University of Glasgow | School of Computing Science

Algorithm Selection for Maximum Common Subgraph

Paulius Dilkas

School of Computing Science
Sir Alwyn Williams Building
University of Glasgow
G12 8QQ

Level 4 Project — 26th November 2017

Education Use Consent

I hereby give my permission for this project to be shown to other University of Glasgow students and to be distributed in an electronic format. **Please note that you are under no obligation to sign this declaration, but doing so would help future students.**

Name: _____ Signature: _____

Contents

Chapter 1

Introduction

Definition 1.0.1. An undirected *multigraph* is a pair (V, E), where V is a set of vertices and E is a set of edges, together with a map $E \to V \cup V^2$, which assigns one or two vertices to each edge [3]. If an edge is assigned to a single vertex, it is called a *loop*. When several edges map to the same pair of vertices, they are referred to as *multiple edges*.

We will refer to undirected multigraphs simply as graphs. The following 3 definitions are adapted from [15] for multigraphs.

Definition 1.0.2. Two graphs $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ are said to be *isomorphic* if there is a bijection $f\colon V_1 \to V_2$ such that for all $v \in V_1 \cup V_1^2$, the number of edges in E_1 that are mapped to v is equal to the number of edges in E_2 that are mapped to $f(v)$, where $f(v) = (f(v_1), f(v_2))$ if v is a tuple (v_1, v_2).

Definition 1.0.3. An *induced subgraph* of a graph $G = (V, E)$ is a graph $H = (S, E')$, where $S \subseteq V$ is a set of vertices and $E' \subseteq E$ is a set of edges that are mapped to $S \cup S^2$.

Definition 1.0.4. A *maximum common induced subgraph* between graphs G_1 and G_2 is a graph $G_3 = (V_3, E_3)$ such that G_3 is isomorphic to induced subgraphs of both G_1 and G_2 with $|V_3|$ maximised.

In this paper we will be dealing with the maximum common induced subgraph problem defined for undirected multigraphs, even though most of the benchmark instances do not have multiple edges.

Chapter 2

Algorithms

The clique encoding [11] solves the maximum common subgraph problem by creating a new (association) graph and transforming the problem into an instance of maximum clique, which is then solved by a sequential version of the maximum clique solver by McCreesh and Prosser [12], which is a branch and bound algorithm that uses bitsets and greedy colouring. Colouring is used to provide a quick upper bound: if a subgraph can be coloured with k colours, then it cannot have a clique of size more than k.

$k\downarrow$ algorithm [6] starts by trying to solve the subgraph isomorphism problem, i.e. finding the pattern graph in the target graph. If that fails, it allows a single vertex of the pattern graph to not match any of the target graph vertices and tries again, allowing smaller and smaller pattern graphs until it finds a solution. The number of vertices of the pattern graph that are allowed this additional freedom is represented by k. More specifically, the algorithm creates a domain for each pattern graph vertex, which initially includes all vertices of the target graph and k wildcards. The domains are filtered with various propagation techniques. Then the search begins with a smallest domain (not counting wildcards), a value is chosen, and domains are filtered again to eliminate the chosen value.

MCSPLIT [14] is a branch and bound algorithm that builds its own bit string labels for vertices in both pattern and target graphs. Once it chooses to match a vertex u in graph G_1 with a vertex v in graph G_2, it iterates over all unmatched vertices in both graphs, adding a 1 to their labels if they are adjacent to u or v and 0 otherwise. That way a vertex can only be matched with vertices that have the same labels. The labels are also used in the upper bound heuristic function using the rule that if a particular label is assigned to m vertices in G_1 and n vertices in G_2, then up to $\min\{m, n\}$ pairs can be matched for that label.

MCSPLIT $\downarrow$ is a variant of MCSPLIT mentioned but not explained in the original paper [14]. It is meant to be similar to $k\downarrow$ in that it starts by trying to find a subgraph isomorphism and keeps decreasing the size of common subgraphs that it is interested in until a solution is found. Based on the source code[1], there are a few key differences between MCSPLIT $\downarrow$ and MCSPLIT:

- Instead of always looking for larger and larger common subgraphs, we have a goal size and exit early if a common subgraph of that size is found.
- The goal size is decreased if the search finishes without a solution.
- Having a big goal size allows the heuristic to be more selective and prune more of the search tree branches.

[1] https://github.com/jamestrimble/ijcai2017-partitioning-common-subgraph/blob/master/code/james-cpp/mcsp.c

Chapter 3

Problem Instances

In order to determine which algorithm should be used for which problem instance, we run all algorithms on two databases that contain a large variety of graphs differing in size, various characteristics, and the way they were generated.

The McSplit paper [14] used the same datasets to compare these (and a few constraint programming) algorithms and found McSplit to win with unlabelled graphs, the clique encoding to win with labelled graphs, and McSplit $\downarrow$ to win with the `largerGraphs` dataset. However, in some cases the difference in performance between McSplit and the clique encoding or between McSplit $\downarrow$ and $k\downarrow$ was very small.

The algorithms were compiled with gcc 6.3.0 and run on Intel Xeon E5-2697A v4 (2.60 GHz) processors with 512 GB of memory and a 1000 s time limit. A Makefile was created to run multiple experiments in parallel with, e.g., `make -j 64`, which generates pairs of graph filenames for all datasets, runs the selected algorithms with various command line arguments, redirects their output to files that are later parsed using `sed` and regular expressions into the CSV format. For each algorithm, we keep the full names of pattern and target graphs, the number of vertices in the returned maximum common subgraph, running time as reported by the algorithms themselves, and the number of explored nodes in the search tree. Entries with running time greater than or equal to the timeout value are considered to have timed out. The aforementioned node counts are collected but not currently used. Afterwards, the answers of different algorithms are checked for equality (for algorithms that did not time out).

The clique algorithm requires $O(n^2m^2)$ memory for a pair of graphs with n amd m vertices [6, 11]. To avoid segmentation faults, its virtual memory usage was limited to 7 GB with `ulimit -v` and the unlabelled instances (which contain much larger graphs) were restricted to $m \times n < 30{,}000$.

$k\downarrow$ was further modified to accept graphs with vertex labels by adding an additional constraint for matching labels on line 8 of the `klessSubgraphIsomorphism` function [6].

3.1 Labelled graphs

All of the labelled graphs are taken from the ARG Database [5, 16], which is a large collection of graphs for benchmarking various graph-matching algorithms. The graphs are generated using several algorithms:

- randomly generated,
- 2D, 3D, and 4D meshes,

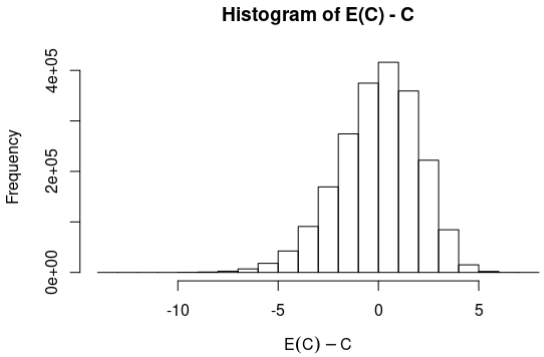

Figure 3.1: Histogram of the difference between the expected number of vertices assigned each label and the actual number (for all labelled graphs)

- and bounded valence graphs.

Furthermore, each algorithm is executed with several (3–5) different parameter values. The database includes 81400 pairs of labelled graphs. Their unlabelled versions are used as well.

3.1.1 Characteristics of Graph Labelling

For the purposes of this paper, we look at two types of labelled graphs: those that have their vertices labelled and those that have both vertices and edges labelled. We define them as follows (the definitions are loosely inspired by [1]):

Definition 3.1.1. A graph $G = (V, E)$ is a *(vertex) labelled graph* if it has an associated vertex labelling function $\mu\colon V \to \{0, \ldots, N-1\}$ for some $N \in \{2, \ldots, |V|\}$.

Definition 3.1.2. A graph $G = (V, E)$ is a *fully labelled graph* if it is a vertex labelled graph and it has an associated edge labelling function $\zeta\colon E \to \{0, \ldots, M-1\}$ for some $M \in \{2, \ldots, |E|\}$.

Specifically, note that:

- If a graph is labelled, then all its vertices (and possibly edges) are assigned a label.
- We are only considering finite sets of labels, represented by non-negative integers.

Now we need a way to choose N and M. For that we formally define how labelling is implemented in the ARG database:

Definition 3.1.3. A graph $G = (V, E)$ is said to have a $p\%$ *(vertex) labelling* if

$$N = \max\left\{2^n : n \in \mathbb{N},\, 2^n < \left\lfloor \frac{p}{100\%} \times |V| \right\rfloor \right\}.$$

The default value for p is 33%.

The publications associated with the database [5, 16] say nothing about how the labels are distributed among the N values. We calculate the number of vertices that were assigned each label for each graph (represented by C) and compare those values with the numbers we would expect from a uniform distribution (represented by $E(C)$). We plot a histogram of the difference $E(C) - C$ in Figure 3.1 and observe that the difference is normally distributed around 0.

3.2 Unlabelled graphs

We also include a collection of benchmark instances for the subgraph isomorphism problem[1] (with the biochemical reactions dataset excluded since we are not dealing with directed graphs). It contains only unlabelled graphs and consists of the following sets:

images-CVIU11 Graphs generated from segmented images. 43 pattern graphs and 146 target graphs, giving a total of 6278 instances.

meshes-CVIU11 Graphs generated from meshes modelling 3D objects. 6 pattern graphs and 503 target graphs, giving a total of 3018 instances. Both `images-CVIU11` and `meshes-CVIU11` datasets are described in [2].

images-PR15 Graphs generated from segmented images [18]. 24 pattern graphs and a single target graph, giving 24 instances.

LV Graphs with various properties (connected, biconnected, triconnected, bipartite, planar, etc.). 49 graphs are paired up in all possible ways, giving $49^2 = 2401$ instances.

scalefree Scale-free networks generated using a power law distribution of degrees (100 instances).

si Bounded valence graphs, 4D meshes, and randomly generated graphs (1170 instances). This is the unlabelled part of the ARG database. `LV`, `scalefree`, and `si` datasets are described in [17, 21].

phase Random graphs generated to be close to the satisfiable-unsatisfiable phase transition (200 instances) [13].

largerGraphs Larger instances of the `LV` dataset. There are 70 graphs, giving $70^2 = 4900$ instances. The separation was made and used in [6, 9, 14].

Remark 3.2.1. This set of instances was taken from the repository[2] for the McSplit paper [14] and has some minor differences from the version on Christine Solnon's website.

Remark 3.2.2. Since $k \downarrow$ comes from the subgraph isomorphism problem background, it treats the two (pattern and target) graphs differently. Therefore, when graphs are not divided into patterns and targets, we run the algorithms with both orders ((G_1, G_2) and (G_2, G_1)).

[1] `http://liris.cnrs.fr/csolnon/SIP.html`
[2] `https://github.com/jamestrimble/ijcai2017-partitioning-common-subgraph`

Chapter 4

Features

The initial set of features is based on the algorithm selection paper for the subgraph isomorphism problem [9] and consists of the following:

1. number of vertices,
2. number of edges,
3. mean degree,
4. maximum degree,
5. density,
6. mean distance between all pairs of vertices,
7. maximum distance between all pairs of vertices,
8. standard deviation of degrees,
9. number of loops,
10. proportion of all vertex pairs that have a distance of at least 2, 3, and 4,
11. whether the graph is connected.

We exclude feature extraction running time as a viable feature by itself since it would not provide any insight into what properties of the graph affect which algorithm is likely to achieve the best performance. Since $k\downarrow$ and McSplit$\downarrow$ both start by looking for (complete) subgraph isomorphisms, they are likely to outperform other algorithms when both graphs are very similar and the maximum common subgraph has (almost) as many vertices as the smaller of the two graphs. Thus, for each feature f in features 1–7 (excluding the rest to avoid division by 0), we also add a feature for the ratio $\frac{f(G_p)}{f(G_t)}$, where G_p and G_t are the pattern and target graphs, respectively.

We analyse three different types of labelling and treat them as separate problems:

- no labels,
- vertex labels,
- vertex and edge labels.

For the last two types, we add a feature corresponding to p defined in Definition 3.1.3 and collect data for the following values of p: 50%, 33%, 25%, 20%, 15%, 10%, 5%. The values correspond to having about 2, 3, 4, 5, 10, and 20 vertices/edges with the same label on average, respectively.

4.1 Distributions of Features

In this section we plot and discuss how the selected features are distributed in different datasets. Most of the data is plotted, except for the number of loops of the labelled dataset and connectedness of both labelled and unlabelled data. For connectedness, 99.81% of the labelled graphs are connected, compared to 93.19% of the unlabelled graphs. As both numbers are quite high, they may not be ideal for establishing if connectedness is a significant factor in determining which algorithm performs the best, however, the numbers seem quite representative of real data, where connected graphs are a lot more common. In fact, applications in chemistry are often only interested in connected graphs [4].

With the labelled graphs, all graphs are assigned into pairs (A and B) with no significant differences between them. Thus all of the statistical information is provided only for pattern graphs and is plotted in Figures 4.1 and 4.2. As the number of loops varies between two values, it is not plotted: 0.98% of the graphs have a single loop, other graphs have no loops. Other than the plot for number of vertices, which is manually chosen by the creators of the database, all the distributions in Figure 4.1 are centered around lower values, with some instances providing significantly higher values. More importantly, we have some graphs that are quite dense and some graphs with higher mean distance values.

For unlabelled graphs, we will only consider graphs that are part of a pair of graphs solved by at least one algorithm. Since most of the data is heavily skewed, Figures 4.3 and 4.4 show the distributions as density plots. Because of this choice of which graphs to include, all of the easy instances are solved and thus end up in the sample, while only some of the harder instances are solved by at least one algorithm. Harder instances typically have more vertices, which means they are also capable of higher values for many other features, hence all of the density plots in Figure 4.3 are right skewed. The same applies to the number of loops in Figure 4.4: almost all graphs have a small number of them, while some (presumably larger) graphs have significantly more. The same story is represented in the heatmap in Figure 4.5, where the average values are coloured grey, largest values black, and smallest values white. The rows that look almost completely white represent features that have several significantly higher values.

To sum up, even though the distributions are far from normal, most of them still provide a good range of different values. There is one important difference between the plots of proportions of vertex pairs with distance $\geq k$ for $k = 2, 3, 4$ for labelled and unlabelled datasets (in Figures 4.2 and 4.4, respectively). With labelled graphs, the mean value keeps shifting to the left, while with unlabelled graphs, the plot for $k = 4$ still has its highest peak around 0.9, which means adding features for $k \geq 5$ could be valuable.

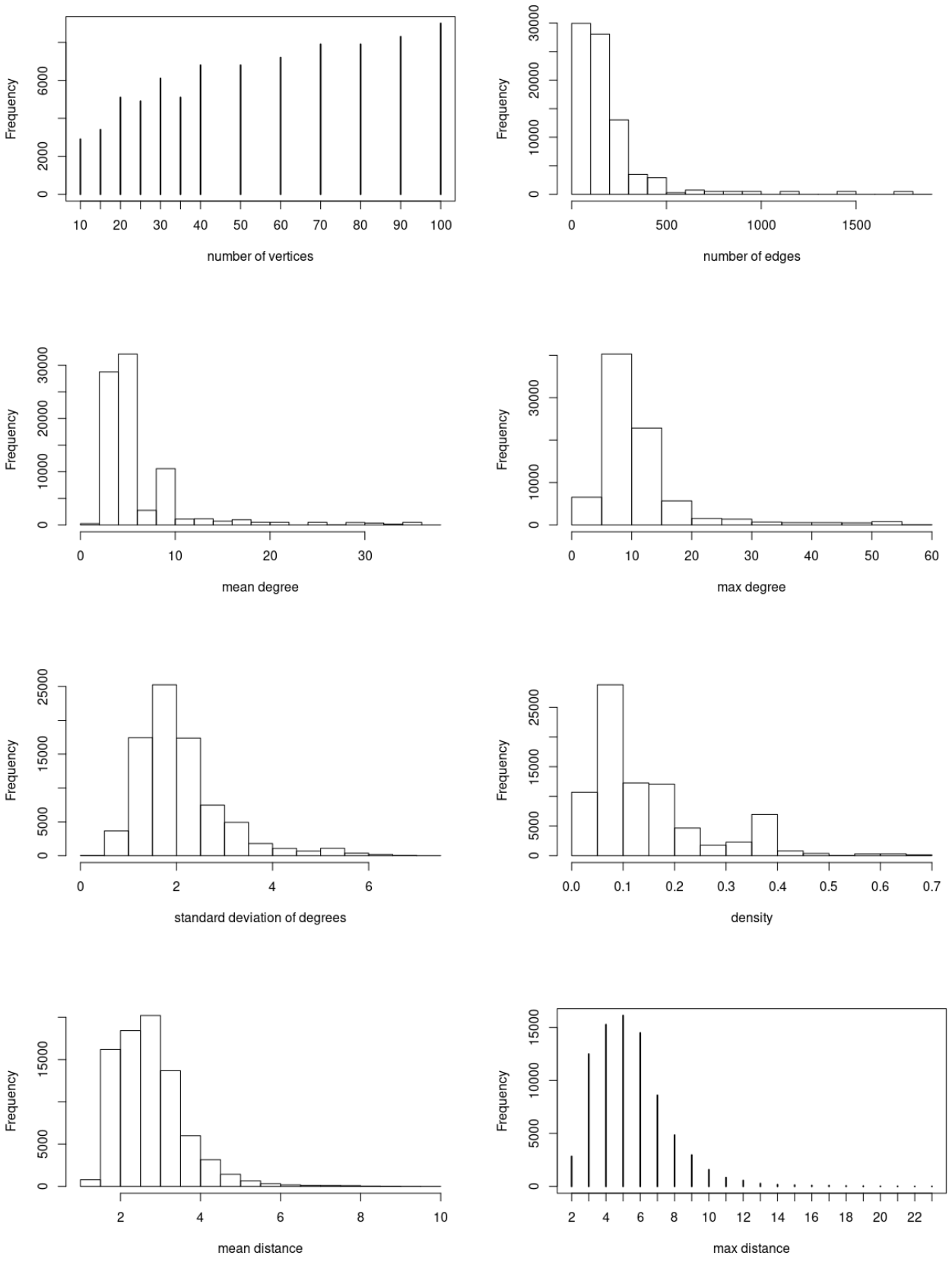

Figure 4.1: Plots of how various features are distributed for the labelled graphs

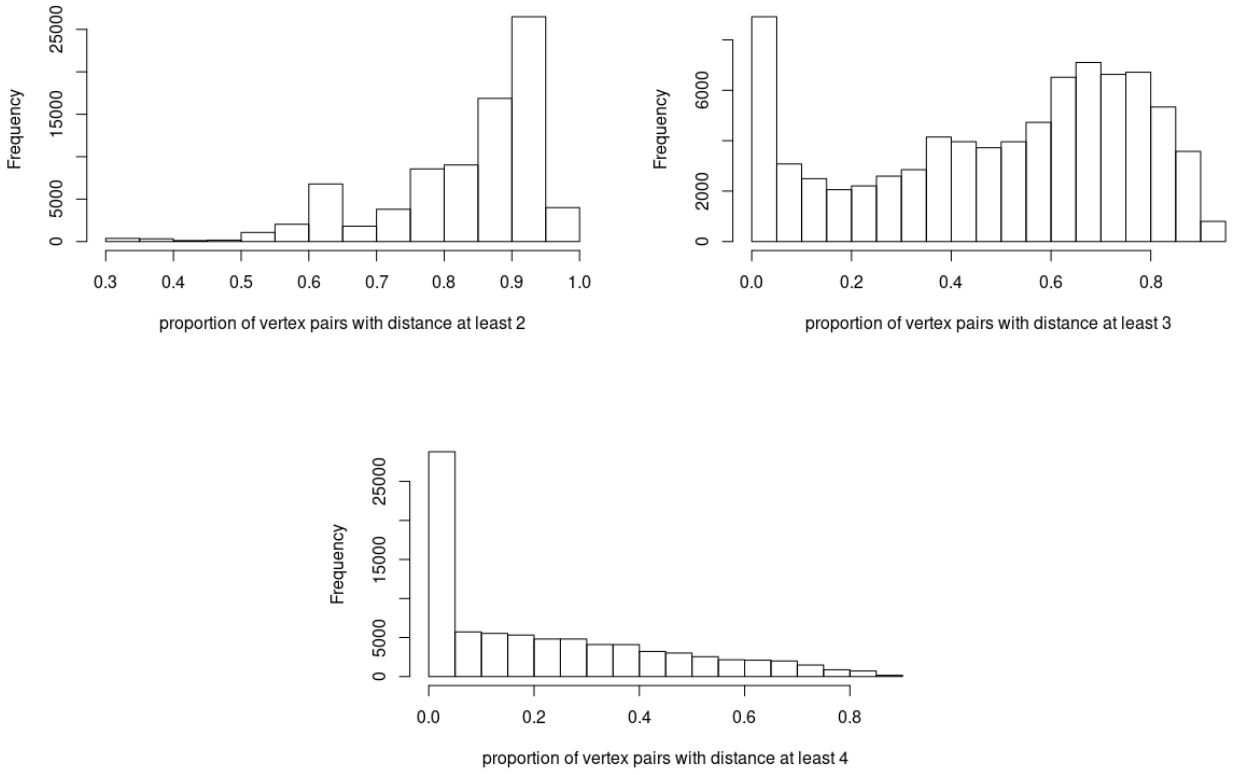

Figure 4.2: Plots showing typical distances between pairs of vertices for the labelled graphs

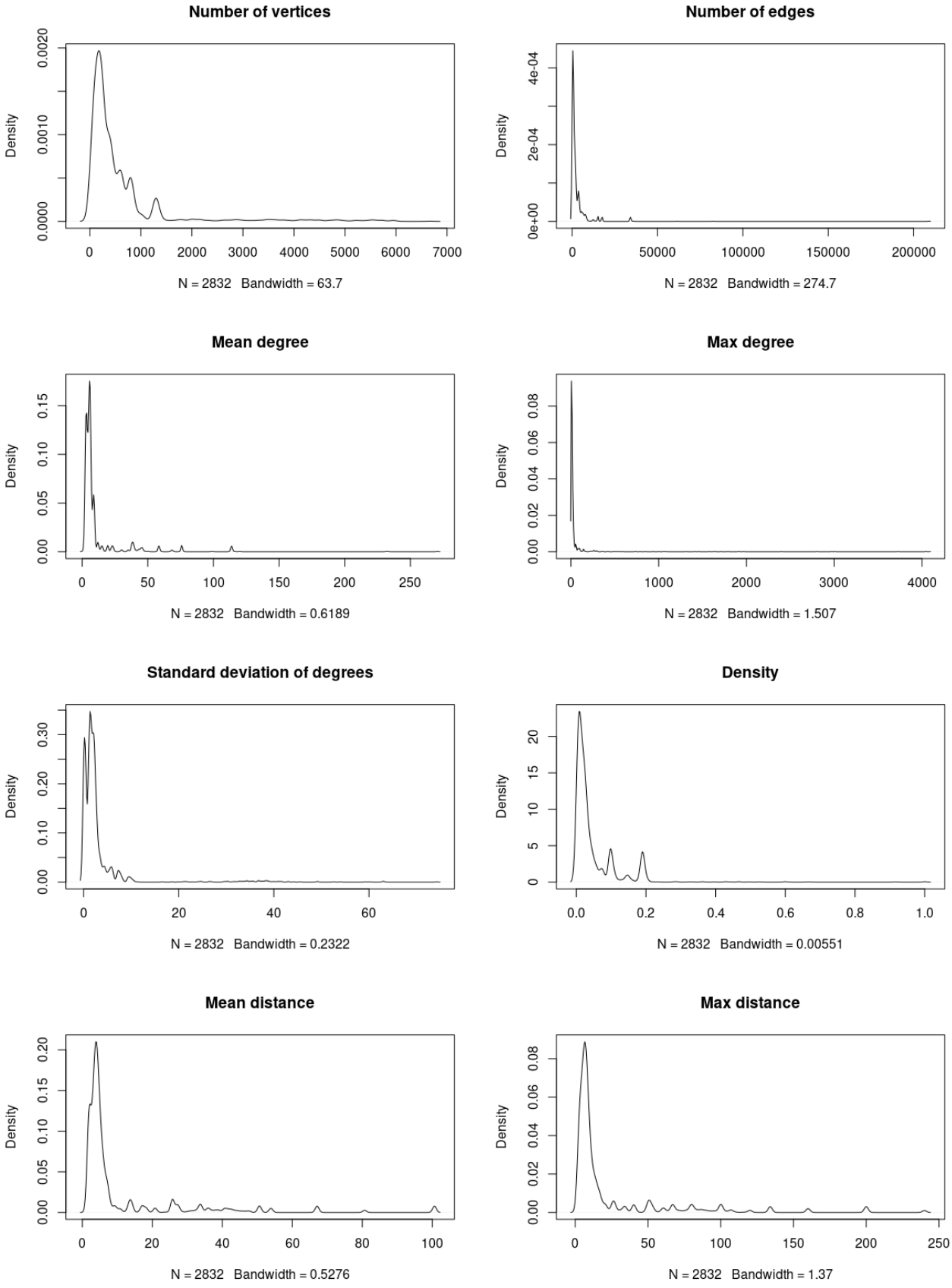

Figure 4.3: Plots of how various features are distributed for the unlabelled graphs

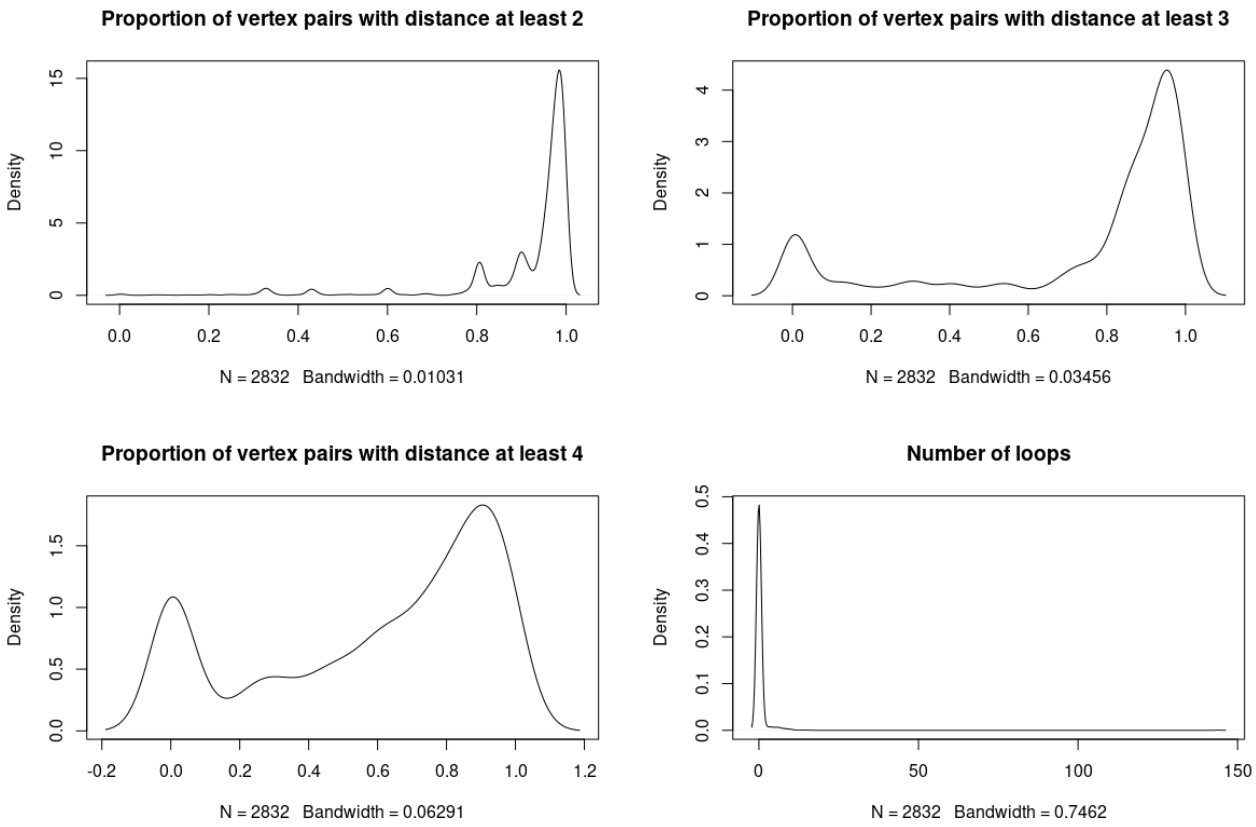

Figure 4.4: Plots showing typical distances between pairs of vertices and the number of loops for the unlabelled graphs

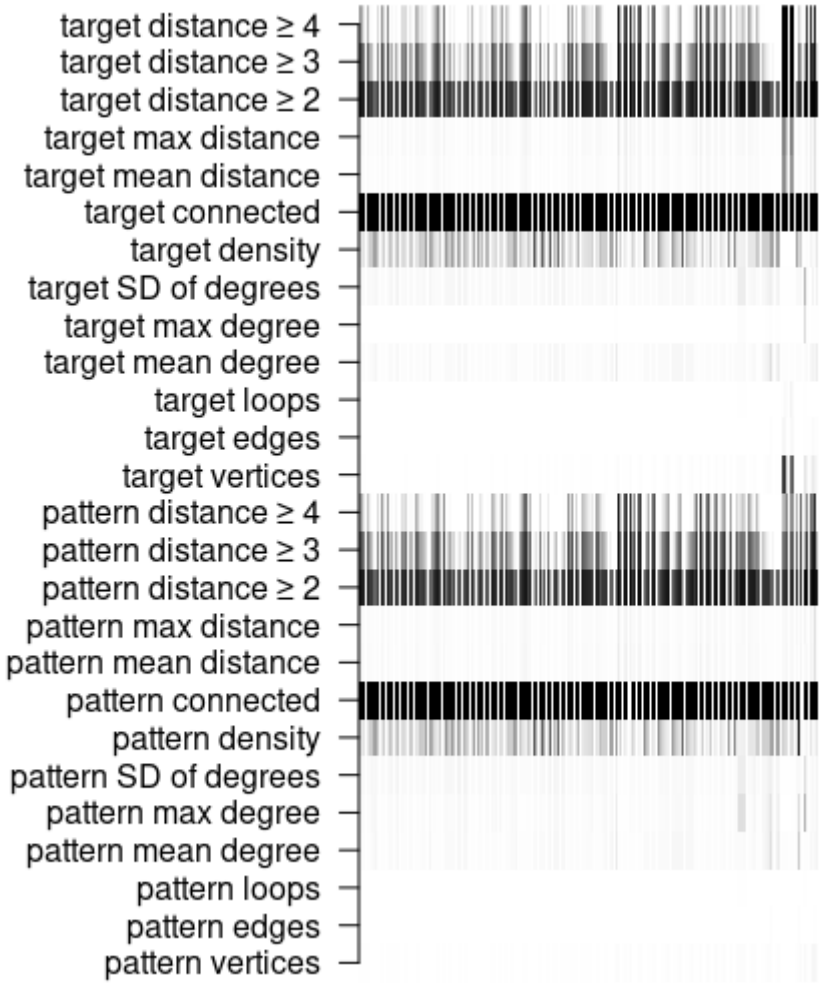

Figure 4.5: A heatmap for normalised features with black denoting the maximum value and white denoting the minimum for each feature

Chapter 5

Machine Learning

After running the algorithms on all of the data for different types of labelling and p values, a machine learning (ML) algorithm can be trained to predict which algorithm should be chosen for each pair of graphs. For that we are using an R package called LLAMA [8], which was created for exactly this purpose. For each pair of graphs it takes:

- A list of features. We treat the features of pattern and target graphs separately, giving more than 20 features per problem instance.
- A list of performance measures for all algorithms, i.e., the values that we are trying to optimise. In this case (as in most cases), this corresponds to running time. The values are capped at the timeout value (1,000,000 ms). Furthermore, instances that were not run on the clique algorithm are also set to the timeout value. Finally, we filter out instances where all of the algorithms timed out.
- A list of boolean values, denoting whether each algorithm successfully finished or not. Timeouts, the clique algorithm running out of memory, and instances that were not run with the clique algorithm because of their size are all marked as false.
- A dataframe, measuring the running time taken to compute each feature for each problem instance. Alternatively, a single number for the approximate time taken to compute all features for any instance. This parameter is taken into account when comparing the algorithm portfolio against specific algorithms. As the main goal of this work is to gain insight about how the algorithms compare rather than to prove an algorithm portfolio as a superior approach, this parameter is not used.

After constructing the required dataframes as described above, the data needs to be split into training and test sets. We use a technique called 10-fold *cross-validation*, which splits the data into 10 parts [19]. 9/10$^{\text{ths}}$ of the data is used to train the ML algorithm, while the remaining 1/10$^{\text{th}}$ is used to evaluate how good the trained model is. This process of training and evaluation is repeated 10 times, letting each of the 10 parts be used for evaluation exactly once. The goodness-of-fit criteria are then averaged out between the 10 runs.

The 10 folds could, of course, be chosen completely randomly. However, research suggests that stratified cross-validation typically outperforms random-sampling-based cross-validation and results in a better model [7]. Suppose we have a dataset of N elements. *Stratified sampling* partitions it into a number of subpopulations $s_1, \ldots, s_n$ with $n_1, \ldots, n_N$ elements, respectively (typically based on the value of some feature or collection of features). It then draws from each subpopulation independently, ensuring that approximately n_i/N of the sample comes from subpopulation s_i for $i = 1, \ldots, n$ [10]. In this case the data is partitioned into four groups based on which algorithm performed the best.

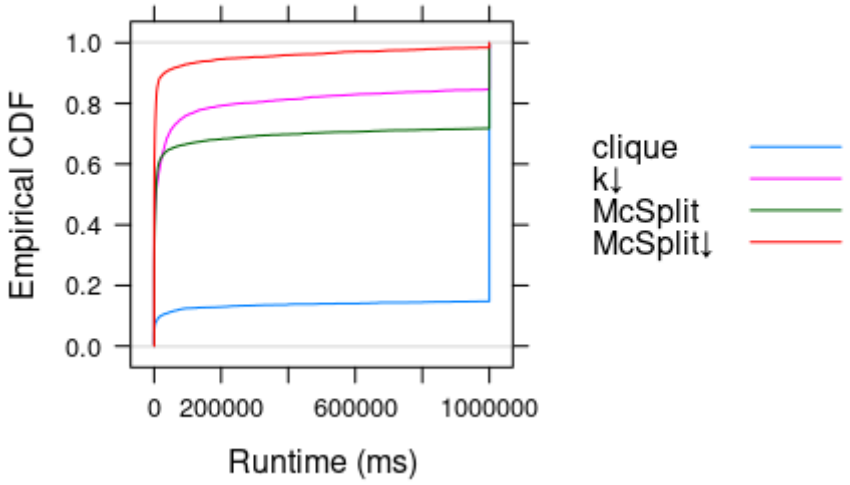

Figure 5.1: Comparison of the runtimes of algorithms for the unlabelled dataset

The cross-validation folds are then passed to the ML algorithm. We are using a classifier called random forest as it is recommended in the manual [8] and successfully used in a similar study [9].

As we are trying to use hundreds of megabytes of data, the R code was optimised to reduce memory consumption by removing temporary variables as soon as they are no longer needed and parallelised with the help of the `parallelMap` package.

5.1 Unlabelled Graphs

In order to compare the runtimes of the algorithms, we use empirical cumulative distribution function (ECDF) plots [20]: for each unit of time on the x axis, the y value represents what part of the problem instances was solved in that amount of time or less. Figures (TODO: add the MCS figure) and 5.1 show the ECDF plots for unlabelled instances described in Sections 3.1 and 3.2, respectively. We can check that the ordering of the algorithms in these plots is the same as in Figures 3.(a) and 4 in the McSplit paper [14]. Next, Figure 5.2 provides the same plot for all of the unlabelled data and Figure 5.3 compares the runtimes of the algorithms on a per-instance basis. Out of $45{,}465$ instances that were solved by at least one algorithm, the clique encoding won 85 times, $k\downarrow$ won 81 times, McSplit won $15{,}017$ times, and McSplit $\downarrow$ won $21{,}484$ times. In the remaining 8798 cases 2 or more algorithms performed equally well (usually with a runtime of less than 10 ms). Note that even though $k\downarrow$ performs considerably better than the clique encoding according to Figures 5.2 and 5.3, it is the best algorithm in only 81 instances, less than the clique encoding. Given this information, we would expect the ML algorithm to suggest using McSplit and McSplit $\downarrow$ most of the time, only suggesting $k\downarrow$ and the clique encoding in very specific situations.

5.2 Vertex-Labelled Graphs

5.3 Vertex- and Edge-Labelled Graphs

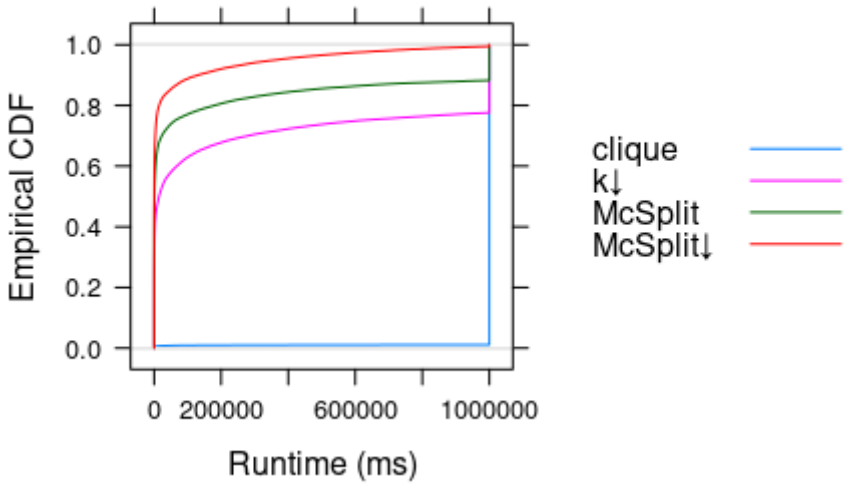

Figure 5.2: Comparison of the runtimes of algorithms for all of the unlabelled data

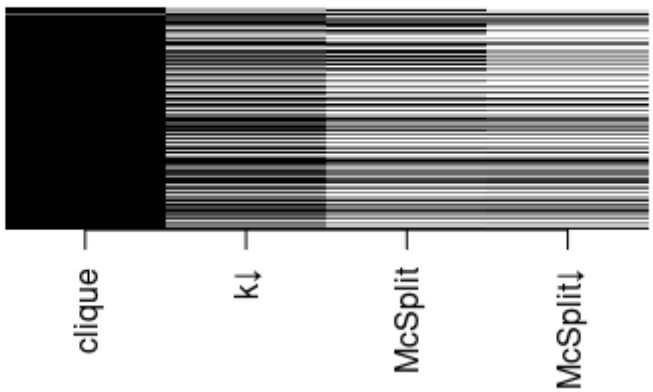

Figure 5.3: A heatmap of $\log_{10}$ runtimes: light colours for low running times and black for timing out

Bibliography

[1] Zeina Abu-Aisheh. *Anytime and Distributed Approaches for Graph Matching*. PhD thesis, Université François-Rabelais de Tours, 2016.

[2] Guillaume Damiand, Christine Solnon, Colin de la Higuera, Jean-Christophe Janodet, and Émilie Samuel. Polynomial algorithms for subisomorphism of nd open combinatorial maps. *Computer Vision and Image Understanding*, 115(7):996–1010, 2011.

[3] Reinhard Diestel. *Graph Theory, 5th Edition*, volume 173 of *Graduate texts in mathematics*. Springer-Verlag, 2016.

[4] Hans-Christian Ehrlich and Matthias Rarey. Maximum common subgraph isomorphism algorithms and their applications in molecular science: a review. *Wiley Interdisciplinary Reviews: Computational Molecular Science*, 1(1):68–79, 2011.

[5] Pasquale Foggia, Carlo Sansone, and Mario Vento. A database of graphs for isomorphism and sub-graph isomorphism benchmarking. In *Proceedings of the 3rd IAPR TC-15 International Workshop on Graph-based Representations*, 2001.

[6] Ruth Hoffmann, Ciaran McCreesh, and Craig Reilly. Between subgraph isomorphism and maximum common subgraph. In Satinder P. Singh and Shaul Markovitch, editors, *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence, February 4-9, 2017, San Francisco, California, USA.*, pages 3907–3914. AAAI Press, 2017.

[7] Ron Kohavi. A study of cross-validation and bootstrap for accuracy estimation and model selection. In *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence, IJCAI 95, Montréal Québec, Canada, August 20-25 1995, 2 Volumes*, pages 1137–1145. Morgan Kaufmann, 1995.

[8] Lars Kotthoff. LLAMA: leveraging learning to automatically manage algorithms. Technical Report arXiv:1306.1031, arXiv, June 2013.

[9] Lars Kotthoff, Ciaran McCreesh, and Christine Solnon. Portfolios of subgraph isomorphism algorithms. In Paola Festa, Meinolf Sellmann, and Joaquin Vanschoren, editors, *Learning and Intelligent Optimization - 10th International Conference, LION 10, Ischia, Italy, May 29 - June 1, 2016, Revised Selected Papers*, volume 10079 of *Lecture Notes in Computer Science*, pages 107–122. Springer, 2016.

[10] Sharon L. Lohr. *Sampling: Design and Analysis*. Advanced (Cengage Learning). Cengage Learning, 2009.

[11] Ciaran McCreesh, Samba Ndojh Ndiaye, Patrick Prosser, and Christine Solnon. Clique and constraint models for maximum common (connected) subgraph problems. In Michel Rueher, editor, *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*, pages 350–368. Springer, 2016.

[12] Ciaran McCreesh and Patrick Prosser. The shape of the search tree for the maximum clique problem and the implications for parallel branch and bound. *TOPC*, 2(1):8:1–8:27, 2015.

[13] Ciaran McCreesh, Patrick Prosser, and James Trimble. Heuristics and really hard instances for subgraph isomorphism problems. In Subbarao Kambhampati, editor, *Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence, IJCAI 2016, New York, NY, USA, 9-15 July 2016*, pages 631–638. IJCAI/AAAI Press, 2016.

[14] Ciaran McCreesh, Patrick Prosser, and James Trimble. A partitioning algorithm for maximum common subgraph problems. In Carles Sierra, editor, *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI 2017, Melbourne, Australia, August 19-25, 2017*, pages 712–719. ijcai.org, 2017.

[15] John W. Raymond and Peter Willett. Maximum common subgraph isomorphism algorithms for the matching of chemical structures. *Journal of Computer-Aided Molecular Design*, 16(7):521–533, 2002.

[16] Massimo De Santo, Pasquale Foggia, Carlo Sansone, and Mario Vento. A large database of graphs and its use for benchmarking graph isomorphism algorithms. *Pattern Recognition Letters*, 24(8):1067–1079, 2003.

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

[18] Christine Solnon, Guillaume Damiand, Colin de la Higuera, and Jean-Christophe Janodet. On the complexity of submap isomorphism and maximum common submap problems. *Pattern Recognition*, 48(2):302–316, 2015.

[19] Andrew R. Webb. *Statistical Pattern Recognition, 2nd Edition*. John Wiley & Sons, October 2002.

[20] Martin B. Wilk and Ramanathan Gnanadesikan. Probability plotting methods for the analysis of data. *Biometrika*, 55(1):1–17, 1968.

[21] Stéphane Zampelli, Yves Deville, and Christine Solnon. Solving subgraph isomorphism problems with constraint programming. *Constraints*, 15(3):327–353, 2010.