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 — 9th January 2018

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

We start with a definition of a graph suitable for the empirical data described in Chapter 3 and used in Chapter 4.

Definition 1.1. Let S be a set and k a non-negative integer. Then $[S]^k$ denotes the set of *k-subets* of S [42], i.e.,

$$[S]^k = \{A \subseteq S : |A| = k\}.$$

Definition 1.2. 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 [8]. 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*.

For the purposes of this project, 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 1.3. A *(vertex-)labelled graph* is a 3-tuple $G = (V, E, \mu)$, where V, E are as in Definition 1.2 and $\mu\colon V \to \{0, \ldots, N-1\}$ is a vertex labelling function, for some $N \in \{1, \ldots, |V|\}$.

Definition 1.4. A *fully labelled graph* is a 4-tuple $G = (V, E, \mu, \zeta)$, where the first three elements are as in Definition 1.3 and $\zeta\colon E \to \{0, \ldots, M-1\}$ is an edge labelling function, for some $M \in \{1, \ldots, |E|\}$.

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.
- A vertex-labelled graph is just a special case of a fully labelled graph with $M = 1$.
- An unlabelled graph is just a special case of of a vertex-labelled graph with $N = 1$.

The last two observations allow us to tailor subsequent definitions to fully labelled graphs without having to define everything 3 times for the three cases of labelling. We formulate all of the definitions in terms of undirected multigraphs as some of the graphs in Chapter 3 contain multiple edges and many contain at least one loop. In the rest of the dissertation, we will use the word "graph" to denote undirected multigraphs. When vertex and/or edge labels are immaterial to a definition, we will omit the labelling functions from the full 4-tuple and denote a graph as just $G = (V, E)$. Next we adapt some basic graph theory definitions to work with our definition of graphs and labelling. As most definitions in the literature are typically defined for simple graphs with no labels, the cited sources are heavily adapted to the full generality of graphs relevant to this dissertatation.

Definition 1.5. Two graphs $G_1 = (V_1, E_1, \mu_1, \zeta_1)$ and $G_2 = (V_2, E_2, \mu_2, \zeta_2)$ are said to be *isomorphic* if there are bijections $f\colon V_1 \to V_2$, $g\colon E_1 \to E_2$ such that [33]:

- $\forall e \in E_1$, if e maps to some $v \in V_1 \cup [V_1]^2$, then $g(e)$ maps to $f(v)$, where $f(v) = \{f(v_1), f(v_2)\}$ if $v = \{v_1, v_2\}$ (preserving the structure);
- $\forall v \in V_1$, $\mu_2(f(v)) = \mu_1(v)$ (preserving vertex labels) [2];
- $\forall e \in E_1$, $\zeta_2(g(e)) = \zeta_1(e)$ (preserving edge labels).

Definition 1.6. Let $f\colon X \to Y$ be a function. Then the *restriction* of f to $A \subseteq X$ is a function $f|_A\colon A \to Y$ such that $\forall a \in A$, $f|_A(a) = f(a)$ [39].

Definition 1.7. A graph $G_2 = (V_2, E_2, \mu_2, \zeta_2)$ is a *subgraph* of graph $G_1 = (V_1, E_1, \mu_1, \zeta_1)$ if [8]:

- $V_2 \subseteq V_1$,
- $E_2 \subseteq E_1$ (with the edge-mapping function restricted to E_2),
- $\mu_2 = \mu_1|_{V_2}$,
- $\zeta_2 = \zeta_1|_{E_2}$.

Definition 1.8. An *induced subgraph* of a graph $G = (V, E)$ is a subgraph $H = (S, E')$, where $E' \subseteq E$ is a set of edges mapped to $S \cup [S]^2$ [33].

Definition 1.9. A *clique* C in a graph $G = (V, E)$ is a subset of V such that $\forall v_1, v_2 \in C$ with $v_1 \neq v_2$, there is an edge in E mapping to $\{v_1, v_2\}$ [31].

Finally, we define the main problem of this project:

Definition 1.10. 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 [33]. 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 $U_2 \subseteq V_2$.

The purpose of this project is to create and explore machine learning-based algorithm *portfolios*, i.e., machine learning models whose main goal is to provide a mapping, assigning the best-performing algorithm to each problem instance [3, 34].

Chapter 2

The Algorithms

We consider four algorithms that were shown to be competitive in a paper by McCreesh *et al.* [28]: $k \downarrow$ [14], MCSPLIT, MCSPLIT $\downarrow$ [28], and the clique encoding [25]. The way some of these algorithms work relates to two other problems: the maximum clique problem and the subgraph isomorphism problem, both defined as follows:

Definition 2.1. Given a graph G, the *maximum clique problem* is an optimisation problem asking for a clique in G with maximum cardinality [31].

Definition 2.2. 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 [6].

We will illustrate how each algorithm works by considering the graphs in Figure 2.1. Since G_p has only one blue edge, while all edges of G_t are blue, any common subgraph can have up to one edge. It is then obvious that the maximum common subgraph is isomorphic to the subgraph induced by $\{u_1, u_2, u_3\}$ in G_p and it is isomorphic to multiple induced subgraphs of G_t.

$k \downarrow$ algorithm [14] 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.

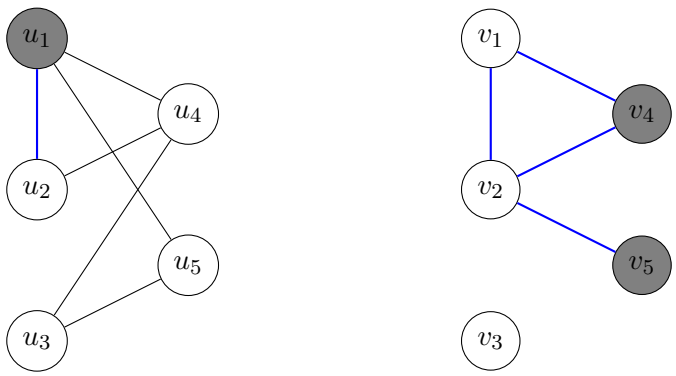


Figure 2.1: Two graphs with binary ($\{0, 1\}$) labels on both vertices and edges with pattern graph (G_p) on the left and target graph (G_t) on the right. Grey vertices and thick blue edges represent label 1.

The clique encoding [25] solves the maximum common subgraph problem by creating a new (association) graph and transforming the problem into an instance of the maximum clique problem [21], which is then solved by a sequential version of the maximum clique solver by McCreesh and Prosser [26], 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.

Remark 2.1. Two versions of the clique algorithm had to be combined to support both binary and plaintext formats used by the databases described in Chapter 3.

MCSPLIT [28] 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 [28]. 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.

2.1 Modifications to $k\downarrow$

The $k\downarrow$ algorithm was modified to accept graphs with vertex labels by adding an additional constraint for matching labels on line 8 of the `klessSubgraphIsomorphism` function [14] and extending the `Graph` class with two new fields: a `vector`, assigning a label to each vertex, and a `vector` of `vectors`, which, for each label, list all vertices that have that label. After generating most of the data, it was noticed that the vertex-labelled version of $k\downarrow$ was winning precisely 0 times.

TODO: replace with the new version once we know it's correct In order to make $k\downarrow$ more competitive, the neighbourhood degree sequence filtering was improved to make use of the additional information that labels provide. It is also part of line 8 of the same function in the original paper [14], however, it is not enabled by default[2] and is not used in the experiments of the MCSPLIT paper [28]. We take the definition directly from [14]:

Definition 2.3. "The *neighbourhood degree sequence* (NDS) of a vertex p, $S(p)$, is the (non-ascending) sequence of degrees of its neighbours."

Definition 2.4. For two sequences $S = (s_1, \ldots, s_n)$ and $T = (t_1, \ldots, t_m)$ and some non-negative integer k, we say that $S \preceq_k T$ if $n - k \leq m$ and there is a subsequence S_k of S with $|S_k| \leq k$ such that [14]

$$\forall s_i \in S \setminus S_k, \text{ there exists a distinct } j \in \{1, \ldots, m\} \text{ such that } s_i - k \leq t_j.$$

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

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

As a simplification of the definition above, we state the following lemma, based on Corollary 1 from the $k\downarrow$ paper [14] and the C++ implementation[2].

Lemma 2.1. *Let $S = (s_1, \ldots, s_n)$ and $T = (t_1, \ldots, t_m)$ be two sequences and k an integer such that $k \geq \max\{0, n-m\}$. Then $S \preceq_k T$ if and only if*

$$s_i \leq t_{i-k} + k \quad \forall i = k+1, \ldots, n.$$

Proof. A simple extension of Corollary 1 from [14] by a reordering argument. □

Data: a non-negative integer kOriginal, a list of NDSs for each label pNds, a list of NDSs for each label tNds
Result: a boolean value, indicating whether p and t can be matched

```
kLeft ← kOriginal − Σ_{i=|tNds|}^{|pNds|−1} |pNds[i]|;
if kLeft < 0 then
    return false;
end
for i ← 0 to min{|pNds|, |tNds|} − 1 do
    for k ← max{|pNds[i]| − |tNds[i]|, 0} to ∞ do
        if k > kLeft then
            return false;
        end
        if ⋀_{j=k}^{|pNds[i]|−1} tNds[i][j − k] + k ≥ pNds[i][j] then
            kLeft ← kLeft − k;
            break;
        end
    end
end
return true;
```

Algorithm 1: NDS filtering with vertex label support

We extend these ideas in Algorithm 1 by considering an NDS for each label. The algorithm checks whether a vertex p in the pattern graph can be matched with a vertex t in the target graph. kOriginal is the value of k in the main $k\downarrow$ algorithm, denoting how many of the pattern graph vertices can be mapped to nothing. pNds and tNds store the non-ascending NDSs for each label of the pattern and target graph, respectively. Some of them can be empty and the graphs can have different numbers of labels. kLeft is our counter for how many neighbours of p we can still map to nothing. We note that if p has a neighbour with label i and t has no neighbours with such a label, we must use one of the k values, i.e., reduce kLeft by 1. We also note that if kLeft becomes negative, then there is no way for p and t to be matched and we must return false. Then, for each label i, we find the smallest integer $k \in \{\max\{|\text{pNds}[i]| - |\text{tNds}[i]|, 0\}, \ldots, \text{kLeft} - 1\}$ such that $\text{pNds}[i] \preceq_k \text{tNds}[i]$ according to Lemma 2.1. If we are unable to find such a value, then there is no way for p and t to be matched and we return false. Otherwise we subtract from kLeft the number of neighbours of p that could not be mapped to any of the neighbours of t and move on to label $i+1$.

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 [28] 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 [10, 36], 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

In Definitions 1.3 and 1.4 we used N and M to denote the number of different labels for vertices and edges, respectively. The ARG Database supports interpreting the same graphs to have different numbers of different labels via this parameter[1]:

Definition 3.1. A graph $G = (V, E)$ is said to have a $p\%$ *(vertex) labelling* if

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

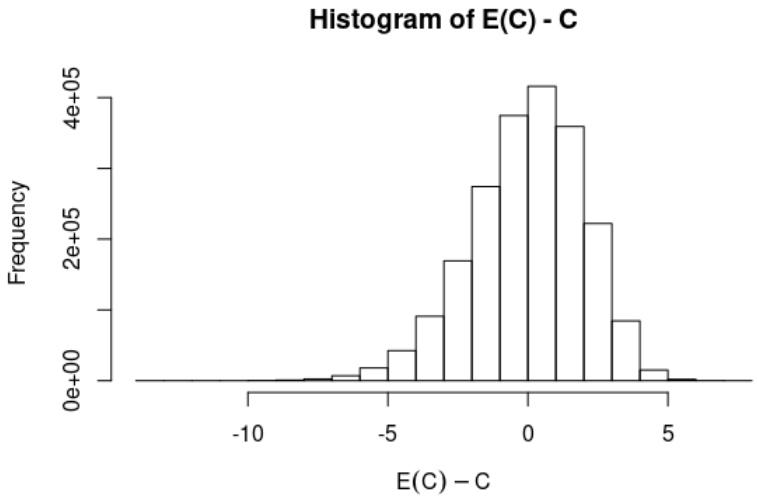


Figure 3.1: Histogram of the difference between the expected number of vertices assigned each label and the actual number (for all labelled graphs)

The default value for p is 33%. The publications associated with the database [10, 36] 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[2] (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 [7].

images-PR15 Graphs generated from segmented images [38]. 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).

[1] `http://mivia.unisa.it/datasets/graph-database/arg-database/documentation/`, "How to read labeled graphs"

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

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 [37, 45].

phase Random graphs generated to be close to the satisfiable-unsatisfiable phase transition (200 instances) [27].

largerGraphs Larger instances of the `LV` dataset. There are 70 graphs, giving $70^2 = 4900$ instances. The separation was made and used in [14, 20, 28].

Remark 3.1. This set of instances was taken from the repository[3] for the McSplit paper [28] and has some minor differences from the version on Christine Solnon's website.

Remark 3.2. 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)).

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

Chapter 4

Generating Data

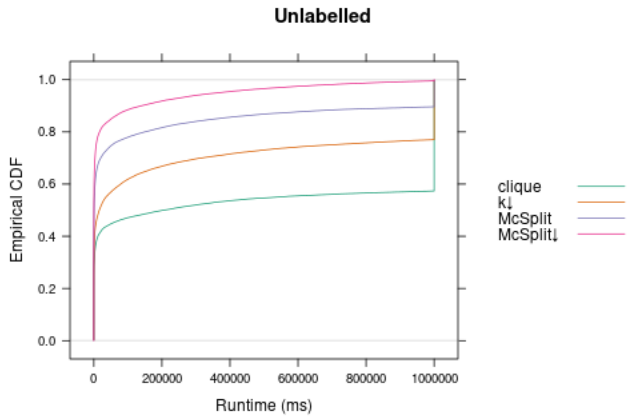
A machine learning (ML) model requires data to learn from. We are using an R package called LLAMA [17, 18], which helps to train and evaluate ML models in order to compare algorithms and was used to create algorithm portfolios for the travelling salesperson problem [19] and the subgraph isomorphism problem [20]. 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 [20] to handle the binary format of the ARG Database [10, 36], run it on all graphs, and record the features in a way described in Section 4.2.

4.1 Running Time of Algorithms

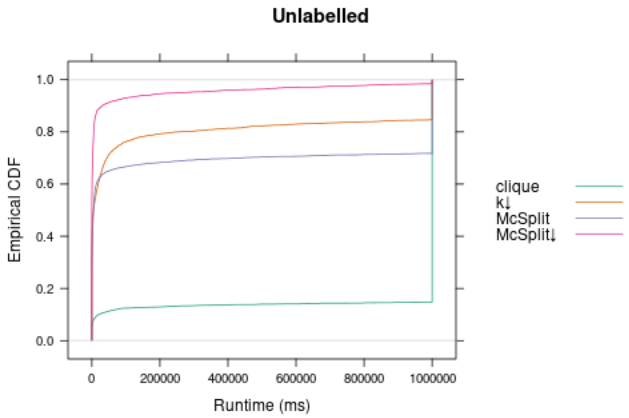
The algorithms were compiled with gcc 6.3.0 and run on Intel Xeon E5-2697A v4 (2.60 GHz) processors with a 1000 s time limit. A Makefile was created to run multiple experiments in parallel with, e.g., `make -j 64`, which generates pairs of graph filenames for all datasets, runs the selected algorithms 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).

Some limitations had to be enforced to avoid running out of memory. First, the clique algorithm requires $O(n^2m^2)$ memory for a pair of graphs with n amd m vertices [14, 25], so its virtual memory usage was limited to 7 GB with `ulimit -v` and the instances from Section 3.2 (which contain much larger graphs) were restricted to $m \times n < 16{,}000$. Second, having almost 10^5 problem instances and analysing 7 different kinds of labelling (according to Definition 3.1) results in too much data for the ML algorithm. Therefore, we sample $30{,}000$ out of $81{,}400$ instances from Section 3.1. The sample is drawn once and used for training ML models for both types of labelling (vertex labels and both vertex and edge labels). As with 50% labelling most instances are solved within the time limit, sampling from the whole database still leaves us with enough relevant data.

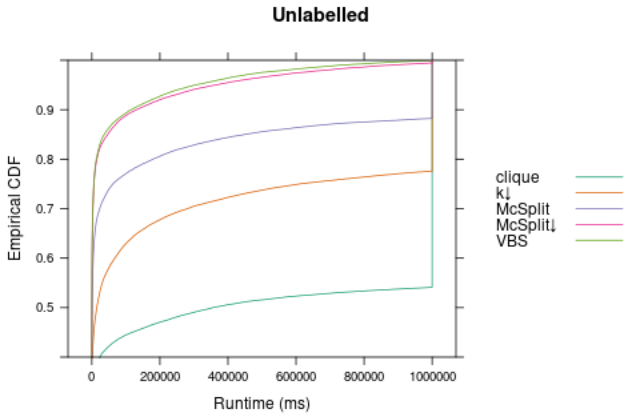
In the rest of this section we explore and compare how the algorithms performed on the three different subproblems under consideration, those of having no labels, vertex labels, and both vertex and edge labels. We introduce *empirical cumulative distribution function (ECDF)* plots [44]: for each unit of time on the horizontal axis, the value on the vertical 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

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 [28]. Namely, MCSPLIT outperforms $k\downarrow$ in Figure 4.1a, and the opposite happens in Figure 4.1b. In Figure 4.1c we also plot a curve for the *virtual best solver (VBS)*, i.e., a perfect algorithm portfolio that always chooses the best-performing algorithm for each problem instance. Note that the difference between MCSPLIT $\downarrow$ and VBS is very small. Therefore, a portfolio cannot provide significant performance benefits for this subproblem. We also provide a heatmap in Figure 4.2 to compare the runtimes of the algorithms on a per-instance basis.

Remark 4.1. 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 4.2. More specifically, Table 4.1 shows the numbers of times that each algorithm's runtime was lower than or equal to the runtimes of other algorithms. Therefore, if 2 or more lowest runtimes are equal (as can often happen with single-digit runtimes), both algorithms are marked as winning in the table.

Given this information, we would expect the ML algorithm to suggest using MCSPLIT and MCSPLIT $\downarrow$ most

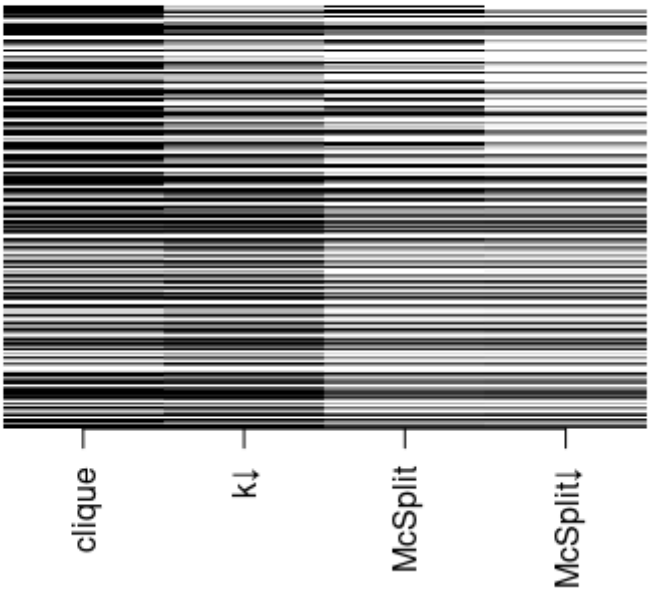


Figure 4.2: A heatmap of $\log_{10}$ runtimes: light colours for low running times and black for timing out

Dataset	clique	$k\downarrow$	McSplit	McSplit $\downarrow$
images-CVIU11	0	32	79	1081
images-PR15	0	0	0	24
largerGraphs	0	14	30	167
LV	90	30	489	439
meshes-CVIU11	0	13	0	23
phase	0	0	0	0
scalefree	0	0	0	80
si	0	10	102	1135
ARG Database	1443	141	21965	27305
Total	1533	240	22665	30254

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

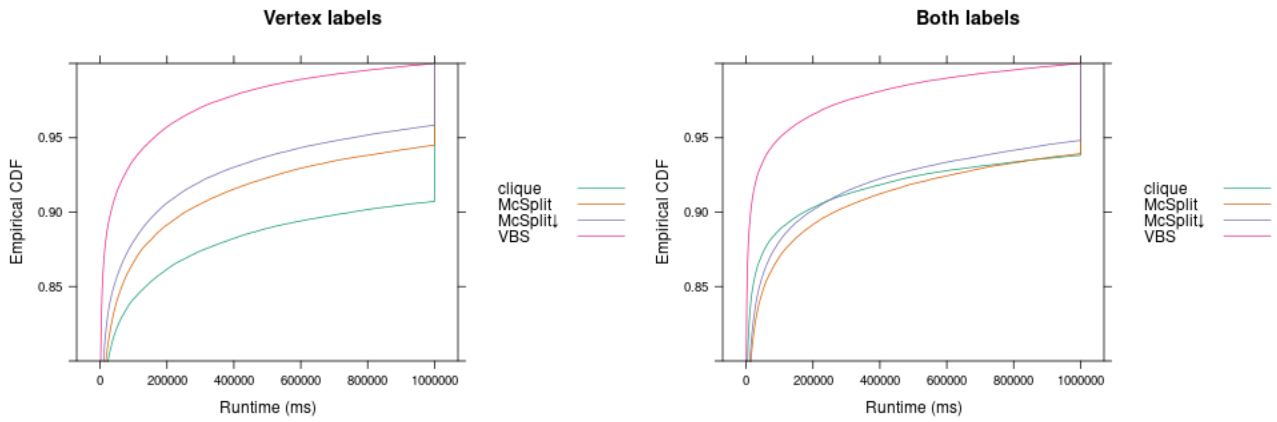


Figure 4.3: Cumulative plots with the vertical axis starting at 0.8

of the time, occasionally consider the clique encoding, and mostly forget about $k\downarrow$.

4.1.2 Labelled Graphs

We are analysing two different subproblems, dealing with graphs with just vertex labels and with both vertex and edge labels. However, since the results are fairly similar, we describe them in the same section to highlight the differences.

We plot the ECDFs in Figure 4.3. The situation with vertex-labelled graphs is quite straightforward: McSplit $\downarrow$ is slight better than McSplit, which is better than the clique encoding. Moreover, the VBS curve is significantly higher, providing plenty of room for an ML model to outperform individual algorithms (unlike with unlabelled data). This latter fact is true for graphs with both vertex and edge labels as well, whereas the other 3 curves provide a more interesting story. The clique algorithm is briefly winning for shorter timeout values, then is between McSplit $\downarrow$ and McSplit until it drops to the 3$^{\text{rd}}$ place right before the final timeout at 1,000,000 ms. This is likely because the clique encoding is better with higher labelling percentages (we will see this shortly) and such graphs are generally easier to solve (because there are fewer "matchable" combinations of vertices, each pair of vertices is likely to have different labels).

Just like with unlabelled graphs we could split the data into different subsets based on how (and by whom) the graphs were generated, now we can analyse how the situation changes with different labelling percentages. In Figure 4.4 we analyse two performance measures (proportion of instances solved and total runtime) as well as the number of times each algorithm "wins", for each labelling percentage. The two performance statistics tell exactly the same story, and it is the same for both types of labelling. The only difference is that the clique algorithm's major drop in performance starts at 20% for vertex-labelled graphs and at 15% for both vertex- and edge-labelled graphs. With higher labelling percentages, the clique encoding is better than McSplit, which is marginally better than McSplit $\downarrow$. Whereas with lower levels of labelling, two differences emerge: McSplit experiences a drop in performance below McSplit $\downarrow$, and the performance of the clique algorithm starts to drop exponentially. The latter is exactly what makes the clique encoding finish last in Figure 4.3, even though it is in the lead for most labelling percentages.

Remark 4.3. Note that we are only considering instances solved by at least one algorithm. Out of the 30.000 instances selected by our random sampling procedure, the number of such instances ranges from 56% with 5% labelling to 99% with 50% labelling for vertex-labelled graphs (the percentages for both vertex and edge labels are similar). Thus a (roughly) horizontal line should not be interpreted as the algorithm performing equally well for different labelling percentages: the performance of all algorithms improves with higher labelling percentages as the problem becomes significantly easier. In this case we are more interested in the differences between

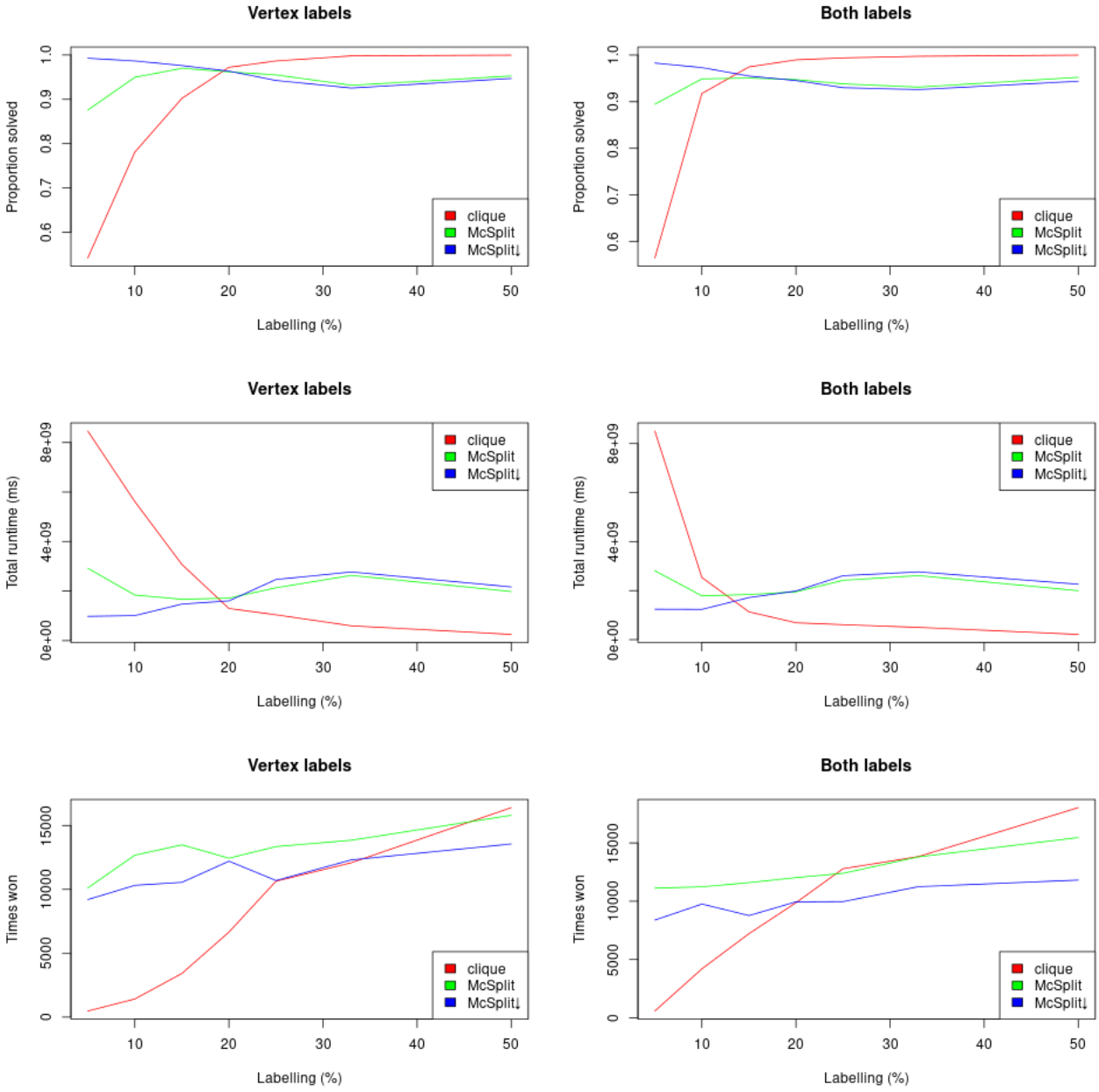


Figure 4.4: For both types of labelling and for the 3 algorithms we plot how the 3 characteristics change with respect to the labelling percentage: proportion of instances solved within the time limit, total runtime, and number of times each algorithm outperformed the others

individual algorithms.

As for the last two plots of Figure 4.4, MCSPLIT is winning more often than MCSPLIT $\downarrow$. This is not surprising for higher labelling percentages (based on the other two types of plots), but this remains true for lower percentages as well. Perhaps this is because by definition MCSPLIT $\downarrow$ is better at handling problem instances with a big answer. With a smaller labelling percentage, MCSPLIT is more likely to time out on those instances, contributing to MCSPLIT $\downarrow$ solving more instances than MCSPLIT. Perhaps MCSPLIT $\downarrow$ is also much faster on those instances, ensuring its lower total runtime, while consistenly falling slightly behind MCSPLIT on easier instances, resulting in a lower win count. The one significant difference between the two types of labelling is that the clique encoding wins slightly less than MCSPLIT $\downarrow$ with vertex labels, but slightly more than MCSPLIT with both vertex and edge labels, for 25%–33% labelling. Finally, the main observation from this plot is that the overall highest point is only at 16,401 (18,031) for vertex-labelled graphs (both vertex and edge labels), just slightly above half of the number of instances (30,000) and other than the clique encoding falling behind with $\leq 15\%$ labelling, the 3 algorithms winning rates stay similar. This makes the problem especially potent for a algorithm selection approach.

4.2 Graph Features

The initial set of features is based on the algorithm selection paper for the subgraph isomorphism problem [20] 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 4.1. 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 [8]. The standard formula used for *simple* graphs (i.e., graphs with no multiple edges or loops [30]) is

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

Even though some of our graphs do contain multiple edges and loops, we stick to this formula as it was used in [20] and it does not break the ML algorithm in any way to have the theoretical possibility of density greater than 1.

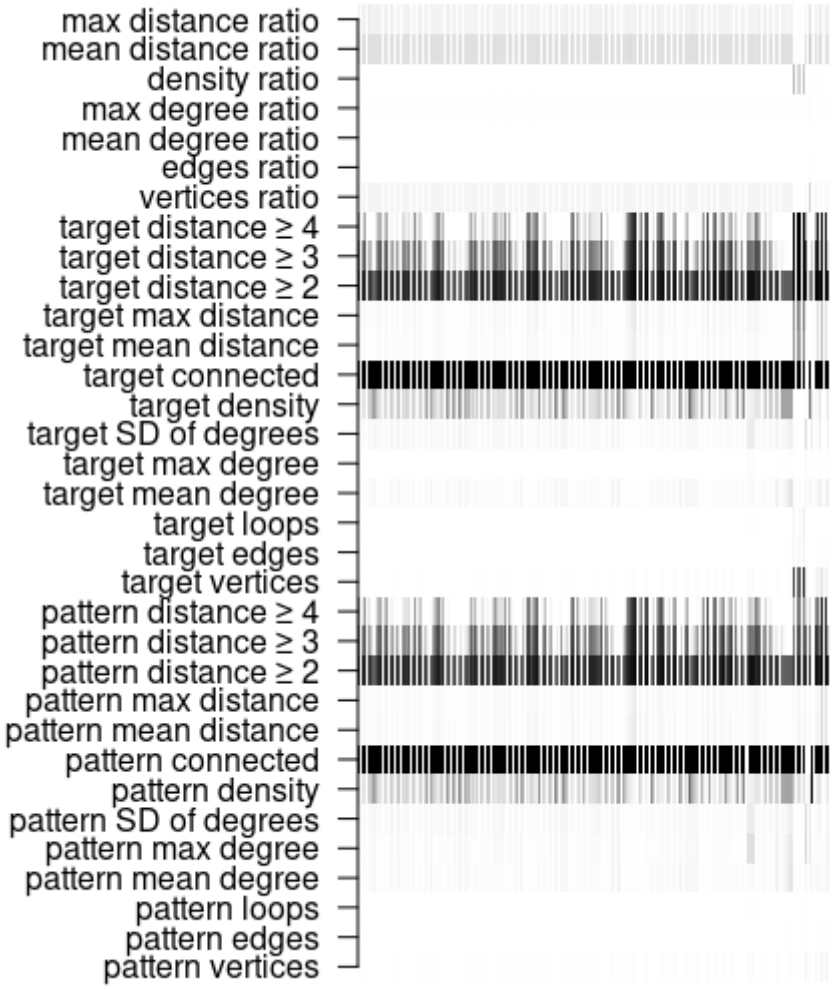


Figure 4.5: A heatmap for normalised features with black denoting the maximum value and white denoting the minimum for each feature

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 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 4.4. When working with both vertex and edge labels, we only consider using the same value of p for both vertices and edges. This "convention" seems to have originated in a paper by the creators of the ARG Database [5] and was replicated in subsequent papers on maximum common subgraph algorithms [25, 29].

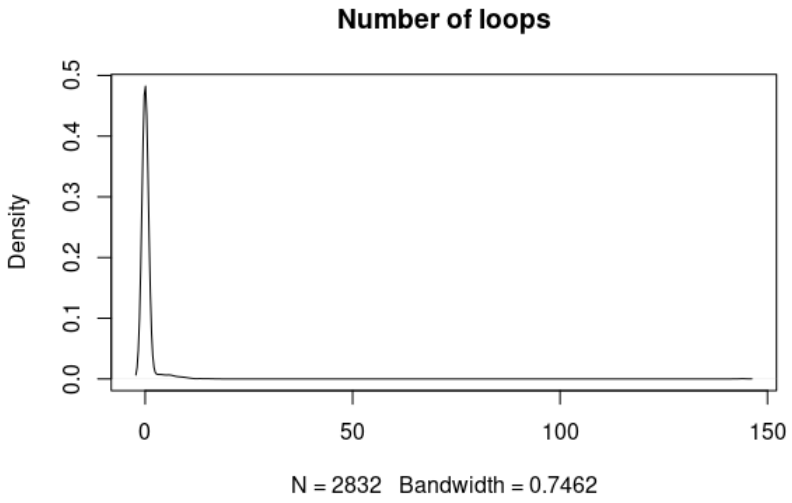


Figure 4.6: Density plot of the number of loops in graphs from Section 3.2

4.2.1 Distributions of Features

In this section we plot and discuss how the selected features are distributed in both databases. As the graphs from Section 3.2 contain some very hard instances, we only consider graphs that are part of a pair of graphs solved by at least one algorithm. In order to visualise highly skewed data, we sometimes use density plots. Furthermore, we take $\log$ transformations for ratio features. We also plot a heatmap of all features for all of the data in Figure 4.5. The rows that are almost completely white represent features that have several significantly higher values that skew the mean.

Firstly, one feature is not plotted as by definition it has only two possible values. 99.81% of graphs from Section 3.1 are connected, compared to 93.19% of graphs from Section 3.2. 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, many applications in chemistry are only interested in connected graphs [9]. Similarly, the number of loops for graphs from Section 3.1 is not plotted as it varies between two values: 0.98% of the graphs have a single loops, while the remaining majority of graphs have no loops. On the other hand, as shown in Figure 4.6, some (although not many) graphs from Section 3.2 have significantly more loops.

Most of the features for graphs from Section 3.1 are displayed in Figure 4.7. Other than the plot for the number of vertices, which is controlled by the creators of the database, all the other distributions are centred around lower values, with some outliers on the high end. More importantly, we have some graphs that are quite dense and some graphs with higher mean distance values.

The same plots for graphs from Section 3.2 in Figure 4.8 show a similar story, albeit for clearer reasons. Since we filter out graphs that none of the algorithms were able to handle, our sample consists of all of the easy instances and some harder instances that were solved by one or two algorithms. 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 are right skewed.

Figure 4.9 shows density plots of proportions of pairs of vertices with distance at least k for $k = 2, 3, 4$. For both databases, as k increases, the distributions shift to the left as expected. However, there is one important difference: even with $k = 4$ the plot for graphs from Section 3.2 has its highest peak around 0.9, which means that adding features for $k \geq 5$ could be valuable.

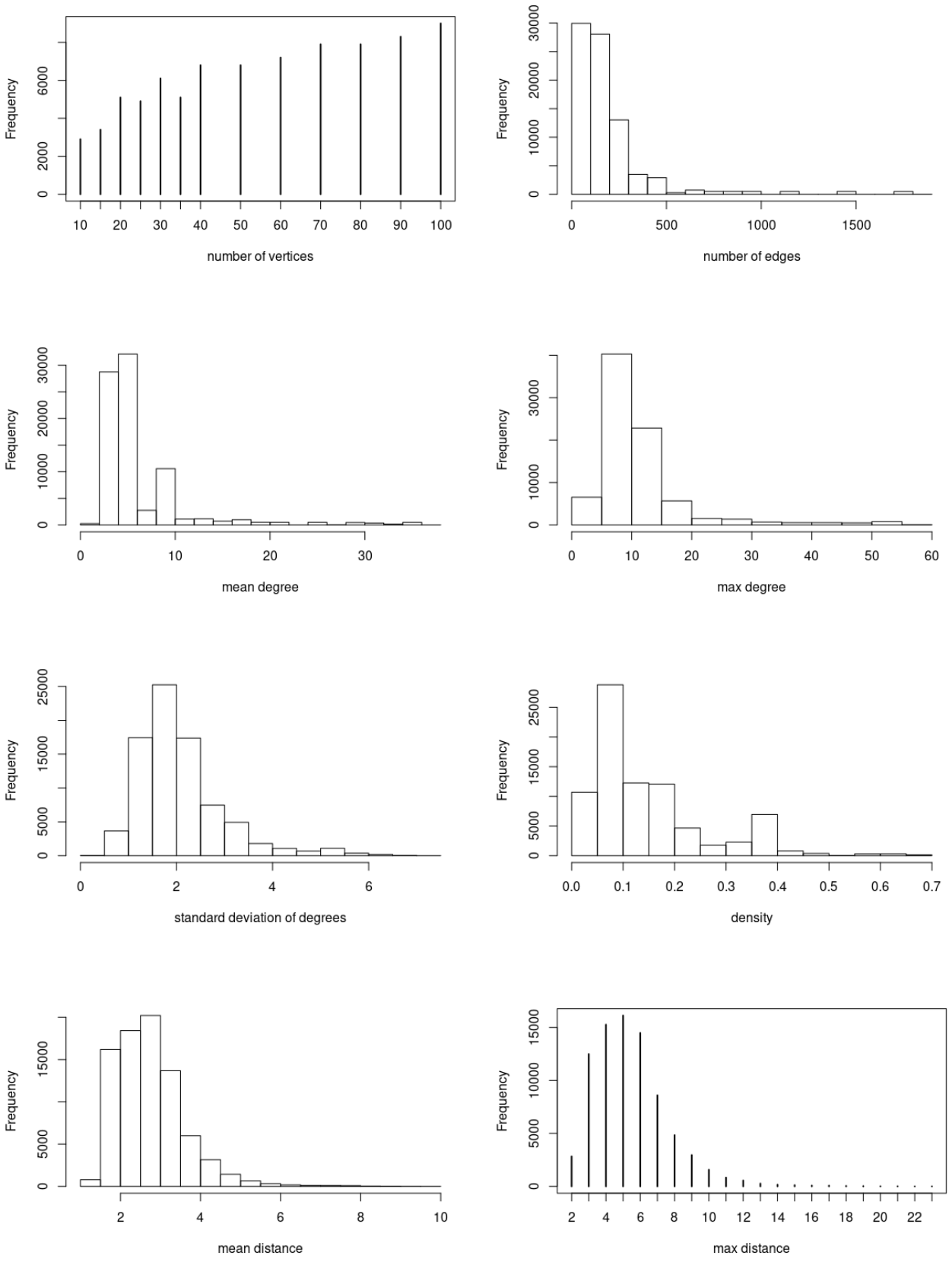


Figure 4.7: Plots of how various features are distributed for graphs from Section 3.1

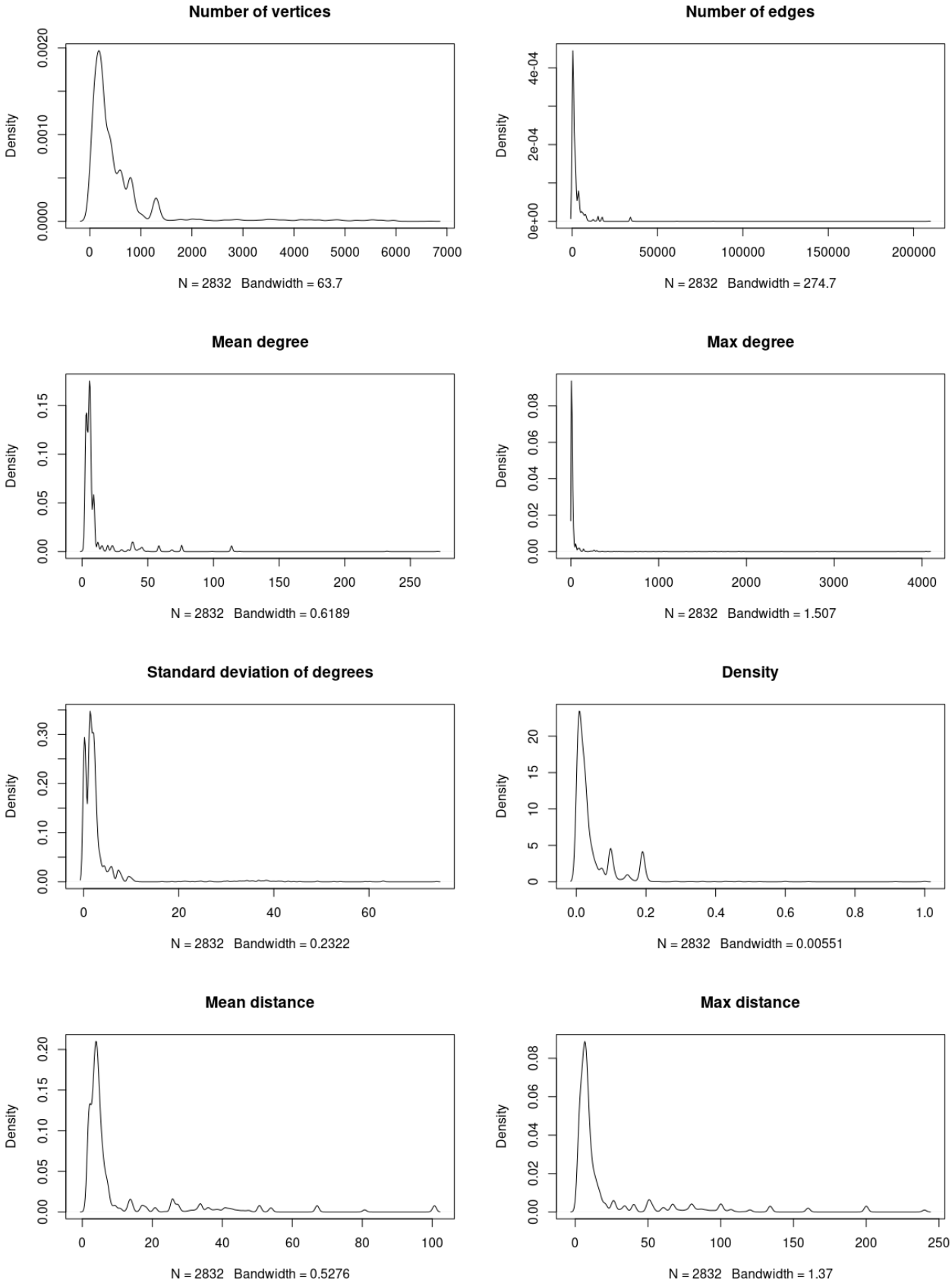


Figure 4.8: Plots of how various features are distributed for graphs from Section 3.2

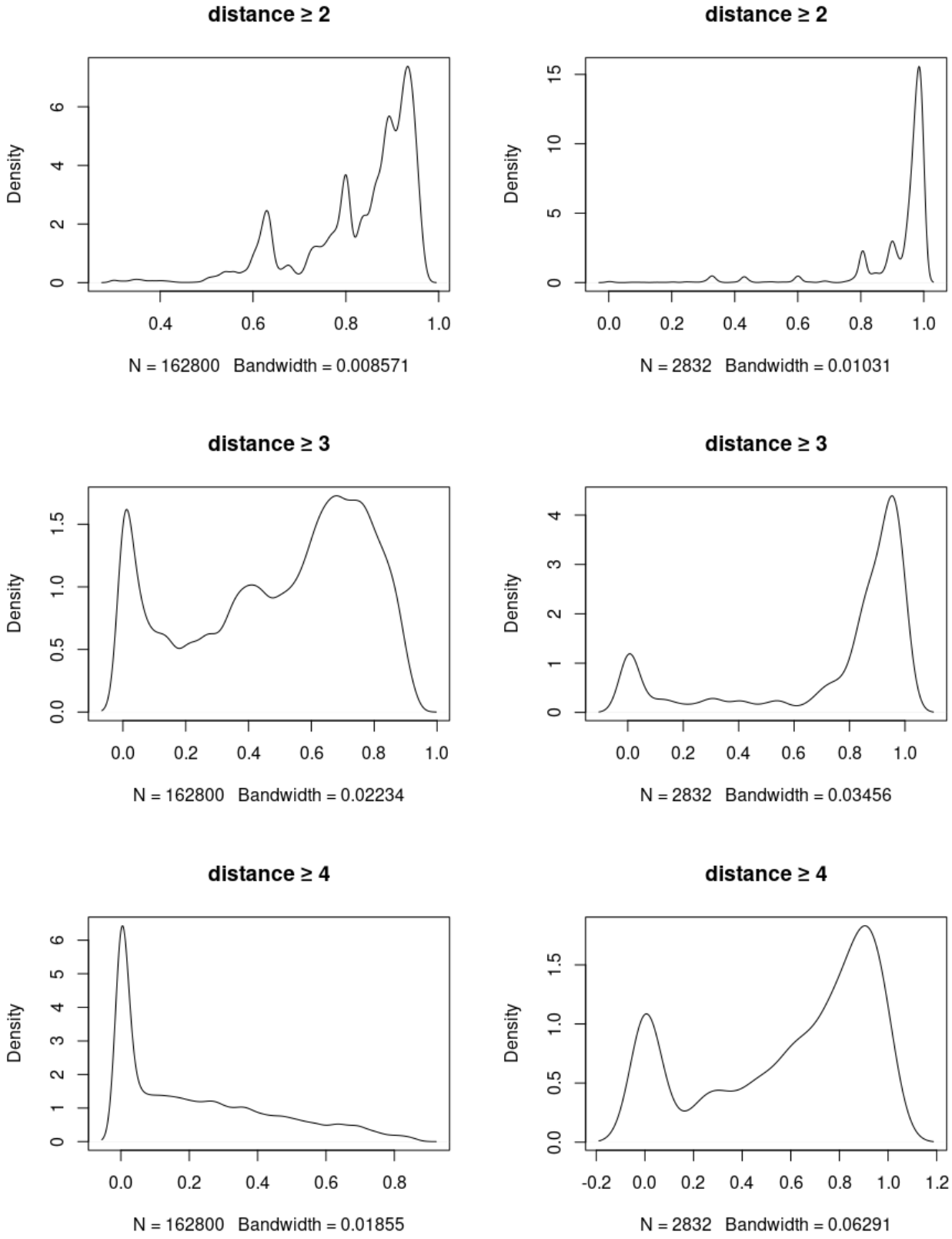


Figure 4.9: Comparison of typical distances between pairs of vertices between the two graph databases with graphs from Section 3.1 on the left and graphs from Section 3.2 on the right

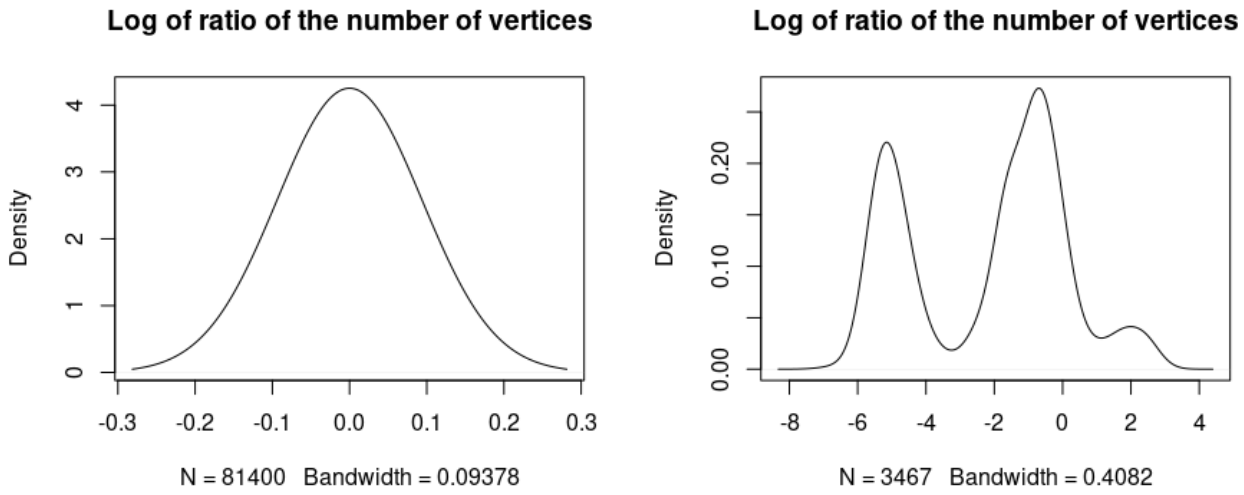


Figure 4.10: The density plots of log-transformed ratio of the number of vertices between pattern and target graphs for both databases with graphs from Section 3.1 on the left and graphs from Section 3.2 on the right

Finally, the density plots of log-transformed ratio features are in Figures 4.10, 4.11, and 4.12. Almost all of these plots for graphs from Section 3.2 (with the exception of the ratio of mean degree) are clearly bimodal with one of the two modes centred around 0. Hence we can infer the existance of two subpopulations: one where pattern and target graphs have very similar properties (the majority for most features) and one where pattern and target graphs are very different. As for the graphs from Section 3.1, the plot of the ratio of the number of vertices in Figure 4.10 is perfectly symmetrical and centred around 0, since the number of vertices is a controlled variable for this database. All of the remaining plots for ratio features have most of the data very close to 0. Thus, the differences between pattern and target graphs are very small. Furthermore, all the distributions are symmetrical—the pattern graph is just as likely to be larger/denser/etc. as the opposite.

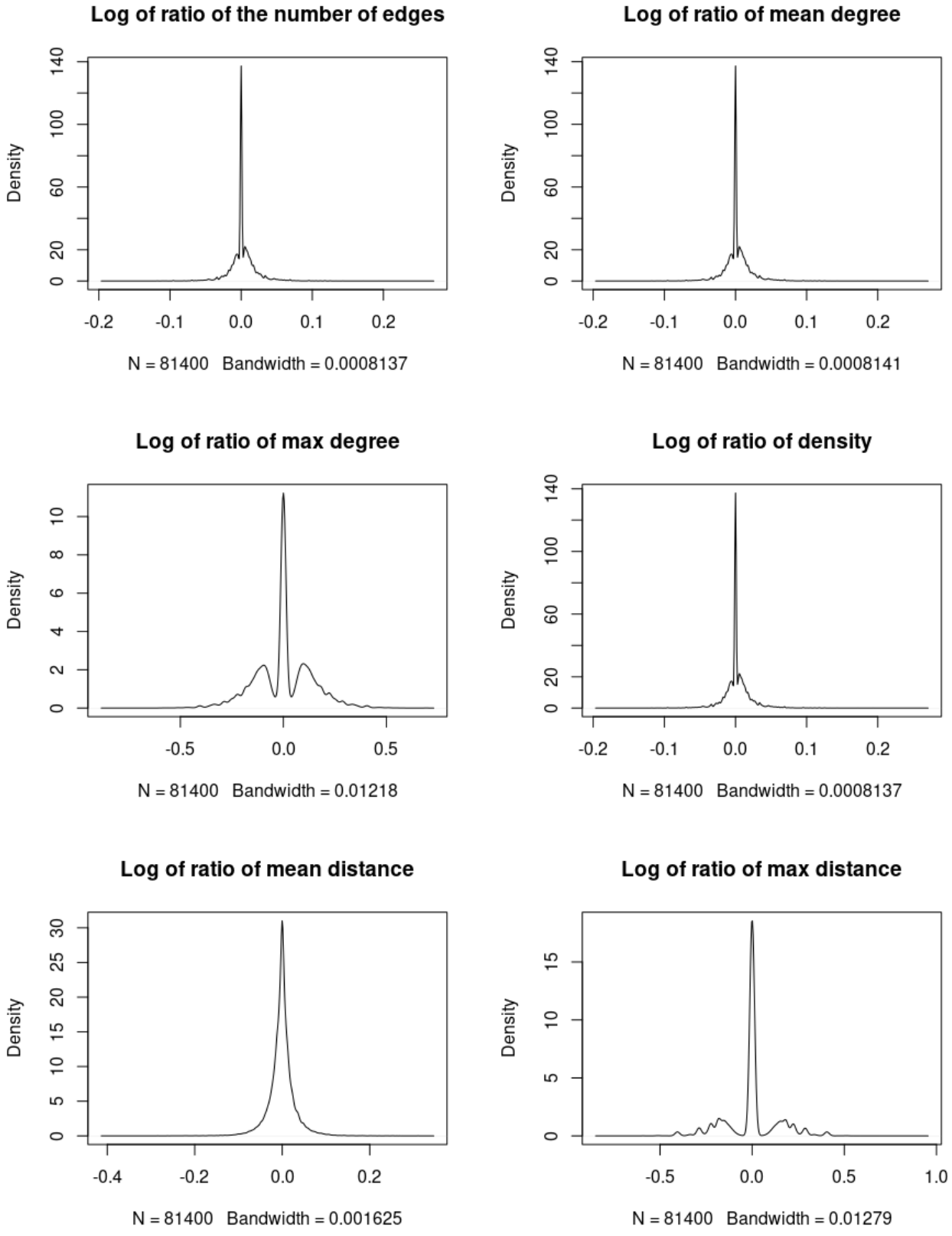


Figure 4.11: The other density plots of the log-transformed ratio features for graphs from Section 3.1

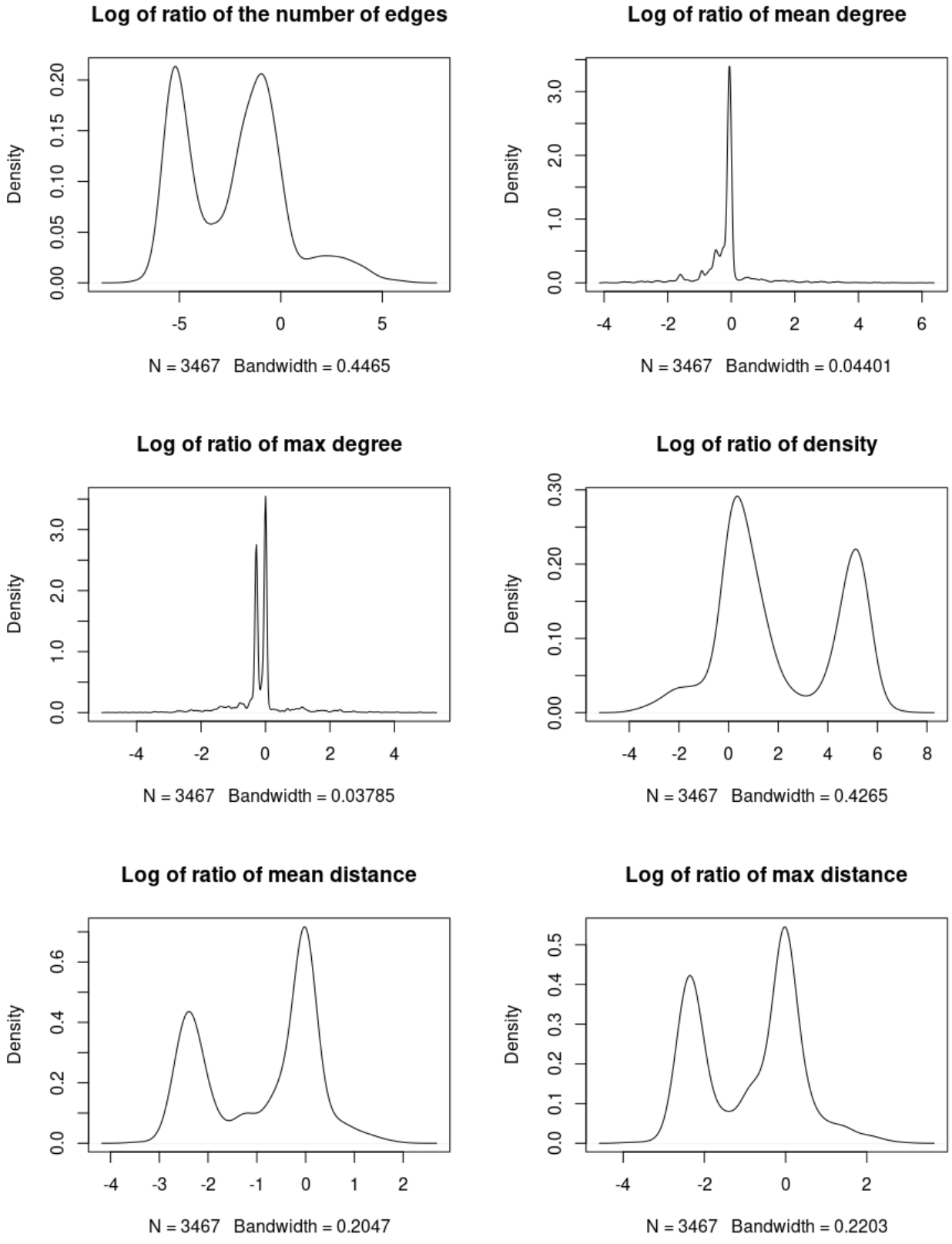


Figure 4.12: The other density plots of the log-transformed ratio features for graphs from Section 3.2

Chapter 5

Machine Learning Models & Their Evaluation

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 [18] can take:

- A list of features. With separate features for pattern and target graphs as well as ratio features, we have 33 features in total.
- 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, a single number for the approximate time taken to compute all features for any instance, or a list with one or more costs per problem instance. We use the last option with a single cost per instance. This parameter is used to ensure a fair comparison when comparing the portfolio against other algorithms: the runtime of the portfolio is defined as the runtime of its chosen algorithm together with the feature extraction time. While costs for graphs from Section 3.2 reached up to 65 s, the maximum cost for graphs from Section 3.1 is only 6 ms.

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 [41]. $9/10^{\text{ths}}$ 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 [16]. 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 $i = 1, \ldots, n$ [24]. In this case the data is partitioned into four groups based on which algorithm performed best.

The cross-validation folds are then used to generate predictions on all of the data, where each prediction is made by a model that did not have that observation in its training data set, and for various statistics provided by the LLAMA package [18]. The predictions are then used to compare the ML model with individual algorithms and the VBS using ECDF plots and numeric statistics. LLAMA [17, 18] supports algorithm portfolios based on three different types of ML algorithms:

Classification The ML algorithm predicts which algorithm is likely to perform 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 nearest cluster.

We are using a classification algorithm called random forests [4] and its implementation in R [22]. We chose this algorithm as it is recommended in the LLAMA manual [17] and successfully used in a similar study [20]. We use the default number of trees (500), and our trees have about 6000, $20{,}000$, and $21{,}000$ vertices on average for unlabelled, vertex-labelled, and both vertex- and edge-labelled cases.

In order to discuss and analyse the ML algorithm in more detail, we introduce some new terminology. Each problem instance with features and running times in the training dataset is called an *observation*. The *class* of an observation is the algorithm with lowest running time for that problem instance. Previously discussed features of graphs are sometimes referred to as *(independent) variables*. The *(problem) instance space* is the Cartesian product of the domains of features [35], where a *domain* of X (denoted $\operatorname{dom} X$) is a set of all possible values that X can take.

A *(classification) decision tree* is "a classifier expressed as a recursive partition of the instance space" [35]. Typically, it can be represented as a rooted binary tree, where each internal node *splits* the instance space into two regions based on the value of one of the features. For example, for some feature X and a particular value $x \in \operatorname{dom} X$, the left child might be assigned all observations with $X < x$, while the right child would get observations with $X \geq x$. For each leaf node, we can count how many of its observations belong to each class and assign the most highly represented class to that node.

Remark 5.1. Other possibilities include a node having more than two children and a split being made in a more complicated way. Although trees with such properties fit the definition of a decision tree, standard machine learning algorithms are more restrictive [15, 35].

When a decision tree is used to make a classification prediction, a data point travels from node to node (starting at the root node) according to the splitting rules. When it reaches a leaf node, the class assigned to that node is outputted as the tree's prediction.

A *random forest* builds a collection of decision trees [15]. Suppose we have p variables. Then each time a split is considered, the variable to split on is chosen from $\sqrt{p}$ rather than p variables. Therefore, the strongest predictors are sometimes not even considered, ensuring a level of diversity among the trees. An individual tree's prediction is called a *vote*. A classifying random forest predicts by collecting the votes from all of the trees and predicting the class with the highest number of votes.

Lastly, we used the `parallelMap` package to train the model using multiple threads. Furthermore, the R code was heavily optimised to remove temporary variables as soon as they are no longer needed in order to reduce memory consumption.

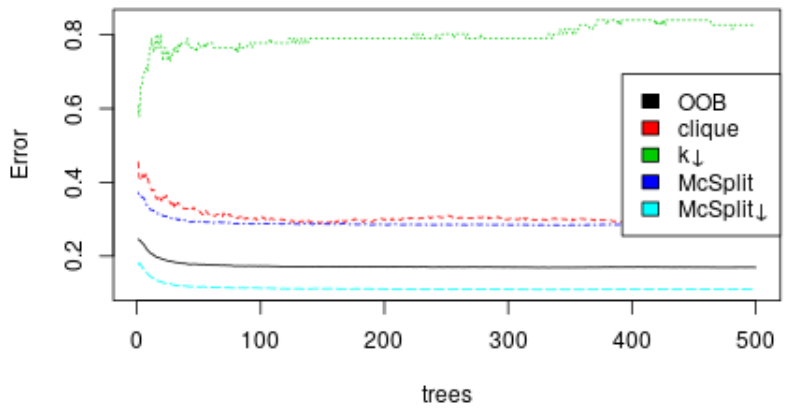


Figure 5.1: Convergence plot of various error measures as the number of trees in a random forest increases. The plot shows the OOB error and $1 -$ recall for each algorithm.

As we saw in Section 4.2, some of the feature data is highly skewed. Thus we should formally address the question of using transformations (such as $\log$ used for some of the plots or $x \mapsto 1/x$, giving the labelling percentage a different interpretation) before feeding the data into the ML algorithm. While it is a commonly held belief that predictor transformations are unnecessary for decision tree-based learning algorithms [11, 13, 40], they can affect the predictions of previously unseen data [12]. To see this, consider a split between some values $x_1, x_2 > 0$ at $b = \frac{x_1+x_2}{2}$ (which is the type of split used by the `randomForest` package we are using [12]) and consider the same situation after applying the transformation $x \mapsto x^2$. The new boundary is $b' = \frac{x_1^2+x_2^2}{2}$ and it is easy to find $x > b$ such that $x^2 < b'$. Then, if this is the final split and x_1 and x_2 have different predictions, x will have different predictions with and without the transformation. In this project we do not use transformations as having over $100{,}000$ data points and 500 trees reduces this effect and good algorithm portfolios are created without even mentioning transformations [19, 20]. However, we would encourage future researchers to consider $\log$-transforming highly skewed features, especially if using an algorithm more sensitive to the distribution of data.

5.1 Unlabelled Graphs

5.1.1 Error Rates

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 [15]. The remaining 1/3 is referred to as *out-of-bag (OOB)* observations . 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 [15]. 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 $1 -$ recall, where, for an algorithm A, *recall* [32] 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 McSplit ↓, McSplit, the clique encoding, and $k \downarrow$ converge to 11%, 29%, 30%, and 80%, respectively. Unlike the errors of all the other classes, the error of $k \downarrow$ has an upward trend and converges to a very high value. Perhaps the model eventually learns not to predict $k \downarrow$ very often and the data points with $k \downarrow$ winning are treated as randomness in the data more so than a statistically significant trend.

5.1.2 Variable Importance

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 [15]. We will refer to a part of feature space that an observation can be in while at some node n as a *region*.

Definition 5.1. 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) [15].

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 index 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 5.2. 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.

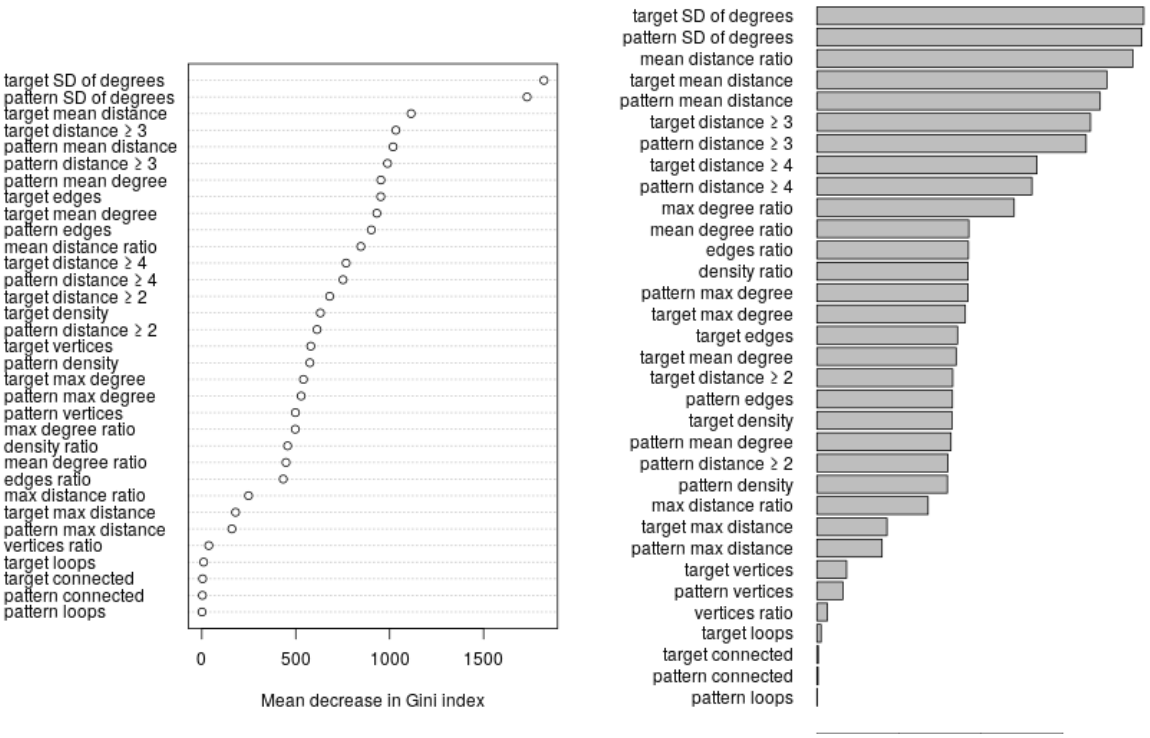
The variable importance measure of feature f in Figure 5.2a is calculated as the amount by which the Gini index decreases after passing nodes that use feature f, averaged over all trees in the random forest [15]. Looking at the figure more closely, the standard deviations of degrees of both target and 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 5$^{\text{th}}$ place and the best ratio feature, the mean distance ratio, is only 10$^{\text{th}}$. 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 pattern graphs having fewer vertices. The variable usage plot in Figure 5.2b tells a similar story: the orders are not identical, but there are no big outliers.

5.1.3 Margins

Definition 5.2. Let $c_1, \ldots, c_n$ be n classes and let p be a data point that belongs to class c_p. Let $v_1, \ldots, v_n$ denote the number of votes for each class when given p as input. The *margin* of p is

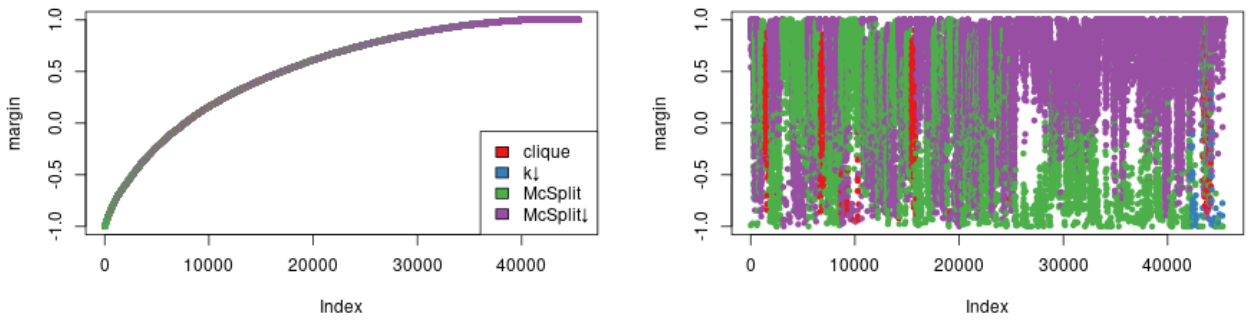
$$\frac{v_p}{\sum_{i=1}^{n} v_i} - \max_{i \neq p} \frac{v_i}{\sum_{j=1}^{n} v_j},$$

which is a number in $[-1, 1]$ [23].



(a) Dotchart of variable importance calculated based on the Gini index and sorted from most important to least important

(b) How often was each variable used to make splitting decisions?



(a) Sorted

(b) Unsorted

Figure 5.3: Margins of all the data points

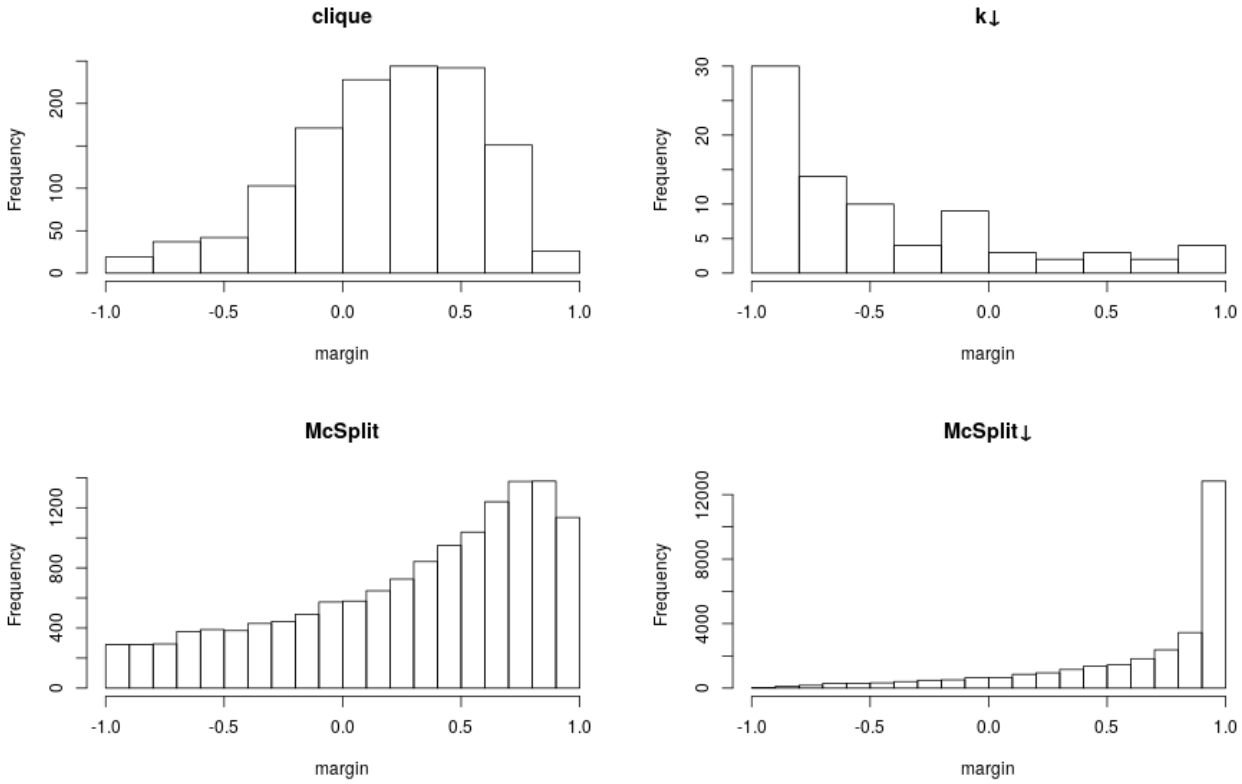


Figure 5.4: Histograms of margins for each winning algorithm

A value above 0 means that the forest as a whole predicted correctly. A margin of 1 would mean that all trees voted correctly. In Figure 5.3 we plot the margins in the following way: for each problem instance, the colour signifies the winning algorithm and the height shows the margin. Figure 5.3a shows the points sorted by the margin, while in Figure 5.3b they are left in the original order of the data (which mostly corresponds to the order of files in the databases, but with some variation since experiments are started in order but end at different times). We can recognise the same error rates as in Figure 5.1 as well as areas where MCSPLIT and MCSPLIT $\downarrow$ dominate. We also plot the histograms of how the margins are distributed for each algorithm in Figure 5.4. We note that:

- Instances best handled with the clique encoding are usually recognised, but with significant uncertainty.
- We are usually wrong about $k\downarrow$ (probably because it is a winning algorithm in only 0.44% of all cases).
- When faced with an instance that is best handled with MCSPLIT $\downarrow$, the vast majority of the trees vote correctly.
- MCSPLIT detection rates are decent, but far behind those of MCSPLIT $\downarrow$.

5.1.4 Partial Dependence

Since the standard deviations of degrees in both target and pattern graphs are the most important features, we plot partial dependence plots of the standard deviation of degrees in the target graph for MCSPLIT $\downarrow$ and the clique encoding in Figure 5.5. The plotted function [23] is defined as

$$f(x) = \log p_k(x) - \frac{1}{K}\sum_{i=1}^{K} \log p_i(x),$$

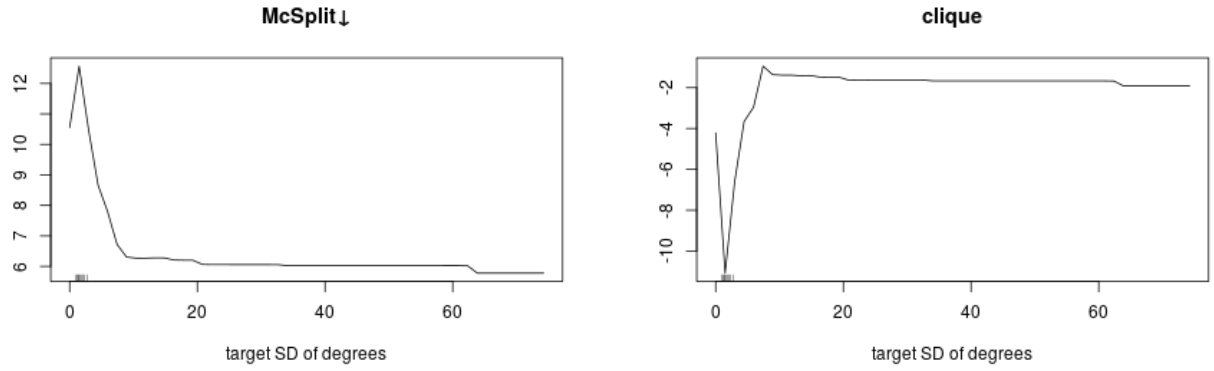


Figure 5.5: Partial dependence plots of the standard deviation of degrees in the target graph

where:

- x is the value on the horizontal axis (in this case standard deviation of degrees in the target graph),
- $p_i(x)$ is the proportion of votes for class i for a problem instance with a standard deviation of degrees in the target graph equal to x,
- K is the number of classes,
- and k is the main class under consideration (MCSPLIT $\downarrow$ and the clique encoding).

Essentially, $f(x)$ compares the proportion of votes for class k with the average value over all classes. We can deduce that a low standard deviation of degrees is a strong sign that MCSPLIT and MCSPLIT $\downarrow$ should perform well. On the other hand, the clique encoding is expected to perform better on graphs with high variance in degrees. However, $\max f(x)$ for the clique encoding is just barely above 0 and much lower than $\min f(x)$ for MCSPLIT $\downarrow$, meaning that the standard deviation of degrees does not provide enough information to choose the clique encoding over MCSPLIT or MCSPLIT $\downarrow$.

Remark 5.3. We omit the plots for the other two algorithms as the plot for MCSPLIT looks the same as the one for MCSPLIT $\downarrow$ and prediction success rate for $k\downarrow$ is so low that a plot for $k\downarrow$ would be meaningless.

Remark 5.4. The plots for the standard deviation of degrees of the pattern graph are omitted since they are identical to those of the target graph.

5.1.5 Runtime Comparison

In order to compare the LLAMA model with other algorithms, we treat the VBS as the upper bound and the single best solver MCSPLIT $\downarrow$ as the lower bound. Out of 45468 instances solved by at least one algorithm, our model managed to solve 45290, compared to 45223 solved by MCSPLIT $\downarrow$. In other words, it was able to close 27.3% of the gap between our lower and upper bounds in terms of instances solved within the time limit. Figure 5.6 shows how the ML model compares to the VBS and the single best solver MCSPLIT $\downarrow$ (note that the vertical axis starts at 0.9 rather than 0). Unsurprisingly, the model outperforms MCSPLIT $\downarrow$, but does not reach the best possible performance represented by the VBS.

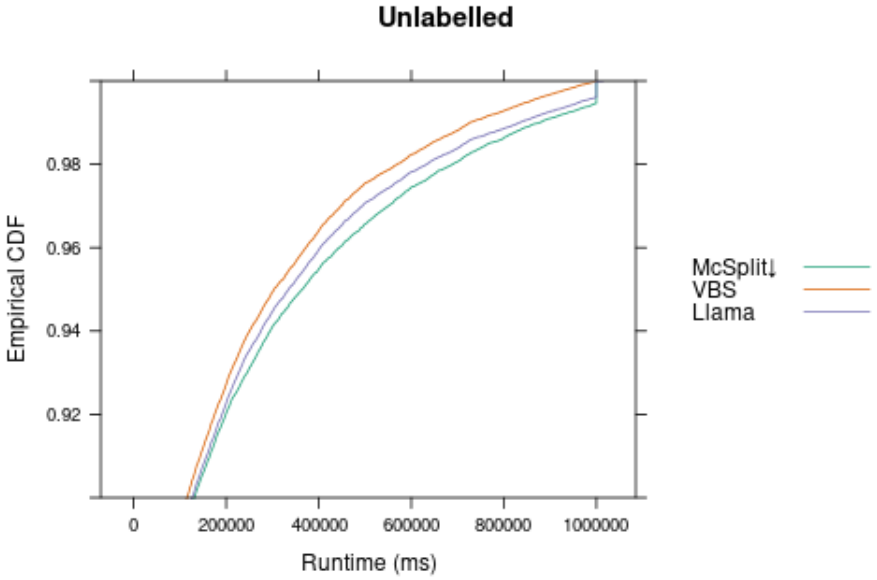


Figure 5.6: LLAMA model compared to the VBS and MCSPLIT ↓

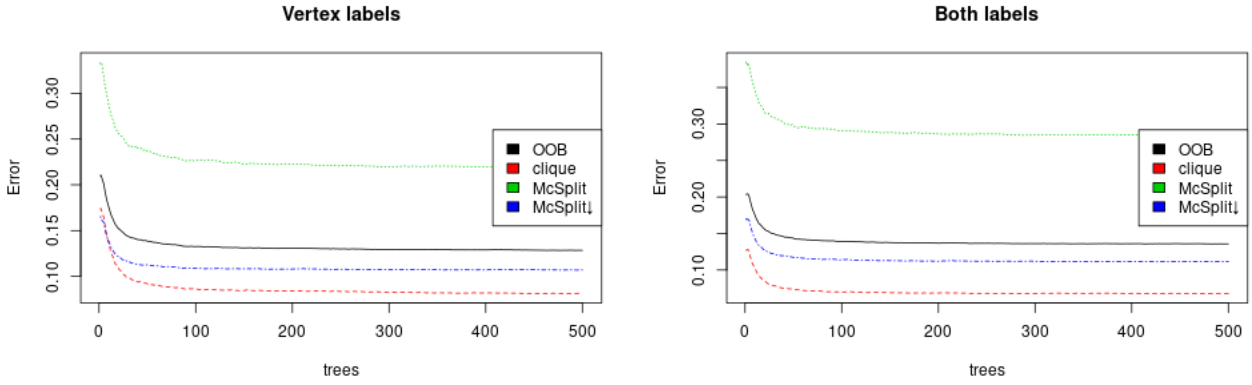


Figure 5.7: Convergence plots of various error measures as the number of trees in a random forest increases. The plots show the OOB error and $1 -$ recall for each algorithm.

Error type	Final value for vertex labels (%)	Final value for both labels (%)
OOB	13	14
clique	8	7
MCSPLIT	22	29
MCSPLIT ↓	11	11

Table 5.1: The values that all 4 errors (approximately) converge to, for both types of labelling

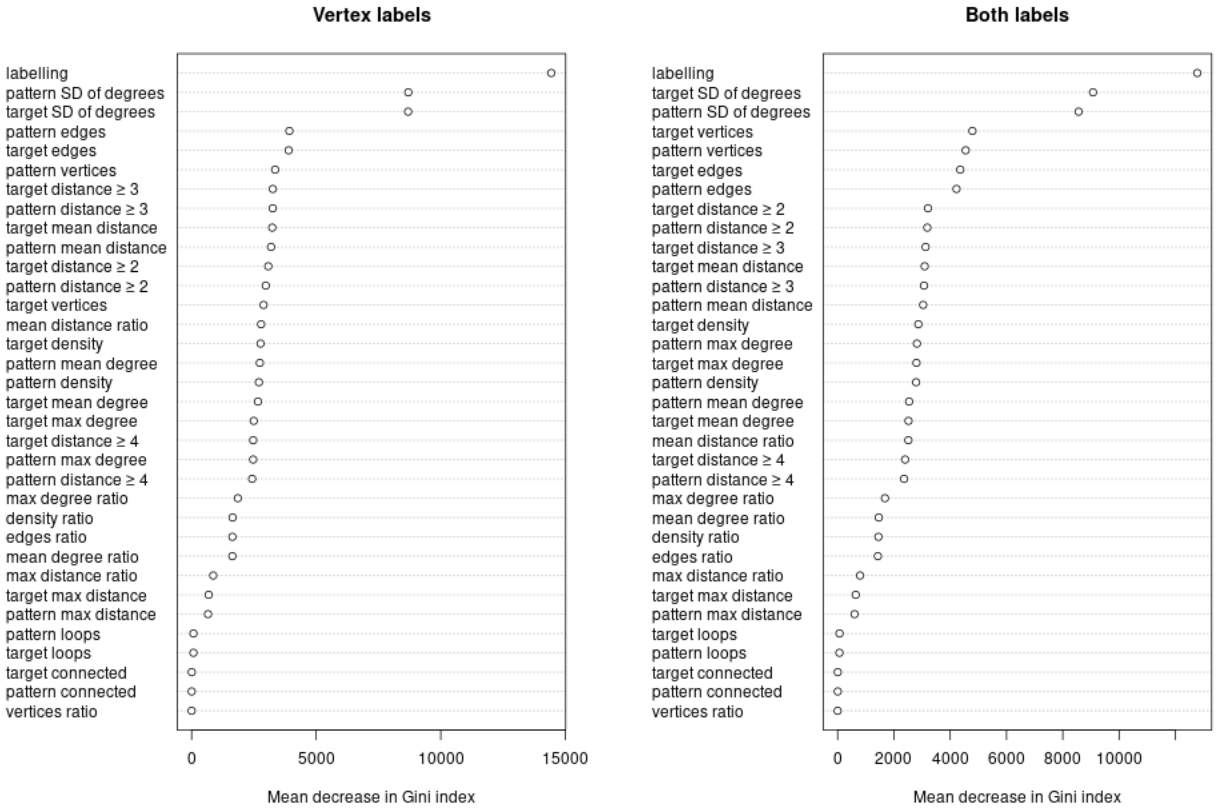


Figure 5.8: Variable importance for both types of labelling, sorted from most to least important

5.2 Labelled Graphs

Once again we plot the error rates (defined in Section 5.1.1) in Figure 5.7. The final (approximately limit) values are summarized in Table 5.1. This time all of the errors converge downwards and are below 50%. Comparing vertex-labelled and both vertex- and edge-labelled subproblems, the only noticable difference is that McSplit has a lower error rate for vertex-labelled graphs, which is also the worst-recognised algorithm for both types of labelling.

According to the variable importance measures plotted in Figure 5.8, the standard deviation of degrees for both pattern and target graphs still act as top predictors, however, they are overshadowed by the labelling feature, which is to be expected considering how impactful it is to the performance of the clique algorithm and the difficulty of the problem in general. For graphs with both vertex and edge labels, the vertex/edge counts make up the next most important predictors, while for vertex-labelled graphs, the number of vertices in the target graph is seems to be less important (perhaps simply due to chance). Comparing this with the variable usage statistic in Figure 5.9 (which is almost identical for the two subproblems), the top 3 spots remain the same, the numbers of edges drop to the middle of the list, and the numbers of vertices drop to the 6^{th}–7^{th} places from the bottom. Apparently, even though all 4 of these predictors end up doing a great job at splitting the data to reduce the Gini index (a variation of which is used by the random forest algorithm in choosing which predictor to split on [43]), they are rarely used.

We show the margin scatter plots and histograms in Figures 5.10 and 5.11. Again, the data for both types of labelling is close to identical. Note that there are plenty of data points in all 3 colours and the ML models are usually very convinced when predicting McSplit ↓ and the clique encoding and are less sure when dealing with problem instances best handled with McSplit.

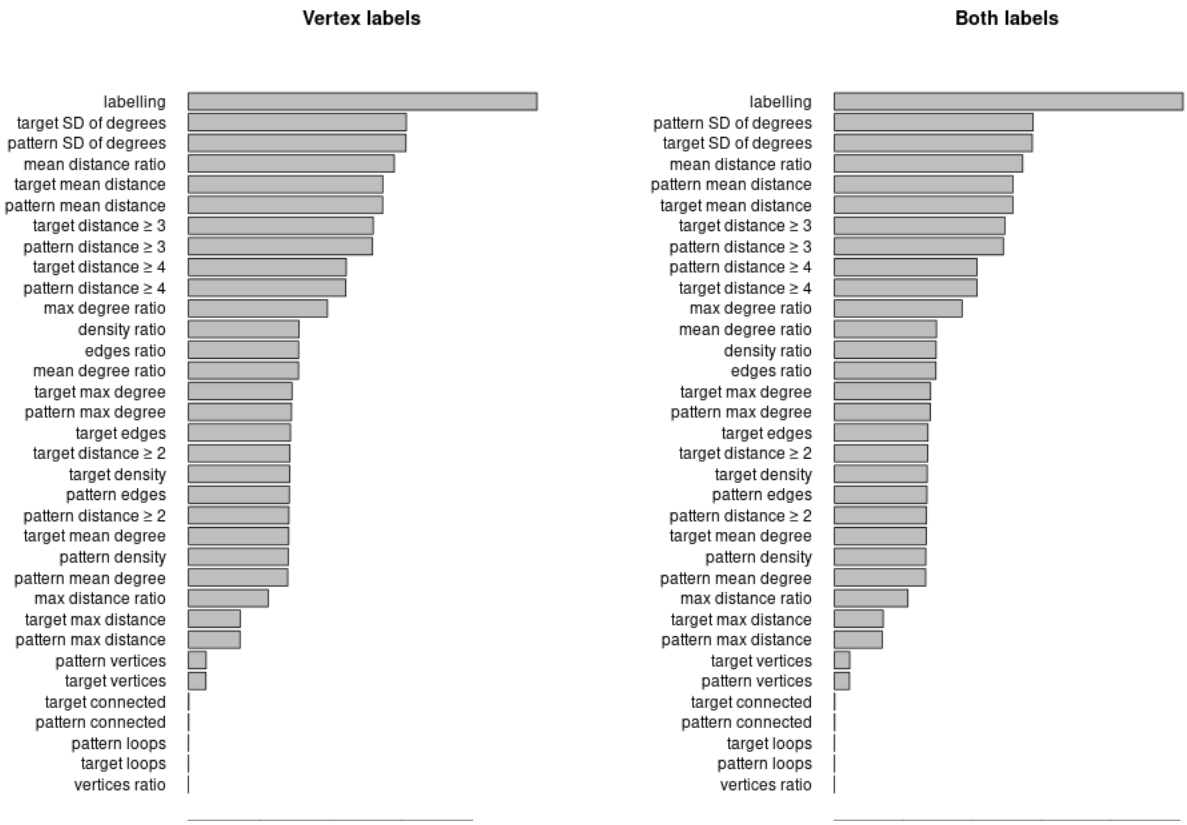


Figure 5.9: How often was each variable used to make splitting decisions?

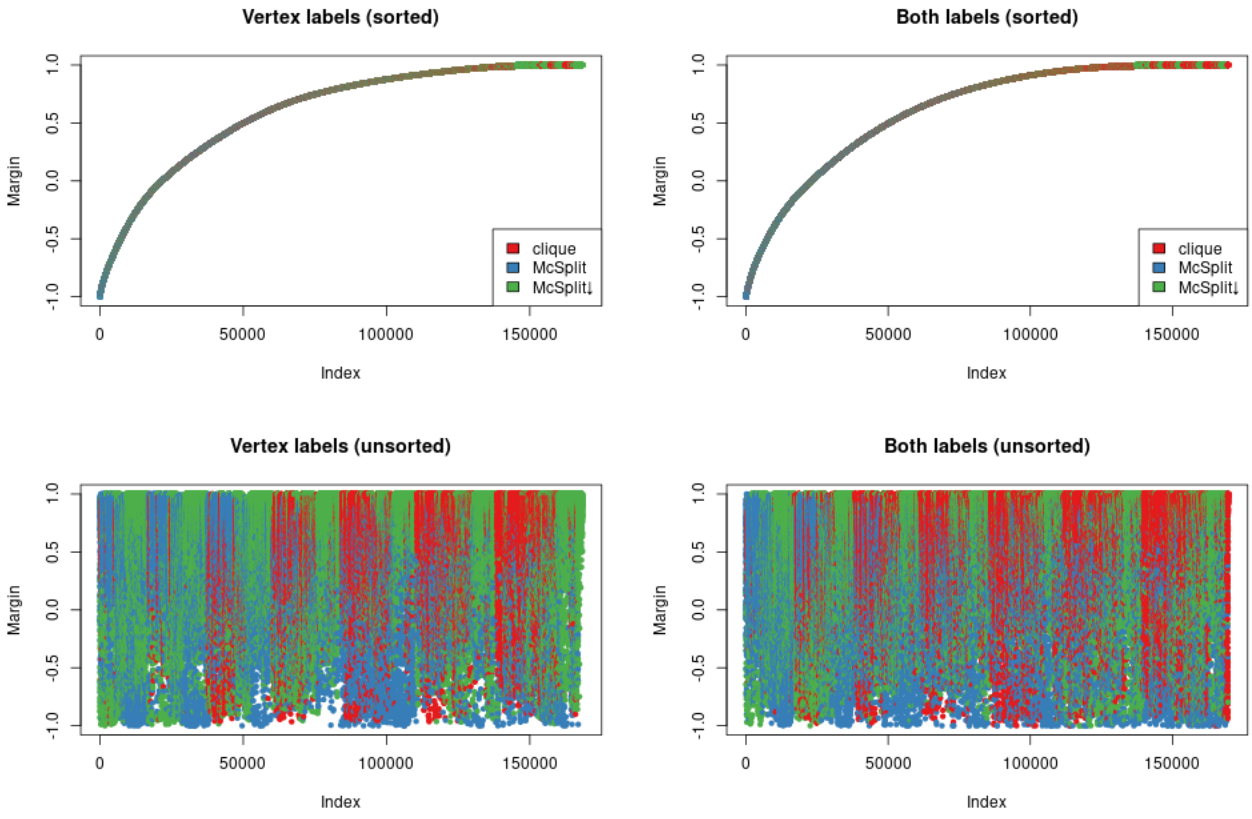


Figure 5.10: Sorted and unsorted margins of all data points, for both types of labelling

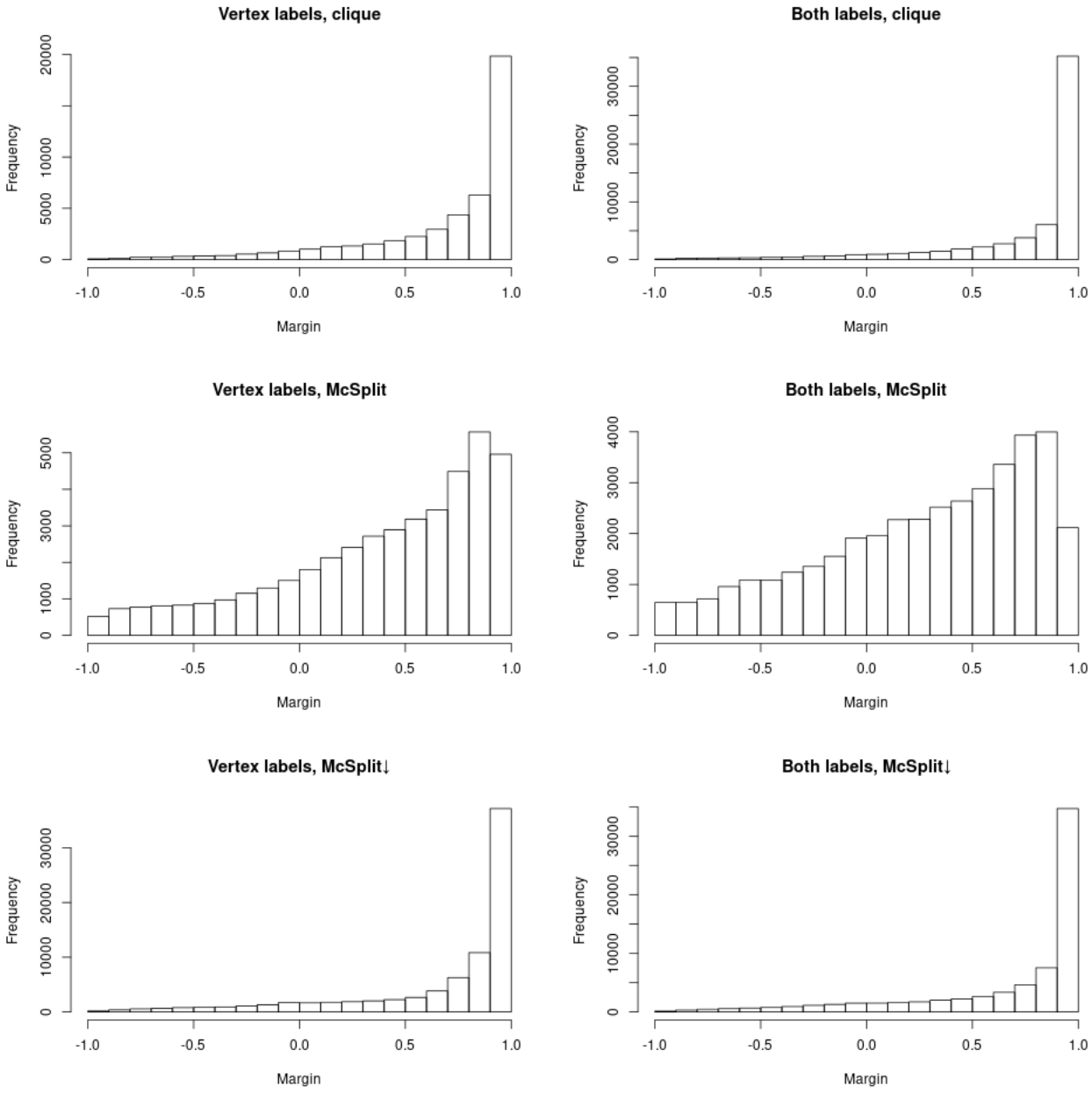


Figure 5.11: Histograms of margins, for each algorithm and type of labelling

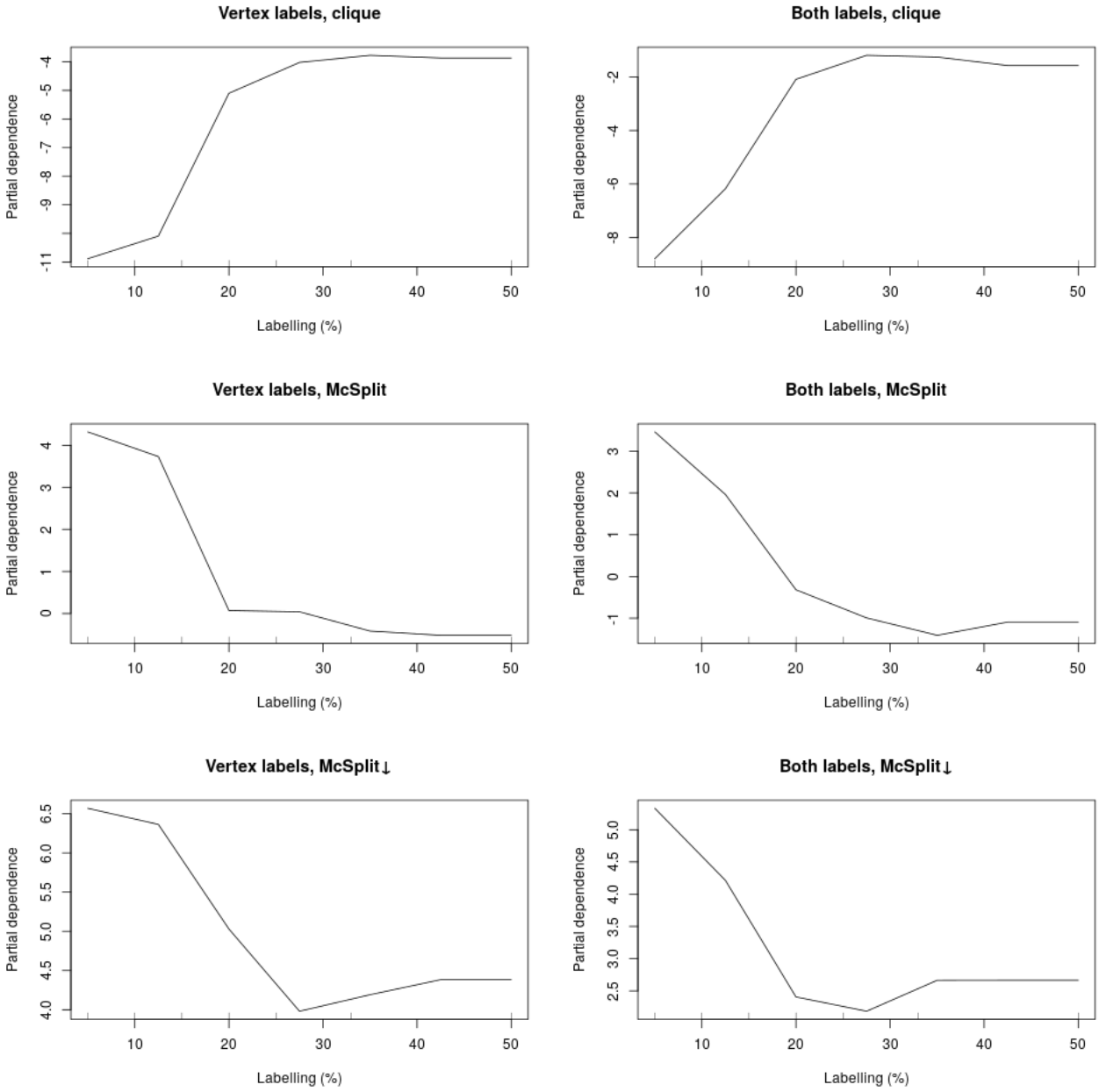


Figure 5.12: Partial dependence plots of the labelling percentage

Figure 5.13: Partial dependence plots of the standard deviation of degrees in the target graph

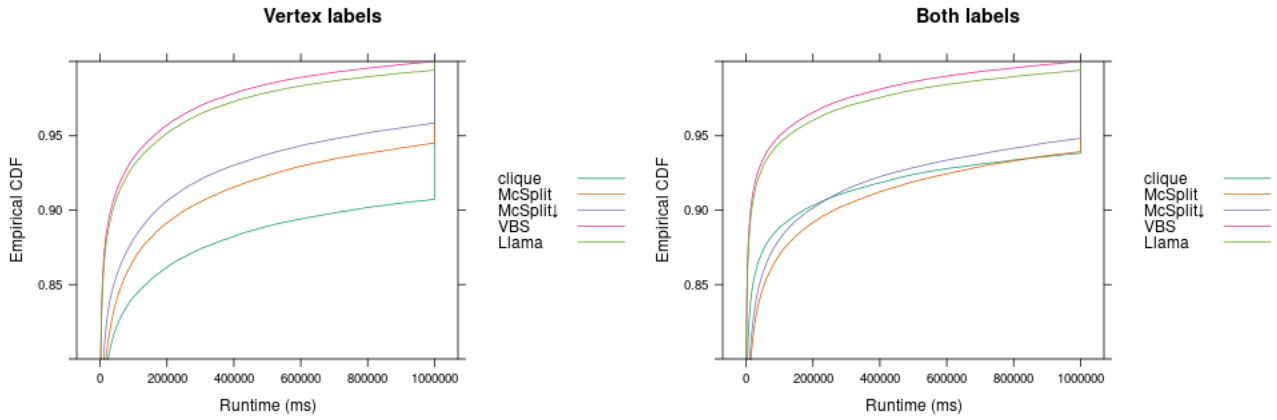


Figure 5.14: LLAMA models compared to the other algorithms and the VBS

This time we show the partial dependence plots for the top 2 most influential predictors (the labelling percentage and the standard deviation of degrees in the target graph) in Figures 5.13 and 5.12. The results for the clique encoding are the most straightforward to interpret. Clearly, the algorithm performs much better with higher labelling percentages (more different labels). The interesting bit (just like back in Figure 4.4) is that the curve stays constant between 20% and 50%. In other words, anywhere above 20% labelling, a higher labelling percentage does not make the clique encoding more preferable than it already is. The partial dependence on the standard deviation of degrees of the target graph says what we already knew from Section 5.1.4: the clique encoding prefers graphs with more variance in degrees. The one smalll difference is that this time the change is not as steep, but this can be easily explained by the fact that the highest standard deviation of degrees is around 7 in this section compared to around 70 in Section 5.1.4.

On the other hand, both MCSPLIT and MCSPLIT ↓ prefer lower labelling percentages. While MCSPLIT ↓ prefers graphs with smaller standard deviations of degrees, the situation with MCSPLIT is suddenly less clear, which is reflective of the fact that the ML model has less confidence about MCSPLIT (compared to the other 2 algorithms).

Finally, for the runtime comparison, we plot the ECDFs for all individual algorithms, the VBS, and the LLAMA models in Figure 5.14. While there are important differences among individual algorithms' curves (discussed in Section 4.1.2), the way LLAMA behaves with respect to the VBS and MCSPLIT ↓ curves is about the same: LLAMA's curve is quite close to optimal, and the gap between LLAMA and MCSPLIT ↓ is significantly wider than the gap between MCSPLIT ↓ and MCSPLIT. To add more numbers to this picture, LLAMA closed 86% and 88% of the difference between its lower and upper bounds for vertex-labelled and both vertex- and edge-labelled graphs, respectively.

Bibliography

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

[2] László Babai, Paul Erdös, and Stanley M. Selkow. Random graph isomorphism. *SIAM J. Comput.*, 9(3):628–635, 1980.

[3] Bernd Bischl, Pascal Kerschke, Lars Kotthoff, Marius Thomas Lindauer, Yuri Malitsky, Alexandre Fréchette, Holger H. Hoos, Frank Hutter, Kevin Leyton-Brown, Kevin Tierney, and Joaquin Vanschoren. Aslib: A benchmark library for algorithm selection. *Artif. Intell.*, 237:41–58, 2016.

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

[5] Donatello Conte, Pasquale Foggia, and Mario Vento. Challenging complexity of maximum common subgraph detection algorithms: A performance analysis of three algorithms on a wide database of graphs. *J. Graph Algorithms Appl.*, 11(1):99–143, 2007.

[6] 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.

[7] 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.

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

[9] 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.

[10] 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.

[11] Jerome H. Friedman. Recent advances in predictive (machine) learning. *J. Classification*, 23(2):175–197, 2006.

[12] Tal Galili and Isaac Meilijson. Splitting matters: how monotone transformation of predictor variables may improve the predictions of decision tree models. *CoRR*, abs/1611.04561, 2016.

[13] Trevor Hastie, Robert Tibshirani, and Jerome H. Friedman. *The elements of statistical learning: data mining, inference, and prediction, 2nd Edition*. Springer series in statistics. Springer, 2009.

[14] 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.

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

[16] 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.

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

[18] Lars Kotthoff, Bernd Bischl, Barry Hurley, and Talal Rahwan. *Leveraging Learning to Automatically Manage Algorithms*. CRAN, December 2015.

[19] 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.

[20] 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.

[21] Giorgio Levi. A note on the derivation of maximal common subgraphs of two directed or undirected graphs. *CALCOLO*, 9(4):341, Dec 1973.

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

[23] Andy Liaw and Matthew Wiener. *Breiman and Cutler's Random Forests for Classification and Regression*. CRAN, October 2015.

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

[25] 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.

[26] 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.

[27] 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.

[28] 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.

[29] Samba Ndojh Ndiaye and Christine Solnon. CP models for maximum common subgraph problems. In Jimmy Ho-Man Lee, editor, *Principles and Practice of Constraint Programming - CP 2011 - 17th International Conference, CP 2011, Perugia, Italy, September 12-16, 2011. Proceedings*, volume 6876 of *Lecture Notes in Computer Science*, pages 637–644. Springer, 2011.

[30] Takao Nishizeki and Md. Saidur Rahman. *Planar Graph Drawing*, volume 12 of *Lecture Notes Series on Computing*. World Scientific, 2004.

[31] Panos M. Pardalos and Jue Xue. The maximum clique problem. *J. Global Optimization*, 4(3):301–328, 1994.

[32] 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.

[33] 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.

[34] John R. Rice. The algorithm selection problem. *Advances in Computers*, 15:65–118, 1976.

[35] Lior Rokach and Oded Maimon. *Data Mining with Decision Trees - Theory and Applications. 2^{nd} Edition*, volume 81 of *Series in Machine Perception and Artificial Intelligence*. WorldScientific, 2014.

[36] 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.

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

[38] 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.

[39] Robert R. Stoll. *Set Theory and Logic*. Dover books on advanced mathematics. Dover Publications, 1979.

[40] Roman Timofeev. Classification and regression trees (CART): Theory and applications. Master's thesis, CASE - Center of Applied Statistics and Economics, Humboldt University, Berlin, December 2004.

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

[42] Eric W. Weisstein. k-subset. *MathWorld – A Wolfram Web Resource*, April 2004.

[43] Soren H. Welling. Does Breiman's random forest use information gain or Gini index? Cross Validated, August 2015.

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

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