Final Year Project

Maximum Clique Enumeration

William O'Meara

Student ID: 16440764

A thesis submitted in part fulfilment of the degree of

BSc. (Hons.) in Computer Science

Supervisor: Deepak Ajwani

UCD School of Computer Science

University College Dublin

May 20, 2020

Table of Contents

1 Project Specification 4

1.1 Core 4

1.2 Advanced 4

2 Introduction 5

2.1 Maximum Clique Enumeration 6

3 Related Work and Ideas 7

3.1 Combinatorial optimization problems 7

3.2 Machine learning and Combinatorial Optimization 8

3.3 Maximum clique enumeration 9

3.4 Maximum Clique Enumeration 10

3.5 Maximal Clique Enumeration 11

3.6 Probabilistic preprocessing 11

3.7 Multi-stage sparsification 12

4 Data Considerations 13

5 Outline of Approach 14

5.1 Edge Framework 14

5.2 Graph Theoretic Features 15

5.3 Chi Square Statistic 21

5.4 Model Selection 22

5.5 Feature Selection 23

5.6 Domain oblivious training 24

6 Evaluation 26

6.1 Sparse Graphs 26
6.2 Medium Graphs 27
6.3 Dense Graphs 29
7 Conclusions and Future Work 31
7.1 Dynamic classification threshold 31
7.2 Multi stage sparsification 31
8 Acknowledgements 33
9 Appendix 37

Abstract

A number of optimization problems in diverse domains such as data mining, decision-making, planning, routing and scheduling are NP-hard. Which means no efficient polynomial-time algorithms are known for these problems. This project focuses on the NP-hard problem of maximum clique enumeration. Maximum clique enumeration is a fundamentally computationally difficult problem that arises in various network analysis tasks. I am exploring a supervised learning framework to predict whether an edge in the graph is part of some maximum clique. The edges that are predicted to not be part of any maximum clique are pruned away, reducing the size of the problem for the exact optimization algorithm. There exist accurate solvers for this problem however they can only deal with instances containing a few thousand elements. There also exist some effective machine learning approaches focusing on this problem. This project shows that better results can be achieved through the use of advanced feature engineering and the use of an edge framework rather than a node framework.

Chapter 1: Project Specification

1.1 Core

1. Develop a Python code to (i) extract simple local features of edges in a graph and (ii)learn a classification model to map the edges to a binary score predicting whether orb not the edge is part of some maximum clique
2. Evaluate the accuracy of the solutions obtained on a range of graphs

1.2 Advanced

1. Careful feature engineering to identify features of edges that improve the accuracy of the prediction
2. Improve the accuracy further by adaptively boosting the classification model

Chapter 2: **Introduction**

A number of optimization problems are computationally hard (NP-hard) which means there is no existing polynomial time algorithm that can solve every instance of the problem optimally, many any researchers consider that such algorithms may not even exist. However in recent years researchers have been exploring if machine learning techniques can be used to solve these types of problems.

Recent developments in deep learning and graph convolutional networks (neural network that operates on graphs) have the ability to learn the output of an optimization problem based on small training examples e.g.[1][2][Vinyals et al., 2015; Bello et al., 2016). However, these techniques are not widely used as they require large amounts of training data and obtaining this data requires solving the NP-hard optimization problem which can is computationally expensive. This results in the fact that these techniques can only really be used effectively on small data sets.

Recently Lauri and Dutta [3] proposed a probabilistic preprocessing framework to address these NP-hard problems. The idea is that instead of directly learning the output of the NP-hard optimization problem their approach is to prune away a part of the input. This smaller input can then be solved with state-of-the-art-solvers. They considered the problem of maximum clique enumeration (A maximum clique of a graph G is a complete subgraph of maximum possible size for G.) and their approach pruned 75% -98% of the vertices on sparse graphs. Their approach however did suffer from poor pruning on dense instances, poor accuracy on larger synthetic instances and non-transferability of training models across domains (Using knowledge acquired from one task to solve related ones).

In this project build upon this work and build upon the work of Grassia et al. [4] who looked at the same problem and produced a higher accuracy pruning trade-off, both on sparse and dense graphs as well as achieving cross-domain generalizability using multi-stage learning methodology. I will attempt to improve on this design by changing the framework from a node framework to an edge framework, this means instead of extracting information from the nodes (vertices) I will look to gather the information from the edges. The advantages of this is that an edge framework can look at the relative positioning of the nodes and when we observe an edge, we can extract information from its two vertices knowing they are connected to each-other. I also wish to explore the possibility of improved graph theoretic features which could improve the accuracy of the solution.

2.1 Maximum Clique Enumeration

The goal of the maximum clique enumeration (MCE) problem is to list all maximum cliques in a given graph. A clique is a subset of vertices such that every two vertices are adjacent. A maximal clique is a clique that cannot be extended by including one more adjacent vertex, meaning it is not a subset of a larger clique. A maximum clique is the clique of the largest size in a given graph, a maximum clique is therefore always maximal, but the opposite is not always true. It is one of the most studied combinatorial problems arising in various domains such as in the analysis of social networks[5][6] financial networks [7] and behavioural networks [8]. It is also relevant in clustering [9][10] and cloud computing [11]

Even approximating the maximum clique problem is a NP-hard problem[12] and the problem of identifying if a graph has a clique size of k is not solvable in polynomial time [13] because of this even small instances of this problem can be non-trivial to solve. Furthermore, there is no polynomial time algorithm that preprocesses an instance of k-clique to have only f(k) vertices where f is a function depending solely on k [14]. These results indicate that it is unlikely that an efficient preprocessing method for MCE exists that can reduce the size of the input instance drastically while guaranteeing to preserve all maximum cliques. This has led researchers to focus on heuristic pruning approaches.

Chapter 3: Related Work and Ideas

3.1 Combinatorial optimization problems

Combinatorial optimization is the mathematical study of the arrangement, grouping, ordering or selection of discrete objects, usually finite in number [15][16]. Combinatorial optimization problems have been solved using many different strategies. Exact algorithms can be used to find the optimal solution to these types of problems however many of the combinatorial optimization problems are NP-hard and using exact solvers on NP-hard problems is computationally expensive. These solvers, through the use of various heuristics can be effective when the graph has certain structural properties but detecting these properties can be computationally expensive. This is often the case for extremely large graphs that you may encounter in fields such as data mining or computational biology [17]

Therefore, applications rely on approximation algorithms and/or domain specific heuristics and/or meta-heuristics for solving these problems.

Approximation Algorithms: Approximation algorithms are much more efficient algorithms that can find approximate solutions to NP-hard optimization problems with a knowledge that the approximation of the returned solution is accurate. These approximations can then be used to solve the NP-hard problem more easily due to the reduced problem set. Approximation algorithms involve a mathematical proof ensuring the correctness of the returned solution in its worse possible case [18]. One example of an approximation algorithm is Minimum Vertex Cover Problem [19].

Domain-Specific Heuristics: A heuristic is a function that ranks options in search algorithms at each step based on the current information available to it. Heuristics seek near-optimal solutions at a reasonable cost while not being able to guarantee feasibility or optimality. When dealing with combinatorial optimization there are generic methods like Branch and Bound or there are domain-specific heuristics that focus more on the problem in particular rather than a generic solution.

Metaheuristics: A metaheuristic is a high-level procedure that finds heuristics that may provide a sufficiently good solution to an optimization problem. Techniques which constitute metaheuristic algorithms can vary from simple local search procedures to complex learning processes. An example of a metaheuristic search method solving combinatorial optimization problems is simulated annealing and the travelling salesman problem. [20]

Integer Linear Programming Solvers/ Constraint Programming Solvers: An *Integer linear program* is a linear program with the additional constraint that the variables are only allowed to take on integer values. The idea behind an integer linear program is to maximize some parameter while keeping certain constraints to be true. Solving integer linear programs is often NP-hard [21] The strategies for approximating or solving integer linear programs usually try to solve relaxations of the original program, which can be obtained by dropping some of the inequalities(this cannot be done in constraint programming solvers).

A *constraint programming* optimization model has the same structure as an integer linear programming model; a set of variables, a function to maximize (or minimize) and a set of constraints. The differences between a constraint programming model and a integer linear programming model is that constraint programming models can use specialized constraints such as the all-different constraint, it also has no limitation on the arithmetic constraints and it can only support discrete decision variables while a linear programming model supports either discrete of continuous decision variables.

3.2 Machine learning and Combinatorial Optimization

It is only in recent years that researchers have considered if machine learning techniques can be used to tackle NP-hard optimization problems.

The metaheuristic literature navigates the search space to find good solutions. To avoid getting stuck in local minima, they often used randomization. These techniques combine the greedy search for the best solution with controlled random exploration. For instance, in simulated annealing, a lot of random exploration is allowed in the initial stage, but as the number of iterations increase, the search focuses more and more on the greedy best solution found so far and the randomness in exploration is reduced. In contrast to the random exploration, learning techniques use training examples to guide the search towards good (or even optimal) solutions. Thus, machine learning techniques have the potential to converge to good solutions quicker than the heuristic methods described.

Some of these new techniques includes supervised learning, reinforcement learning and unsupervised learning. We outline one approach from each of these learning techniques:

- *Supervised Learning:* In supervised learning a set of input (features) is provided and the task is to find a function that for every input a predicted output is returned that is as close as possible to the optimal solution. A good example of supervised learning

in the context of combinatorial optimization is the approach of Oriol et al. [1]. They used a sequence-to-sequence Recurrent Neural Network (RNN) and Attention mask to learn the output directly from the input. Their input is a sequence of tokens (e.g., corresponding to positions of point set P) and the output is a subset of those tokens (e.g., a set of points in convex hull of P) and their approach uses the sequence-to-sequence learning to learn the output sequence directly from the input sequence.

Another example of supervised learning being used to solve combinatorial optimization problem is the approach by Kool et al. [22], who introduced a model and training method which greatly contributed to improved results on learned heuristics for Travelling Salesman Problem. They also introduced a powerful starting point for learning heuristics for other combinatorial problems defined on graphs.

- *Reinforcement Learning:* In reinforcement learning we interact with the environment through a markov decision process. At each step the system is presented with the current state of the environment and chooses an action according to in most cases its stochastic policy. Reinforcement learning has been used to tackle combinatorial optimization problems, in particular the traveling salesman problem [2]

  Khalil et al. [23] presented an end to end machine learning framework for automatically creating greedy heuristics for NP-hard combinatorial optimization problems on graphs. Their approached relied on a combination of graph embedding and reinforcement learning. The embedding of a graph is a drawing of the graph in such a way that its edges may only intersect at their endpoints. This proposed framework was effective in learning greedy heuristics and the high performance across all graph types suggested that the framework could be a promising new mechanism for graph problems.

- *Unsupervised learning:* is a type of machine learning algorithm that is used to draw logical conclusions from data sets consisting of only input data. An example of how unsupervised learning has been used to tackle combinatorial optimization problems is in Probst et al. [24] where in depth analysis is carried out using a Restricted Boltzmann Machine within an Estimation of Distribution Algorithm for combinatorial optimization.

3.3 Maximum clique enumeration

Take a graph G = (V, E) where V and E represent vertices and edges respectively. A clique C is a subset of vertices in G such that every two distinct vertices in C are adjacent in G.

The clique 3.2 represents a clique as all the vertices are connected to eachother while the clique 3.2 is not a clique as the nodes A and E are not connected.

A *maximal clique* is a clique that cannot be extended by including any more adjacent vertices. The clique 3.3 is an example of a maximal clique as if we were to include D, D and A would not be connected and likewise with E.

A *maximum clique* of a graph G is a clique such that there is no other clique with more vertices. The clique 3.4 is a maximum clique as it cannot be extended by including any more adjacent vertices (maximal) and it is the largest maximal clique.

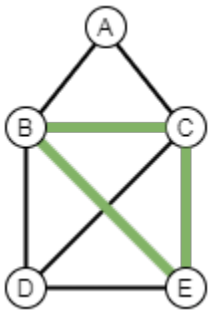

Figure 3.1: Clique

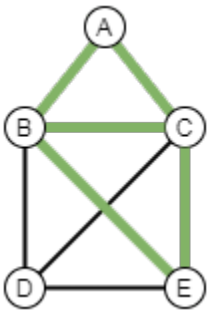

Figure 3.2: Not a clique

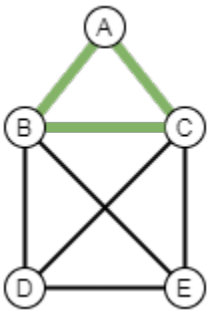

Figure 3.3: Maximal

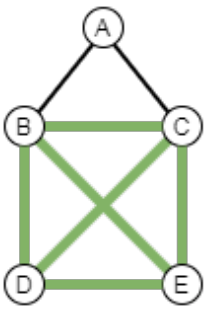

Figure 3.4: Maximum

3.4 Maximum Clique Enumeration

The *Maximum clique enumeration problem* asks to identify all maximum cliques in a finite, simple graph. It is one of the most studied combinatorial problems arising in various domains such as in the analysis of social networks [5], financial networks [7] and behavioural networks [8]. It is also relevant in clustering [25] and cloud computing [26]. One of the most interesting examples of its use is its application in biological research, for example with proteins it is important to know if a pair of proteins contains similar folding patterns as this may give us information regarding common evolutionary origins. The folding patterns of proteins can be converted to complex undirected graphs with edges and vertices and if we were to compare the maximal clique enumeration of two of these graphs, we can gather information on common evolutionary origins between two protein structures.[27]

State-of-the-art solvers exist for maximum clique enumeration, they employ various heuristics that are effective when the input graph has certain structural properties. These solvers work as follows: (1) they quickly find a large clique of size k. (2) compute the core number of each vertex of the input graph G (how many edges each vertex has) and (3) delete every vertex of G with core number less than k - 1.There are however two large downsides to these solvers, first one is that they are dependent on k (the size of a large clique found) since maximum clique size is NP-hard to approximate, maximum clique sized with no formal guarantees are used. Secondly even if k was equal to the size of a maximum clique in a graph, the core number of most vertices could be much higher than k - 1 which would

result in little or no pruning. This can often be true in very dense graphs.

It is for these reasons that a preprocessing approach is beneficial which was shown by Lauri and Dutta [3] and the further work done by Grassia et al.[4] with their multi-stage sparsification learning framework for maximum clique enumeration.

3.5 Maximal Clique Enumeration

There exist algorithms for maximal clique enumeration [28]. While maximum clique enumeration of a graph G is a complete subgraph(s) of maximum possible size for G, maximal clique enumeration is the enumeration of every clique that cannot be extended by including one more adjacent vertex meaning it is not a subset of a larger clique. This set of graphs would include the maximum clique along with all the other maximal cliques as a maximum clique is always maximal however the converse is not always true. Maximal clique enumeration and Maximum clique enumeration have very different practical applications, furthermore the runtime of maximal clique is much higher as it is required to return all the maximal cliques not just the maximum clique.

3.6 Probabilistic preprocessing

A probabilistic preprocessing framework for fine-grained search space classification was proposed by [3]. This framework was for reducing the search space for NP-hard enumeration problems and for altering state-of-the-art solvers with cues from the input distribution.

The idea was to look at the individual vertices of a graph G = (V, E) as classification problems the problem of learning a preprocessor reduces to that of learning a mapping : $V- > 0, 1$ from a set of L training examples $T =< f(v_i).y_i >$i=1 , L where v_i is a vertex and y_i $0, 1$ is a class label and d f : $V- > R^d$ a mapping from a vertex to a d-dimensional feature space.

To learn the mapping from T a probabilistic classifier P is used which outputs either 0 or 1 for a given f(u) for a given vertex u. Then all the vertices from G that are not predicted by P to not be in a solution with a probability of at least q are pruned away, q trades off accuracy for pruning rate. This framework showed that there was potential for learning a heuristic processor for instance size pruning, however the speedups obtained were limited and the training models were not transferable across domains.

3.7 Multi-stage sparsification

In order to improve upon the probabilistic preprocessing framework for fine-grained search space classification proposed by Lauri and Dutta [3] a Multi-stage sparsification approach was implemented by Grassia et al.[4]. One core problem with the probabilistic preprocessing approach is that the model focuses too much on pruning out the easy cases such as vertices of low degree and not the cases of vertices with high degree and high core number which are much harder to diagnose. The Multi-stage sparsification approach attempts to address this. Each stage the approach focuses on gradually harder cases that were difficult to prune by the classifier in earlier stages. Consider a graph $G \subset G_1$ where G_1 is the input set of networks. Let M be the set of all maximum cliques in G and $V(M)$ is the set of all vertices contained in a maximum clique. The positive examples set by the learning algorithm consist of all the vertices that are in some maximum clique V(M) and the negative ones are the set $V \setminus V(M)$. A probabilistic classifier P_1 is trained on the training data in stage 1. Then in the following stage after the removal of all vertices that were predicted by P_1 to be in the negative class (with a probability over a certain threshold) we focus on G_2 the resulting subgraphs of G_1. We repeat the process on G_2 and on the resulting G_3 and so on until we reach G_l.

In contrast to the known exact pre-processing methods their approach required no estimate for the maximum clique size at runtime which is a task NP-hard to even approximate. They showed that their approach could routinely prune over 99 of vertices in sparse graphs and more importantly this approach could prune around 30 percent of the vertices on dense graphs which is considerably more than existing methods.

In my project I plan on improving on the Multi-stage sparsification design by changing the framework from a node framework to an edge framework and through the use of new features that can improve accuracy.

Chapter 4: Data Considerations

In my project I will be using data from websites such as http://networkrepository.com/ and https://snap.stanford.edu/data/. For many of the graphs in these datasets, the ground truth of maximum clique has been obtained using state of the art heuristics and solvers.

In order to demonstrate the usefulness of my framework I plan on focusing on various networks such as biological networks and social networks for both my training set and for testing.

The size of the data I will be dealing with will be very large, as the focus of this project is to use machine learning to explore big instances of optimization problems. The data will likely be structured, and it will not be too difficult to convert the text data into a graph using python packages like network.

Chapter 5: Outline of Approach

In this project I focused on learning to prune instances of maximum clique enumeration. The key idea here was to view it as a classification problem such that the learned model can discriminate between nodes that are part of the maximum clique and nodes that are not. The recently proposed multi-stage sparsification framework [4] learns to prune nodes that are not part of the maximum clique (relatively easy cases of nodes far from decision boundary) so that the solvers can focus on the hard part of the problem.

I will explore if the accuracy of the framework by Grassia et al [4] can be improved by classifying edges instead of nodes. The advantage of this is that an edge framework can look at the relative positioning of the nodes and when we observe an edge, we can extract information from its two vertices knowing that they are connected. Some of these edge-based features have been explored by Grassia et al. [4] but I explore the improved graph theoretic features which improve the accuracy of the solution.

In contrast to previous work I use an adaptive boosting classifier [29] which in combination with the new graph theoretic features I introduce produced impressive results.

5.1 Edge Framework

Here I describe how the edge framework will functions; The classifier learns the characteristics of edges in terms of various neighborhood features and based on this it prunes the set of edges that are likely not contained in any maximum clique.

If we were to consider a graph $G \varepsilon G_1$ where G_1 is the input set of networks, let M be the set of all maximum cliques in G and E(M) is the set of all edges contained within the maximum clique. A probabilistic classifier (a classifier that will predict, given information from an input a probability distribution over a set of classes) will be trained using a training set T1. The positive examples of the training set will consist of all the edges that are in the maximum clique (E(M)) and the negative ones are the set E(M) (the set of edges not in the maximum clique). This will be a skewed dataset as in nearly all cases of sparse graphs there will be much more edges not in the maximum clique than there are in the maximum clique. On the other hand in the case of very dense graphs it can often be the case where the number of maximum clique instances are larger than the number of cases

not in the maximum clique. Therefore, we under-sample the larger class of data in order to get a balanced dataset. A probabilistic classifier P_1 is trained on the training data from the balanced dataset. In the next stage we prune all the edges that were predicted by P1 to be in the negative class (with a probability over a certain threshold q). After removing the edges predicted to be in the negative class by P1 We then focus on G_2 which is the resulting subgraphs.

5.2 Graph Theoretic Features

In machine learning a feature is a property of a problem based on which one predicts results. In this project I did extensive feature engineering to improve upon the features used by Lauri and Dutta and Grassia et.al. [4][3]. This is the main part of my project and for this reason I go into detail about each of the features and in mt evaluation section I explore which features were successful and which ones were not. These are the features that I experimented with in my project. In these examples consider an edge e = u, v. While a subset of these features was identified in by Grassia et al., many of the features were identified by me.

I denote the features identified by me by having a * after the name. The reason for having a lower value and a higher value for many of the features is that u and v are interchangeable, so it does not matter which one goes first. Having the lower value first followed by the higher value is a way for the feature to remain consistent across all edges.

- *F1: Number of vertices*
- *F2: Number of edges*
- *F3: Lower Degree∗ :* I denote the lower degree of an edge in a graph as the lower of the two degrees between u and v.
- *F4: Higher Degree∗ :* I denote the higher degree of an edge in a graph as the higher of the two degrees between u and v.
- *F5: Average degree over u and v: This value represents the average of the degree of u and the degree of v.*
- *F6: Absolute difference between degree of u and v∗ :* If this value is high an edge is less likely to be in a maximum clique as one of u or v has a low degree
- *F7: Jaccard similarity:* Jaccard similarity is the number of common neighbors of two vertices u and v divided by the number of vertices that are neighbors of at least one

of u and v. This statistic helps us understand the similarities between two vertices connected by the observed edge. If they have many of the same connections, they are more likely to be in a maximum clique.

- *F8: Dice similarity*: Dice similarity is twice the number of common neighbors of u and v, divided by the sum of the actual number of neighbors of u and v. This statistic will also help us to deduce how similar two connected vertices are.

- *F9: Inverse log-weighed similarity*: The inverse log-weighted similarity of two vertices is the number of their common neighbors, weighed by the inverse logarithm of their degrees. The logic behind it is that two vertices should be considered more similar if they share low degree common neighbors than vertices with high degree, as vertices with high degree are more likely to appear similar simply by chance. In the context of maximum clique enumeration this is important for example there may exist a vertex with high degree not in the maximum clique and it may be considered more valuable than a vertex in the maximum clique with just the exact number of edges necessary.

- *F10: Cosine similarity*: is the number of common neighbors of u and v divided by the geometric mean of their degrees. It is another method of measuring similarity between two nodes.

- *F11: Average local clustering coefficient* (LCC) over u and v: The local clustering coefficient of a vertex is the fraction of its neighbors that the vertex forms a triangle with. So, we get the average of the LCC of u and v.

- *F12: Lower local clustering coefficient* (LCC)$*$: The local clustering coefficient of a vertex is the fraction of its neighbors that the vertex forms a triangle with. So, we get the value of the LCC which is lowest between u and v.

- *F13: Higher local clustering coefficient* (LCC)$*$: The local clustering coefficient of a vertex is the fraction of its neighbors that the vertex forms a triangle with. So, we get the value of the LCC which is highest between u and v.

- *F14: Average eigencentrality over u and v:* eigencentrality is a measure of the importance of a node in the whole network. It assigns values to all nodes in the networks based on the concept that nodes with connections to important nodes (nodes with many edges) gives a higher score than connections with less important nodes. A high eigencentrality means that a node is connected to many nodes who are also connected to many nodes. Areas with high eigencentrality are likely to have a maximum clique.

- *F15: Lower eigencentrality$*$:* Eigencentrality is a measure of the importance of a node in the whole network. The eigencentrality of a vertex is measure of the importance of a node in the whole network. So, we get the value of the eigencentrality which is lowest between u and v.

- *F16: Higher eigencentrality∗ :* Eigencentrality is a measure of the importance of a node in the whole network. The eigencentrality of a vertex is measure of the importance of a node in the whole network. So, we get the value of the eigencentrality which is highest between u and v.

- *F17: The number of length two paths between u and v*: This value is high when it is part of a large clique. This is because in a clique everything is connected to everything else. In the clique all the vertices in the clique will have many two length paths to one another. Imagine a path that goes from u to some vertex x that is in the clique and as x is in the clique it is also connected to v. This is an example of a two-length path. The number of two length paths is the same as the number of common neighbors between two nodes.

- *F18: Local edge-chromatic density over u and v*: The **local chromatic density of a vertex** is the minimum ratio of the number of distinct colors appearing in the neighborhood of v and any optimal coloring of G. The neighborhood of a vertex in a graph G is the subgraph containing all of the vertices that are adjacent to v. A proper vertex coloring of G is a labelling of a vertex set where the labels are called colors and no two adjacent vertices get the same color. An optimal coloring of G is a proper vertex coloring, but the number of labels is minimal. A vertex with a high local chromatic density means the neighborhood of v is dense (it takes more colors to ensure no two adjacent vertices are the same in a dense region). A vertex in a dense region is more likely to be part of a maximum clique.

  The *local edge-chromatic density* is the number of distinct colors on the common neighbors of **u and v** divided by the total number of colors used in any optimal proper coloring. Grassia et al. [Grassia et al] showed us that the local edge-chromatic density could be converted into a deterministic rule: an edge e = { u, v} can be removed if the common neighbors u and v see less than k-2 colors in any proper coloring of the input graph G, where k is an estimate for the number of maximum cliques in G. They also noticed that the local edge-chromatic density can be applied in cases where the chromatic density of a vertex cannot. We observe this in 5.1 In this example if k = 3 the chromatic density of a vertex cannot delete vertex C or D but local edge-chromatic density can delete e = {C, D}.

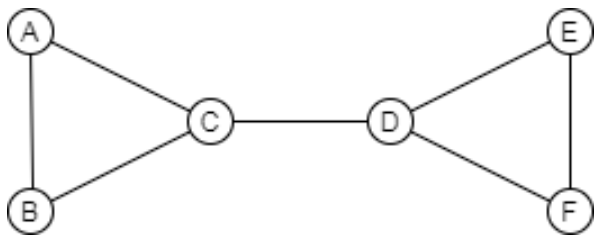

Figure 5.1

- *F19: The average degree of the neighbors of u and v*:* This value represents the average number of connections between every node connected to both u and v.
- *F20: The geometric mean of the degree of the neighbors of u and v*:* This value represents the geometric mean of the number of connections between every node connected to both u and v. The geometric mean gets the central tendency of a set of numbers by using the product of their values. [30]
- *F21: The mean of the eigencentrality of the neighbors of u and v** This value represents the mean of the eigencentrality between every node connected to both u and v.
- *F22: The geometric mean of the eigencentrality of the neighbors of u and v** This value represents the geometric mean of the eigencentrality between every node connected to both u and v.
- *F23: The mean of the local common clustering of the neighbors of u and v*:* This value represents the mean of the local common clustering between every node connected to both u and v.
- *F24: The geometric mean of the local common clustering of the neighbors of u andv*:* This value represents the geometric mean of the local common clustering between every node connected to both u and v.
- *F25: The local s-metric*:* The s-metric is defined as the sum of the products deg(u)*deg(v) for every edge (u, v) in a graph G [31]. I define the local s-metric as the s-metric for the subgraph $S \subset G$ created by adding nodes common to both u and v.
- *F26: The local s-metric over the total s-metric*:* I get the local s-metric and divide it by the total s-metric which is the sum of the products deg(u)*deg(v) for every edge (u, v) in the subgraph $S \subset G$ created by adding nodes common to both u and v divided by the sum of the products deg(u)*deg(v) for every edge (u, v) in G [31]
- *F27: The sum of the degrees of neighbors common to u and v*:* This value represents the sum of the number of connections between every node connected to both u and v. The value of getting the sum rather than the average is that it takes both the degree of the vertices and the number of vertices into consideration.
- *F28: The sum of the log of the degrees across neighbors common to u and v*:* This value represents the sum of the log of the number of connections between every node connected to both u and v. The value of getting the log of the sum is that the logarithms are a useful way to express large numbers.
- *F29: The sum of the reciprocal for the local common clustering across neighbors of u and v*:* This value represents the reciprocal of the local common clustering of every

node connected to both u and v. The reciprocal for a number x is a number which when multiplied by x yields the multiplicative identity of 1. The reciprocal of x is calculated using the following formula.

$$reciprocal\ of\ x = 1/x$$

The sum of the reciprocal of the local common clustering across all neighbors of u and v is calculated by iterating across the common neighbors of u and v and summing the reciprocal of the local common clustering for each node.

- *F30: The sum of the log values of the local common clustering across neighbors of u and v*:* This value represents the multiplicative inverse of the local common clustering of every node connected to both u and v. The value of getting the log of the sum is that the logarithms are a useful way to express large numbers.

  The sum of the sum of the log values of the local common clustering across neighbors of u and v is calculated by iterating across the common neighbors of u and v and summing the log of the local common clustering for each node.

- *F31: The absolute difference in local common clustering between u and v*:* If this value is high an edge is less likely to be in a maximum clique as one of u or v has a low local common clustering.

- *F32: The sum of the eigencentrality across neighbors of u and v*:* Eigencentrality is a measure of the importance of a node in the whole network. This value represents the sum of the eigencentrality of each neighbor common to u and v. The sum of the of the eigencentrality across all neighbors of u and v is calculated by iterating across the common neighbors of u and v and summing the eigencentrality for each node. The eigencentrality for each node is calculated at the start of preprocessing and stored in a dictionary using networkx, this was we do not have to recalculate the eigencentrality when we encounter the same node multiple times.

- *F33: The absolute difference in eigencentrality between u and v*:* If this value is high the eigencentrality differs greatly between u and v. If it is low the eigencentrality is closer together and the nodes are more likely to be in a maximum clique as both of them have similar a centrality level in the graph.

- *F34: The sum of the reciprocal of eigencentrality across common neighbors of u and v*:* This value represents the multiplicative inverse of the eigencentrality of every node connected to both u and v. The sum of the reciprocal of the eigencentrality across all neighbors of u and v is calculated by iterating across the common neighbors of u and v and summing the reciprocal of eigencentrality for each node.

- *F35: The sum of the log value eigencentrality across common neighbors of u and v*:* This value represents the sum of the natural log of every node connected to both u and v.

- *F36: The number of local colors*:* This value represents the number of distinct colors appearing in the subset of nodes common to both u and v. A proper vertex coloring of G is a labelling of a vertex set where the labels are called colors and no two adjacent vertices get the same color. An optimal coloring of G is a proper vertex coloring, but the number of labels is minimal. Here we find the optimal coloring of the subgraph which only includes nodes common to both u and v. To calculate the local coloring, I use the greedy color algorithm provides by networkx. This provides a relatively inexpensive way to get the local coloring, however as it is a heuristic, we do not always get the optimal solution.
- *F37: The average degree centrality*:* The degree centrality for a node is the fraction of other nodes it is connected to. It is calculated by getting the number of edges attached to a node and dividing by the total number of nodes in a graph minus one. The average degree centrality of an edge e is calculated by getting the average of the degree centrality of u and the degree centrality of v. The average degree centrality can be a powerful tool when trying to measure how central an edge is in a graph.
- *F38: The Lower degree centrality*:* The degree centrality for a node is the fraction of other nodes it is connected to. The degree centrality of a node is a measure of the importance of a node in the whole network. So, we get the value of the degree centrality which is lowest between u and v.
- *F39: The Higher degree centrality*:* The degree centrality for a node is the fraction of other nodes it is connected to. The degree centrality of a node is a measure of the importance of a node in the whole network. So, we get the value of the degree centrality which is highest between u and v.
- *F40: The absolute difference degree centrality between u and v*:* If this value is high an edge is less likely to be in a maximum clique as one of u or v has a low degree centrality.
- *41: Lower number of triangles*:* In graphs a triangle graph is a graph with three vertices and three edges in the form of a triangle for an example see figure 5.2. The number of triangles of a node in a graph is the number of triangles in the graph that include the node under examination [32]. This value is high when it is part of a large clique. As in a clique everything is connected to everything else all the vertices in the clique will have many triangles as it would simply follow a three length path starting at the node under examination then passing through two other nodes in the clique, then back to the node under examination. So, here I get the lowest number of triangles in the graph between triangles containing u and triangles containing v.
- *F42: Higher number of triangles*:* Here I get the highest number of triangles in the graph between the number of triangles containing u and the number of triangles containing v.

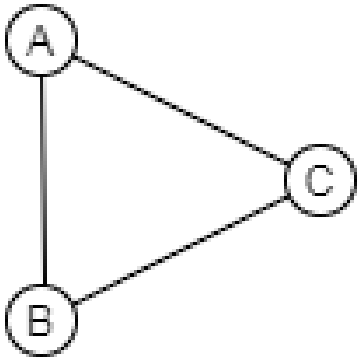

Figure 5.2

- *F43: The average number of triangles between of u and v*:* This value represents the average number of triangles in the graph between the number of triangles containing u and the number of triangles containing v.
- *F44: The absolute difference in triangles between u and v*:* If this value is high the number of triangles that can be made differs greatly between u and v. If it is low the number of triangles are closer together and the nodes are more likely to be in a maximum clique as they can both take advantage of the fact that many triangles can be made inside a large clique.
- *F45: The average number of triangles across of the neighbors of u and v*:* This value represents the average number of triangles across the nodes that are common to both u and v.
- *F46: The sum of triangles across of the neighbors of u and v*:* This value represents the sum of triangles across the nodes that are common to both u and v. This property takes into account both the number of triangles of the common neighbors and the number of neighbors as if you were to only take the average number of triangles across the neighbors you could have scenarios where you are connected by a small number of neighbors which have a large number of triangles. This property could deceptive. The sum of triangles is calculated by iterating across the common neighbors of u and v and summing number of triangles for each node.

5.3 Chi Square Statistic

A chi-square statistic is a method of measuring how expectations compare to actual observed data. The formula for chi-square statistic is

$$\tilde{\chi}^2 = \frac{1}{d}\sum_{k=1}^{n}\frac{(O_k - E_k)^2}{E_k}$$

where O_k is the observed occurrences of the outcomes i and E_k is the expected number of occurrences of i. The chi-squared test can determine if there is a significant difference between the expected data vs the actual data. The idea behind using the following features is that if a vertex was present in a large clique its features would deviate from the expected distribution characterizing the graph.

I extracted the features noted in the appendix as Chi square values for this project.

5.4 Model Selection

This project uses a machine learning framework for the reducing the problem size of the computationally difficult problem called maximum clique enumeration. The two most promising machine learning strategies that I found were Gradient boosting classifiers and Adaptive boosting classifiers. I used the sklearn learn to implement both of the names classifiers [29].

Both Gradient boosting and Adaptive boosting use the strategy of combining multiple weak learners into a single strong machine learning model.

Adaptive Boosting

The core principle of Adaptive boosting is to fit a sequence of weaker learners on the training data. The predictions from all of them are then combined through a weighed majority vote to produce the final prediction. Adaptive boosting uses boosting by applying weights to each of the training samples, initially all these weights are even however for each iteration the sample weights are modified, and the learning algorithm is reapplied to the weighted data. This modification involves selecting training examples that were incorrectly predicted by the boosting model in the previous step and giving these selected examples a larger weight. The weights for the correctly predicted samples are then decreased. This means that examples that are hard to predict receive increasing importance and because of this the weak learners are forced to concentrate on the more difficult examples [33]. Therefore Adaptive boosting is a powerful tool for identifying elements inside the maximum clique as instead of focusing on examples that are easy to predict it trains more on the difficult examples which leads to less false positives and an overall increase in accuracy.

Gradient Boosting

Gradient boosting is a state-of-the-art prediction algorithm that produces a model by combining many weak models together in order to create a strong predictive model overall [34]. Both Adaptive boosting and gradient boosting are boosting algorithms which mean they convert a set of weak learners into a single strong learner but unlike Adaptive Boosting, Gradient boosting trains on the errors directly instead of updating the weights like in Adaptive boosting. In previous examples of this problem Gradient Boosting strategies have been used however in my project I have decided to replace this strategy with the Adaptive boosting strategy [35] which in combination with the features I have selected produces a higher pruning rate than the Gradient Boosting model.

5.5 Feature Selection

In the context of supervised learning, when you are training a machine learning model the features that you use to train the model are the main influence on the performance you can achieve. Adaptive boosting trees naturally output feature importances.

The strategy that I used to select my features was based on feature importances outputted by the Adaptive boosting model. Due to the large number of features and the relatively long running time of the Adaptive boosting classifier I decided not to employ an exhaustive search strategy of all the features and instead employed the Greedy forward search strategy.

The first step that I took was to remove highly correlated features. Many of the features described in section 5.0.2 are highly correlated with each-other and these features provide little extra information so for time and space complexity it is desirable to remove them.

Greedy Forward search works by making changes to the set of features in the training set and only keeping the new set if there is an increase in accuracy on the validation set. Using the feature importance's property from Adaptive Boosting [35] I selected the feature with highest importance. From this point on I kept adding the feature which best improved the model. If a feature improved the pruning rate of the validation set, I kept it and if it did not improve the pruning rate, I ignored it. Once this is done I keep removing the least important feature, according to the importance outputted by the Adaptive Boosting Classifier until the pruning rate was not largely effected. (I measured this by a decrease in 2 percent on the validation set) The process of removing features helps with memory, computational cost and the accuracy of a model.

5.6 Domain oblivious training

Something to note about maximum clique enumeration is that different features have different levels of importance depending on how dense a graph is.

In [Lauri and Dutta, (2019)] paper [3], it was assumed that the classifier was to be trained with networks coming from the same domain, because of this the testing was also performed on networks also from that domain. Grassia et al[4] showed that a classifier can be trained on networks from various domains. In order to do this they divided the graphs into two categories, sparse graphs and dense graphs. This was based on some metric of edge density that was not specified. Graphs with edge density over 0.5 were placed in the dense graphs category and graphs with edge density less that 0.5 were placed in the sparse category. So when making predictions on an unseen dataset they would first find the edge density and make predictions on sparse training data or dense training data accordingly. In my project I measure graph density using the following function.

$$d = \frac{2m}{n(n-1)},$$

Where n is the number of nodes and m is the number of edges in G.

This value represents a fraction of the actual number of edges in a graph over the total possible number of edges in a graph. A value of 0 would consist of a graph with no edges and a value of 1 would be a complete graph. Using this function I expand upon the work done by Grassia et. al [4] by selecting a training set within a certain range of the graph density of the testing set. This is essentially the same as what Grassia et. al. did except instead of having two training sets (sparse and dense) and selecting which to train on based on the edge density of the test set being above or below 0.5, I select a subset of a total training set that is within a certain range of the test set. This results in the selection of much more informative training data and as the training data size is smaller it also results in decreased training time.

Furthermore the addition of the graph density function isolated a large disparity between certain graphs, in particular the ones identified to be dense graphs according to Grassia et al. For example heart1 and brock200-1 are both dense graphs according to Grassia et al. According to the density function heart1 has an graph density of 0.1 while brock200-1 has a graph density of 0.75. This is a large disparity and features that predict maximum clique in heart1 will not nessicarily be the same as the ones that predict maximum clique in brock200-1. I eliminate this problem by more finely selecting the training data based on the density of the unseen / test data.

In an attempt to effectively compare my results to Grassia et al. while also representing the

fact that I am using this new strategy to effectively distinguish between graph types I will add a new graph type called medium density that will represent a subset of graphs identified by Grassia et al. to be dense. Medium density graphs will be graphs with edge density from 0.05 to 0.25. The dense graphs will include graphs with density larger then 0.25 and the sparse graphs will be graphs with graph density less that 0.05 the sparse graphs will largely be the same as the sparse graphs identified by Grassia et al. This also allows me to select a classification threshold for each density type based on the observations I make regarding the lower bound of false positives on the data.

Chapter 6: Evaluation

In this section I show what features proved to be effective on what types of graphs and my computational results. To allow for a clear comparison I follow closely the definitions and practices followed in both Lauri and Dutta (2019)[3] and Grassia et al.(2019)[4]. However it should be noted that in my project, instead of pruning away the search space in order to get a desired level of pruning, I prune away the search space to get the highest level of pruning while retaining *all optimal solutions*. Because of this I sacrifice pruning rate in some very dense graphs however, through the selection of more effective features and better domain oblivious training I will show large improvements in some medium density graphs.

In the tables representing the pruning rates w is the size of the largest clique in the graph and n.w is the number of cliques in the graph of such size.

6.1 Sparse Graphs

In my project I define sparse graphs as graphs with graph density lower than 0.05. The results I achieved on sparse networks closely reflect the results achieved by Grassia et al[4]. It should be noted however that the use of a multi stage sparsification methodology by Grassia et al. adds an extra 1%-2% pruning which I do not gain in my results.

Local edge-chromatic density alternative

One criticism of the work done by Grassia et al. was that the computation of the local edge chromatic density is an computationally complex task. In some examples its computation is not worth the information gained. The complex nature of this feature derives in the fact that it requires calculating the colouring of a graph. Getting the colouring of a graph is another example of a np-hard problem and in some cases it can take even more time than the maximum clique enumeration problem. For this reason in an attempt to improve upon the work done by Grassia et al. I have produced my results on sparse graphs without any features that require the colouring of a graph like local edge chromatic density or the local chromatic density which dominate in importance in both Grassia et al.[4] and Lauri and Duttas work [3].

The effect of removing the edge chromatic density can be observed when we take bio-

WormNet-v3 under examination. When I choose the features outlined by Grassia et. al.[4] excluding the local-edge-chromatic density I get an optimal pruning rate of 87%. By optimal I mean the maximum possible pruning rate without any misclassifications. When I introduce the local edge chromatic density as a feature I get an optimal pruning rate of 99.1%. This increase in pruning rate is representative of how important the local edge-chromatic density is as a feature.

When I applied my newly introduced features outlined in figure 6.1 I received an optimal pruning rate of 94.8%. I see this as a great success as the features that I use are much less costly than the local edge-chromatic density and we only sacrifice 4%. The table 6.2 represents the pruning rates achieved using the new features described in Table 6.1.

Feature Name	Importance
Sum of triangles across common neighbours	0.22
Average of eigencentrality across common neighbours	0.15
Lower degree	0.15
Sum of LCC across common neighbours	0.14
Sum of the reciprocal of eigencentrality across common neighbours	0.11
Jaccard Similarity	0.11
Lower triangles	0.08
Absoloute difference in LCC	0.04

Table 6.1: Sparse Features Excluding Local edge-chromatic density

Instance	|V|	|E|	w	n.w	Pruning
bio-WormNet-v3	16K	763K	121	18	0.94
web-wikipedia2009	2M	5M	31	3	0.96
web-google-dir	3M	21M	24	196	0.98

Table 6.2: Results for sparse networks

6.2 Medium Graphs

In my project I defined Medium graphs as graphs with graph density between 0.05 and 0.15. I applied the feature selection strategy described in an outline of my approach and the features selected are represented in Table 6.3 Once these features were selected I built a model and ran it on the test networks using these features.

Across all the testing data the lowest confidence threshold that maintained all elements in the maximum clique was 66 percent. For this reason I selected the confidence threshold to

be 76 percent in an attempt to maintain this trend. The results I obtained are documented in Table 6.4. These results are very positive compared to results achieved by grassia et algrassia2019learning with networks like HFE18 96 in increasing from 0.27% to 0.66% and movielens-1m increasing from 0.23% to 0.56%. Increases in pruning rates this large will result in large speedups for state of the art solvers.

It should be noted however that these large improvements were not seen across all Medium graphs, in particular cegb2802 saw a decrease in pruning rate from 0.46% to 0.124%. This could be a result of Grassia et al. use of multi-stage sparsification methodology or the fact that they may have used more informative training data than I did for that network. Also my model retains all optimal solutions while the model proposed by Grassia et al. only retains 38 out of 101 maximum cliques. As I do not have access to the exact training data they used I cannot make accurate comparisons. I note the training data I used for all of the tests in the appendix below.

Feature Name	Importance
Chi Square value of the sum of the LCC across common neighbours	0.14
Chi Square value of the number of local colours	0.13
Number of local colours	0.11
Average degree centrality	0.09
Sum of triangles across common neighbours	0.08
Sum of LCC across common neighbours	0.08
Higher degree centrality	0.07
Geometric mean of degree centrality	0.05
Chi Square value of the sum of the log value of the degrees across common neighbours	0.05
Sum of the log value of the degrees across common neighbours	0.04
Sum of the reciprocal of the LCC across all neighbours	0.04
Dice similarity	0.03
Absoloute difference in degree centrality	0.03
Chi square of local s-metric over total s-metric	0.02
Chi square value of dice similarity	0.02
Chi square value of absoloute difference in degree centrality	0.02

Table 6.3: Medium density Features

Instance	|V|	|E|	w	n.w	pruning rate
movielens-1m	6K	1 M	31	147	0.56
heart1	3.6 K	1.4M	200	45	0.72
cegb2802	2.8K	137.3K	60	101	0.12
HFE18 96 in	4K	993.3K	20	2	0.66

Table 6.4: Medium Graphs

6.3 Dense Graphs

In this project dense graphs are defined as graphs with graph density larger than 0.2. Comparing results to Grassia et. al. [4] it is clear that I receive a much lower pruning rate but it should be noted that all optimal solutions are retained compared to previous solutions where many of the cliques are lost and in some cases the size of the largest maximum clique is not retained. Other factors that make it difficult to compare the difference in pruning rate include the fact that I may be using less informative training data as I do not have access to the training data used by Grassia et al. Furthermore, the multi stage sparsification approach taken by Grassia et al. may have added considerable accuracy to the results they achieved.

In this solution I am aiming for the optimal solution which means zero misclassifications. In this project in an attempt to improve upon previous work my focus was to prune away zero elements inside the maximum clique. Even though in some cases this results in smaller pruning rates, this sacrifice can be worth it when trying to achieve an optimal solution.

In results documented by Grassia et al. brock200-1 has a pruning rate of 55% however the size of the maximum clique is not retained while in my solution the size of the maximum clique is retained. When I allow misclassifications in my results we can achieve similar results to the ones documented by Grassia et. al. Loosing 0.4% of the maximum clique I can achieve pruning rates of 11% and can achieve pruning rates of up to 40% while loosing 20% of instances inside the maximum clique.

These results show promise if in future work we wanted to further explore pruning while still retaining all optimal solutions. Depending on the problem being solved it may be more important to retain all optimal solutions while sacrifice some pruning rather than allowing for the removal of some elements of the maximum clique from the search space.

My results are documented in Table 6.5 along with the feature importance in 6.6. The classification threshold I selected for these graphs was 51.3%.

Instance	|V|	|E|	w	n.w	pruning rate
p-hat500-3	300	33.4 K	36	10	0.08
brock200-1	200	14.8 K	21	2	0.08

Table 6.5: Dense Graphs

Feature	Importance
Lower LCC	0.264
Chi Square value of lower triangles	0.17
Chi Square value of higher LCC	0.168
Higher LCC	0.156
Chi value of lower LCC	0.131
Chi square of lower degree centrality	0.037
Chi square value of dice similarity	0.024
Average degree of neighbours degree	0.015
Chi square of harmonic mean of common neighbours degree	0.013
Sum of triangles across common neighbours	0.011
Number of edges in graph	0.011

Table 6.6: Dense Graph Features

Chapter 7: Conclusions and Future Work

This project uses a machine learning framework for the reducing the problem size of the computationally difficult problem called maximum clique enumeration. It builds upon an earlier framework of Grassia et al [4] by identifying new features and introducing an alternative to the local-edge chromatic density when dealing with sparse graphs. Furthermore the introduction of the medium density graph along with new graph theoretic features results in better pruning rates on specific graphs.

7.1 Dynamic classification threshold

One of the main limitations of my solution to this problem and in the solution introduced by Grassia et al. is that the pruning rate is limited by a certain classification threshold. This classification threshold is limited by the minimum threshold of the most difficult graph that you are trying to prune away (without any miss classifications).

For example in the medium density graphs section movielens-1m first false positive occurs at a classification threshold of 53%. If it were pruned away at this optimal threshold the pruning rate of the search space would be 0.93%. But as other graphs like cegb2802 have a much lower optimal classification threshold we have to lower the baseline classification threshold across all medium sized graphs which results in the reduction of movielens-1m pruning rate from 0.93% to 0.56%.

If you could map the classification threshold against certain properties like graph density or the global clustering coefficient you could dynamically select a pruning rate for a certain graph instead of having to default to the worst case scenario. This could result in graphs that are not as difficult to predict like movielens-1m having a much higher pruning rate.

7.2 Multi stage sparsification

In order to improve upon the probabilistic preprocessing framework for fine-grained search space classification proposed by Lauri and Dutta [3] a Multi-stage sparsification approach

was implemented by Grassia et al.[4]. One core problem with the probabilistic preprocessing approach is that the model focuses too much on pruning out the easy cases such as vertices of low degree and not the cases of vertices with high degree and high core number which are much harder to diagnose. The Multi-stage sparsification approach attempts to address this. Each stage the approach focuses on gradually harder cases that were difficult to prune by the classifier in earlier stages. In future work I would like to employ this multi stage sparsification strategy in combonation with the new features that I have produced in order to see if the accuracy of the solution can be improved.

Chapter 8: Acknowledgements

Thanks to Deepak Ajwani, my project supervisor who was very helpful throughout this project.

Github

https://github.com/williamomeara7/MaximumCliqueEnumeration.git

Bibliography

1. Vinyals, O., Fortunato, M. & Jaitly, N. *Pointer networks* in *Advances in neural information processing systems* (2015), 2692–2700.
2. Bello, I., Zoph, B., Vaswani, A., Shlens, J. & Le, Q. V. *Attention augmented convolutional networks* in *Proceedings of the IEEE International Conference on Computer Vision* (2019), 3286–3295.
3. Lauri, J. & Dutta, S. *Fine-Grained Search Space Classification for Hard Enumeration Variants of Subset Problems* in *Proceedings of the AAAI Conference on Artificial Intelligence* **33** (2019), 2314–2321.
4. Grassia, M., Lauri, J., Dutta, S. & Ajwani, D. Learning Multi-Stage Sparsification for Maximum Clique Enumeration. *arXiv preprint arXiv:1910.00517* (2019).
5. Wasserman, S., Faust, K. & Iacobucci, D. *Social network analysis: Theory and methods* 1995.
6. Palla, G., Derényi, I., Farkas, I. & Vicsek, T. Uncovering the overlapping community structure of complex networks in nature and society. *nature* **435,** 814–818 (2005).
7. Boginski, V., Butenko, S. & Pardalos, P. M. Statistical analysis of financial networks. *Computational statistics & data analysis* **48,** 431–443 (2005).
8. Lagrange, B. *Biométhane* (Edisud, 1979).
9. Stix, V. Finding all maximal cliques in dynamic graphs. *Computational Optimization and applications* **27,** 173–186 (2004).
10. Yang, L., Cao, J., Tang, S., Han, D. & Suri, N. Run time application repartitioning in dynamic mobile cloud environments. *IEEE Transactions on Cloud Computing* **4,** 336–348 (2014).
11. Wang, C., Schwan, K., Laub, B., Kesavan, M. & Gavrilovska, A. *Exploring graph analytics for cloud troubleshooting* in *11th International Conference on Autonomic Computing ({ICAC} 14)* (2014), 65–71.
12. Zuckerman, D. *Linear degree extractors and the inapproximability of max clique and chromatic number* in *Proceedings of the thirty-eighth annual ACM symposium on Theory of computing* (2006), 681–690.
13. Wan, L., Wu, B., Du, N., Ye, Q. & Chen, P. *A new algorithm for enumerating all maximal cliques in complex network* in *International Conference on Advanced Data Mining and Applications* (2006), 606–617.

14. Chalermsook, P. *et al. From gap-eth to fpt-inapproximability: Clique, dominating set, and more* in *2017 IEEE 58th Annual Symposium on Foundations of Computer Science (FOCS)* (2017), 743–754.

15. Schrijver, A. *A course in combinatorial optimization* (TU Delft, 2017).

16. Lawler, E. L. *Combinatorial optimization: networks and matroids* (Courier Corporation, 2001).

17. Eblen, J. D., Phillips, C. A., Rogers, G. L. & Langston, M. A. *The maximum clique enumeration problem: algorithms, applications, and implementations* in *BMC bioinformatics* **13** (2012), S5.

18. Williamson, D. P. & Shmoys, D. B. The Design of Approximation Algorithms. 2010. *preprint http://www. designofapproxalgs. com* (2010).

19. Vazirani, V. V. in *Approximation algorithms* 74–78 (Springer, 2003).

20. Cemy, V. Thermodynamical approach to the traveling salesman problem: An efficient simulation algorithm. *Journal of optimization theory and applications* **45,** 41–51 (1985).

21. Dvořák, P., Eiben, E., Ganian, R., Knop, D. & Ordyniak, S. Solving integer linear programs with a small number of global variables and constraints. *arXiv preprint arXiv:1706.06084* (2017).

22. Kool, W., Van Hoof, H. & Welling, M. Attention, learn to solve routing problems! *arXiv preprint arXiv:1803.08475* (2018).

23. Khalil, E., Dai, H., Zhang, Y., Dilkina, B. & Song, L. *Learning combinatorial optimization algorithms over graphs* in *Advances in Neural Information Processing Systems* (2017), 6348–6358.

24. Probst, M., Rothlauf, F. & Grahl, J. Scalability of using restricted boltzmann machines for combinatorial optimization. *European Journal of Operational Research* **256,** 368–383 (2017).

25. Chum, J. *et al.* Ionospheric signatures of the April 25, 2015 Nepal earthquake and the relative role of compression and advection for Doppler sounding of infrasound in the ionosphere. *Earth, Planets and Space* **68,** 24 (2016).

26. Jiang, X., Jiang, H., Shen, Z. & Wang, X. Activation of mitochondrial protease OMA1 by Bax and Bak promotes cytochrome c release during apoptosis. *Proceedings of the National Academy of Sciences* **111,** 14782–14787 (2014).

27. Butenko, S. & Wilhelm, W. E. Clique-detection models in computational biochemistry and genomics. *European Journal of Operational Research* **173,** 1–17 (2006).

28. Eppstein, D., Löffler, M. & Strash, D. *Listing all maximal cliques in sparse graphs in near-optimal time* in *International Symposium on Algorithms and Computation* (2010), 403–414.

29. Pedregosa, F. *et al.* Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research* **12,** 2825–2830 (2011).

30. Tao, D., Li, X., Wu, X. & Maybank, S. J. Geometric mean for subspace selection. *IEEE Transactions on Pattern Analysis and Machine Intelligence* **31,** 260–274 (2008).

31. Bounova, G. & De Weck, O. Overview of metrics and their correlation patterns for multiple-metric topology analysis on heterogeneous graph ensembles. *Physical Review E* **85,** 016117 (2012).

32. Nair, B. R. & Vijayakumar, A. About triangles in a graph and its complement. *Discrete Mathematics* **131,** 205–210 (1994).

33. Freund, Y., Schapire, R. & Abe, N. A short introduction to boosting. *Journal-Japanese Society For Artificial Intelligence* **14,** 1612 (1999).

34. Biau, G. & Cadre, B. Optimization by gradient boosting. *arXiv preprint arXiv:1707.05023* (2017).

35. Pedregosa, F. *et al.* Scikit-learn: Machine learning in Python. *the Journal of machine Learning research* **12,** 2825–2830 (2011).

Chapter 9: **Appendix**

Chi Square Feature
The average chi-squared value of the Lower degree of u and v
The average chi-squared value of the Higher degree of u and v
The chi-squared value of the average degree of u and v
The chi-squared value of the absolute difference between degree of u and v
The chi-squared value of the Jaccard similarity between u and v
The chi-squared value of the Dice similarity between u and v
The chi-squared value of the Inverse log-weighted similarity between u and v
The average chi-squared value of Cosine similarity between u and v
The average chi-squared value of the local clustering coefficient of u and v
The average chi-squared value of the lower local clustering coefficient between u and v
The average chi-squared value of the higher local clustering coefficient between u and v
The average chi-squared value of the eigencentrality of u and v
The average chi-squared value of the lower eigencentrality between u and v
The average chi-squared value of the higher eigencentrality between u and v
The chi-squared value of the number of length two paths between u and v
The chi-squared value of the local edge-chromatic density over u and v
The chi-squared value of the average degree of the common neighbors of u and v
The chi-squared value of the geometric mean of the degree of the common neighbors of u and v
The chi-squared value of the local s-metric
The chi-squared value of the local s-metric over the total s-metric
The chi-squared value of the sum of the degrees of neighbors common to both u and v
The chi-squared value of the sum of the log of the degrees of neighbors common to both u and v
The chi-squared value of the sum of the reciprocal of the local common clustering across neighbors of u
The chi-squared value of the sum of the log values of the local common clustering across neighbors of u
The chi-squared value of the absolute difference local common clustering between u and v
The chi-squared value of the sum of the eigencentrality across neighbors of u and v
The chi-squared value of the absolute difference in eigencentrality between u and v u and v
The chi-squared value of the sum of the reciprocal of eigencentrality across common neighbors of u and
The chi-squared value of the log value of the eigencentrality across common neighbors of u and v

Chi Square Feature The chi-squared value of the number of local colors
The chi-squared value of the average degree centrality
The chi-squared value of the lower degree centrality
The chi-squared value of the higher degree centrality
The chi-squared value of the absolute difference in degree centrality between u and v
The chi-squared value of the lower number of triangles
The chi-squared value of the higher number of triangles
The chi-squared value of the sum of the log value eigencentrality across common neighbors of u and v*
The chi-squared value of the number of local colours
The chi-squared value of the average degree centrality.
Average chi-squared value of the lower degree centrality.
The chi-squared value of the higher degree centrality*:
The chi-squared value of the absolute difference in triangles between u and v*:
The chi-squared value of the lower number of triangles*:
The chi-squared value of the Higher number of triangles*:
The chi-squared value of the average number of triangles between of u and v*:
The chi-squared value of the absolute difference in triangles between u and v*:
The chi-squared value of the average number of triangles across of the neighbors of u and v*:
The chi-squared value of the sum of triangles across of the neighbors of u and v*

Testing files	Training Files	Validation File
Heart1	heart1-train	HFE18 96 in
movielens-1m	heart1-train	heart1
HFE18 96 in	heart1-train	heart1
cegb2802	heart1-train	heart1
p-hat500-3	p-hat-train	phat500-2
brock200-1	brock-train	brock200-2
bio-Wormnet-v3	bio-train	bio-CE-CX
web-wikipedia2009	web-train	web-google-dir
web-google-dir	web-train	web-wikipedia2009