA, March 31 - April 04, 2012*, 13–22. ACM.

Ndiaye, S. N., and Solnon, C. 2011. CP models for maximum common subgraph problems. In Lee, J. H., ed., *Principles and Practice of Constraint Programming - CP 2011 - 17th International Conference, CP 2011, Perugia, Italy, September 12-16, 2011. Proceedings*, volume 6876 of *Lecture Notes in Computer Science*, 637–644. Springer.

Pesant, G., ed. 2015. *Principles and Practice of Constraint Programming - 21st International Conference, CP 2015, Cork, Ireland, August 31 - September 4, 2015, Proceedings*, volume 9255 of *Lecture Notes in Computer Science*. Springer.

Régin, J. 1994. A filtering algorithm for constraints of difference in csps. In *Proceedings of the 12th National Conference on Artificial Intelligence, Seattle, WA, USA, July 31 - August 4, 1994, Volume 1.*, 362–367.

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

Sevegnani, M., and Calder, M. 2015. Bigraphs with sharing. *Theoretical Computer Science* 577(0):43 – 73.

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

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

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

Zampelli, S.; Deville, Y.; and Dupont, P. 2005. Approximate constrained subgraph matching. In van Beek, P., ed., *Principles and Practice of Constraint Programming - CP 2005, 11th International Conference, CP 2005, Sitges, Spain, October 1-5, 2005, Proceedings*, volume 3709 of *Lecture Notes in Computer Science*, 832–836. Springer.

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