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 — 29th 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 [5]. 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 [21] 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 $(v_1, v_2) \in V_1^2$.

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. The *maximum common (induced) subgraph problem* is the problem of finding a maximum common subgraph between two given graphs $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$, usually expressed as a bijection between two subsets of vertices $U_1 \subseteq V_1$ and $U_2 \subseteq V_2$.

Similarly to Definition 1.0.3, we define a more general notion of a subgraph.

Definition 1.0.5. A graph $G_2 = (V_2, E_2)$ is a *subgraph* of graph $G_1 = (V_1, E_1)$ if $V_2 \subseteq V_1$ and $E_2 \subseteq E_1$ [5].

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. We will also refer to a closely related graph problem, the subgraph isomorphism problem.

Definition 1.0.6. Given two (finite) graphs G_1 and G_2, the *subgraph isomorphism problem* is the decision problem of determining whether G_1 is isomorphic to a subgraph of G_2 [3].

Chapter 2

The Algorithms

The clique encoding [16] 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 [17], 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 [8] 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 [19] 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 [19]. 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

We use two graph databases that contain a large variety of graphs differing in size, various characteristics, and the way they were generated. The McSplit paper [19] used the same data to compare these (and a few constraint programming) algorithms and found McSplit to win with unlabelled graphs described in Section 3.1 of this paper, the clique encoding to win with labelled graphs, and McSplit $\downarrow$ to win with unlabelled graphs from Section 3.2. However, in some cases the difference in performance between McSplit and the clique encoding or between McSplit $\downarrow$ and $k \downarrow$ was very small.

3.1 Labelled Graphs

All of the labelled graphs are taken from the ARG Database [7, 22], 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,
- 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 $M \in \{2, \ldots, |E|\}$.

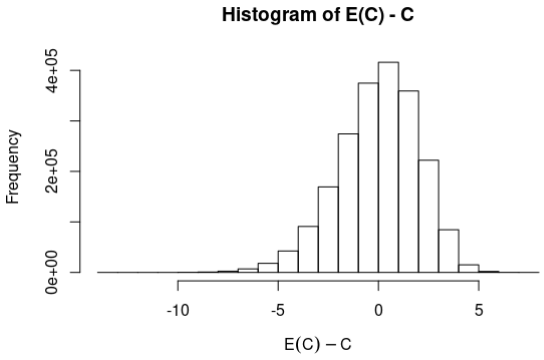


Figure 3.1: Histogram of the difference between the expected number of vertices assigned each label and the actual number (for all labelled graphs)

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 [7, 22] 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.

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

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 [4].

images-PR15 Graphs generated from segmented images [24]. 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 [23, 27].

phase Random graphs generated to be close to the satisfiable-unsatisfiable phase transition (200 instances) [18].

largerGraphs Larger instances of the `LV` dataset. There are 70 graphs, giving $70^2 = 4900$ instances. The separation was made and used in [8, 13, 19].

Remark 3.2.1. This set of instances was taken from the repository[2] for the McSplit paper [19] 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)).

[2] `https://github.com/jamestrimble/ijcai2017-partitioning-common-subgraph`

Chapter 4

Generating Data

A machine learning (ML) model requires data to learn from. We are using an R package called LLAMA [11], which helps to train and evaluate ML models in order to compare algorithms and was used to create algorithm portfolios for the travelling salesperson problem [12] and the subgraph isomorphism problem [13]. First, we run each algorithm on all pairs of pattern-target graphs and record the running times (described in Section 4.1). Then, we adapt a graph feature extractor program used in [13] to handle the binary format of the ARG Database [7, 22], run it on all graphs, and record the features in a way described in Section 4.2.

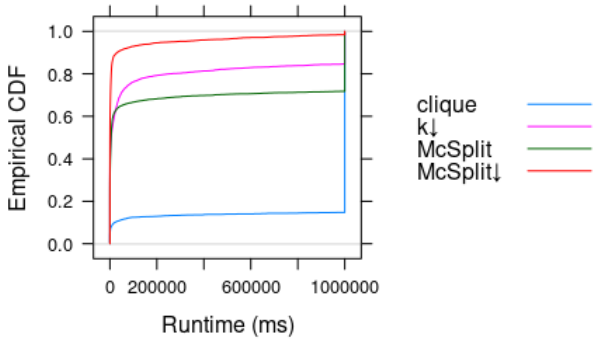
4.1 Running Time of Algorithms

The algorithms were compiled with gcc 6.3.0 and run on Intel Xeon E5-2697A v4 (2.60 GHz) processors with 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).

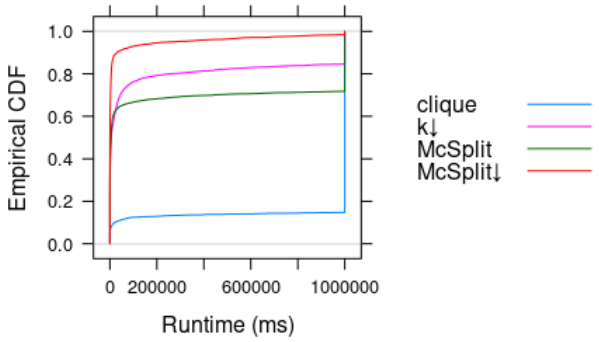
The clique algorithm requires $O(n^2m^2)$ memory for a pair of graphs with n amd m vertices [8, 16]. To avoid segmentation faults, its virtual memory usage was limited to 7 GB with `ulimit -v` and the unlabelled instances (which contain much larger graphs) were restricted to $m \times n < 30{,}000$.

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

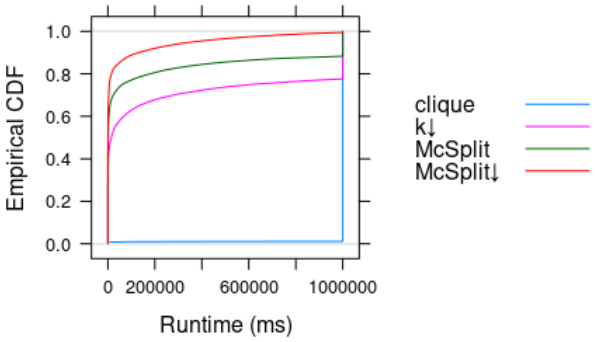
In the rest of this section we explore and compare how the algorithms performed on the three different subproblems under consideration, those of having no labels, vertex labels, and both vertex and edge labels. We introduce *empirical cumulative distribution function (ECDF)* plots [26]: for each unit of time on the x axis, the value on the y axis represents what part of the problem instances was solved in that amount of time or less.



(a) Data from Section 3.1, the ARG Database



(b) Data from Section 3.2



(c) All unlabelled data

Figure 4.1: Comparison of the runtimes of algorithms on unlabelled data

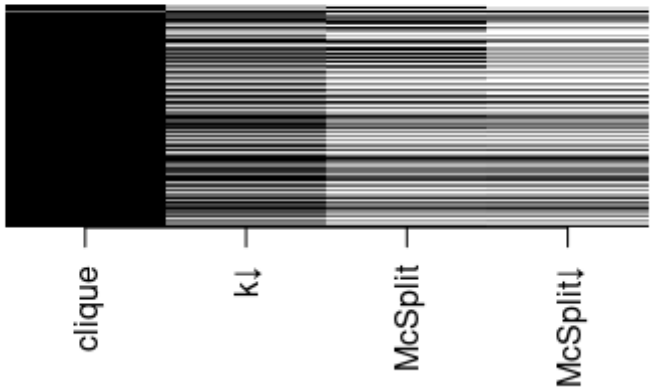


Figure 4.2: A heatmap of $\log_{10}$ runtimes: light colours for low running times and black for timing out

Dataset	clique	$k\downarrow$	McSplit	McSplit $\downarrow$
images-CVIU11	0	32	78	1080
images-PR15	0	0	0	24
largerGraphs	0	14	6	143
LV	85	12	256	206
meshes-CVIU11	0	13	0	23
phase	0	0	0	0
scalefree	0	0	0	80
si	0	10	11	1044
ARG Database	0	0	14666	18884
Total	85	81	15017	21484

Table 4.1: The number of times each algorithm was the best, for each dataset

4.1.1 Unlabelled Graphs

We plot the ECDF plots for unlablled graphs in both databases in Figure 4.1. We can check that the orderings of the algorithms in parts (a) and (b) of Figure 4.1 are the same as in Figures 3a and 4 in the McSplit paper [19]. We also provide a heatmap in Figure 4.2 to compare the runtimes of the algorithms on a per-instance basis.

Remark 4.1.1. Not every problem instance gets a line of pixels on the heatmap. Therefore, the column for the clique encoding may look darker than it actually is. Furthermore, there are problem instances where $k\downarrow$ performs better than the other algorithms, even though it may not be apparent from the heatmap.

Table 4.1 shows that most of the datasets have multiple algorithms that managed to outperform the others for some problem instances. Thus, looking at the differences between different datasets will not be enough to predict the best algorithm.

Remark 4.1.2. More specifically, Table 4.1 shows the numbers of times that each algorithm's runtime was lower than the runtimes of other algorithms. Therefore, if 2 or more lowest runtimes are equal (as can often happen with single-digit runtimes), neither algorithm is marked as winning in the table.

Remark 4.1.3. Note that even though $k\downarrow$ performs considerably better than the clique encoding according to Figures 4.1 and 4.2, it is counted as 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.

4.1.2 Vertex-Labelled Graphs

4.1.3 Vertex- and Edge-Labelled Graphs

4.2 Graph Features

The initial set of features is based on the algorithm selection paper for the subgraph isomorphism problem [13] 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.

Definition 4.2.1. For a graph G with n vertices and m edges, the *(edge) density* is defined to be the proportion of potential edges that G actually has [5]. The standard formula used for simple graphs is

$$\frac{m}{\binom{n}{2}} = \frac{2m}{n(n-1)}.$$

Even though some of our graphs contain multiple edges and loops, we stick to this formula as it was used in [13] and it does not break the ML algorithm in any way to have the theoretical possibility of density greater than 1.

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.

Remark 4.2.1. When working with both vertex and edge labels, we only consider using the same value of p for both vertices and edges. Although this may not be ideal, it reflects how the ARG Database was constructed. Furthermore, there are many ways to label a graph and it is unclear which types of labelling are worth investigating.

4.2.1 Distributions of Features

In this section we plot and discuss how the selected features are distributed in both databases. Most of the data is plotted, except for the number of loops of the ARG Database and connectedness of both databases. 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 [6].

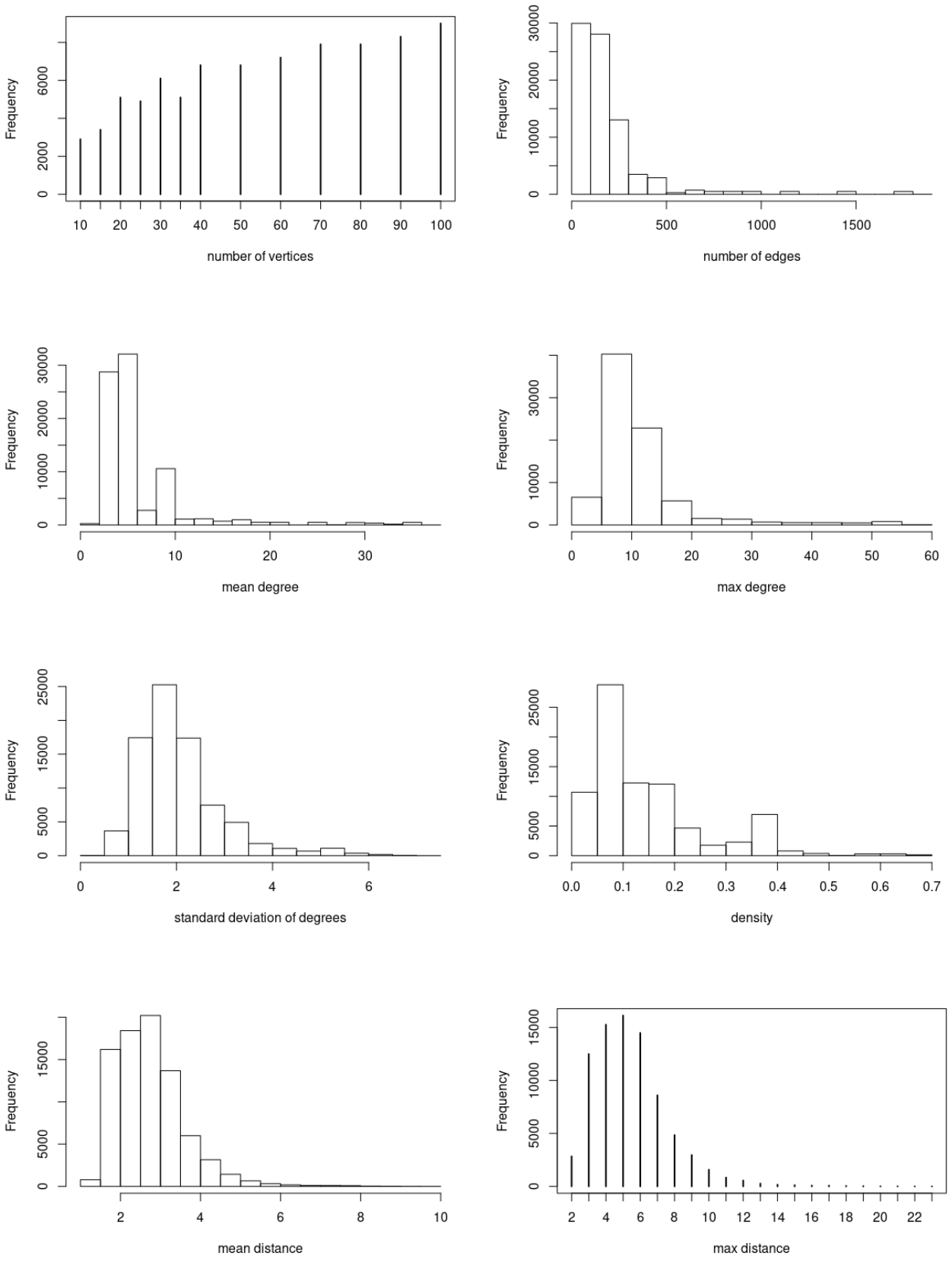


Figure 4.3: Plots of how various features are distributed for the labelled graphs

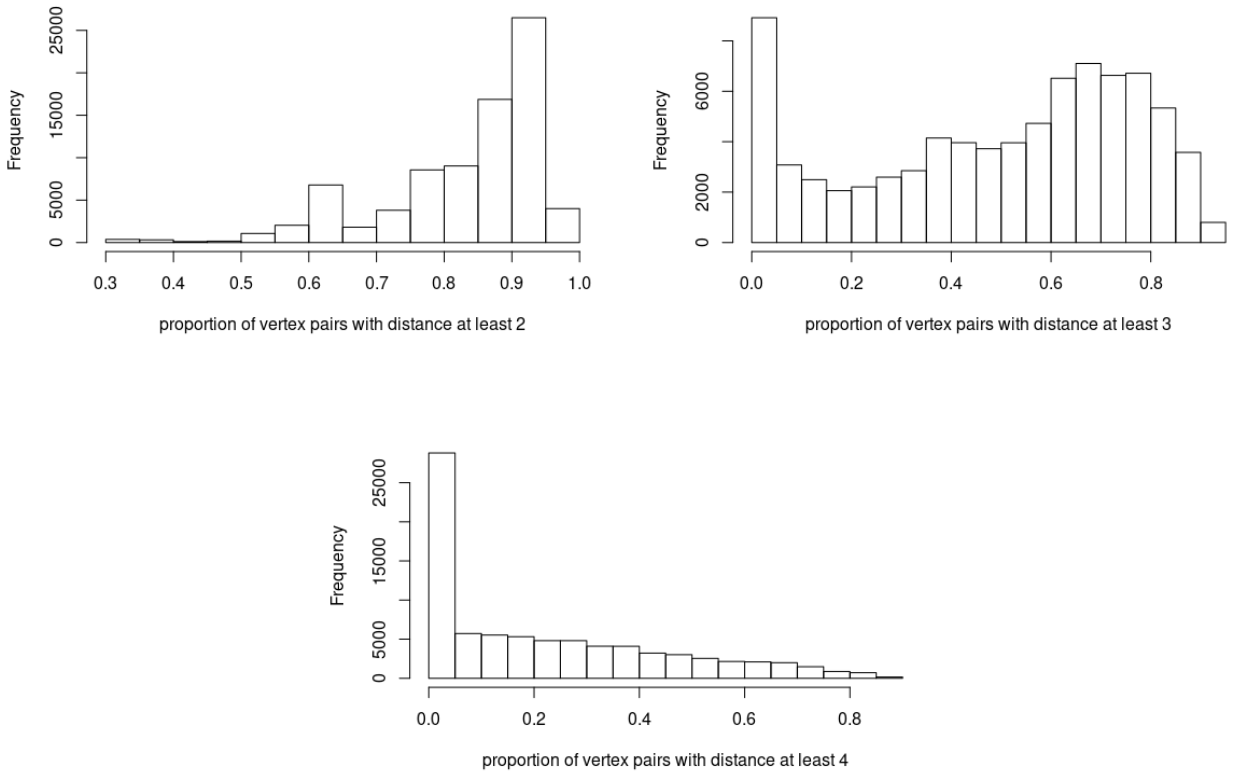


Figure 4.4: Plots showing typical distances between pairs of vertices for the labelled graphs

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 4.3 and 4.4. 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 4.3 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 4.5 and 4.6 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 4.5 are right skewed. The same applies to the number of loops in Figure 4.6: 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 4.7, 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 the graphs from Sections 3.1 and 3.2 (in Figures 4.4 and 4.6, respectively). With the ARG Database, the mean value keeps shifting to the left, while with the other database, the plot for $k = 4$ still has its highest peak around 0.9, which means adding features for $k \geq 5$ could be valuable.

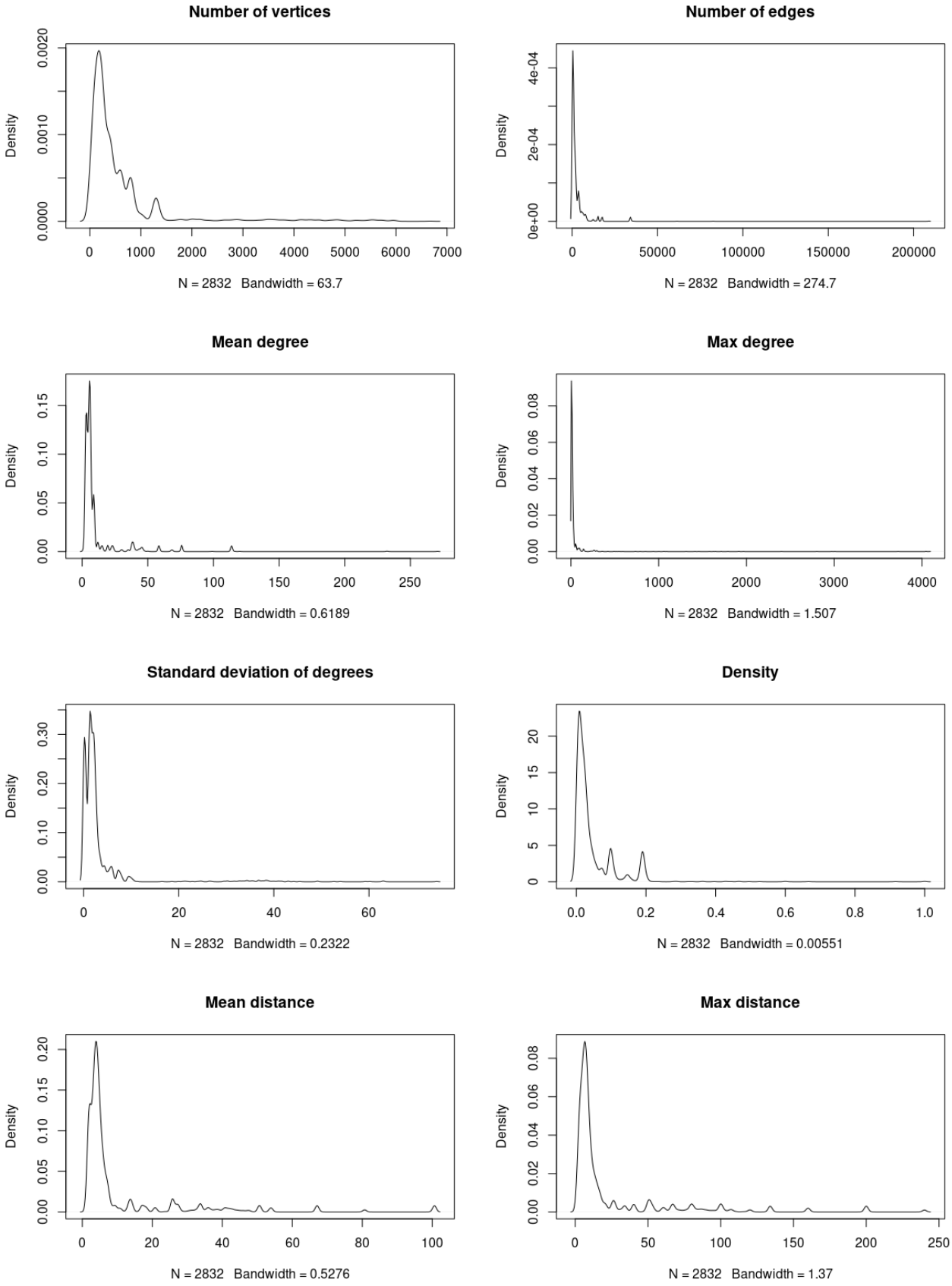


Figure 4.5: Plots of how various features are distributed for the unlabelled graphs

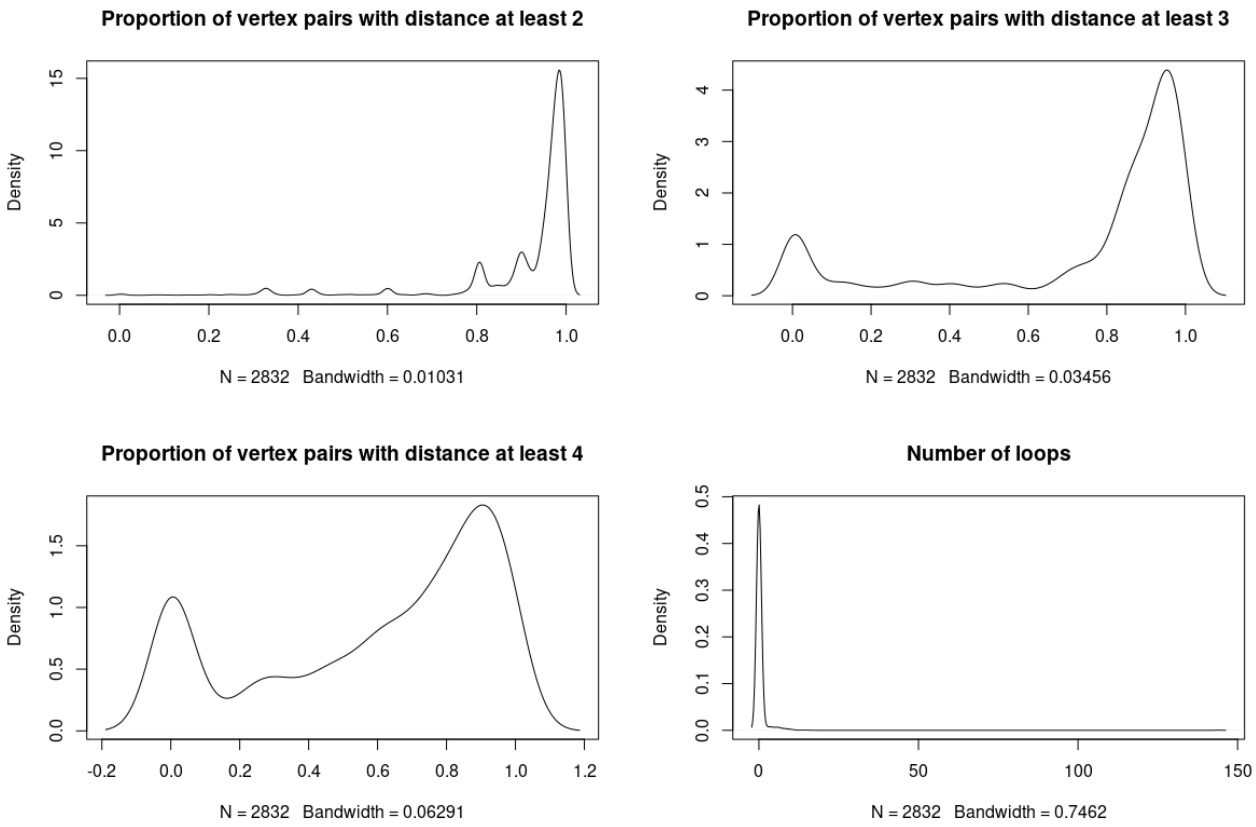


Figure 4.6: Plots showing typical distances between pairs of vertices and the number of loops for the unlabelled graphs

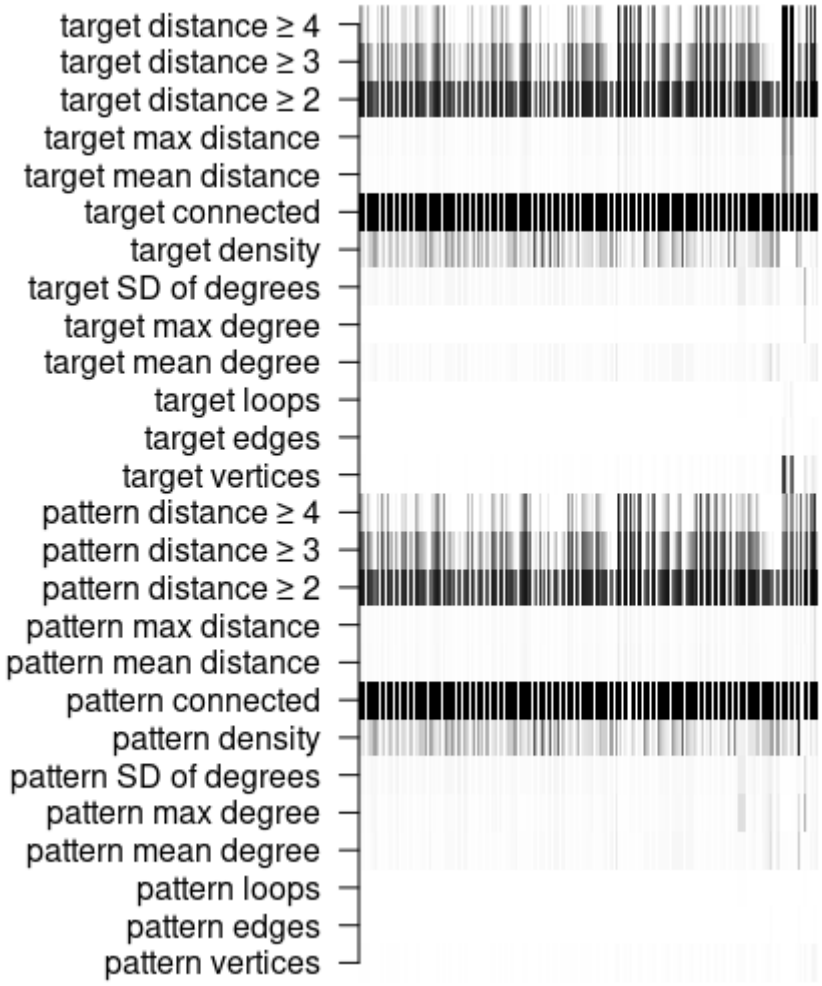


Figure 4.7: A heatmap for normalised features with black denoting the maximum value and white denoting the minimum for each feature

Chapter 5

Machine Learning

After running the algorithms on all of the data for different types of labelling and p values, an ML algorithm can be trained to predict which algorithm should be chosen for each pair of graphs. For each pair of graphs, LLAMA can take:

- 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 [25]. 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 [10]. 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 $i = 1, \ldots, n$ [15]. In this case the data is partitioned into four groups based on which algorithm performed the best.

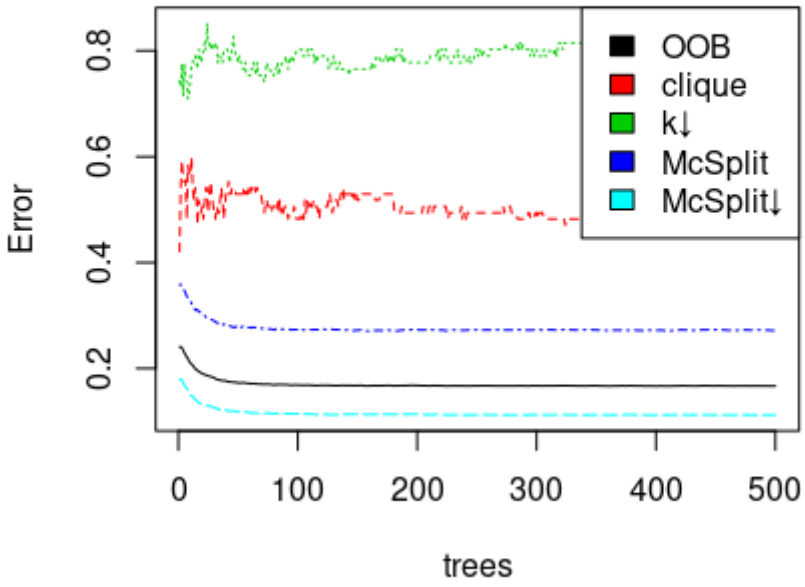


Figure 5.1: Convergence plot of various error measures as the number of trees in a random forest increases. The plot shows the OOB error and 1 − recall for each algorithm.

The cross-validation folds are then passed to the ML algorithm. LLAMA supports algorithm portfolios based on three different types of ML algorithms:

Classification The ML algorithm predicts which algorithm is likely to perform the best on each problem instance.

Regression Each algorithm's data is used to train a separate ML model, predicting the algorithm's performance. The winning algorithm can then be chosen based on those predictions.

Clustering All instances of the training data are clustered and the best algorithm is determined for each cluster. New problem instances can then be assigned to the closest cluster.

We are using a classification algorithm called random forests. The ideas behind the main algorithm is described in [2]. More information about its implementaion in R can be found in [14]. We chose this algorithm as it is recommended in the LLAMA manual [11] and successfully used in a similar study [13].

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

Random forests support a convenient way to estimate the test error without cross-validation or any other kind of data splitting. Each tree in a random forest uses around 2/3 of the data. The remaining 1/3 is referred to as

out-of-bag (OOB) observations [9]. For each observation in the data, we can predict the answer (the vote on which algorithm is expected to win) using all trees that have the observation as OOB. The majority vote is then declared to be the answer. The *OOB error* is the relative frequency of incorrect predictions. As each prediction was made using trees that had not seen that particular observation before, OOB error is a valid estimate of test error. The black line in Figure 5.1 shows how the error converges with the number of trees to about 17%.

The other lines in the figure, one for each algorithm, are defined as $1-\text{recall}$, where for an algorithm A *recall* [20] is

$$\frac{\text{the number of instances that were correctly predicted as } A}{\text{the number of instances where } A \text{ is the correct prediction}}.$$

The error rates for the clique encoding and $k\downarrow$ are quite high, converging to 47% and 81%, respectively. This is most likely due to having very few observations with them as the winning algorithms. The error rates for McSplit and McSplit $\downarrow$ converge to 27% and 11%, respectively.

Next we are going to explore how important each feature is in making predictions, but for that we need to introduce some new definitions. Consider a single tree T in a random forest. The root of T can be reached by any observation, regardless of the values of its features. After passing some node n, some feature is restricted, i.e., it is imposed an upper or lower limit on the kind of values it can have for it to move towards a particular child of n. The part of feature space that an observation can have while at some node n is called a *region*.

Definition 5.1.1. Suppose we have K classes. Consider some region m. The *Gini index* is then defined as

$$G = \sum_{k=1}^{K} \hat{p}_{mk}(1-\hat{p}_{mk}),$$

where $\hat{p}_{mk}$ represents the proportion of observations in region m that are from class k (i.e., have algorithm k as the best algorithm) [9].

As we move down a tree, we want the region to be restricted to a single class. Then the observations from the training data that satisfy the conditions imposed by the parent nodes would be classified with perfect accuracy. The Gini is at its lowest when all the proportions $\hat{p}_{mk}$ are close to either 0 or 1, meaning that almost all the observations in the region belong to a single class. Hence the Gini index is often used to evaluate the quality of a split.

Remark 5.1.1. Note that $G = 0$ when any single $\hat{p}_{mk} = 0$, regardless of the values of other proportions. Therefore $G = 0$ does not automatically imply that the tree is a good classifier.

The variable importance measure of feature f in Figure 5.2 is calculated as the amount by which the Gini index decreases after passing nodes that use feature f, averaged over all trees in the random forest [9]. Looking at the figure more closely, the standard deviations of degrees of both target and pattern graphs are by far the most important predictors. Unsurprisingly, the worst predictors are the features with very low variance: number of loops and connectedness of both graphs. Perhaps more surprisingly, the ratio features are not as successful as one might have hoped: the ratio of the numbers of vertices is at the bottom 5th place and the best ratio feature, the mean distance ratio, is only 10th. Last thing to note is that features of the pattern graph are always behind the same features of the target graph and usually not far behind. Perhaps this is due to some datasets having less pattern graphs, or them having less vertices.

5.2 Vertex-Labelled Graphs

5.3 Vertex- and Edge-Labelled Graphs

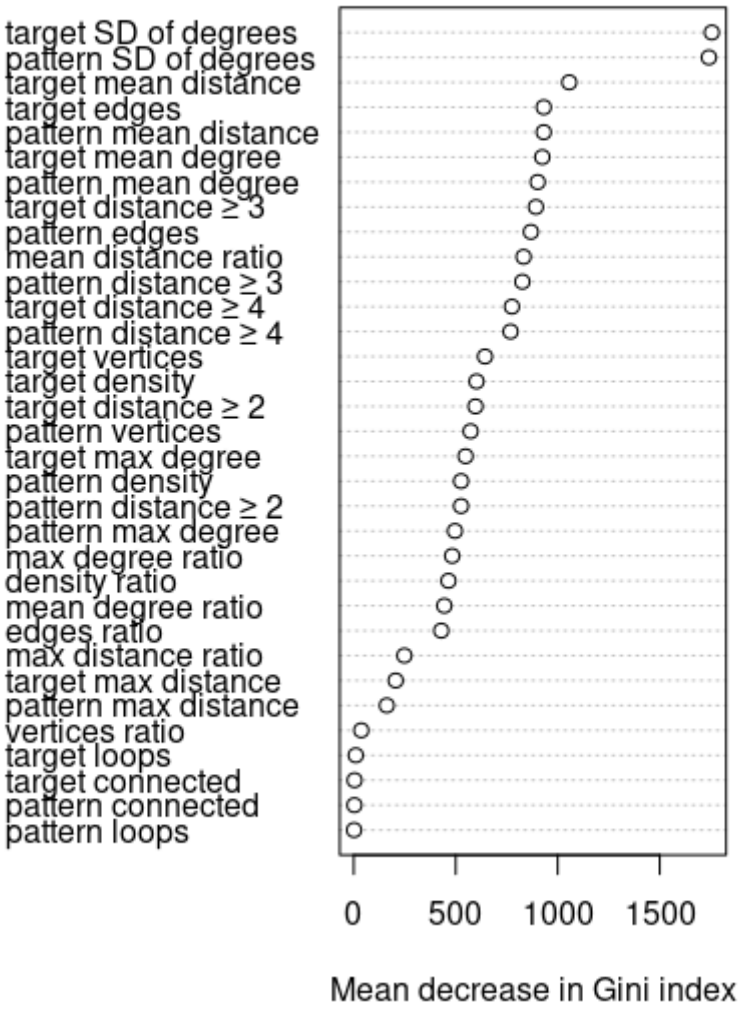


Figure 5.2: Dotchart of variable importance calculated based on the Gini index and sorted from most important to least important

Bibliography

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

[2] Leo Breiman. Random forests. *Machine Learning*, 45(1):5–32, 2001.

[3] Stephen A. Cook. The complexity of theorem-proving procedures. In Michael A. Harrison, Ranan B. Banerji, and Jeffrey D. Ullman, editors, *Proceedings of the 3rd Annual ACM Symposium on Theory of Computing, May 3-5, 1971, Shaker Heights, Ohio, USA*, pages 151–158. ACM, 1971.

[4] 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.

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

[6] 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.

[7] 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.

[8] 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.

[9] Gareth James, Daniela Witten, Trevor Hastie, and Robert Tibshirani. *An Introduction to Statistical Learning: With Applications in R*. Springer Publishing Company, Incorporated, 2014.

[10] 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.

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

[12] Lars Kotthoff, Pascal Kerschke, Holger Hoos, and Heike Trautmann. Improving the state of the art in inexact TSP solving using per-instance algorithm selection. In Clarisse Dhaenens, Laetitia Jourdan, and Marie-Eléonore Marmion, editors, *Learning and Intelligent Optimization - 9th International Conference, LION 9, Lille, France, January 12-15, 2015. Revised Selected Papers*, volume 8994 of *Lecture Notes in Computer Science*, pages 202–217. Springer, 2015.

[13] 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.

[14] Andy Liaw and Matthew Wiener. Classification and regression by randomforest. *R News*, 2(3):18–22, 2002.

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

[16] 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.

[17] 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.

[18] 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.

[19] 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.

[20] David M. W. Powers. Evaluation: from precision, recall and f-measure to roc, informedness, markedness and correlation. *Journal of Machine Learning Technologies*, 2(1):37–63, 2011.

[21] 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.

[22] 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.

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

[24] 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.

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

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

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