Comenius University in Bratislava
Faculty of Mathematics, Physics and Informatics

Minimal uncolourable subgraphs of snarks

Bachelor Thesis

2024
Emma Baďurová

Comenius University in Bratislava
Faculty of Mathematics, Physics and Informatics

Minimal uncolourable subgraphs of snarks

Bachelor Thesis

Study Programme:	Computer Science
Field of Study:	Computer Science
Department:	Department of Computer Science
Supervisor:	doc. RNDr. Ján Mazák, PhD.

Bratislava, 2024
Emma Baďurová

16652457

Univerzita Komenského v Bratislave
Fakulta matematiky, fyziky a informatiky

ZADANIE ZÁVEREČNEJ PRÁCE

Meno a priezvisko študenta:	Emma Baďurová
Študijný program:	informatika (Jednoodborové štúdium, bakalársky I. st., denná forma)
Študijný odbor:	informatika
Typ záverečnej práce:	bakalárska
Jazyk záverečnej práce:	anglický
Sekundárny jazyk:	slovenský

Názov: Minimal uncolourable subgraphs of snarks
Minimálne nezafarbiteľné podgrafy snarkov

Anotácia: Práca sa zaoberá skúmaním minimálnych nezafarbiteľných podgrafov nezafarbiteľných kubických grafov. Vytvorí sa úplný zoznam takýchto malých podgrafov a následne tieto grafy budú podrobené ďalšiemu skúmaniu; dúfame, že sa nám podarí zovšeobecniť ich štruktúru a objaviť vlastnosti, ktoré by priniesli nové poznatky o nezafarbiteľnosti kubických grafov.

Cieľ: 1.Vytvorenie kompletného zoznamu minimálnych nezafarbiteľných podgrafov snarkov približne do 20 vrcholov.
2. Analýza grafov v zozname a odvodenie ich spoločných vlastností.
3. Zovšeobecnenie niektorých podgrafov a ich využitie pri konštrukciách snarkov.

Vedúci:	doc. RNDr. Ján Mazák, PhD.
Katedra:	FMFI.KI - Katedra informatiky
Vedúci katedry:	prof. RNDr. Martin Škoviera, PhD.
Dátum zadania:	31.10.2023

Dátum schválenia: 08.11.2023

doc. RNDr. Daniel Olejár, PhD.
garant študijného programu

....
študent vedúci práce

16652457

Comenius University Bratislava
Faculty of Mathematics, Physics and Informatics

THESIS ASSIGNMENT

Name and Surname:	Emma Baďurová
Study programme:	Computer Science (Single degree study, bachelor I. deg., full time form)
Field of Study:	Computer Science
Type of Thesis:	Bachelor´s thesis
Language of Thesis:	English
Secondary language:	Slovak

Title: Minimal uncolourable subgraphs of snarks

Annotation: The thesis consists in investigating minimal uncolourable subgraphs of uncolourable cubic graphs. First, an exhaustive list of such small graphs will be created. These graphs will undergo further investigation; we hope to generalize their structure and discover some of their properties that would give new insights into uncolourability of cubic graphs.

Aim: 1. Create a list of minimal uncolourable subgraphs of snarks, up to perhaps 20 vertices.
2. Analyze the computed cores and derive their common properties.
3. Generalize some of the subgraphs into families of larger snarks.

Supervisor:	doc. RNDr. Ján Mazák, PhD.
Department:	FMFI.KI - Department of Computer Science
Head of department:	prof. RNDr. Martin Škoviera, PhD.
Assigned:	31.10.2023
Approved:	08.11.2023

doc. RNDr. Daniel Olejár, PhD.
Guarantor of Study Programme

....

Student Supervisor

Acknowledgments: I would like to thank doc. RNDr. Ján Mazák, PhD. for supervising this thesis, providing consultations, and giving quick responses. I also thank my partner for his continuous support and interesting ideas. Lastly, I am grateful to my SSD, which stored all the generated data and did not malfunction even after my laptop did.

Abstrakt

Táto práca sa zameriava na zostavenie zoznamu minimálnych nezafarbiteľných podgrafov snarkov a analýzu ich vlastností. Snarky sa stávajú čoraz populárnejšími, pretože majú potenciál poskytnúť protipríklady k niekoľkým hypotézam v teórii grafov. Aby sme vyriešili absenciu zoznamu minimálnych nezafarbiteľných podgrafov, vygenerovali sme tieto podgrafy algoritmom, ktorý postupne odstraňuje hrany a overuje minimalitu a nezafarbiteľnosť výsledných podgrafov. Minimálne grafy sú tie s najmenším počtom prvkov, pričom sú stále nezafarbiteľné. Konkrétne, 3-hranovo-nezafarbiteľné. Na optimalizáciu času výpočtu sme použili memoizáciu. Ako vstupy sme stiahli rôzne sady grafov z online databázy. Naše výsledky zahŕňajú údaje o vygenerovaných podgrafoch, ako je ich rozdelenie podľa počtu vrcholov alebo časov trvania jednotlivých generovaní, atď. Zistenia sú prezentované v mnohých tabuľkách a obrázkoch.

Kľúčové slová: snark, minimálny nezafarbiteľný podgraf, 3-hranovo nezafarbiteľné

Abstract

This work focuses on compiling a list of minimal uncolorable subgraphs of snarks and analyzing their properties. Snarks are becoming increasingly popular because of their potential to provide counterexamples to several conjectures in graph theory. To solve the absence of a list of minimal uncolorable subgraphs, we generated these subgraphs with an algorithm that successively removes edges and verifies the minimality and uncolorability of the resulting subgraphs. Minimal graphs are those with the least number of elements while still being uncolorable. Specifically, 3-edge-uncolorable. We used memoization to optimize the calculation time. As inputs, we downloaded different sets of graphs from an online database. Our results include data about the generated subgraphs, such as their distribution according to the number of vertices or the durations of each generation, etc. The findings are presented in numerous tables and figures.

Keywords: snark, minimal uncolorable subgraph, 3-edge uncolorable

Disclaimer

The text in this thesis was grammatically edited using Grammarly and reformulated/rephrased with ChatGPT. AI was used exclusively for these purposes.

Contents

Introduction

This thesis aims to create a list of minimal uncolorable subgraphs of snarks and to analyze the subgraphs and their properties. The thesis is divided into four chapters in which we progressively go through definitions, implementation, data, and results.

In the first chapter, we briefly examine the historical context, explain the origin of the term "snark," and define fundamental concepts such as minimality, coloring, and snarks. We also review previous works related to this topic.

Chapter two starts with a few more definitions and graph formats, in which the input and output data is stored, how the formats work, and the libraries that are used. Next, we provide information about the generation algorithm, which is based on the gradual removal of edges and verification of whether the generated subgraphs are uncolorable and minimal. A minimal graph is a graph that has the smallest possible number of elements while still meeting the required properties. In this context, uncolorable graphs are those that are 3-edge-uncolorable, meaning they cannot be properly edge-colored with up to three colors. We also cover the memoization used in the algorithm and discuss applied and potential improvements. Lastly, we provide proof of the correctness of the algorithm.

Chapter three focuses on the input and output data. We describe various sets of graphs obtained from the internet graph database [11], including their counts and vertex numbers. A brief section at the end of the chapter discusses the formats of output files and provides a link to the repository containing all related files.

The final chapter begins with an explanation of the number of vertices in the subgraphs, followed by results for each input set. This chapter includes numerous tables with data such as the numbers of generated subgraphs, generation run times, the distribution of graphs based on vertex count, etc. A few selected figures are also provided.

While there is an existing database of snarks with up to 36 vertices (with additional snarks available based on girth), there is no comprehensive list of minimal uncolorable subgraphs of snarks. Snarks are of significant interest because they are believed to potentially contain counterexamples to several conjectures. Therefore, a list like this could provide new insights into this field.

Chapter 1

Snarks

In this chapter, we will briefly look at some historical context and reasoning for the name *"snark"*, some basic definitions and the current state of the art.

1.1 History

In 1880 P.G. Tait wanted to find a new way to prove the four-color theorem [8]. But, instead of coloring maps, he translated the original problem to the edge-coloring of cubic graphs problem [24]. He believed, that all cubic graphs were 3-edge-colorable, but he overlooked two kinds of cubic graphs of which some cannot be 3-edge-colored: those with bridges and non-planar cubic graphs.

Martin Gardner was the first to popularize the term "snark" [13], after he looked into nontrivial graphs, that cannot be 3-edge-colored and published a column in *Scientific American* in 1976. He deduced the title from *Lewis Carroll's The Hunting of the Snark* [24].

1.2 Definitions

To fully understand the topic, we need to start with some definitions. All definitions are sourced from [18] and common graph theory textbooks.

First, we need to define ***edge-coloring***. Edge-coloring is an assignment of colors to graph edges. Formally, it's a mapping $\varphi\colon E \to X$ from a set of edges E of graph G to a set of colors X. More specifically we need to define, ***proper edge-coloring***, which is edge-coloring, where no two adjacent edges have the same color.

Chromatic index $\chi(G)$ of a graph G is the smallest number of colors needed to achieve *proper edge-coloring.* Graph is said to be ***k-edge-colorable***, if $\chi(G) \leq k$.

Edge-colorability (edge-uncolorability) is the property of a graph G, that has any edge-coloring ϕ (does not have an edge-coloring).

Since in this thesis, we will only work with 3-edge-colorability and 3-edge-uncolorability, we will refer to it as *colorability* and *uncolorability* respectively.

An ***edge cut*** is a *set* of edges of a graph G of which, if removed (or "cut"), disconnects the graph. A ***cyclic edge-cut*** of a graph G is an edge cut, which separates two cycles.

Edge-connectivity of a graph is the minimum *number* of edges $\lambda(G)$ whose deletion from a graph G disconnects G. In other words, it is the size of a minimum edge cut. The edge connectivity of a disconnected graph is therefore 0, while that of a connected graph with a bridge equals 1.

A graph G is ***cyclically k-edge connected*** if the deletion of fewer than k edges from G does not create two components both of which contain at least one cycle.

A ***cyclic edge connectivity*** $\lambda_c(G)$ is the size of a smallest cyclic edge cut, i.e., a smallest edge cut A such that $G - A$ has two connected components, each of which contains at least one graph cycle.

A graph $H(V', E')$ is a subgraph of a graph $G(V, E)$, where V, V' are sets of vertices and E and E' are sets of edges. $V' \subseteq V$, $E' \subseteq E$. If H cannot be further reduced, so that the properties of the original graph G are preserved, H is a ***minimal*** subgraph of the graph G.

A ***girth*** of a graph is the length of the shortest cycle in the graph. If there are no cycles in the graph, the girth is defined as infinity.

Several definitions of snarks exist differing in strength, but the most important part is the colorability requirement, which is the same across all definitions. We will go with this form of the definition: ***snark*** is a *minimal cyclically 4-edge connected cubic graph* with an *edge chromatic index 4* and *girth at least 5*.

Multipoles are graphs in which dangling (only one end of an edge is incident with a vertex) or isolated (neither end of an edge is incident with a vertex) edges are permitted.

An end of an edge that is incident with no vertex is labeled as a ***semiedge***.

k-pole is a multipole with k semiedges.

Lemma 1 (Parity lemma)**.** Let M be a k-pole and let k1, k2, and k3 be the numbers of semiedges of color c_1, c_2, and c_3, respectively. Then $k1 \equiv k2 \equiv k3 \equiv k \pmod 2$.

As stated in [18], snarks are getting more attention lately since they might contain counterexamples to several profound and long-standing conjectures. Therefore, understanding their structure is necessary to (dis)prove any of those conjectures. But they are rare because almost all cubic graphs are hamiltonian, therefore 3-edge-colorable [22].

We use two names for the inputs in the texts. A ***group*** represents input graphs with a given number of vertices and a ***set*** represents the groups of a given girth together.

1.3 Previous works

Generating snarks

In 2013 G. Brinkmann, J. Goedgebeur, J. Hägglund, and K. Markström published a paper called *Generation and properties of snarks* [6]. They came up with a new algorithm for all non-isomorphic snarks of a given order. The implementation was 14 times faster than previous snark-generating programs and could generate all snarks up to 36 vertices. Until then, only snarks with up to 28 vertices were published. In their analysis of the sets of generated snarks, they proved some conjectures such as some of the strongest versions of the cycle double cover conjecture for all snarks of these orders, Jaeger's Petersen coloring conjectures imply that there are no small counterexamples to Fulkerson's conjecture. However, they successfully found some counterexamples of eight previously published cycle-covering conjectures and the general structure of cubic graphs conjectures.

Their algorithm is based on the one G. Brinkmann, J. Goedgebeur and B. D. McKay described in [7] in 2011. Originally was the algorithm intended for generating all cubic graphs without the restriction of girth, but it could be efficiently modified to generate cubic graphs with girth at least 4 or 5. At first, from K_4 are generated pairwise non-isomorphic connected cubic graphs by building prime graphs with operations (a),(b), and (c) shown in Figure 1.1. Afterward was the operation (d) applied.

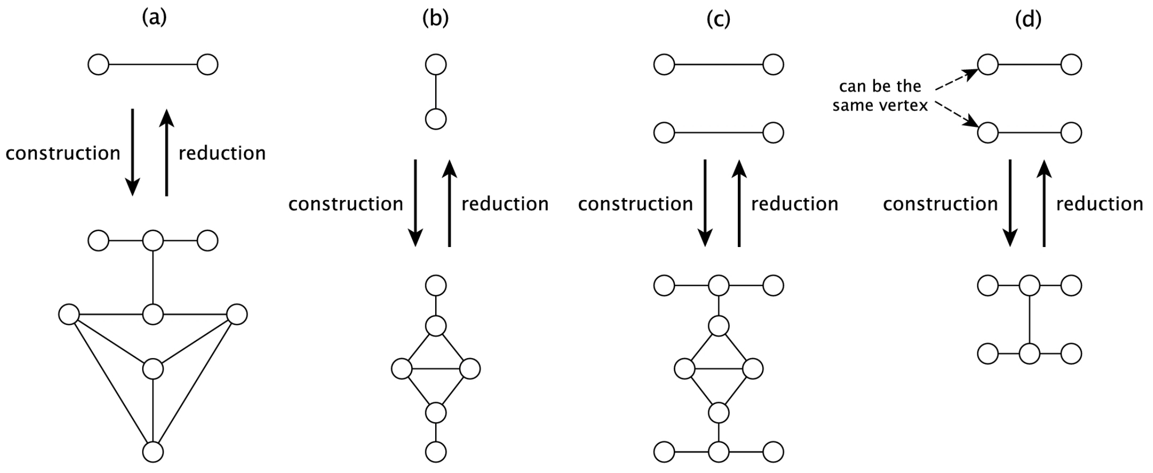

Figure 1.1: The construction operations for cubic graphs from [6].

Older snark-generating algorithms worked by adding filters for graphs to a program generating all cubic graphs of a given order. But the efficiency of the approach that, at first, generated a larger group of cubic graphs and then uses filters on the output to select only the ones that meet the given criteria depends on the filter complexity itself and also on the ratio between the number of the generated graphs and the filtered ones. In [6] they provide exact numbers and scores for this generating approach.

Classification of snarks

Ján Mazák, Jozef Rajník, and Martin Škoviera then in 2021 classified all snarks up to order 36 and explained the reason for their uncolorability in their paper *Morfology of small snarks* [18]. They achieved this by computer-assisted structural analysis of cyclically 5-connected critical snarks. Their results showed, that most of the analyzed snarks are built up from pieces of the Petersen graph called *Petersen clusters*, and depending on the reason of their uncolorability, they can be naturally distributed into a small number of classes. Finally, they generalized some snarks to infinite families.

Database of snarks

In 2013, a comprehensive database of graphs [5, 11] was created by G. Brinkmann, K. Coolsaet, J. Goedgebeur and H. Mélot. Their motivation was to have a searchable database and offer a complete list of some graph classes and a list of other interesting and relevant graphs, which can be used in graph theory problems or as counterexamples to conjectures. Aside from other useful functionalities, it contains a list of all snarks (of girth at least 5) up to 36 vertices, which will be useful in further research. In 2022 were the used technologies updated and some new features added (see [11]).

Related thesis

A related thesis was written in 2018 by Tomáš Vician, called *Cores of uncolorable cubic graphs* [23]. The motivation for this thesis was to create a database or generator of all small uncolorable cubic graphs and minimal uncolorable subgraphs of these small snarks. Since many conjectures can be proved or disproved using snarks, it is useful to have a database of them and their different variants to analyze various properties and structures.

But generating graphs with brute force would have large complexity, so he tried another method: to iteratively generate graphs using functions, that add graph elements (vertices, edges, graphs) to a graph in different ways and store canonical forms to avoid storing isomorphic graphs. Finally, the generated graphs were then reduced to “cores” - minimal uncolorable subgraphs.

His approach is based on an article from R. Nedela and M. Škoviera called *Decompositions and reductions of snarks* [21]. The idea was that snarks can be trivial and non-trivial. Trivial snark is, for example, a snark, that does contain a bridge or one, that is a “trivial” variation of another snark. Also, as stated in [21]: “It has been customary to consider trivial any snark that has a digon (Figure 1.2) or a triangle because these are trivial features that can be added to or removed from a snark without altering its uncolorable property”. Based on the two theorems they provided and previous work, two types of operations to simplify a given snark needed to be considered - *reduction* and *decomposition* [21, 18].

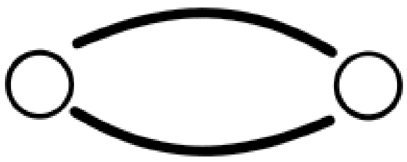

Figure 1.2: A digon.

The result of the thesis was some insight into the small snark generation. The method he used for generating the cores was not very efficient and he only generated them for 6-vertex graphs.

Both, [6] and [23] use the method of adding elements to a graph and removing them, but they differ in what kind of elements they are adding or removing. In [6] they work with more complex structures than in [23], where only some edges, loops, triangles, and squares are added and two graphs are connected on nodes or on the edges.

Chapter 2

Generating subgraphs

The goal of this thesis is to create a list of minimal uncolorable subgraphs of snarks with more than 10 vertices. Such subgraphs can also be called *edge-critical.*

Definition 1 (Edge-critical graph)**.** An edge-critical graph is a type of graph where the chromatic index is decreased by removing any edge.

These subgraphs we are interested in can also be referred to as "critical" or "bicritical. These terms are used in various contexts with subtle differences, so specifically, for our purposes, we define these terms as follows. But before we define the two terms, "critical" and "bicritical," we must first establish one definition that is used in those definitions [18].

Definition 2 (Removable, non-removable edge)**.** A pair of distinct vertices $\{u, v\}$ of a graph G will be called removable if $G - \{u, v\}$ is not 3-edge-colorable; otherwise it will be called non-removable.

Definition 3 (Critical graph)**.** A graph G is critical if every pair of adjacent vertices in G is non-removable.

Definition 4 (Bicritical graph)**.** A bicritical graph is a graph where every pair of distinct vertices is non-removable.

In the related thesis [23] an algorithm for generating cubic graphs with up to 10 vertices and subgraphs with up to 6 vertices was described. The process of generating the graphs took a while, so working with more than 10 vertices requires either faster or better implementation, or a programming language, that is faster than Python - which was used before.

The approach for generating the given subgraphs we used is based on the gradual removal of edges. At first, we downloaded groups of input graphs from the internet graph database House of Graphs [11].

2.1 Graph formats

All the graphs were in *graph6* format, which is a format for storing undirected graphs in a compact manner [19]. The way *graph6* works is that it represents the graph as a string. More specifically, as numbers between 63 and 126, since those can be easily converted into printable ASCII characters. The first part of the string is $N(n)$, which is a number based on the number of vertices n and it's calculated as stated in Table 2.1

n	$N(n)$
$0 <= n <= 62$	single byte $n + 63$
$63 <= n <= 258047$	four bytes - 126 $R(x)$, where x is the big-endian 18-bit binary form of n
$258048 <= n <= 68719476735$	eight bytes - 126 126 $R(x)$, where x is the big-endian 36-bit binary form of n

Table 2.1: Calculation of N(n)

The second part of the string is $R(x)$, where x is the upper triangle from the adjacency matrix of a graph written as a bit vector, divided into groups of six bits using ordering (0,1), (0,2), (1,2), (0,3), (1,3), (2,3), ..., (n-2,n-1) (if its length is not a multiple of 6, zeros are added to the end of the bit vector). Then the sixes are read as big-endian binary numbers and 63 is added to each one of them. These values are stored one per byte. $N(n)$ $R(x)$ is the graphs representation.

Example 1. Let graph G have five vertices ($n = 5$) and edges 0-2, 0-4, 1-3 and 3-4. So the adjacency matrix would look like this:

$$\begin{bmatrix} 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 \end{bmatrix}$$

Then $x = 0\ 10\ 010\ 1001$. Since $n < 62$, $N(n) = 68$ and $R(x) = R(010010\ 1001\mathbf{00})$ $= 81\ 99$. The graph is *68 81 99* or *DQc* in ASCII format.

Another format we used is *sparse6*. But since this format was used sparsely we will mention it very briefly. The *sparse6* format differs from the *graph6* format not only in the notation but also in the way of calculating the second part of the output string because in this format self-loops and multiple edges are permitted. The notation of *sparse6* format for one graph is the character ':', the number of vertices, a list of edges, and an end-of-line character, so the only difference from *graph6* format notation is a colon.

2.2 Libraries

There are a few characteristics that the subgraphs must have and to verify that they do, we used functions from a graph library `ba-graph` [1] provided by J. Mazák. The library provides functions for working with graphs, such as graph conversion from/to *graph6/sparse6* formats, checking if the graph is edge-colorable, whether two graphs are isomorphic, and many more. This library uses `nauty` [20]. `Nauty` is a program written in C for computing automorphism groups of graphs and digraphs (directed graphs). It can also produce canonical labels. But since we do not directly work with `nauty`, we will not provide detailed information on it.

2.3 Generation algorithm

We wanted to generate minimal subcubic graphs, which were uncolorable. That means that for every graph we generated, we had to check if it was minimal in the sense of having the smallest number of elements (vertices, edges) so that the graph is still uncolorable. The graph is minimal if its *every* subgraph is *colorable* or if there is *no such subgraph* that is *uncolorable*. Otherwise, we would be able to either remove a vertex or an edge to make a smaller graph that is uncolorable. Of course, if the graph itself was colorable, all its subgraphs were also colorable, so those graphs could be removed right at the beginning. If the graph was not minimal we tried recursively removing every edge, until there were no more edges to be removed to generate a subgraph that was not colorable.

While recursively going through the graphs, a set of minimal subcubic subgraphs was created by putting possible candidates into it. A graph was a candidate while no uncolorable subgraph was found. If at least one uncolorable subgraph was found, the graph itself was removed from the set and the subgraph was added into the set. To prevent redundancy of isomorphic graphs we converted the graphs into canonical forms before they were added. The conversion was done with a function from `ba-graph` library based on `nauty`'s [20] *canonical labeling*. More on how the *Canonical Graph Labeling Algorithm* works can be found in [16].

This approach works, but for symmetric graphs, we reach many of the same graphs while going into recursion. To make the time complexity better, we added a form of memoization. For every group of graphs, we memoized every subgraph we encountered, so that we did not compute the same branch of the recursion every time we came to the same subgraph. This slightly improved the time of the runs, but more on that in 2.4. Finally, the algorithm is listed in 2.1.

2.4 Memoization

When generating the subgraphs, a lot of isomorphic graphs were reached. However, since it was not necessary to recompute the same subgraph multiple times we added memoization. To memoize the subgraphs we used a `hash set`. Only canonical forms of the graphs were put into it to prevent inserting different variants of the same graph.

When generating the minimal subgraphs we recursively removed every edge from a given graph. So when we were processing a graph we inserted it into the `hash set`. The `hash set` is a set of graphs that were already processed so they can be skipped when encountered for the second time. Right at the beginning of the generation function, we verified whether we had already seen the given graph and returned to the function call or that we had not processed the graph before and added the graph to the `hash set`.

Algorithm 2.1: Generation algorithm

```
void generate_minimal_subgraphs(graph g, set memo, set subgs){
    if(canonical g in memo){return;}
    put canonical form of g into memo;

    if(g is not colorable){
        put g into subgs;

        for(every edge of g){
            new_g = remove edge from g;

            if(new_g is not colorable){
                remove canonical g from subgs if present;
                put canonical form of new_g into subgs;
                generate_minimal_subgraphs(new_g, memo, subgs);
            }
        }
    }
}
```

But since the graphs can share some subgraphs, we kept the `hash set` for the whole group of graphs with the same number of vertices. A slight difference can be seen even in the computation of subgraphs for the smallest snark - Petersen graph (figure 2.1). Without the use of memoization, the program ran for 223 milliseconds, and with memoization, it only took 9 milliseconds. In this case, the memoization approach was almost 25 times better. When we ran the program on groups of cubic graphs with a girth of at least 5 with 10, 12, 14, 16, and 18 vertices right one after another, with memoization, it finished computing after 16.266 seconds, but without the memoization, it did not finish even after a few hours.

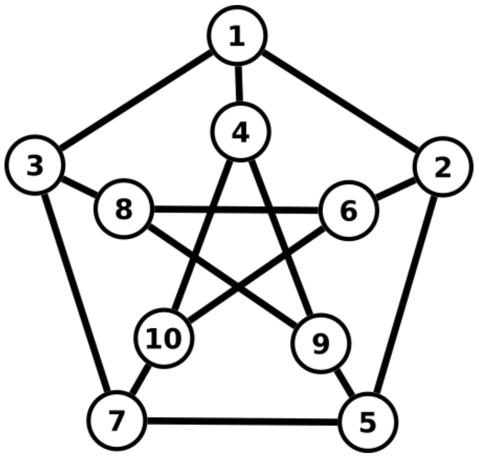

Figure 2.1: Petersen graph

This approach was good enough for the algorithm to run in a feasible amount of time for the input we had. An improvement that could be made is to make the `hash set` shared for all groups of input graphs with various numbers of vertices. But using this method the information about the origin of the subgraph would be lost since a subgraph might have multiple graphs from which it was created. A graph would be inserted into the `hash set` at first sighting and ignored on every other, so if any of its subgraphs were minimal and uncolorable, we would not know they are reachable from the new starting graph.

2.5 Proper memoization

To not have to compute every subgraph all over again and to keep track of the origin of subgraphs we modified the memoization we already had. When already seen graph G_s was encountered while computing another graph's G subgraphs, we simply added all subgraphs of the already-seen graph G_s to the subgraphs of the graph G. To keep track of which subgraph belongs to which graph we had to change the `hash set` to `hash map`. Time-wise, on input snarks with a girth of at least 5 and the number of vertices equal to 10, 18, 20, 22, and 24 without the proper memoization, we were able to run the program in 4 hours. With the proper memoization, the computation took only 2 hours and 41 minutes which is almost 1.5 times better. The algorithm is shown in listing 2.2

One last update that could be made to make the time of each generation smaller, is to save the `hash map` into a file and load it every time the program is run. That could speed up the run since all computations of subgraphs for given input graphs would only have to be done once.

Another type of improvement that could be implemented is threading.

2.6 Correctness of the generation algorithm

We prove the correctness of the algorithm using induction. We do not take the memoization into account, since it does not change the algorithm in terms of logic.

As the base case, we take a graph consisting of two vertices connected by an edge, which is trivially colorable. For its colorability, our algorithm evaluates the condition on line 9 as `false` (listing 2.2). Therefore returns an empty set of subgraphs, since colorable graphs do not have uncolorable subgraphs.

Now let's assume that for every cubic graph G' with *less* than n edges the algorithm returns a correct set of uncolorable subgraphs (the induction hypothesis). The inductive step looks as follows: graph G has n edges. Then at first, the function checks if G is colorable.

- If yes, then its subgraphs are irrelevant, so we can terminate.
- If no, then G is a candidate to be a minimal subgraph, so we iteratively remove every edge of G one after another, creating a new G_{new} every time an edge is removed. If G_{new} is not colorable, G is no longer a candidate because it is not minimal. Then we recursively call the function for every G_{new}. From the induction hypothesis, the recursion call returns a correct set of uncolorable subgraphs of G_{new} which are added to the set of minimal subgraphs of G.

We cover every subgraph because we are recursively removing edges in all possible ways. The memoization only keeps us from repeating the computations on similar graphs and does not change the result. The output set includes only minimal subgraphs. This is ensured by removing any graph that contains a subgraph, which is uncolorable, from the set. The algorithm will always terminate because there is only a finite number of edges that can be removed.

Algorithm 2.2: Generation algorithm with proper memoization

```
void generate_minimal_subgraphs(graph g, map memo, set subgs){
    if(canonical g in memo){
        for(every subgraph in g's subgraphs){
            add subgraph to subgs;
        };
        return;
    }

    if(g is not colorable){
        put g into subgs;

        for(every edge of g){
            new_g = remove edge from g;

            if(new_g is not colorable){
                remove canonical g from subgs if present;

                new_subgs = new empty map;
                generate_minimal_subgraphs(new_g, memo, new_subgs);

                for(every new subgraph in new_subgs){
                    add new subgraph to subgs;
                }
            }
        }
    }
    set subgs as value for canonical g in memo;
}
```

Chapter 3

Input data and output format

In this chapter, we will describe the input sets of graphs - what they were and the reasoning behind choosing them. This thesis contains graphs, that are visualized with internet tools [4], [2], and [3].

3.1 Input

For input, we downloaded various graphs from the internet graph database House of graphs [11]. The database has a lot of filtering options for searching graphs and contains many categories into which the graphs are split. In each category, there are graphs for downloading. The two main categories that interested us were *cubic* graphs [9] and *snarks* [10].

All input graphs have an even number of vertices. That is because we are working with cubic graphs for which the following lemma holds [15, 17].

Lemma 2 (Handshaking lemma)**.** For any graph $G = (V, E)$ holds

$$\textstyle\sum_{v \in V} deg(v) = 2|E|.$$

Therefore, any k-regular graph, where $k > 0$ is an odd number, must have an even number of vertices.

Cubic graphs with girth at least 5

Graphs in the first set on which we ran the program were *cubic with a girth of at least 5*. The set consisted of graphs with numbers of vertices equal to 10, 12, 14, 16, 18, and 20 (table 3.1). Our definition of snarks requires a *girth of at least 5*, so we took this group as our first input to achieve the property of being a subgraph of a snark. And since by removing edges we can increase the girth of a graph but not decrease, our subgraphs will also have a girth of 5 or more.

# vertices	10	12	14	16	18	20
# graphs	1	2	9	49	455	5783

Table 3.1: Number of cubic graphs (girth ≥ 5) by the number of vertices in a graph

With the specific requirements we need the graphs to have, there are downloadable graphs with up to 26 vertices. Graphs from 28 to 32 vertices are not accessible for download, but at least the number of them is known. The counts of the cubic graphs with girth ≥ 5 and with the number of vertices greater than 38 are still unknown.

Snarks with girth at least 5

In the second set were snarks with a girth greater than 4. The reason why we chose this set of graphs is the same as the one before, but we wanted to check, whether some graphs are only reachable from snarks. Numbers of vertices in these graphs were 10, 18, 20, 22, 24, 26, and 28 (table 3.2).

# vertices	10	18	20	22	24	26
# graphs	1	2	6	20	38	280

Table 3.2: Number of snarks (girth ≥ 5) by the number of vertices in a graph

With the properties of this group, there are also more snarks if vertices are added. From 28 vertices to 38 there are graphs for download. For greater numbers of vertices not even the counts are known.

Weakening requirements

Then we tried to weaken our requirements and let girth be at least 4 in the case of snarks. That resulted in bigger input groups - tables 3.3.

# vertices	10	18	20	22
# graphs	1	2	6	31

Table 3.3: Number of snarks (girth ≥ 4) by the number of vertices in a graph

When lower demands are made, it is clear that more graphs will meet the requirements. Therefore the computation of the graphs is harder and downloadable are only snarks with up to 36 vertices. However, for snarks, the counts for graphs with no more than 36 are known.

3.2 Output files

All generated outputs are saved in text files. We've categorized the file types into four groups:

1. Files, sorted by some certain property such as the number of vertices, girth, etc. containing only *graph6* format graphs.
2. Files containing (all) graphs in the *graph6* format, regardless of any specific properties.
3. Files containing graphs in relation to other graphs, such as a graph and its subgraphs or a subgraph, and the graphs in which the subgraph is contained.
4. Files containing additional information but without actual graphs.

All files associated with the implementation and generated data can be found at `https://github.com/emmiiika/bach-thesis` and in appendix.

Chapter 4

Results

This chapter is dedicated to the results of the generation algorithm and some statistical data. We will go through each set of input graphs independently. For each input set, we generated subgraphs for all graphs in the set and measured the time for each generation. In the following text, we will refer to girth only as g.

In every input set, we can see that there are mostly subgraphs with an odd number of vertices. One reason for this occurrence is that removing a single vertex from a graph with an even number of vertices results in subgraphs with an odd number of vertices. For cubic graphs holds the parity lemma [18].

In other words, the numbers of semiedges of colors c_1, c_2, and c_3 must have the same parity. They must also share the same parity as the total number of semiedges. In our situation, immediately after removing one vertex, three edges become connected to other vertices at only one end. Since the numbers of edges of the three different colors must share the same parity as the total number of edges, these three edges must be colored differently. Otherwise, one color would be used on an even number of edges, which would not be congruent with the total number of edges modulo 2.

In each input set, subgraphs predominantly contain an odd number of vertices. This frequently occurs because removing a single vertex from a graph with an even number of vertices typically results in subgraphs with an odd vertex count. Additionally, when a graph is uncolorable, removing one vertex does not alter the uncolorability of its subgraph (a corollary of lemma 1). Put another way, if the graph $G - v$, where v is any vertex of G is colorable, then the three semiedges must each have a different color in the coloring ϕ. Consequently, the same coloring ϕ can be applied to the entire graph G without violating the property of proper coloring. Therefore we can *always remove at least one vertex from an uncolorable cubic graph* to get an uncolorable subgraph.

4.1 Cubic graphs with $g \geq 5$

As the first input, we took the cubic graphs with g of at least 5. We processed graphs containing from 10 to 20 vertices. There are no cubic graphs with $g \geq 5$ with less than 10 vertices. The ten-vertex graph is the Petersen graph and its generated minimal uncolorable subgraph is shown in figure 4.1. As shown, only one vertex (and its edges) can be removed to keep the graph uncolorable.

# vertices	10	12	14	16	18	20
# graphs	1	2	9	49	455	5783
# colorable graphs	0	2	9	49	452	5769
# subgraphs	1	0	0	0	11	76
Time [ms]	27	1	8	49	20346	974357

Table 4.1: Statistics for cubic graphs ($g \geq 5$)

The graphs with 12, 14, and 16 vertices are all colorable, therefore they do not have any uncolorable subgraphs. Since the algorithm checks whether a graph is uncolorable right at the beginning, it does not try to remove any edges and directly skips the colorable graph making the calculation time less than a few milliseconds. The row "time" contains the overall computation times per each group.

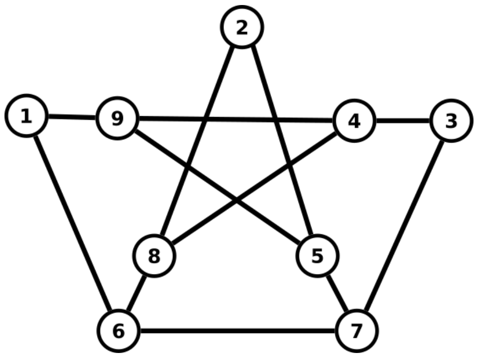

Figure 4.1: Minimal uncolorable subgraph of the Petersen graph

Given the number of colorable graphs, all the generated subgraphs were generated from 18 different uncolorable graphs. Few of the graphs share some subgraphs. For example, the Petersen subgraph is a subgraph of 9 other graphs. In figure 4.2 are shown 2 graphs with the Petersen subgraph as a subgraph. Seven other subgraphs are also subgraphs of multiple graphs and the remaining 76 subgraphs have only one origin graph.

Overall, after filtering away some redundant subgraphs, 84 different subgraphs were generated from 6299 cubic graphs. The computation took 16 minutes (table 4.2). If needed, graphs with 22 vertices could also be taken as input, which would be finished

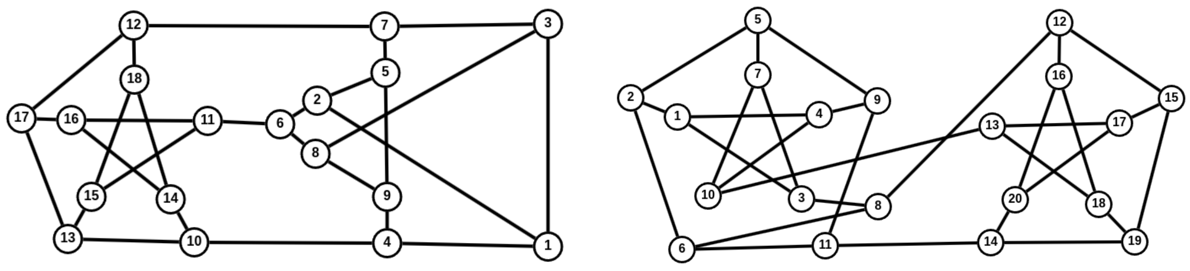
Figure 4.2: Graphs containing the Petersen subgraph

in a reasonable time. There are 90938 cubic graphs with 22 vertices which is similar to the number of cubic graphs with g at least 4.

# all graphs	6299
# all colorable graphs	6281
# all subgraphs	84
Time sum [min]	16.5798

Table 4.2: Total numbers for cubic graphs ($g \geq 5$)

# vertices	9	17	19
# subgraphs	1	10	73

Table 4.3: Distribution of subgraphs of cubic graphs ($g \geq 5$) based on the number of vertices.

Based on the number of vertices the distribution of the graphs is shown in 4.3. It may seem as if we are only able to remove one vertex from the different groups of input graphs, but many graphs have subgraphs containing fewer vertices than just one less (table 4.6). From the 10-vertex graph, we are able to get only a 9-vertex subgraph, but some of the 18-vertex graphs have 9- and 17-vertex subgraphs while some of the 20-vertex have 9-, 17-, and 19-vertex subgraphs.

graph	subgraph
10	9,
18	9, 17,
20	9, 17, 19,

Table 4.4: # vertices in (sub)graphs

graph	subgraph
15	12,
27	12, 23, 24,
30	12, 23, 26, 27,

Table 4.5: # edges in (sub)graphs

Table 4.6: Number of vertices and edges in graphs and their subgraphs

Not only are vertices removed from the graphs, but edges as well. The input graph with 15 edges (Petersen graph) has one subgraph with 12 edges. From graphs with 27 edges, subgraphs with 12, 23, and 24 edges originated. Last are graphs with 30 edges, which have subgraphs with 12, 23, 26, and 27 edges. However, not every graph with a given number of edges (or vertices) necessarily contains subgraphs with all the possible numbers of edges (or vertices).

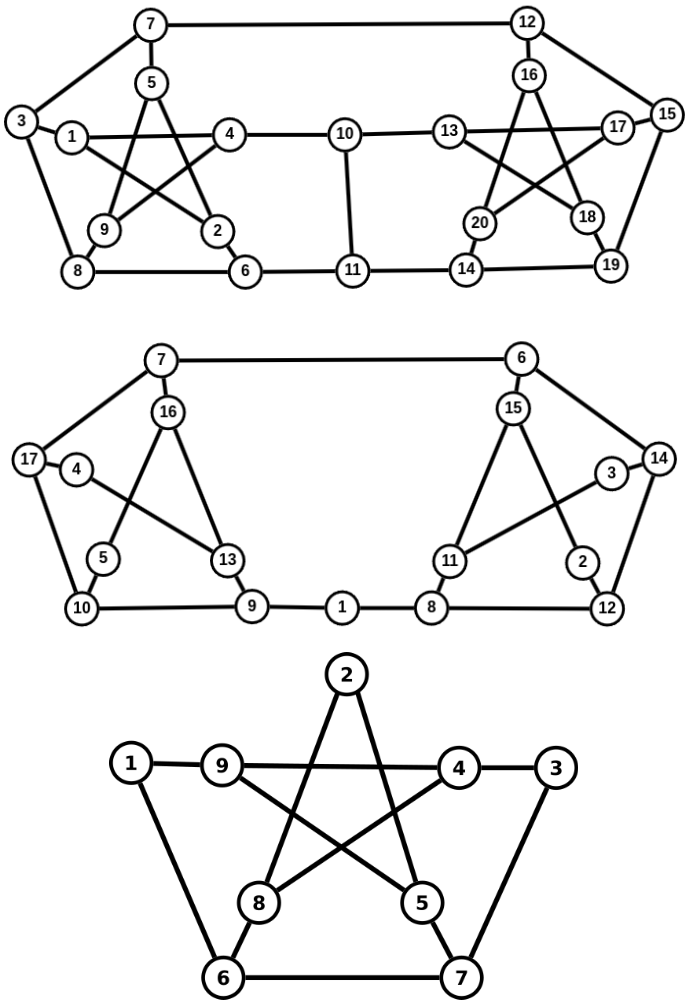

Figure 4.3: 20-vertex, 30-edge graph, and its two subgraphs

Figure 4.3 depicts a graph with 20 vertices and 30 edges with its only two subgraphs. The first subgraph has 17 vertices and 23 edges and the second one has 9 vertices and 12 edges. The change to a canonical form causes the difference in the notion of the vertices throughout the graphs.

The graphs differ in the number of subgraphs they have (table 4.7). The smallest number of subgraphs is 1 and the subgraph that the three graphs share is the Petersen subgraph. The three graphs are the Petersen graph and two others shown in figure 4.4. On the other hand, the largest number of subgraphs for this input set is 18, but only two graphs share this number. The subgraphs of the graphs in the rows of the table are not necessarily the same - the graphs can only share the number of subgraphs, not the subgraphs themselves.

# subgraphs	# graphs
1	3
2	1
3	2
5	5
6	1
7	2
8	1
10	1
18	2

Table 4.7: Cubic graphs by number of subgraphs

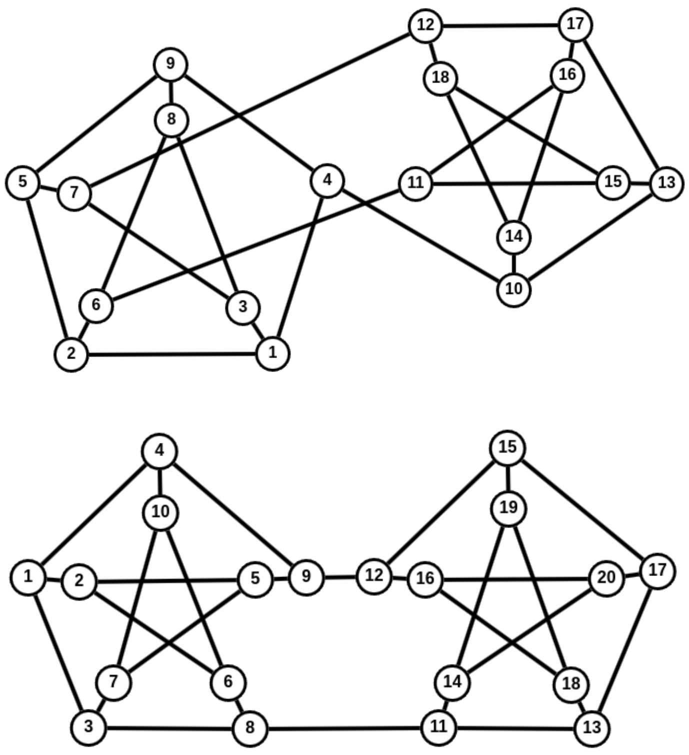

Figure 4.4: Two of three cubic graphs that have a single min. uncol. subgraph

This input set is also the only one that includes other graphs besides the Petersen graph that contain the Petersen subgraph (table 4.8). There are 10 such graphs, and for 3 out of the 10, it is the only subgraph. Eight out of all generated subgraphs have only different subgraphs.

	# graphs
not contained in	8
contained in	10
its only subgraph	3

Table 4.8: Number of cubic graphs which (do not) contain Petersen subgraph

We examined the number of vertices and edges that needed to be added to the subgraphs to form the nearest snark. Nearest snark means that the number of elements needed to be added is the smallest possible. In this input set, there are only two cases: one where 1 vertex and 3 edges need to be added and the second one where 1 vertex and 4 edges need to be added. As shown in table 4.9, 57 subgraphs can be extended in the first way, and 7 in the second way. Not all subgraphs can be extended into snarks because the subgraphs were generated from the cubic graphs which are a superset of snarks. We tried to distinguish between the snarks with girth at least 4 and 5, but this input set had the same numbers in both cases.

To find these results we took the list that contains all the generated subgraphs from each input set separately. For each subgraph in the set, we have a list of graphs in which the subgraph is contained. The graphs in each line of the file were grouped by the same canonical form of the subgraph. Then we iterated the list of graphs, finding the smallest one. Lastly, we calculated the difference between the number of elements in the subgraph and the graph. For every subgraph that can be extended into a snark, is the found snark the smallest one. If there existed a smaller one, it would have to be in the list of graphs of which the subgraph is a subgraph.

# vertices	# edges	# subgraphs
1	3	57
1	4	7
		sum = 64

Table 4.9: Subgraphs of cubic graphs by the number of elements needed to complete the nearest snark with $g \geq 4$ and $g \geq 5$ (in terms of the number of elements)

4.2 Snarks with $g \geq 5$

The second set of input graphs consisted of snarks with $g \geq 5$. Although many snarks with $g \geq 5$ are available for download, those with 26 vertices required a significant amount of computation time, as shown in table 4.10 and 4.11. From the definition, snarks are uncolorable, therefore no graph can be skipped and the time for computation is higher. But again there are no snarks with less than 10 vertices.

# vertices	10	18	20	22	24	26
# graphs	1	2	6	20	38	280
# subgraphs	1	10	55	189	471	3853
Time [ms]	8	560	6478	47896	434492	9241544

Table 4.10: Statistics for snarks ($g \geq 5$)

# all graphs	347
# all subgraphs	4525
Time sum [h]	2.703049444

Table 4.11: Total numbers for snarks ($g \geq 5$)

An example from this input set is the *Flower snark* with 20 vertices (figure 4.5). Only three vertices (and their 3 edges) are removable from this snark but always only one at a time (figure 4.6).

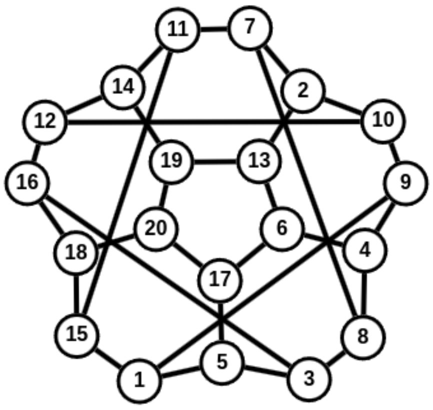

Figure 4.5: Flower snark with 20 vertices

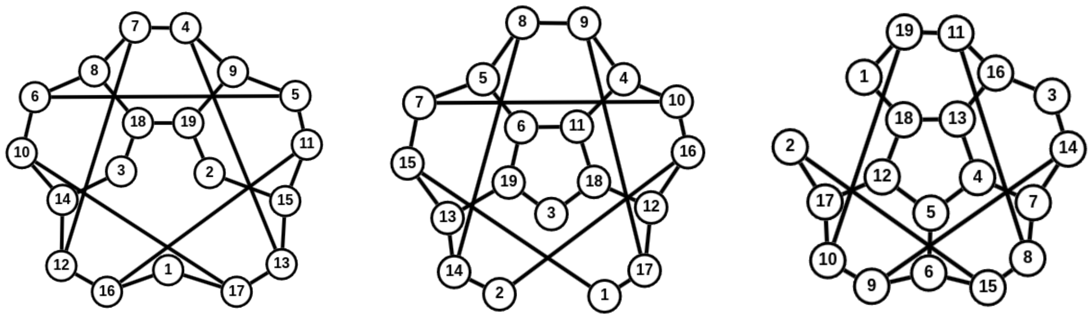

Figure 4.6: Subgraphs of 20-vertex Flower snark

Comparing the snarks and cubic graphs only on graphs with 10, 18, and 20 vertices shows, that some cubic graphs are uncolorable but are not snarks. Specifically, one 18-vertex and eight 20-vertex graphs. As expected, since snarks are a subset of cubic graphs there is also a smaller number of subgraphs of snarks than the ones of cubic graphs.

Regarding the distribution of subgraphs according to the number of vertices, we have 3 graphs with an even number of vertices (table 4.16, figure 4.8). They were created by removing 2 edges. In some cases, even vertices are removed from graphs to

# vertices	10	18	20
# all graphs	1	455	5783
# uncolorable graphs	1	3	14
# subgraphs	1	11	76

Table 4.12: Cubic graphs $g \geq 5$

# vertices	10	18	20
# graphs	1	2	6
# subgraphs	1	10	55

Table 4.13: Snarks $g \geq 5$

Table 4.14: Difference between cubic graphs and snarks

achieve these subgraphs. The 22-vertex one is achieved by removing two edges from two 24-vertex graphs or 2 vertices and 5 edges from the other 3 graphs. To get the 24-vertex subgraph only 2 edges from 2 graphs or 2 vertices and 5 edges from 3 graphs are removed. The 26-vertex graph is obtained by removing 2 edges from two different graphs. Both 22- and 24-vertex graphs have 5 graphs of origin and even share two origin graphs. The 24- and 26-vertex graphs also share 2 graphs of origin. We believe it's important to clarify that the edges we are removing are not adjacent.

In the table 4.15 are the origin graphs which the subgraphs with the even number of vertices share. The *graph6* format names of the subgraphs are long, so we will use the following notion:

(a) U???WCC@?O‘C?O?OO?‘?OOC?O_?_Aa?DA?AGG?P? (22 vertices)

(b) W???GOB?GI@O@?A?_A??_C?O_AA?GC?O@??P_?C__@GC?D? (24 vertices)

(c) Y???GODA_@_@?_@??GOOA?@??CA?AC?C?_?_C?C?_?KC?@G_?AGO?@C? (26 vertices)

shared by	origin graphs in *graph6*
(a) and (b)	W?HI@eO???C?_?OCa_?CP_???CGO?O?g??GC???w@_??@O@
	W??Y?EO??CCAP??Bo@A?‘G???_HO???_G@?G???{O??AG?@
(b) and (c)	Y?HI@eO?????_?OCa_?CP_???SG@_??SCG???@?H?G???@C?A?C???B_
	Y??Y?EO??CCAP??Bo@A?‘A???_KO??a?C_????@H??O?A?A?O?G???B_

Table 4.15: Origin graphs shared by subgraph with the even number of vertices

Based on the graphs we have generated, we can say that the subgraph with 22 vertices could be the first subgraph of snarks with an even number of vertices. We cannot say this with absolute certainty, because we only generated subgraphs for snarks with up to 26 vertices. There might be bigger graphs that will have even smaller subgraphs but based on the data from 4.19 it seems like a reasonable guess.

From the edge and vertex removal point of view, the graphs and their subgraphs can be divided as shown in table 4.19. This splitting does not mean that there is only

# vertices	9	17	19	21	22	23	24	25	26
# subgraphs	1	10	53	181	1	455	1	3822	1

Table 4.16: Distribution of subgraphs of snarks ($g \geq 5$) based on the number of vertices.

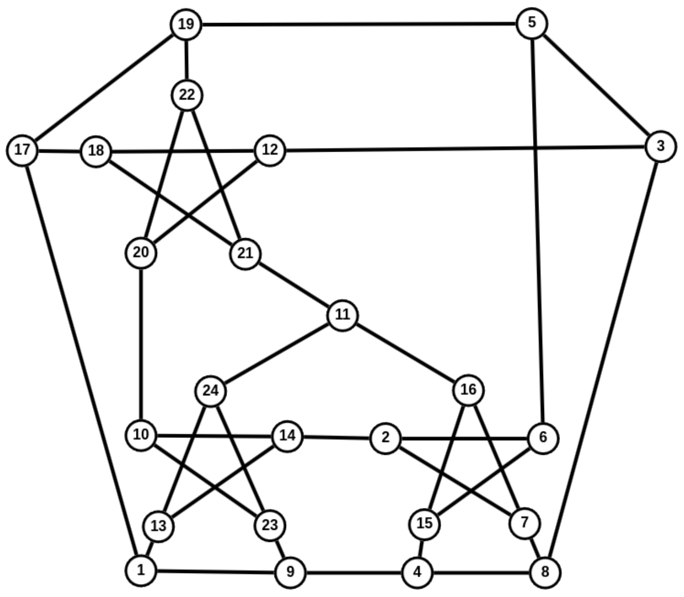

Figure 4.7: Common origin snark for 22- and 24-vertex subgraphs

one subgraph with a certain number of vertices/edges in every group - there can be multiple subgraphs with the same number of vertices/edges in one group.

It can be seen that every subgraph group with a certain number of vertices is contained in the next one except for the 9-vertex, 22-vertex, and possibly 24-vertex, and 26-vertex subgraph. This suggests that some subgraphs with an even number of vertices could be present in both graphs with the same number of vertices and graphs with two additional vertices.

On the other hand, in the table with numbers of edges, the subgraphs that are created by the removal of three edges are not contained in the next group of graphs.

This input set also differs from the previous one in the distribution of graphs by the number of subgraphs (table 4.20) - more graphs from which the subgraphs are generated, and the Petersen subgraph is a subgraph only of the Petersen graph, which does not have any other subgraphs. The biggest number of subgraphs is 26 shared by 52 graphs. An interestingly high number of graphs (137) have exactly 18 subgraphs.

This input set includes only one graph to which the Petersen subgraph is a subgraph - the Petersen graph. Every other graph in this set contains only different subgraphs. This data is shown in table 4.21.

Table 4.24 shows, that *every* generated subgraph of snarks with $g \geq 5$ can be expanded into a snark with the same girth but only some into snarks with $g \geq 4$. This

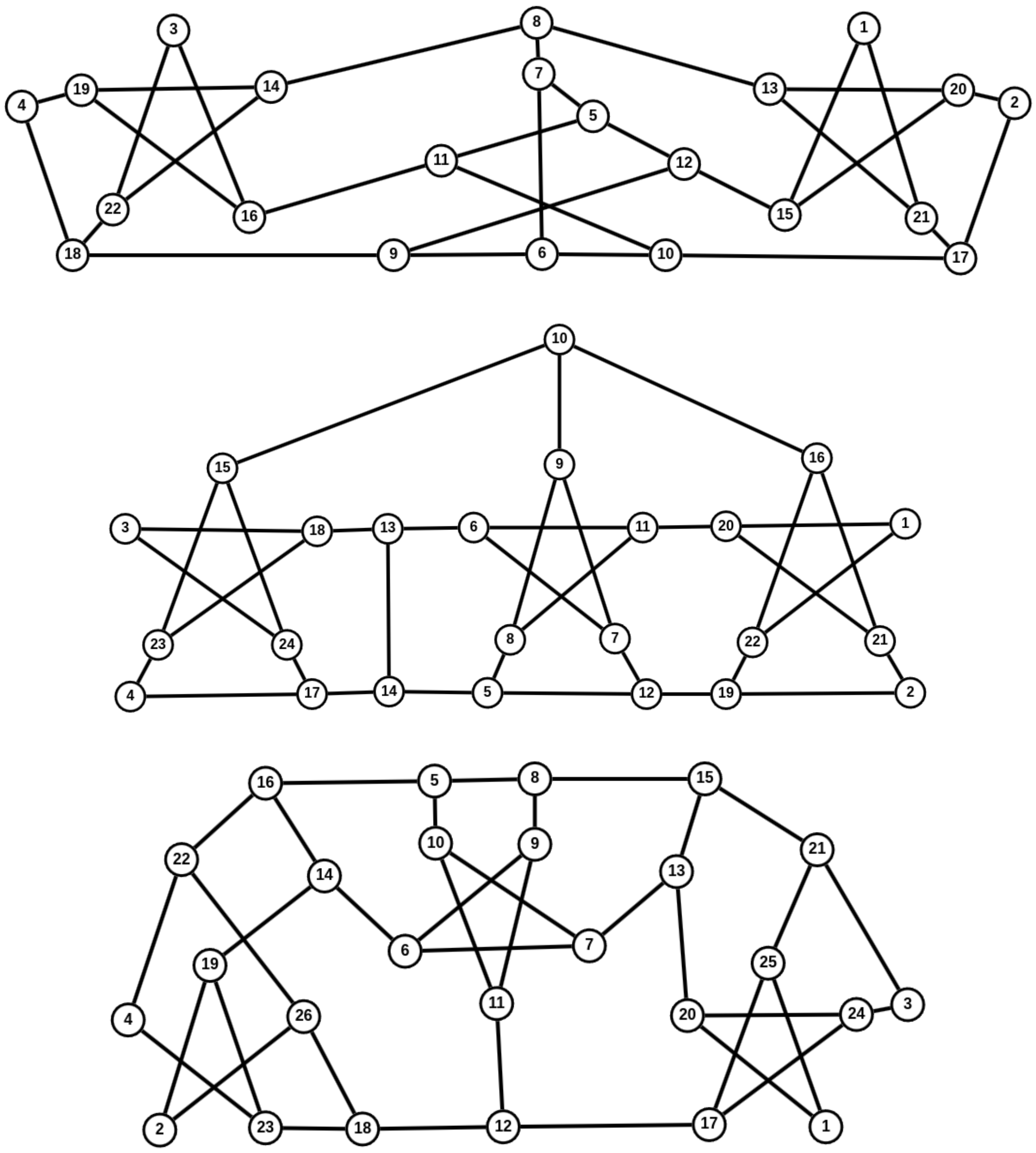

Figure 4.8: 22-, 24-, and 26-vertex subgraphs of snarks

is because of the difference in the sizes of our input sets.

The largest number of elements needed to reach the nearest snark in the first table is 1 vertex and 4 edges, which was the case for 29 subgraphs. Second's table largest difference between a subgraph and a graph for 14 graphs was 1 vertex and 5 edges.

graph	subgraph
10	9
18	17
20	17, 19
22	17, 19, 21, 22
24	17, 19, 21, 22, 23, 24
26	17, 19, 21, 23, 24, 25, 26

Table 4.17: # vertices in (sub)graphs

graph	subgraph
15	12
27	23, 24
30	23, 26, 27
33	23, 26, 29, 30, 31
36	23, 26, 29, 31, 32, 33, 34
39	23, 26, 29, 32, 34, 35, 36, 37

Table 4.18: # edges in (sub)graphs

Table 4.19: Number of vertices and edges in graphs and their subgraphs

# subgraphs	# graphs
1	1
3	3
4	2
5	8
6	6
7	13
8	7
9	5
10	38
11	11
12	17
13	2
14	15
15	13
16	4
17	1
18	137
20	1
24	8
25	3
26	52

Table 4.20: Snarks ($g \geq 5$) by number of subgraphs

	# graphs
not contained in	346
contained in	1
its only subgraph	1

Table 4.21: Number of snarks ($g \geq 5$) which (do not) contain Petersen subgraph

4.3 Snarks with $g \geq 4$

As previously mentioned, when the requirements are lowered, more graphs fulfill them. In table 4.25 are the numbers of snarks with $g \geq 4$. The number of subgraphs is also slightly larger which is reflected in the almost 2 times longer computation time for 22-vertex snarks and 3 times larger for 24-vertex snarks. That is why this input set consists only of graphs with 10 to 24 vertices. There are 1297 snarks with $g \geq 4$ and 26 vertices, so the computation would be feasible but would take at least 3 times longer.

# vertices	# edges	# subgraphs
0	2	2
1	3	671
1	4	29
		sum = 702

Table 4.22: Nearest snark with $g \geq 4$

# vertices	# edges	# subgraphs
0	2	3
1	3	4198
1	4	310
1	5	14
		sum = 4525

Table 4.23: Nearest snark with $g \geq 5$

Table 4.24: Subgraphs of snarks ($g \geq 5$) by the number of elements needed to complete the nearest snark (in terms of the number of elements)

# vertices	10	18	20	22	24
# graphs	1	2	6	31	155
# subgraphs	1	10	55	323	2031
Time [ms]	8	553	6387	84334	1284895

Table 4.25: Statistics for snarks ($g \geq 4$)

The following tables 4.26, 4.29, 4.30, 4.31, 4.32, and 4.35 show similar data as the ones in previous sections, but for snarks with $g \geq 4$. The accompanying text is omitted due to the considerable overlap with the previous set of snarks. No new findings have been identified in the graph groups.

# all graphs	195
# all subgraphs	2385
Time sum [min]	22.936283333

Table 4.26: Total numbers for snarks ($g \geq 4$)

# vertices	10	18	20	22	24
# graphs	1	2	6	20	38
# subgraphs	1	10	55	189	471

Table 4.27: Snarks $g \geq 5$

# vertices	10	18	20	22	24
# graphs	1	2	6	31	155
# subgraphs	1	10	55	323	2031

Table 4.28: Snarks $g \geq 4$

Table 4.29: Difference between snarks with $g \geq 5$ and $g \geq 4$

# vertices	9	17	19	21	22	23	24
# subgraphs	1	10	53	315	1	2004	1

Table 4.30: Distribution of subgraphs of snarks ($g \geq 4$) based on the number of vertices.

# subgraphs	# graphs	# subgraphs	# graphs
1	1	9	4
3	2	10	24
5	4	11	10
6	1	12	10
7	16	15	10
8	3	18	110

Table 4.31: Snarks ($g \geq 4$) by number of subgraphs

	# graphs
not contained in	194
contained in	1
its only subgraph	1

Table 4.32: Number of snarks ($g \geq 4$) which (do not) contain Petersen subgraph

4.4 Flower snarks

Certain snarks are distinguished by specific names and flower snarks are one such example. One specific snark was already mentioned in 4.2. We ran the generation algorithm specifically on 5 flower snarks with 20, 28, 36, 44, and 52 vertices called J_5, J_7, J_9, J_{11}, and J_{13}, respectively. All of these snarks have three uncolorable minimal subgraphs.

These graphs are very symmetrical (see figure 4.5). From the parity lemma, we know, that from uncolorable cubic graphs, we can remove one vertex without altering its uncolorability. Therefore the flower snark must have at least one vertex that is removable so that the subgraph is minimal and uncolorable. As stated, the input flower snarks all have 3 non-isomorphic subgraphs. Thus, three vertices can be removed. Each one on a different layer (figure 4.9) of the snark.

# vertices	# edges	# subgraphs
1	3	2262
1	4	121
0	2	2
		sum = 2385

Table 4.33: Nearest snark with $g \geq 4$

# vertices	# edges	# subgraphs
1	3	671
1	4	29
0	2	2
		sum = 702

Table 4.34: Nearest snark with $g \geq 5$

Table 4.35: Subgraphs of snarks ($g \geq 4$) by the number of elements needed to complete the nearest snark (in terms of the number of elements)

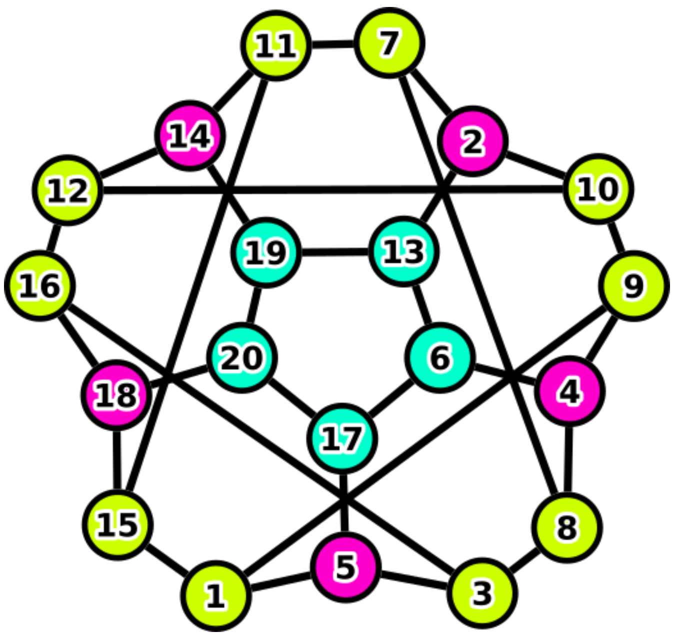

Figure 4.9: Three layers of vertices in flower snarks

Based on the structure and on the construction [12] of the flower snarks we can assume that all flower snarks will have only three minimal uncolorable subgraphs. They have three types of vertices (highlighted in color in figure 4.9): *the ones in the base (magenta vertices), in the n-cycle (cyan vertices), and in the $2n$-cycle* (green vertices). So after removing those one by one, we get 3 different subgraphs. And since the flower snarks are an infinite family of snarks, their subgraphs can also be divided into three infinite families based on the layers, from which one vertex is removed.

4.5 Future work

Some of the interesting questions we left unanswered are listed in this section.

Hamiltonian and hypohamiltonian graphs

Since we are working with graphs and their subgraphs and we also know how many vertices each graph has it could be possible to divide the input graphs into (hypo)hamiltonian and non-(hypo)hamiltonian. A graph $G = (V, E)$ with a vertex set V and an edge set E is said to be ***hypohamiltonian*** if *it is not* hamiltonian but for each $v \in V$ the subgraph $G - v$ *is* hamiltonian. This might be an interesting categorization, for example, because of the connection between these types of graphs and the *traveling salesman problem* [14].

Graphs with a subgraph with the same vertex number

We found three subgraphs that had the same number of vertices as their origin graphs, which is only a fraction of the total number of found subgraphs. The number of results is limited, because of the size and the computation time of the input group. But if a longer run time is possible or a better algorithm for graph generation is used, more graphs with the same vertex number as their origin graphs might be found.

Number of graphs having only one subgraph

The same goes for this question. From our results, it seems that only three different graphs contained only one minimal uncolorable subgraph. Our inputs were bonded above by 20, 26, and 24 vertices so with bigger input graphs the results could change.

Conclusion

This thesis aimed to compile a list of minimal uncolorable subgraphs of snarks, analyze these subgraphs and their properties, and offer potential insights into the study of snarks in graph theory.

After introducing the topic through definitions and previous works, we presented a generation algorithm based on the gradual removal of edges. Due to the lengthy run times on the input sets, we incorporated memoization. We demonstrated the correctness of the algorithm with a proof by induction.

We generated the minimal uncolorable subgraphs of three sets of input graphs: cubic graphs with girth of at least 5 up to graphs with 20 vertices, snarks with girth ≥ 5 up to 26 vertices, and snarks with girth ≥ 4 up to 24 vertices. These subgraphs were then analyzed, and the results are presented in numerous tables and figures in chapter 4.

A special case of snarks was then used as input. We generated the subgraphs for five of the *flower snarks* and identified three types of vertices present in every flower snark, which can be removed to obtain three different subgraphs. Based on this discovery, three distinct infinite families of flower snark subgraphs were identified.

Lastly, we listed some questions that could lead to interesting results if further studied.

Bibliography

[1] ba-graph library. URL: `https://bitbucket.org/relatko/ba-graph/src/master/`.

[2] Create graph online and find shortest path or use other algorithm. `https://graphonline.ru/en/`. (Accessed on 05/25/2024).

[3] House of graphs. `https://houseofgraphs.org/draw_graph`. (Accessed on 05/25/2024).

[4] terpai.web.elte.hu/d6visual.html. `https://terpai.web.elte.hu/d6visual.html`. (Accessed on 05/25/2024).

[5] Gunnar Brinkmann, Kris Coolsaet, Jan Goedgebeur, and Hadrien Mélot. House of graphs: a database of interesting graphs. *Discrete Applied Mathematics*, 161(1-2):311–314, 2013.

[6] Gunnar Brinkmann, Jan Goedgebeur, Jonas Hägglund, and Klas Markström. Generation and properties of snarks. *Journal of Combinatorial Theory, Series B*, 103(4):14–15, 25–26, 2013.

[7] Gunnar Brinkmann, Jan Goedgebeur, and Brendan D McKay. Generation of cubic graphs. *Discrete Mathematics & Theoretical Computer Science*, 13(Discrete Algorithms), 2011.

[8] Yuriy Brun. The four-color theorem. *Undergraduate Journal of Mathematics*, 2002.

[9] K. Coolsaet, S. D'hondt, and J. Goedgebeur. Connected cubic graphs, 2011. URL: `https://houseofgraphs.org/meta-directory/cubic`.

[10] K. Coolsaet, S. D'hondt, and J. Goedgebeur. Snarks, 2011. URL: `https://houseofgraphs.org/meta-directory/snarks`.

[11] Kris Coolsaet, Sven D'hondt, and Jan Goedgebeur. House of graphs 2.0: A database of interesting graphs and more. *Discrete Applied Mathematics*, 325:97–107, 2023. URL: `https://houseofgraphs.org`.

[12] Kousik Dasgupta, Somnath Mukhopadhyay, Jyotsna K Mandal, and Paramartha Dutta. *Computational Intelligence in Communications and Business Analytics: 5th International Conference, CICBA 2023, Kalyani, India, January 27–28, 2023, Revised Selected Papers, Part II*. Springer Nature, 2023.

[13] Martin Gardner. Mathematical games. *Scientific American*, 234(4):126–130, 1976.

[14] Martin Grötschel. On the monotone symmetric travelling salesman problem: Hypohamiltonian/hypotraceable graphs and facets. *Mathematics of Operations Research*, 5(2):285–292, 1980.

[15] David S Gunderson and Kenneth H Rosen. *Handbook of mathematical induction*. CRC Press LLC, 2010.

[16] Stephen G Hartke and AJ Radcliffe. Mckay's canonical graph labeling algorithm. *Communicating mathematics*, 479:99–111, 2009.

[17] Peter M Higgins. *Mathematics for the Curious*. OUP Oxford, 1998.

[18] Ján Mazák, Jozef Rajník, and Martin Škoviera. Morphology of small snarks, 2021. `arXiv:2112.04335`.

[19] Brendan McKay. graph formats, 2016. URL: `https://users.cecs.anu.edu.au/~bdm/data/formats.html`.

[20] Brendan D McKay and Adolfo Piperno. Nauty and traces user's guide (version 2.8. 6), 2022.

[21] Roman Nedela and Martin Ŝkoviera. Decompositions and reductions of snarks. *Journal of Graph Theory*, 22(3):253–279, 1996.

[22] Robert W. Robinson and Nicholas C. Wormald. Almost all cubic graphs are hamiltonian. *Random Structures & Algorithms*, 3(2):117–125, 1992.

[23] Tomáš Vician. *Cores of uncolourable cubic graphs*. Univerzita Komenského v Bratislave FMFI FMFI.KI, 2018.

[24] John J Watkins. Snarks. *Annals of the New York Academy of Sciences*, 576(1), 1989.