Minimal Forbidden Minors for Distinct Treewidths

Finding a reasonable number of minimal forbidden minors in $F(4)$ and an initial list in $F(5)$

Diogo Rio

Data Science and Artificial Intelligence
BSc. Thesis @ Maastricht University
Fall & Spring 2023 (18 ECTS)
Supervisor & Examiner: Steven Kelk
Second Examiner: Matúš Mihalák

Abstract

The aim of this thesis is to explore the efficiency of algorithms dedicated to finding minimal forbidden minors in $F(4)$, where $F(k)$ is the finite set of graphs with the following property: a graph G has treewidth at most k if and only if none of the graphs in $F(k)$ are a minor of G. The treewidth of a graph is a fundamental characteristic that describes its resemblance to a tree. This is crucial in graph theory, as it enables polynomial time solutions for NP-hard problems, for graphs with bounded treewidth. Methods of communicating the discovered forbidden minors to the community in a verifiable manner are developed, as only up the set $F(3)$ are they fully known. Finally, the research aims to analyze the exhaustiveness of the obtained list of forbidden minors in $F(4)$, as well as examine the feasibility of extending the algorithms used in $F(4)$ in generating a non-exhaustive list for $F(5)$.

To achieve these goals, the study employs graph generation and exploration techniques, search-space pruning methods, and connectivity and isomorphism checks. Different approaches are evaluated, and a MySQL database structure is implemented to facilitate the sharing of any research outcomes.

It defines an endeavor to advance the field of algorithmic graph theory and its possible applications in solving complex computational problems.

Keywords: Forbidden Minors, Treewidth, NP-hardness, Pruning, Connectivity, Isomorphism

Acknowledgment

I would like to use this section to thank my supervisor Steven Kelk, for the guidance throughout the project - as well as for suggesting the topic.

Table of Contents

1 Introduction

1.1 Contextualization

Before proceeding, it makes sense to define one of the main concepts of this thesis: treewidth. Formally, given an undirected graph $G = (V, E)$, a tree decomposition of G is a pair (T, X), where T represents a tree and $X = X_i | i \in V(T)$ is a set of subsets of V, denominated bags. This must satisfy the following conditions:

1. Every vertex $i \in V$ is contained in, at least, one bag X_i

2. For every edge $(i, j) \in E$, there exists a bag X_k that contains both i and j

3. For every vertex $i \in V$, the bags containing i induce a connected subtree in T

The treewidth of a graph G is defined as the minimum size of the largest bag *minus* one over all the possible tree decompositions of G. It is the smallest integer k such that there exists a tree decomposition with bags of size at most $k + 1$.

There are records of the existence of around 90 to 100 minimal forbidden minors in $F(4)$. Such a list was given by Sanders in his thesis [1]. This has, however, become unavailable as the years went on. Nowadays the list is no longer accessible and it was never proven whether Sanders' list was exhaustive. What remains known is that a finite limit does exist [2]. As a starting point, it has been proven the sets $F(1)$, $F(2)$, and $F(3)$ are fully enumerated [3] [4]. For $k > 3$, that is not the case, which is precisely where the foundation of this thesis is held.

The importance of this task lies in the possibility of drawing meaningful conclusions about the computational time of NP-hard problems, by analyzing the treewidth of a given graph. Moreover, if an extremely fast minor-checking algorithm is developed and access to the complete $F(k)$ is widely available, researchers may be able to check whether the treewidth is at most k, faster than using an exact algorithm. Ultimately, $F(k)$ can help understand 'why' a graph has certain treewidth, which is mathematically interesting.

1.2 Thesis Outline

This project is aimed to develop an algorithm or a series of them capable of discovering roughly 90 to 100 minimal forbidden minors in $F(4)$. Subsequently, the same methods will be applied to $F(5)$, to scout whether an initial list can be found there. These sets are characterized by the concept of treewidth.

The treewidth of a graph is essentially a parameter that describes 'how far that graph is from being a tree'. This is quite important in algorithmic graph theory, as it allows NP-hard problems to be solved in polynomial time, and even periodically in linear time. That is, of course, if the treewidth of the graphs in question is bounded (or small). This makes understanding whether a graph has or does not have high treewidth fundamental.

Graphs of bounded treewidth have a finite set of forbidden minors [2]. A minor is a subgraph that is obtainable by deleting vertices and edges and contracting edges. If a graph has treewidth at most k, a set $F(k)$ characterizes the treewidth of said graph, in the following sense: to check whether any graph has treewidth at most k, it suffices to guarantee that it has none of the graphs in $F(k)$ as a minor.

In addition, given that it is only known that $F(k)$ is finite, it would be an interesting (and extremely challenging!) research outcome if the filled $F(4)$ was hypothesized to be exhaustive.

The algorithmic challenge comprises generating valid graphs (i.e. connected graphs that are checked for isomorphism either upfront or upon their addition to the database), followed by searching the generated space. The focus is on graphs with treewidth $k+1$, which is a necessary characteristic of the minimal forbidden minors in $F(k)$. By applying efficient search-space pruning techniques, pre-processing, exact treewidth solvers, and connectivity and isomorphism checks, this condition can be met.

In short, this thesis explores the nature of the algorithms that allow the generation of graphs. Similarly, relevant pruning techniques may be applied, looking to significantly reduce the time spent searching the space. Different tools were tested, not always in the same environment or language. The goal regards avoid-

ing run and memory time errors when filling $F(4)$ and $F(5)$. A MySQL database structure is also implemented to provide the possibility of sharing the results obtained by this research.

1.3 Problem Statement

Given the lack of availability of enumerated minimal forbidden minors for treewidth 4 and above, could an efficient collection of algorithmic approaches be developed to both find those forbidden minors, as well as make them widely accessible?

1.3.1 Research Questions

Here are some of the questions the research aims to address:

1. How efficient can algorithms that focus on discovering a list of forbidden minors for $F(4)$ be?

2. How can the list of found forbidden minors be communicated to the community, in a verifiable way?

3. Is there any evidence that this list for $F(4)$ is exhaustive?

4. Can the same algorithmic techniques be used to produce a (non-exhaustive) list for $F(5)$?

2 Methods

The methods used within the thesis are of various natures. Some focus on generating or collecting a valid database, others on analyzing them. There were efforts to efficiently and sequentially fill the database with correct findings, as the algorithms ran throughout the semester. In this section, one can take a deeper look into what comprises the system of the project.

2.1 Graph Generation

Regardless of the chosen approach, a shared necessity was generating a valid set of graphs that would be fed into the algorithms that discern critical forbidden minors. With that in mind, it became a fair starting point to understand the basic fundamentals of the types of graphs that were to be found. In other words, at the most basic of levels, all (minimal) forbidden minors naturally have a few structural characteristics in common. At the very least, they are always connected, with no vertices without any edge connecting them to the rest of the graph. They are also undirected, which means the sense of direction is irrelevant and therefore unnecessary to model, as are edge weights.

Another important characteristic to monitor is graph isomorphism. There was a need to find a way to guarantee that any minor that was identified could not be rearranged into one same one that had been found previously. There are two ways of doing this: either through a post-analysis concept called isomorphism-checking or by modeling unlabelled graphs from the beginning, which revolves around the idea of not giving a particular ID (or label) to each of the graph vertices. The latter would inherently be a method of isomorphism checking but would limit the amount of graph generation tools that could be used. For that reason, and given the fact that applying restrictions from the conception of the thesis would most likely hinder its development, the option chosen was the former. There is already a significant number of efficient isomorphism-checking tools, which translates into more time focused on the task at hand.

2.1.1 Combinatorial (Exhaustive) Graph Enumeration

The combinatorial graph enumeration graph generation approach is a naive one. It consists in exhaustively conceiving all graphs with any provided number of vertices. This approach is designed manually and can therefore be logically built with the fundamental graph characteristics mentioned previously. This means the outcomes of this method would result in having every valid graph, ready for analysis. Exhaustively generating graphs sounds promising - if it weren't for the fact that the number of possible combinations increases exponentially [5]. Brute-forcing through the number of vertices is therefore not computationally feasible in the long term.

Nevertheless, the fact that it guarantees exhaustiveness is quite tempting. There is the

certainty that no graphs are overlooked. For that reason, this approach is kept up to a reasonable number of vertices, established to be 7, where it significantly outperforms random graph sampling. That is easily explainable, as the number of possible combinations of vertices and edges is small enough to justify the safety of exhaustiveness. With random sampling, the loom of having at least a graph missing is always present.

The combinatorial graph enumeration algorithm was based on Arseny Khakhalin's work [6]. It generates graphs recursively, by including or skipping each edge, using a lexicographical order of construction. Again, it performs well up to 7 vertices. Venturing over that number, however, starts to become seriously challenging, since the computational time increases by 2^{n-1} with each vertex increase. An alternative approach was thus worth considering.

2.1.2 The Package 'nauty': Possibilities and Limitations

The package 'nauty' is an old library developed in and for C. It is a set of procedures that aims to, among many other functionalities, quickly compute several graph operations, such as isomorphism and canonical labeling. A very useful program for this thesis would have been 'geng', included within 'nauty'. This program gives the user the ability to efficiently (and exhaustively!) enumerate non-isomorphic graphs. By tweaking certain parameters, it was possible to have it generate the types of valid graphs desired - connected and specific to the number of wanted vertices.

Once again, this presented itself as a promising approach (and as it will be suggested later on, it may very well still be). However, a setback surged. The format with which 'nauty' outputs its results is an unusual one, developed with the intent of reducing the size of the files containing them. It is a smart procedure since by using the .g6 file format, any usual graph representations (edge lists, adjacency matrices or lists, ...) can be diminished tremendously, while the program itself remains able to interpret any of those entries. Unfortunately, this distinguished factor limited making consistent use of 'nauty'. The conversion attempts (from .g6 files to ones containing any usable graph representations) were in vain. The Python functions I tried applying were prone to incorrect conversions, and this was the main reason why this approach was dropped and yet another alternative was considered.

2.1.3 Random Binomial Graph Sampling

Random sampling: a good solution to many computational intractability roadbloacks. After the somewhat inconsequential previous two methods, random sampling seemed like an interesting change of pace. Until it became an unrealistic effort (i.e. until the sample became measly in comparison with the possibly generatable graphs), it could yield a significant number of results for a number of vertices higher than 7. This was worth following, as it can undoubtedly generate some more valid graphs, which is progress towards finding more forbidden minors after all.

The random graph sampling approach used was based on the Erdős–Rényi model. Essentially, the method generates a set of graphs based on but a few parameters: the number of vertices in the graph to be created, the probability of any two vertices being connected by an edge (i.e., whether an edge is created, independently randomly sampled between every pair of vertices), and lastly the actual number of samples to be generated with these conditions.

Sequentially, this model also filters out any graphs that are not connected, pruning the search space from irrelevant graphs. The isomorphism check could be conducted at this stage, however, given the program was developed to run many times, each time trying to find new elements of $F(k)$, it is more efficient to only perform the isomorphism check on all the graphs that survive the conditions that will be explained in the following sections. That is because iterating through such a possibly large sample, in what is arguably the most time-consuming task of the final program, is not beneficial. The isomorphism checking always needs to be done anyway when graphs are on the verge of being added to the database - so

that there is a certainty that any added graphs are indeed *new* minimal forbidden minors.

2.1.4 The Combined Approach

In the end, the graph generation approach which was chosen to be the main one was a junction of both the ones developed.

As suggested earlier, the feasibility of guaranteeing exhaustiveness up to 7 vertices is something worth chasing. For that reason, the combined approach was settled using combinatorial graph enumeration up to that threshold, and random sampling thereon.

2.2 Forbidden Minor Finding

After there is a set of graphs to work with, it is necessary to perform the actual analysis of it. Minimal forbidden minors are thus identified through a method that logically applies the conditions of what it means 'to be minimal'.

2.2.1 Simplifying the Forbidden Minor Finding through Conditioning

Without having to dive too profoundly into the theory of forbidden minors, it is possible to discern how a sub-graph can be minimal. Since the forbidden minors at focus in this thesis are related to the treewidth of a graph, the observable change in the treewidth itself, as we apply vertex/edge deletion and edge contraction, is sufficient to determine the *minimality* of a minor.

All in all, what the method does is apply the three basic conditions of what it means to be a graph minor - all while constantly checking for any treewidth alterations that may originate from it. Recalling, a minor is the resulting sub-graph G obtainable from an original graph H by subjecting H to operations such as deleting a vertex, deleting an edge, or contracting an edge, for each instance of a vertex or an edge. If the treewidth upon doing so does *not* decrease, then one can be sure that *minimality* is yet to be achieved.

In practical terms, what all this theory means is the following:

For each generated graph G, $tw(G)$ is initially checked. If its treewidth is different from the established treewidth reference (which is always $k+1$), G is discarded. Discarding a graph means that the graph cannot possibly be a minimal forbidden minor. These three upcoming steps can be done in arbitrary order. Next, for each vertex in the graph, that vertex is removed. $tw(G)$ is checked again. If there exists a vertex v such that deleting v does not cause the treewidth to decrease, the graph is dropped. The same exact process is also done for each edge in the graph, where that edge is removed. Finally, the same process is yet again conducted for each edge in the graph, this time contracting it. Surviving all these conditions represents the finding of a minimal forbidden minor. Whether a new one or not, that is to be determined at a later stage.

Any discarded graphs are abandoned and not further looked into, as not doing so (i.e. not abandoning them) would heavily impact the run-time and limit the sample size that could realistically be analyzed. Long term, it is heavily inefficient to perform analysis as we deconstruct a graph, as the samples are not representative of all the graphs that exist. The experiment conducted on this further corroborates this decision (see 3.4).

Regardless, a necessary aspect for this method to work is, naturally, having an exact (and not purely heuristic) treewidth solver.

2.2.2 *QuickBB*: The Exact Treewidth Solver

With the goal of knowing the true treewidth of a graph in mind, the program has the $QuickBB$ algorithm [7] incorporated within it. It is a Branch and Bound optimization algorithm that computes the treewidth of any given undirected graph. It works by performing a search in the perfect elimination ordering of graph vertices space, using theory-based pruning and propagation techniques. An important characteristic that is behind the choice of this algorithm is the fact that it improves upon alternative methods, such as $QuickTree$, which is a complete algorithm [7].

Advantages such as its good anytime performance, resulting in unprecedentedly accurate upper bounding on graphs whose optimal treewidth had not been feasible until the development of $QuickBB$, contributed heavily towards adopting (and adapting) this method.

Nowadays, however, it is true that *QuickBB* is outperformed by alternative C++ or Java solvers, such as *BestTW* or *BFHT* [8]. The reason behind using *QuickBB* despite the known outperformers is that newer solvers are not readily available in Python, which was the language used in this thesis. Besides, the computed treewidths in this project were small, after all - which removes the need for current state-of-the-art solvers.

2.2.3 Isomorphism Checking

Another important aspect of assuring the preciseness and validity of the database is isomorphism checking. Formally, two graphs G and H are isomorphic if there exists a bijection between their vertex sets in a way that for any pair of vertices v_1 and v_2 in G, there exists a corresponding pair of vertices u_1 and u_2 in H (and vice-versa), such that the presence or absence of edges is preserved. Essentially, shared graph morphologies, with different labels. Having hidden repeated entries is an endangering issue, as it would contribute towards getting the wrong feel and number of results.

Therefore, firstly, the program - upon being on the verge of adding a minimal forbidden minor into the database - collects all the already existing instances from the specific table it is meant to add into. This collected set then refers to the set of identified forbidden minors in the current program run and performs a one-on-one comparison between every element of both sets, using the $vf2$ algorithm, which performs well on large graphs [9]. This is thus the final condition separating a found forbidden minor from being added into the database. If it does happen to be a newfound minimal forbidden minor, structurally different from all its peers, it progresses onto the database, finishing the entire forbidden minor finding process.

2.3 The Idea of Designing a Database

The designed program is not single-run. What this means is that in order to find a reasonable number of forbidden minors, the program is intended to run multiple times, because otherwise, it would be computationally impossible to achieve results. In answer to this, a database proved to be an interesting approach, since it can be updated and published externally through various means. Any issues that the program faces are independent of the database, keeping it a safe method to store any outcomes. This is called data persistence, which is a vital advantage in computationally intensive programs such as this one.

The fact that instances can be easily distinguishable from one another, by categorizing graphs with additional parameters also helps with the handling of the data. The versatility and scalability advantages that come with a database are of such importance that in truth an alternative option was never seriously considered.

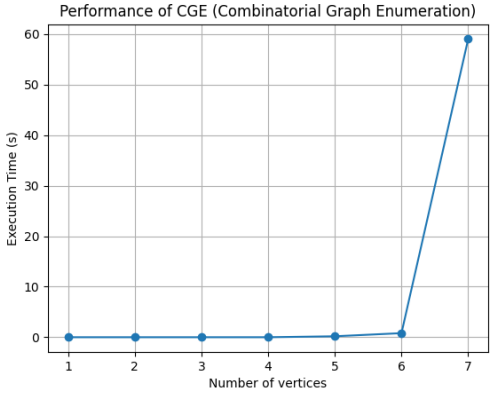
Yet, this is not all. The familiarity with MySQL and the uncertainties at the beginning of how the findings would be communicated in the end were equally major influences.

3 Experiments

Here follow some of the experiments conducted throughout the thesis. All of them were run on a Python-based algorithm, on a computer with an AMD Ryzen 7 4800H octa-core processor, running at 2900 MHz, and 16GB of RAM. Analysis of the results is done more in-depth under section 5.

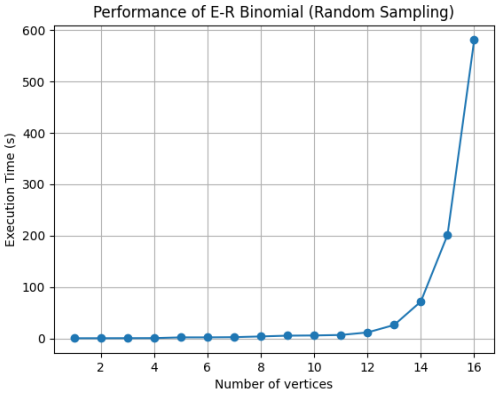
3.1 Exhaustive Vertex-Based Graph Generation

This plot represents the time taken to generate (and analyze) all the graphs, according to their number of vertices, following CGE's brute-force approach.



3.2 Erdős–Rényi Binomial Graph Generation

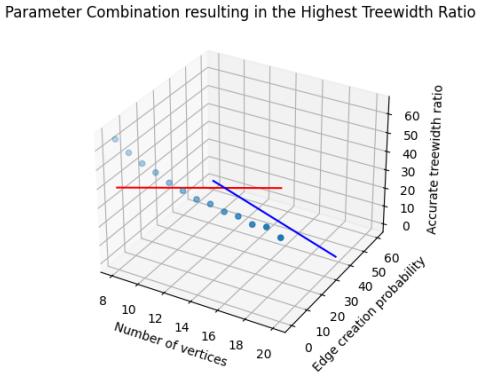
This plot represents the time taken to generate (and analyze) 10,000 graphs with a 50% probability of creating an edge between any two vertices, according to their number of vertices, following the random sampling approach. The impact of the exponentiality on the CPU time is clearly noticeable after a certain number of vertices, both for this and the previous plots.

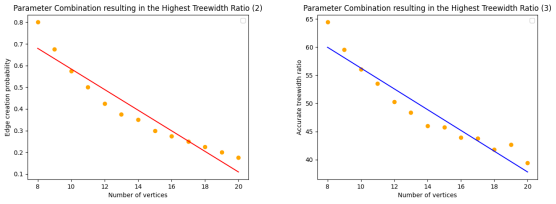


3.2.1 Linear Regression Model for Highest Treewidth Ratios

This plot represents the results of looking for the optimization values of the parameters that are relevant in the random sampling. The model was developed for $F(k)$, where $k = 4$. The step size used was 0.25 on the different edge probabilities that were being tested, and the one that yielded the highest correct treewidth ratio was kept. The optimization process was performed on $n = 10,000$.

In other words, the goal of this model is to maximize the number of relevant graphs that are randomly generated by the Erdős–Rényi approach. It is a method of pruning the search space, inherently considering the optimal edge probabilities that, given any sample, generate graphs whose treewidth is $k+1$, to begin with. Therefore, the base sample is being optimized to provide the highest chances of finding a minimal forbidden minor, according to the number of vertices.

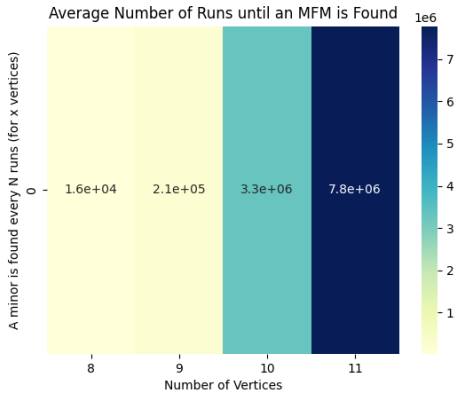




The bottom two plots show the optimal values for edge creation (left) that result in the corresponding highest accurate treewidth ratios (right) generated as a consequence of those optimal values. The top plot is a 3-dimensional visualization of the same parameters. The trend is thus: the lower the number of vertices, the higher the edge creation probability that achieves the highest ratio of graphs with the desired treewidth. As a rule of thumb, if the edge probability is too high, graphs with treewidth strictly higher than $k+1$ will be found; and if it is too low, graphs with treewidth strictly lower than that will be found. Neither of which fit the purposes of the process.

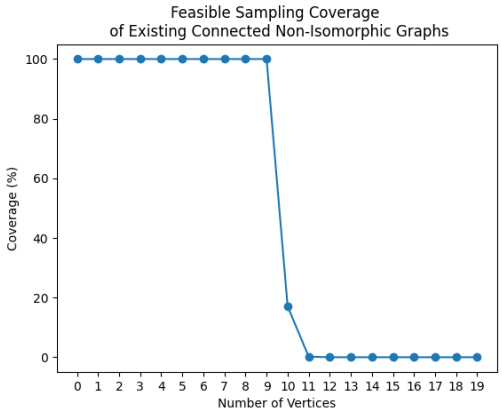
3.2.2 Average Number of Runs until the Finding of an MFM

This plot shows the average number of iterations that the program runs until a minimal forbidden minor is found, within the binomial random sampling approach. Note that this does not account for new minors, it solely represents the frequency of minors, many of which are isomorphic with other found minimal forbidden minors. The optimal binomial parameters for $k = 4$ (plotted in the previous experiment) were used to compute this data.



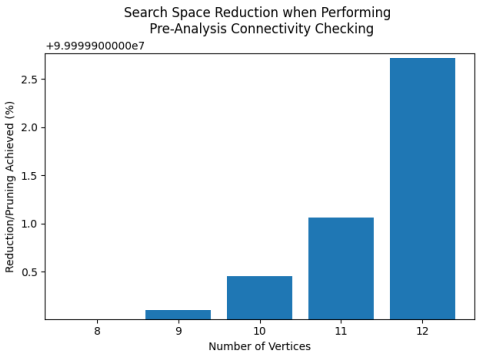
3.2.3 Binomial Sampling Coverage

This plot demonstrates how the size of the sample that is feasible to generate compares with the number of non-isomorphic connected graphs existent in the space, according to their number of vertices [5].



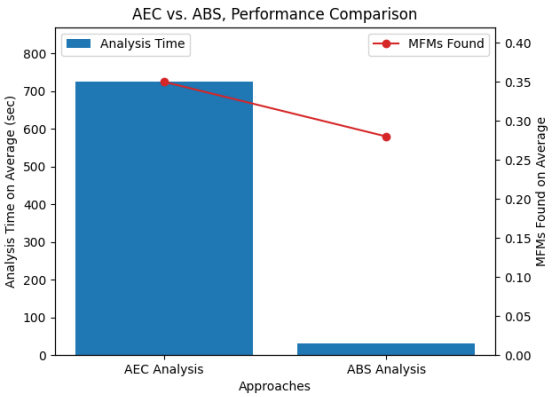
3.3 How does Pre-MFM-Finding Connectivity Checking improve Performance?

This shows the search space pruning that is achievable by performing a connectivity check before analyzing the generated graph datasets.



3.4 Performing MFM-Finding Analysis at every condition (AEC) vs. at basic structure (ABS) alone

This experiment studies the impact of analyzing the generated graphs solely in their original generated format/structure versus at also every graph deconstruction, i.e. at every vertex deletion and at every edge deletion or contraction. It ran on a set of 50,000 generated graphs, with optimal treewidth ratio parameters for 9 vertices (from 3.2.1). It is possible to see the computational time and the practical outcome performance.



4 Results

The complete results are published on this GitHub repository, under the package 'docs/results'. Ideally, as the algorithm is run, the said package is updated accordingly. The graphs are available through their database file, if the user chooses to see the edge lists that define the minimal forbidden minors themselves. In addition, image representations are provided.

4.1 Some Found Minimal Forbidden Minors

In this section, some of the found minimal forbidden minors are displayed. It should be noted that some graphs are more symmetrically drawn than others. Several design layouts were tested, and the subjectively clearest out of each of them was chosen.

4.1.1 The Known: Small Treewidth

The algorithm managed to *exclusively* find all the minimal forbidden minors for treewidth 1, 2, and 3. That means the program did not identify any incorrect minors and instead found all the already known ones, for small treewidths.

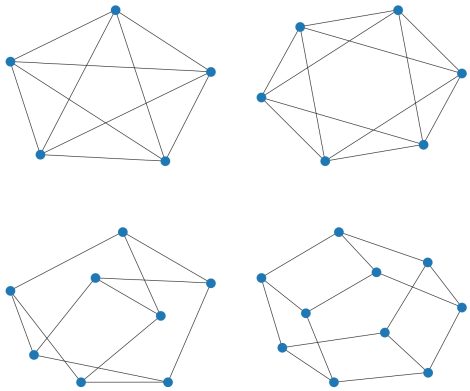
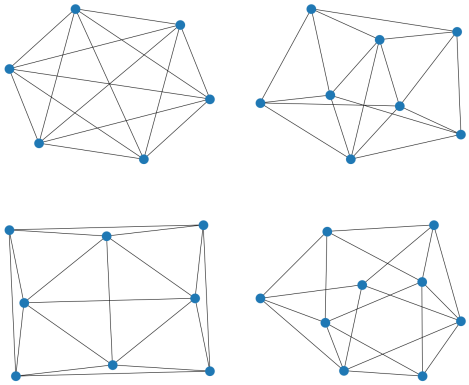


Figure 1: The 4 MFMs of $F(3)$: K_5, the octahedron, Wagner's graph, and the pentagonal prism (top left to bottom right).

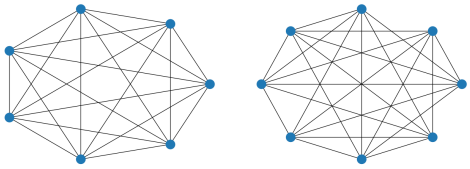
4.1.2 The Unknown: *F(4)*

Here are some of the 41 found minimal forbidden minors for $k = 4$.



4.1.3 The Unknown: *F(5)*

The number of minimal forbidden minors found for $F(5)$ is significantly smaller than that of what was the main focus of this thesis, $F(4)$. The time spent on this set was therefore limited, contrary to $k = 4$, which was the primary goal. Nonetheless, some of the 4 found minimal forbidden minors for $k = 5$ are here presented as well. Displaying the graphs with a circular layout helps understand the differences between minors as they grow more complex.



5 Analysis and Discussion

This section dives into the meaning and deductions that can be derived from the experiments and the results.

Both **3.1)** and **3.2)** represent the eventually noticeable exponential growth of both approaches, but in different manners. It is worth mentioning again that these values include the analysis, i.e. the graphs went through the minimal forbidden minor finder, not the genera-

tion alone - despite what the title may suggest. That's because the generation does take the majority of the CPU time, but it should not be ignored that the analysis is also time-consuming. For the CGE, it is possible to understand how unfeasible the program would become if an enumeration of all candidate graphs with 8 vertices or more was to be opted for. For the binomial graph sampling, that also starts to become an issue worth taking into account at around 14 vertices.

3.2.1) As we can see, for example, for 8 vertices in $F(4)$, by having an edge creation probability of 80%, there is a 64.43% chance of automatically generating graphs with starting treewidth 5 (so, 4+1), which is what is relevant to this case. Using an edge creation probability of 50%, on the other hand, would only have a 2.58% chance of generating graphs that actually mattered. This characterizes the model that maximizes the chances of having relevant minors within the set of generated graphs.

3.2.2) This plot demonstrates the average time it takes for the program to find any minimal forbidden minor (not necessarily a new one), for different numbers of vertices. It is possible to see through the colourful lateral scale that the difference starts to hint at exponential growth, as it progresses from lightly-coloured low regions to much darker ones in the span of one-unit vertex progression (from 10 to 11).

3.2.3) This proves the difficulty of the sampling method in covering all the existing connected non-isomorphic graphs. Given the sample is, on average, only computable with a size of 2,000,000 generations - it is noticeable the measly attempt at covering all of them. And even within the 100% coverage instances, there is no guarantee they all are indeed covered, as graphs are randomly generated after all. Despite having found a reasonable number of MFMs, it cannot be denied that the guarantee of exhaustiveness is easy to counter by taking a look at this graph. Most candidate graphs are most likely never even generated.

3.3) This bar plot allows comprehension of the impact of the connectivity pruning on reducing the graph search space pre-analysis. It is quite insignificant for lower numbers of vertices, which makes sense by recalling the optimal parameters model (see 3.2.1). Again, the tendency is the lower the number of vertices, the higher the edge creation probability parameter, in order to achieve the highest ratio of graphs with the desired treewidth, to begin with. Thus, generating with an 80% chance of creating an edge will naturally result in more connected graphs, as opposed to 42.5% (which is the value for 12 vertices). Pruning becomes increasingly important as graphs with more vertices are generated.

3.4) Analysis done at every condition took around 724.6 seconds, while analysis that was done solely on the generated graphs themselves took 30.7 seconds, on average. The first method does outperform the second at finding minimal forbidden minors, but not significantly. An average of 0.35 and 0.28 found minimal forbidden minors per 50,000-sized run, respectively. It is true that re-analyzing the graphs at every vertex deletion and at every edge deletion or contraction is an approach that can help when samples are representative (e.g. for 8 vertices, it can be a worthy compromise, as the chances of covering the existing valid graphs are larger), but as there is a need to analyze more samples (to cover the maximum ground), the run-time disadvantage heavily hinders the process, and with a non-significant outperformance, result-wise. It is thus fair to say that it represents a compromise that is not worth opting for.

From analyzing this collection of results, it is possible to gain insights into the overall challenges faced by this program. While several optimizations were conducted throughout, it appears they are of limited use in the grand scheme of things. Finding an exhaustive (or close to exhaustive) list of forbidden minors is a challenging task - individually at the very least!

6 Conclusion

The program was capable of re-discovering the minimal forbidden minors for $F(1)$, $F(2)$, $F(3)$, as well as identifying a number of elements of $F(4)$ and $F(5)$. All the generated minimal forbidden minors are publicly available online, and can be subject to verification through a panoply of means, given the panoply of formats published. On the procedure itself,

the obvious limitation of the approach were algorithmic. There is a formidable number potential candidate graphs and it is challenging to efficiently enumerate or sample in such large spaces. Nevertheless, it is possible to identify a number of areas that can be dived upon for improvement in future research. On one hand, it would be useful to tune the graph generation procedure, so that graphs that cannot possibly be minimal forbidden minors (e.g. graphs with nodes of degree 2, graphs with multiple biconnected components, etc.) are never generated. Essentially, more extensive and focused pruning mechanisms. Relatedly, one could explore sampling via the topic of k-trees, which are edge-maximal graphs for a given treewidth. Above all, more ambitious algorithm engineering is bound to achieve the goal of finding close to 90 to 100 forbidden minors in $F(4)$, and likely for higher values of k too. It would help if the constituent parts of the pipeline were to run more quickly, or simply more efficiently. A collaborative approach, which looks to farm out graphs (in the style of SETI@Home [10]) out of a fully enumerated online database would be an interesting way forward. Perhaps using the 'nauty' swift and light C++ generation tool 'geng' (using the .g6 file format), converting those graphs into a format this program can work with and constantly updating the results is one way of guaranteeing exhaustiveness. The hosting of such a service would then be available for any researchers willing to spend CPU resources on the search for minimal forbidden minors. Going forward, a public ever-evolving forbidden minor database and online checking tool would be an exciting outcome of the concepts tackled in this thesis.

References

[1] Daniel P. Sanders. "On Linear Recognition of Tree-Width at Most Four". In: *SIAM Journal on Discrete Mathematics* 9.1 (Feb. 1996), pp. 101–117. ISSN: 0895-4801. DOI: 10.1137/S0895480193243043.

[2] Neil Robertson and Paul D. Seymour. "Graph Minors. XX. Wagner's conjecture". In: *Journal of Combinatorial Theory* 92.2 (2004), pp. 325–357. DOI: 10.1016/j.jctb.2004.08.001.

[3] Hans L. Bodlaender. "Dynamic programming on graphs with bounded treewidth". In: *Automata, Languages and Programming*. Ed. by Timo Lepistö and Arto Salomaa. Berlin, Heidelberg: Springer Berlin Heidelberg, 1988, pp. 105–118. ISBN: 978-3-540-39291-0. DOI: 10.1007/3-540-19488-6_110.

[4] Stefan Arnborg, Andrzej Proskurowski, and Derek G. Corneil. "Forbidden minors characterization of partial 3-trees". In: *Discrete Mathematics* 80.1 (1990), pp. 1–19. ISSN: 0012-365X. DOI: 10.1016/0012-365X(90)90292-P. URL: https://www.sciencedirect.com/science/article/pii/0012365X9090292P.

[5] N. J. A. Sloane. *A001349: Number of connected graphs with n nodes.* The On-Line Encyclopedia of Integer Sequences. [Accessed: March 10th, 2023. Last modified on April 8th, 2014.] 1964. URL: https://oeis.org/A001349.

[6] Arseny Khakhalin. *Draw all graphs of N nodes*. Matplotblog. [Accessed: April 5th, 2023]. 2020. URL: https://matplotlib.org/matplotblog/posts/draw-all-graphs-of-n-nodes/.

[7] Vibhav Gogate and Rina Dechter. "A Complete Anytime Algorithm for Treewidth". In: *Proceedings of the 20th Conference on Uncertainty in Artificial Intelligence*. UAI '04. Banff, Canada: AUAI Press, 2004, pp. 201–208. ISBN: 0974903906. DOI: 10.5555/1036843.1036868.

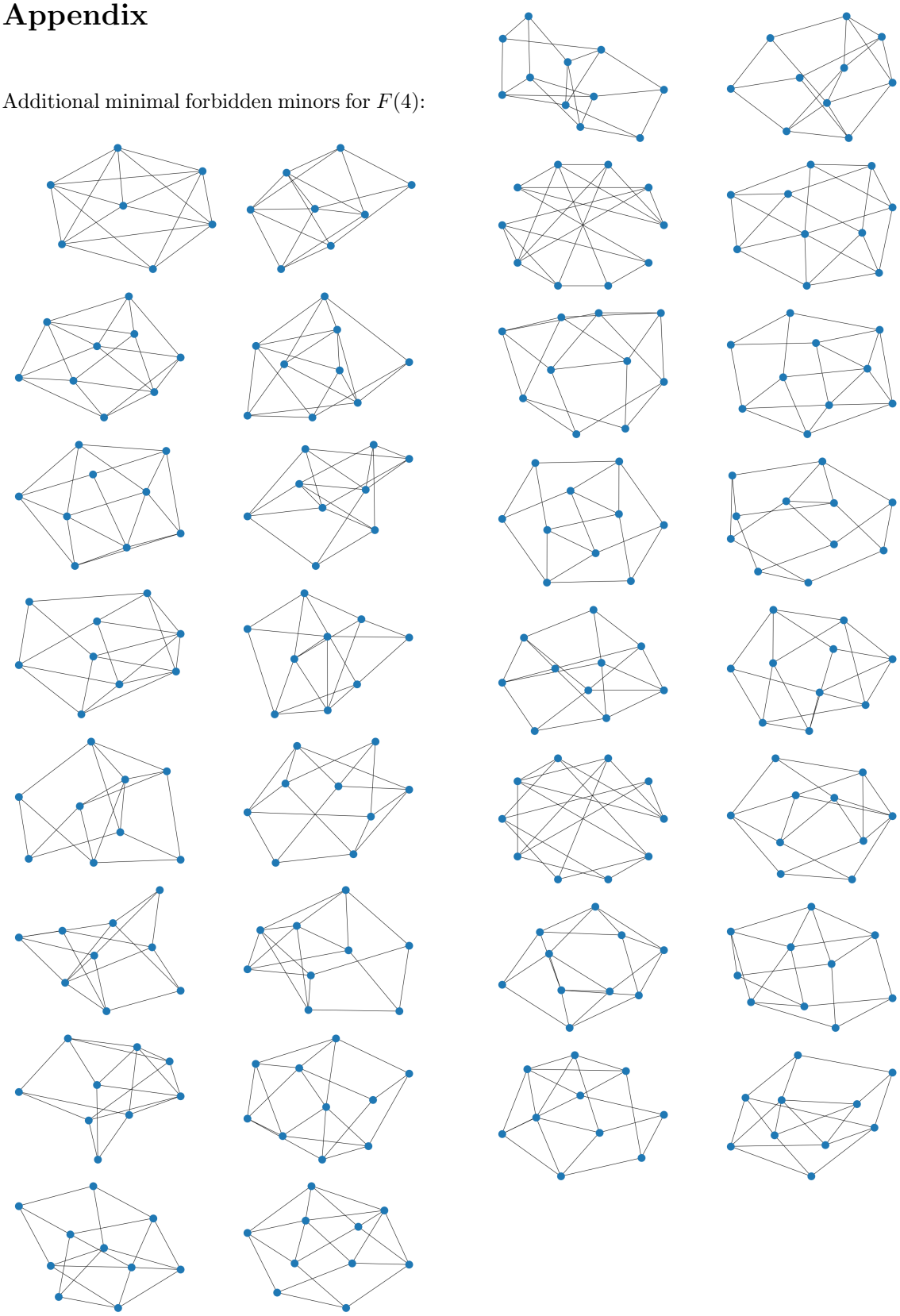
[8] P. Alex Dow and Richard E. Korf. "Best-First Search for Treewidth". In: *Proceedings of the Twenty-Second AAAI Conference on Artificial Intelligence, July 22-26, 2007, Vancouver, British Columbia, Canada*. AAAI Press, 2007, pp. 1146–1151. URL: http://www.aaai.org/Library/AAAI/2007/aaai07-182.php.

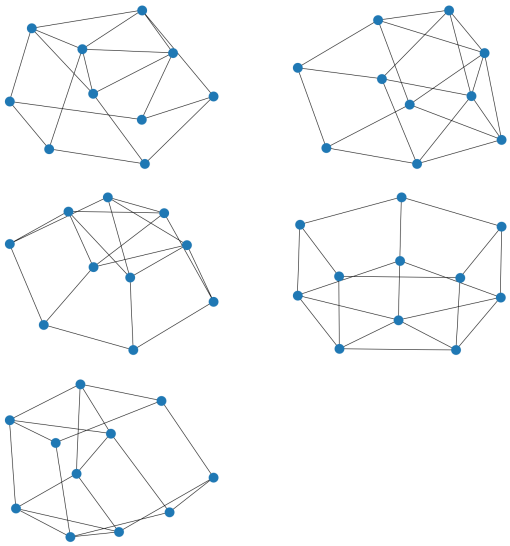
[9] Pasquale Foggia, Carlo Sansone, and Mario Vento. "An Improved Algorithm for Matching Large Graphs". In: Proc of the 3rd IAPR-TC-15 International Workshop on Graph-based Representation, Jan. 2001.

[10] "SETI@home: an experiment in public-resource computing". In: *Communications of The ACM* 45.11 (2002), pp. 56–61. DOI: 10.1145/581571.581573.

Appendix

Additional minimal forbidden minors for $F(4)$:





Additional minimal forbidden minors for $F(5)$: