SEARCH-TREE SIZE ESTIMATION FOR THE SUBGRAPH ISOMORPHISM PROBLEM

Uroš ČIBEJ, Jurij MIHELIČ
Faculty of Computer and Information Science
University of Ljubljana, Slovenia
{uros.cibej, jurij.mihelic}@fri.uni-lj.si

ABSTRACT

This article addresses the problem of finding patterns in graphs. This is formally defined as the subgraph isomorphism problem and is one of the core problems in theoretical computer science. We consider the counting variation of this problem. The task is to count all instances of the pattern G occurring in a (usually larger) graph H. The vast majority of algorithms for this problem use a variation of backtracking. Most commonly they exhaustively search through the space of all possible monomorphisms between G and H. The size of the search tree depends heavily on the choice of the ordering of vertices of G, which are systematically assigned to the vertices of H. We use a method called heuristic sampling to estimate the size of the search tree for each ordering in advance. We use this estimation to select the most suitable order of vertices of G which minimizes the expected tree size. This approach is empirically evaluated on a set of instances, showing the practical potential of the method.

Keywords: *Backtracking, heuristic sampling, sugraph matching, pattern matching, search trees, subgraph isomorphism*

1. INTRODUCTION

Backtracking algorithms are one of the most common approaches to solving hard optimization, enumeration, and decision problems. They are simple to implement and understand, but their behavior is very difficult to predict. Since most of the problems being solved by backtracking are computationally intractable, many pruning techniques and various parameterized tricks are being used to reduce the execution time. The large number of parameters which define a particular variation of the backtracking search becomes a problem when we need to choose the best algorithm for the problem at hand. This article addresses this issue.

One of the fundamental problems classically solved by variations of backtracking algorithms is the subgraph isomorphism problem. The goal of this problem is to find a pattern graph G in a target graph H. Many variations of algorithms for this problem exist, their main differences being the methods for pruning (reducing the size) of the search tree. These variations can have a vastly different behavior on the same instance, e.g. a certain instance might be trivial for one algorithm and impossible for another. This opens up another algorithmic possibility, namely, for each instance the best algorithm could be chosen in advance, if only we knew its behavior before actually running it on the instance. Ideally, this behavior could be predicted analytically, by simply computing a set of features of the problem instance. However, this has been a difficult task, because of the large number of parameters that influence the outcome.

But luckily, there has been an interesting method developed to tackle this problem from a more pragmatical, empirical point of view. The basic method was developed by Donald Knuth [15], where he explored the possibilities to predict the search tree sizes for a few simple problems (such as the knights tour and the Instant Insanity game). The weakness of Knuth's approach is the fact that it assumes a rather homogeneous tree, i.e. all (most) of the subtrees of the search tree are somewhat similar in shape (and size). This is true for such simple problems as Knuth was dealing with, but it is not true for more complex problems.

This is why a more complex method was devised, named heuristic sampling [5]. It is a generalization of Knuth's method, where for a good implementation some specific knowledge of the underlying problem has to be provided in the form of a heuristic function.

The main goal of this article is to implement the heuristic sampling method for the subgraph isomorphism search, devise a suitable heuristic sampling function and evaluate it on a set of backtracking algorithms. We will focus on its predictive strength for choosing the best algorithm for a particular instance of the problem. Some preliminary work on this subject was done by the authors in [8]

The remainder of the paper is organized as follows. Next section introduces the subgraph isomorphism problem, the notation used throughout the paper, and the algorithmic framework that we will be using in the evaluation of the proposed method. Section 3 describes the foundations of search-tree size estimation and its extension, the heuristic sampling. Section 4 describes the empirical results, namely the test set of graphs used in the evaluation and the setup of the experiments, and the analysis of the obtained results. Section 5 explores the possibilities of extension of this work into practical solvers and concludes the paper.

2. SUBGRAPH ISOMORPHISM PROBLEM

From the viewpoint of pattern analysis and recognition, matching graphs and subgraphs is one of the most important tasks in graph processing. One approach to modeling this task is via the *subgraph isomorphism problem*. Given two graphs, namely, a *pattern graph* and a *target graph*, the problem is defined as finding a subgraph (corresponding to the pattern graph) in the target graph.

Formally, the problem can be defined as follows.

Definition 2.1. *A subgraph isomorphism between a pattern $G = \langle V, E\rangle$ and a target $H = \langle U, F\rangle$ is an injective function $f : V \longrightarrow U$ satisfying the condition $(i, j) \in E \Rightarrow (f(i), f(j)) \in F$. Similarly an induced subgraph isomorphism between a pattern $G = \langle V, E\rangle$ and a target $H =$*

$\langle U, F \rangle$ *is an injective function* $f : V \longrightarrow U$ *satisfying the condition* $(i, j) \in E \Leftrightarrow (f(i), f(j)) \in F$.

The two variations of the problem are demonstrated in Fig. 1. From the algorithmic point of view it does not matter if we address the induced or the regular subgraph isomorphism. All the algorithms can find both types of isomorphisms with only minor modifications.

Applications of pattern analysis using the subgraph isomorphism problem abound in various fields, such as chemistry [1, 2], social network analysis [14], and computer vision [16] . Moreover, in these domains graph databases are replacing the traditional relational databases [3]. For an extensive review of graph matching for pattern recognition see [9].

Unfortunately, the subgraph isomorphism problem is computationally very difficult, as it has been shown to be *NP*-complete [10]. Being such a fundamental problem, there are many attempts to overcome this theoretical barrier, and the state-of-the-art algorithms can easily solve non-trivial problems.

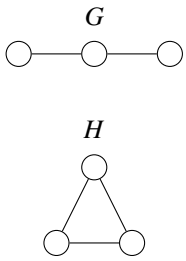

Fig. 1 Finding a pattern graph G in the goal graph H. The graph G is a subgraph in H, but it is not an induced subgraph in H.

2.1. Algorithms

Since this is one of the fundamental problems in computer science, many algorithms have been proposed to solve it. The oldest algorithm that is still a reference for newer approaches is the Ullmanns' algorithm [18]. For many years it was also considered the best for the problem at hand. But it was also treated to be of more theoretical nature, and there were not many attempts to engineer this backtracking into faster algorithms.

Because computers got much faster, and the applications started to deal with more data organized as graphs, the interest for this problem has been increasing, especially in the pattern matching community. New and improved algorithms have started to emerge [7, 11, 17, 19], showing remarkable results on larger and larger instances.

Even though these algorithm utilize very different techniques, all of them follow the same basic pattern of a backtrack search. We can summarize this pattern as a general framework, and we will use it as the basis for our experiments The framework is given in Algorithm 2 and can be described as follows.

Besides the input pattern G and target H, the input to this algorithm is an ordering of vertices in G and the current (partial) assignments (*iso*) which grows and shrinks as the algorithm traverses the search tree. At each step (node of the search tree), the vertex v is the active vertex. This vertex is defined as the first vertex in the current order. The function *getCandidates* retrieves vertices from H which have the same connectivity in H as v has in G, based on the already assigned vertices in *iso*. When we reach the leaf of a search tree, the set of candidates is exactly the number of subgraph isomorphisms at that position. If we are not at the leaf we try to assign v to every possible candidate and proceed recursively.

```
1  Function countIso(G,H, iso, order);
2  v = head(order);
3  candidates = getCandidates(v,iso,G,H);
4  if size(order) = 1 then
5  |  return size(candidates);
6  end
7  nIso = 0;
8  forall u ∈ candidates do
9  |  iso = iso ∪ (v → u);
10 |  nIso = nIso + countIso(G,H, iso,
   |    tail(order));
11 |  iso = iso \ (v → u);
12 end
13 return nIso
```

Fig. 2 countIso() - counting subgraph isomorphisms with a basic backtracking algorithm.

As mentioned earlier, the most influential factor for the speed of the algorithms is the order in which vertices of G are mapped onto H. This impact of this parameter is the easiest to see in the Ullmanns' algorithm, where the vertices of G are simply ordered by their degree (in descending order). If we would use a random order instead, even small instances become impossible to solve (i.e. do not finish in a reasonable amount of time). So the impact of using better orders is felt much more than of any other pruning technique in this algorithm. Other algorithms use more sophisticated orders, usually taking into account also the neighborhood of the currently chosen vertex and other parameters, but the impact on the tree size is similar than with the Ullmanns' algorithm..

Since this is a very impactful technique for reducing the search-tree size, there have been also more in-depth research done, such as in [4, 6]. We will not handle all the available orders but instead focus on a subset of them, since our goal is not to exhaustively evaluate various orders but instead test how good can we predict the performance of different orders.

From a large set of available orders we chose three basic orders, which utilize very local information, and two orders which use more information about the structure of the graph. The three basic orders are as follows.

Degree (DEG) The order which was already proposed by Ullmann in [18] is the ordering of the pattern vertices by their degree (descending). The logic behind this ordering is that the nodes with the highest degree can usually be mapped to the fewest goal nodes, which

reduces the search space already at the top levels of the search tree.

Depth-first search order (DFS) This order follows from a simple traversal of the tree in a depth-first manner. The first vertex is always the one with the highest degree. The trace of visited nodes gives the final order.

Breadth-first search order (BFS) Same logic as the *DFS* order, but with a breadth-first search.

The two more advanced orders can be viewed as an extension of BFS. They start with the highest degree vertex and put the neighborhood in the set of active vertices. This set is used to choose the next vertex based on a specific feature. Once a vertex is chosen, all its neighbors are added to the set of active vertices. This process is repeated until the set of active vertices is empty. The criteria by which the vertex is chosen from the active set defines the order more specifically.

Subdegree (SUB-DEG) The *subdegree* of a vertex is the number of edges from this vertex to the nodes in the set of vertices already chosen in the order. Or more precisely

$$d_{order}(v) = |\{u \in order | (v,u) \in E\}|$$

For this order, the vertex with the maximal subdegree is chosen from the active set of vertices.

Clustering (CLUST) Another criterion for the choice of a vertex from the active set is the clustering coefficient. This is a measure which quantifies how close the neighborhood of a node is to being a clique. This coefficient for a vertex v_i is computed as:

$$c_i = \frac{|\{e_{jk} : e_{jk} \in E \wedge v_j, v_k \in N_i\}|}{k_i(k_i-1)}$$

where N_i is the set of neighbors of v_i and $k_i = |N_i|$. Choosing vertices with a higher clustering coefficient also reduces the search space, because more edges in the neighborhood means more restrictions to the set of candidates (i.e. less candidates) in the assignment phase of the backtracking algorithm.

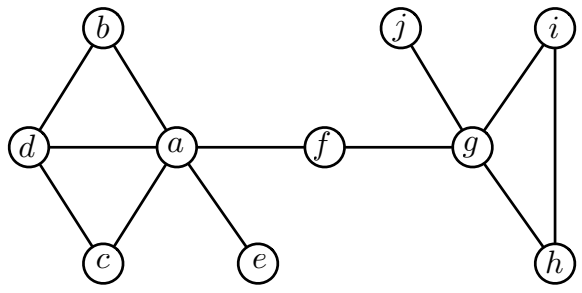

Fig. 3 An example graph, to show the different orderings. The DEG order on this graph is a,g,d,b,c,i,h,f,e,j, the DFS and BFS order are coincidentally the same a,b,d,c,e,f,g,h,i,j, the SUB-DEG order is a,d,b,c,f,g,h,i,e,j and the CLUST ordering is a,b,c,d,e,f,g,h,i,j.

Figure 3 shows an example graph with the 5 described orders.

As we will see in the empirical evaluation, none of these 5 orders dominates others in the sense that it would consistently result in the smallest search trees. Any of these orders can be the best on some instances and our goal is to find the best one before the actual execution of the algorithm.

3. SEARCH-TREE SIZE ESTIMATION

In this section we will describe the method for estimating the search tree size and the specific choices we made in order to use it for the subgraph isomorphism search.

3.1. Knuth's method

Since heuristic sampling is the extension of Knuth's method, we give a quick description of this interesting approach. In a nutshell, the method traverses the tree by choosing a random path until reaching a leaf of the tree. This random walk yields one estimation of the tree size as:

$$S_{est} = 1 + b_1 + b_1 b_2 + b_1 b_2 b_3 + \ldots,$$

where b_i is the number of children the node at depth i has (i.e. the branching rate of that node, see Figure 4). By repeating this random walk and averaging the result, the estimation quickly converges very closely to the actual size on many problems.

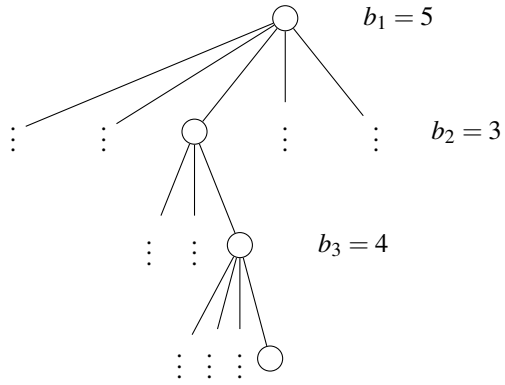

Fig. 4 A random walk in the tree, where the branching at each level gives an estimate of the overall tree size. This is the Knuth's estimation method in a nutshell.

Our first attempts of using Knuth's method for subgraph isomorphism did not yield satisfactory results. The main reason for this is that the simple sampling assumes a very homogeneous structure of the search-tree. Such structure is present in many classical search problems (such as games and basic combinatorial problems), but it is definitely not present in more complex search problems where the structure varies between branches significantly.

3.2. Heuristic sampling

Heuristic sampling (HS) addresses the issues of Knuth's method. Instead of following a single path in the tree, HS proceeds on many representative paths at the same time. This approach captures better the heterogeneous nature of search trees.

The central concept of HS is a heuristic function $h : N \to \mathscr{P}$, N being the set of nodes of the search tree and $\mathscr{P}$ being a partially ordered set. This function is called a stratifier and it should be designed in such a way to reflect the main characteristics of the nodes in the search tree. Intuitively, it is a function which maps each node into a value that should describe the shape of the subtree rooted at that node. Two nodes mapped to the same value by the stratifier should have similar (similarly sized) subtrees.

With the stratifier given, the method must now find an estimation for the number of nodes for each $\alpha \in \mathscr{P}$. And from such estimations the entire size of the tree can simply be computed as:

$$S_{est} = \sum_{\alpha} s_{\alpha}.$$

In order for HS to be successful on subgraph isomorphism problem, we had to find a suitable stratifier. One which captures the shape of the tree well enough, but also does not degenerate into a function that has to search the entire tree. We chose to use the function

$$h(n) \to (depth, deg_n),$$

where $depth$ is the depth at which the node is located and deg_n is the number of children the node has. So two nodes will be considered equal if they are at the same depth and they have the same number of children.

The details of the entire procedure are given in Algorithm 5. The method initiates a priority queue, which will serve as the data structure for holding sample nodes of the search tree. The priority of the elements in the queue is $h(n)$, and the dequeuing removes the element with the maximal value of h. At each step of the iteration, one node is retrieved from the queue and all its children are generated. These children (m) are added to the queue if their $h(m)$ is not yet present in the queue otherwise the weight of the element in the queue is increased (we found a new instance for this stratum) and the old node in the queue is replaced by the newly found with a probability $\frac{w}{w_s}$, where w is the weight of the parent node and w_s is the weight of the newly generated node. A detailed theoretical justification of this method can be found in [5].

```
1  Q = [root, 1)];
2  sol = {};
3  while |Q| > 0 do
4  |   (n, w) = Q.dequeue;
5  |   sol = sol ∪ {(n, w)};
6  |   Q = Q \ (n, w));
7  |   for m ∈ children of n do
8  |   |   α_m = h(m);
9  |   |   if Q contains an element (s, w_s) with
   |   |    h(s) = α_m then
10 |   |   |   w_s = w_s + 1;
11 |   |   |   with probability w/w_s do s = n;
12 |   |   end
13 |   |   else
14 |   |   |   Q = Q ∪ (m, 1);
15 |   |   end
16 |   end
17 end
18 return sol
```

Fig. 5 Heuristic sampling in more detail.

4. EMPIRICAL EVALUATION

In the previous section we described two methods for estimating the search-tree size. We mentioned that heuristic sampling is an extension of the basic method developed by Knuth in order to overcome the shortcomings of the method when dealing with more heterogeneous search trees. Our first goal is to demonstrate that the Knuth's method is not suitable for the subgraph isomorphism problem. On a chosen set of problem instance we obtained the exact sizes of the search tree and both the estimates of the Knuths method and the HS estimate. By showing the large qualitative difference between these two estimates we can justify the dismissal of the Knuths method in our further experiments. Our final goal is to demonstrate that HS can be used as a practical subroutine for a dynamic choice of the ordering for a specific problem instance.

4.1. Experiment Setup

The implementation of this experimental part was done in python, using the igraph library [12] to tackle the basic graph operations and algorithms. In order to fully control the execution of the backtracking algorithm we reimplemented the basic algorithm and we included a few simple pruning techniques. This enabled us to accurately count the tree size and also to implement heuristic sampling since it needs to precisely follow the algorithm it is estimating.

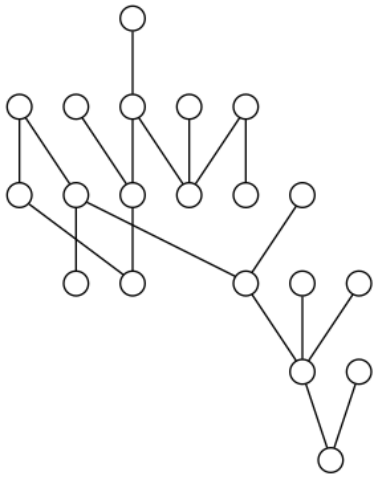

Fig. 6 One of the pattern graphs that we used to search for in larger graphs.

For testing the subgraph isomorphism algorithms, one of the most widely used benchmark set is the Amalfi test-set [13]. It contains a large amount of different graphs, from Erdös-Renyi random graphs, to highly regular meshes. For this evaluation we chose a subset of 100 instances from this benchmark set. All the chosen graphs are random Erdös-Renyi graphs. The main reason for the choice of such a limited subset is that larger instances are too difficult for such a basic backtracking procedure without more sophisticated pruning techniques. So in order to obtain the exact solutions for all the instances, we had to choose smaller graphs.

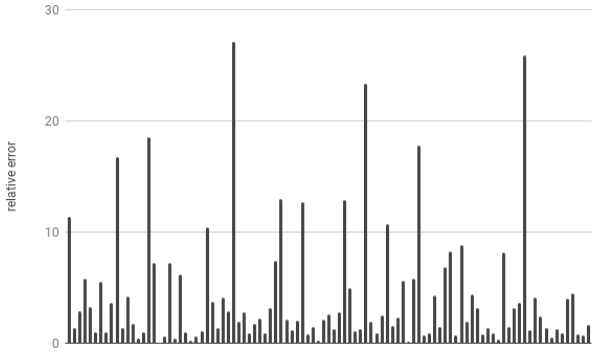

Fig. 7 The relative error of the Knuth's method on each of the 100 instances.

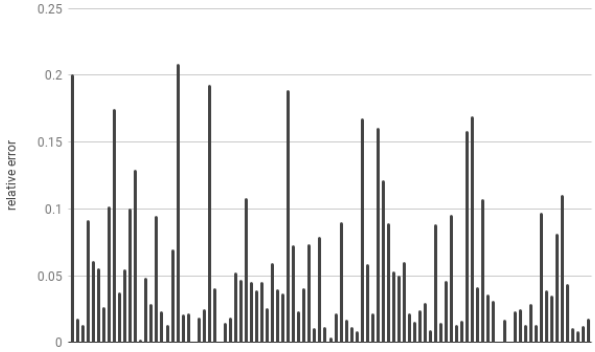

Fig. 8 The relative error of heuristic sampling on each of the 100 instances.

We mentioned earlier that HS addresses the weaknesses of Knuth's method, since it does not assume the search tree to be very homogeneous and tries to capture this heterogeneity by following several different paths. Our first experiment will demonstrate this empirically. Fig. **??** and Fig. 8 show the relative error on every problem instance for the Knuth's method and for heuristic sampling respectively. By relative error we mean

$$\frac{|S - S_{est}}{S}|,$$

where S denotes the search-tree size, and S_{est} denotes the estimated search-tree size. The qualitative difference between the two methods is obvious, since the error differs by nearly two orders of magnitude, i.e., the maximal error of HS is 0.2 (20%), whereas the error of the Knuth's method is more that 20. Such errors are unacceptable, so the Knuth's method is omitted from further experiments.

4.2. Evaluation of HS

We first ran the backtracking algorithm on all instances and measured the size of the search tree for each of them. We could also measure the execution time, since it is nicely correlated with the size of the tree. This is simply because the order remains static during the execution and therefore there is no particular overhead for the computation of different orders. But it is better to choose a machine independent measure, which makes the evaluation more robust, so the tree size was the most natural choice. To obtain a more reliable number, each estimation was repeated 100 times, and the average was computed.

First let us demonstrate that these five orders exhibit a large variance in performance on different instances. Figure 9 shows which order was the winner on 100 instances of the benchmark set. We can see that all orders have instance on which they yield the smallest search tree. The clustering order is the winner on only one instance, but the other orders are significantly more successful.

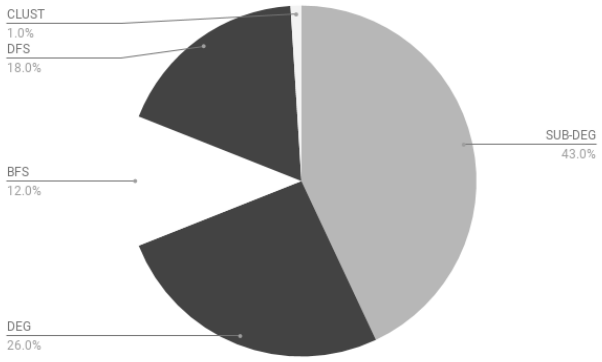

Fig. 9 The best orders for each tested instance. Every order is the best on at least one input instance, showing the diversity and importance of predicting the performance for each order.

Figure 10 shows also how the sizes of the search trees differ for each instance. The graph shows a scatterplot of relative differences for each instance:

$$\frac{s_{max} - s_{min}}{s_{min}},$$

where s_{max}, s_{min} are the largest and smallest tree sizes between the 5 orders. The y-axis in logarithmic, since in some instances the largest tree size was more than 100 times larger than the smallest one. For the majority of instances the difference is more than 100%, which in practice means that in the worst case, the algorithm would run much longer than in the best case.

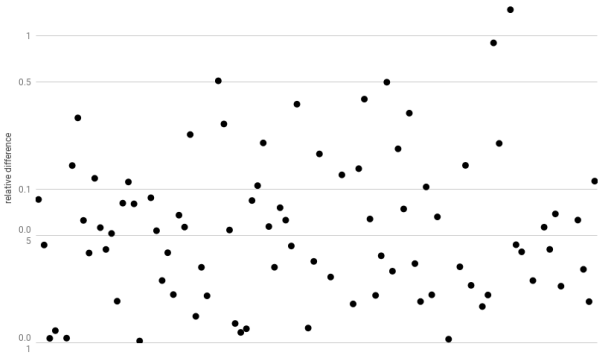

Fig. 10 The scatterplot shows the relative difference of the tree size between the best order and worst order on each of the 100 instances. The y-axis is logarithmic.

Next, we use the heuristic sampling for predicting the tree size for each test instance and each order, altogether 500 predictions. In order to compare the efficiencies between instances we compute the relative error

$$\frac{|T| - S_{est}}{|T|},$$

where T is the tree size and S_{est} is the estimated size. Figure 11 shows the distribution of these relative errors for the 500 predictions. We can see there are a few predictions that were not that good, but the vast majority of predictions is within 20% of the actual size of the tree.

Since we are investigating if the sampling can be successfully used as a predictor in the backtracking algorithms, the error in the prediction is not the most important factor, What is more important is the ranking accuracy of the method. For each instance we are interested in finding the best possible order in advance. For this reason we now show the distribution of the relative difference between the best tree size, and the tree size of the chosen order (both actual, not the predictions):

$$\frac{|T_{best}| - |T_{chosen}|}{|T_{best}|}.$$

The results for this measurement are shown in Figure 12. As we can see, more than 80% of all chosen orders were either exactly the best order or within 5% of the best order. On one instance the predicted winner performed very badly (more than twice the size of the actual best order), but on the rest of the instances the predictions are very feasible for practical applications.

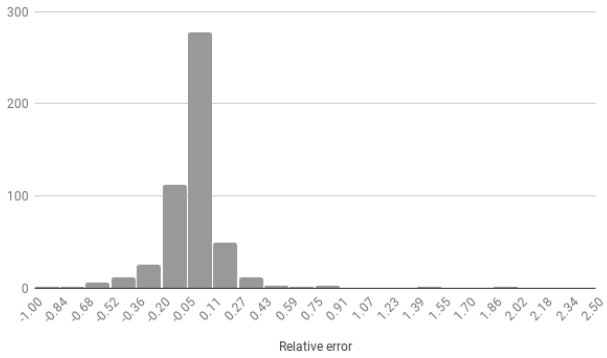

Fig. 11 The histogram of relative errors for each prediction (100 instances and 5 orders, i.e. 500 predictions). Two predictions were more than twice the actual number of tree nodes, however most of the predictions were very accurate.

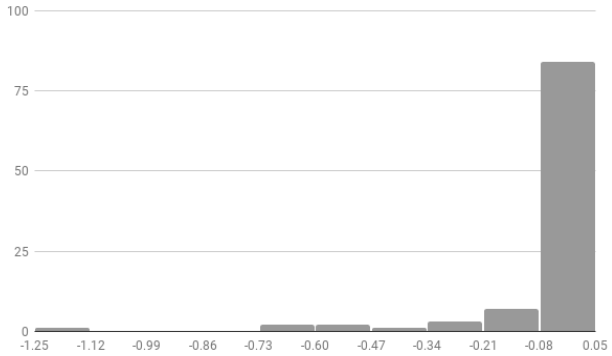

Fig. 12 The relative difference between the chosen order and the best order for an instance. In 80 % of cases the chosen order in either the best one or within 5% of the best. In one case the chosen order was twice as bad as the best one.

sectionConclusions and future work

Subgraph isomorphism is a classical problem in theoretical computer science and due to an explosion of practical application of pattern matching and analysis, it is becoming a highly practical problem as well.

Being an *NP*-hard (or the counting version even #*P*-complete), we do not expect any fast or even feasible algorithms for solving the general case of the problem. However, many pragmatic approaches achieve remarkable results, making many problem instances (sometimes even the majority) much more practical than its theoretical properties would suggest.

In the world of heuristics, the problem is attacked with various rules, subroutines, pruning techniques, etc. This constitutes a bag of different algorithms and there is always a dilemma which choice of parameter makes the best algorithm. Researchers in this area try to find the best parameter configuration by looking at the average performance on a

benchmark set of problem instances. However, by measuring the average performance, the chosen algorithm might not be best for a particular instance. The optimal strategy would be to choose the parameter configuration for every instance, if one would have an oracle for the performance. Exact oracle for the performance of an algorithm, of course, do not exist (in general), so we have to resort to estimation techniques.

Backtracking algorithms are the most common approach to solving the subgraph isomorphism algorithm. One of the most influential parameters on the running time is the ordering of the vertices of the pattern graph. There are many possible orderings and their efficiency varies significantly from instance to instance. We chose to investigate the possibility of predicting the best vertex ordering for each instance in advance. Luckily, various techniques for estimating the search-tree size (which is proportional to the running time) have been developed. We chose to investigate Knuth's basic estimation method and a method called heuristic sampling which addresses the weak points of the basic estimation.

We managed to adapt heuristic sampling for the subgraph isomorphism problem and first show that it significantly outperforms the estimation accuracy over the Knuths method. Furthermore, we showed the practical potential of heuristic sampling as the estimator for the best vertex ordering on a particular problem instance.

This paper is an initial test of the feasibility of the proposed estimation approach. In order for this method to become practical we need to involve all the pruning techniques of the more advanced algorithms and see if the sampling is so efficient also in that context. Another research direction is the possibility for new stratifiers in the heuristic sampling. Stratifiers with even more information about the underlying solution could better describe the shape of the tree, resulting in more accurate estimations.

And finally, since the sampling gives us a lot of information about the future performance of the algorithm it could also be used to make the ordering adapt to changes it sees ahead. During the search, the sampling could provide an estimate of the search tree still to be explored. If the size of the tree is too big, the order can change and thus dynamically adapt to different parts of the graph.

REFERENCES

[1] Dimitris K. AGRAFIOTIS, Victor S. LOBANOV, Maxim SHEMANAREV, Dmitrii N. RASSOKHIN, Sergei IZRAILEV, Edward P. JAEGER, Simson ALEX, and Michael FARNUM. Efficient Substructure Searching of Large Chemical Libraries: The ABCD Chemical Cartridge. *J. Chem. Inf. Model.*, 2011.

[2] John M. BARNARD. Substructure searching methods: Old and new. *J. Chemical Information and Computer Sciences*, 33(4):532–538, 1993.

[3] S. BATRA and C. TYAGI. Comparative analysis of relational and graph databases. *Int'l J. Soft Computing & Engineering*, 2012.

[4] Vincenzo BONNICI and ROSALBA Giugno. On the variable ordering in subgraph isomorphism algorithms. *IEEE/ACM Trans. Comput. Biol. Bioinformatics*, 14(1):193–203, January 2017.

[5] Pang C. CHEN. Heuristic sampling: A method for predicting the performance of tree searching programs. *SIAM Journal on Computing*, 21(2):295–315, 1992.

[6] Uroš ČIBEJ and Jurij MIHELIČ. Search strategies for subgraph isomorphism algorithms. In *International Conference on Applied Algorithms, ICAA*, pages 77–88, 2014.

[7] Uroš ČIBEJ and Jurij MIHELIČ. Improvements to ullmann's algorithm for the subgraph isomorphism problem. *International Journal of Pattern Recognition and Artificial Intelligence*, 29(07):1550025, 2015.

[8] Uroš ČIBEJ and Jurij MIHELIČ. Heuristic sampling for the subgraph isomorphism problem. In *2017 IEEE 14th International Scientific Conference on Informatics*, pages 57–62, Nov 2017.

[9] D. CONTE, P. FOGGIA, C. SANSONE, and M. VENTO. Thirty years of graph matching in pattern recongnition. *Int'l J. Pattern Recognition and Artificial Intelligence*, 18(03):265–298, 2004.

[10] Stephen A. COOK. The complexity of theorem-proving procedures. In *Proc. 3rd annual ACM symposium on Theory of computing - STOC '71*, pages 151–158. ACM Press, 1971.

[11] Luigi P. CORDELLA, Pasquale FOGGIA, Carlo SANSONE, and Mario VENTO. A (sub) graph isomorphism algorithm for matching large graphs. *IEEE transactions on pattern analysis and machine intelligence*, 26(10):1367–1372, 2004.

[12] Gabor CSARDI and Tamas NEPUSZ. The igraph software package for complex network research. *InterJournal*, Complex Systems:1695, 2006.

[13] M. De SANTO, P. FOGGIA, C. SANSONE, and M. VENTO. A large database of graphs and its use for benchmarking graph isomorphism algorithms. *Pattern Recognition Letters*, 24(8):1067–1079, May 2003.

[14] Wenfei FAN. Graph pattern matching revised for social network analysis. In *Proc. 15th International Conference on Database Theory - ICDT '12*, page 8. ACM Press, March 2012.

[15] Donald E. KNUTH. Estimating the efficiency of backtrack programs. Technical report, Stanford, CA, USA, 1974.

[16] Jianzhuang LIU and Yong Tsui LEE. Graph-based method for face identification from a single 2D line drawing. *IEEE Trans. Pattern Analysis and Machine Intelligence*, 23(10):1106–1119, 2001.

[17] Christine SOLNON. AllDifferent-based filtering for subgraph isomorphism. *Artificial Intelligence*, 174(12-13):850–864, August 2010.

[18] Julian R. ULLMANN. An algorithm for subgraph isomorphism. *Journal of the ACM (JACM)*, 23(1):31–42, 1976.

[19] Julian R. ULLMANN. Bit-vector algorithms for binary constraint satisfaction and subgraph isomorphism. *Journal of Experimental Algorithmics (JEA)*, 15:1–6, 2010.

Received April 17, 2018, accepted July 10, 2018

BIOGRAPHIES

Uroš Čibej received his doctoral degree in Computer Science from the University of Ljubljana in 2007. Currently, he is with the Laboratory of Algorithmics. His research interests include location problems, distributed systems, computational models, halting probability, graph algorithms, and computational complexity.

Jurij Mihelič received his doctoral degree in Computer Science from the University of Ljubljana in 2006. Currently, he is with the Laboratory of Algorithmics, Faculty of Computer and Information Science, University of Ljubljana, Slovenia, as an assistant professor. His research interests include algorithm engineering, combinatorial optimization, heuristics, approximation algorithms, and uncertainty in optimization problems as well as system software and operating systems.