University of Bergen

Department of Informatics

Algorithms

New Lower Bounds on the Maximum Number of Minimal Connected Vertex Covers

Student:
Ida Ryland

Supervisor:
Professor Pinar Heggernes

Master Thesis

June 2017

Acknowledgements

First of all, I would like to thank my supervisor Pinar Heggernes for all the time, motivation and guidance I have received while working on this thesis. She has been a great help and inspiration to me. In addition, I would like to thank the whole Algorithms research group and my fellow master students who motivated me to work harder. Last but not least, I would like to thank my family, especially my brother, for all the help and support.

Contents

1 Introduction **1**

1.1 Notation 2

1.2 Vertex covers and connected vertex covers in graphs 2

1.3 Enumeration algorithms using branching 7

2 Enumeration of minimal connected vertex covers **9**

2.1 The algorithm and how we implemented it 9

2.2 Generating graphs 13

2.3 Testing correctness with a trivial algorithm 17

2.4 Implementation of Algorithm 2.1 17

2.5 An improvement of the practical running time 19

3 New lower bounds **23**

3.1 New examples of the existing lower bound 23

3.2 A better lower bound 28

3.3 An even better lower bound 34

4 Further test results **41**

5 Conclusion **49**

5.1 Summary 49
5.2 Further work 51

Chapter 1

Introduction

Graphs are important mathematical structures that are used to model many real-life problems. They can, for instance, be used to model relations between objects in a network. An example of this is the friendship structure on a social media, where the vertices in the graph represent people, or profiles, and edges represent friendships between people. Graphs can also be used to model a highway system between major cities where there is an edge between the vertices representing cities for each interstate highway between them. Such graphs can also have directed edges with a weight on each edge representing toll prices or the length of the highway. In this thesis we will only look at undirected graphs in which edges have no orientation.

An important field of study in graph theory is the study of *vertex covers*. A vertex cover in a graph is a set of vertices that *cover* all the edges. In other words, each edge in the graph has at least one endpoint in the vertex cover. Vertex covers have numerous real-life applications. Imagine you are in charge of the police department in a small town where pickpockets are a growing problem. To catch all the pickpockets you have to make sure that there is always a police officer in the neighborhood. Imagine that the edges of the graph represent the streets of the town, and the vertices represent crossroads. Your task is to place police officers on crossroads in a way such that each road is under supervision of at least one police officer, ensuring that you will be able to catch all pickpockets. The crossroads where police officers are placed correspond to the vertex cover of the graph.

For the problem above, imagine that you also have a strict budget. To save money, you want to use as few police officers as possible, without letting any pickpockets get away. We want to find the vertex cover of the graph of smallest size, i.e., a *minimum vertex cover*. We seek to find the best possible solution among all feasible solutions. Finding a minimum vertex cover is a classical graph problem, and is an example of an NP-hard problem.

In this thesis we are not interested in the solution of finding small vertex covers. Instead we are interested in listing all vertex covers of a certain type and deciding the maximum number of them. Before we can explain these concepts further, we need some notation.

1.1 Notation

We denote a graph by $G = (V, E)$, where V is the set of vertices and E is the set of edges in G. The *neighborhood* of a vertex u is the set of vertices that is adjacent to u and is denoted $N_G(u)$. The *closed neighborhood* of u is $N[u] = N_G(u) \cup \{u\}$. A vertex, $u \in G$, is a *cut vertex* if the removal of u separates the graph into several connected components. For a graph G and a subset $U \subseteq G$ we denote $G[U]$ to be the subgraph of G induced by U. We write $G \setminus U$ to denote $G[V(G) \setminus U]$ and $G - u$ if $U = \{u\}$. A set $U \subseteq V(G)$ is *connected* if $G[U]$ is a connected graph. A set of vertices $U \subseteq V(G)$ is a *vertex cover* if $\forall(u, v) \in E(G)$ either $u \in U$ or $v \in U$. A vertex cover is *connected* if U is a connected set. A (connected) vertex cover U is *minimal* if no proper subset of U is a (connected) vertex cover. A set of vertices is an *independent set* if there is no edges between any pair of these vertices. A set is independent if and only if its complement is a vertex cover.

1.2 Vertex covers and connected vertex covers in graphs

A *vertex cover* of a graph is a subset of vertices such that each edge of the graph is incident to at least one vertex in the vertex cover. More formally, in a graph $G = (V, E)$, a set of vertices $U \subseteq V(G)$ is a vertex cover if U contains at least one endpoint of every edge. A vertex in U is said to *cover* all edges incident to it. The graph $G \setminus U$ contains no edges, and hence $V(G) \setminus U$ is an *independent set*. The illustration in Figure 1.1 shows two different vertex covers of a graph.

A *minimum vertex cover* is a vertex cover of smallest possible size in a graph. A *minimal vertex cover* is a vertex cover U, with the property that no proper subset of U is a vertex cover. Note that a minimal vertex cover is not necessarily minimum. But a minimum can always be found among all minimal vertex covers of a graph. In Figure 1.1 the graph to the left is an

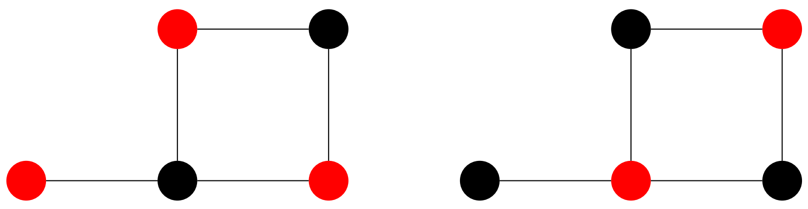


Figure 1.1: The red vertices indicate a minimal (to the left) and a minimum (to the right) vertex cover of a given graph.

example of a minimal vertex cover in a graph that is not minimum.

In the decision version of the vertex cover problem, we are given an instance (G, k), and the objective is to check whether G has a vertex cover of size at most k. This problem is NP-complete, i.e., there is no polynomial time solution to this problem unless P = NP. But since the problem is in NP, a suggested solution to the vertex cover problem can be verified in polynomial time. The naive approach for solving the problem of finding a vertex cover of size at most k is to try all possible subsets of the graph and check whether it is a vertex cover. This is a time consuming operation, as the number of subsets is 2^n for a graph with n vertices and hence constructing all sets has a running time of $\mathcal{O}(2^n)$. Since we know that a possible solution to the problem has to be of size at most k, it is sufficient to only try all possible subsets of size at most k in the graph, and check whether it is a vertex cover. The number of such subsets of a graph $G = (V, E)$ is $\mathcal{O}(|V|^k)$, hence the naive approach has running time $\mathcal{O}(|V|^k)$. Since k is expected to be smaller than $|V|$, this approach has a better running time than $\mathcal{O}(2^n)$. However, there exist much better algorithms to solve this problem. The problem is known to be fixed parameter tractable (FPT), and it can be solved in time $\mathcal{O}(1.2738^k + (k|V|))$ [4]. Note that this running time is exponential only in k, which can be expected to be much smaller than $|V|$. Hence such an algorithm is efficient for small k, regardless of the size of the graph input.

Extremal graph theory is a wide area that studies extreme values of graph parameters for graphs which often has certain properties. Extremal problems are at the very heart of graph theory [2, 1]. Finding the maximal number of objects in a graph given some restrictions has also very important algorithmic applications. For example, the running time of a naive approach will immediately decrease if we find a better upper bound. For many important hard problems we do not have better algorithms than the naive approach. Many exponential time algorithms start from a set of such objects for generating other results [6]. In this thesis we are interested in the maximum number of minimal vertex covers a graph can have, and listing all such sets. For this purpose, let us first note the connection between vertex covers and independent sets.

An *independent set* in a graph is a set of vertices such that for any pair of vertices in the set there is no edge connecting the two. A *maximal independent set* is an independent set that is not a subset of any other independent set. In other words, there is no vertex outside the independent set that may join it because it is maximum with respect to the independent set property. As we will prove in the next observation, a set is independent if and only if its complement is a vertex cover. Thus, if I is a maximal independent set, then the set $V \setminus I$ is a minimal vertex cover.

Observation 1.1. *A set $I \subseteq V$ is a maximal independent set of $G = (V, E)$ if and only if $V \setminus I$ is a minimal vertex cover of G.*

Proof. Assume I is a maximal independent set. Then for every $e \in E(G)$, e has at least one endpoint not in I, thus $V \setminus I$ is a vertex cover. Suppose that $V \setminus I$ is not a minimal vertex cover. Then there exists a vertex u in $V \setminus I$ such that $V \setminus (I \cup \{u\})$ also is a vertex cover. This means that u can not have any neighbors in I, since then there would be at least one edge not covered. Hence, all neighbors of u are in $V \setminus I$. But then $I \cup \{u\}$ is also an independent set, which contradicts the maximality of I.

Assume S is a minimal vertex cover such that every $e \in E(G)$ has at least one endpoint in S. There are no edges between vertices outside of S and the set $V \setminus S$ is an independent set. Suppose that $V \setminus S$ is not a maximal independent set. This means that there is a vertex u such that $(V \setminus S) \cup \{u\}$ is also an independent set. But then $S \setminus \{u\}$ is also a vertex cover. This contradicts the minimality of S, hence the set $V \setminus S$ must be a maximal independent set. □

Earlier work by Miller and Muller [10] , and by Moon and Moser [11] stated that the maximum number of maximal independent sets a graph on n vertices can have is $3^{n/3}$. This bound is tight as a disjoint union of $n/3$ triangles has exactly $3^{n/3}$ maximal independent sets. Any maximal independent set in this graph is formed by choosing one vertex from each triangle. There are three vertices to choose from in each triangle, and the number of triangles is $n/3$. Since the choice of a vertex in each triangle is independent from the other triangles, the number of maximal independent sets in the graph is $3^{n/3}$. Following from Observation 1.1, the maximum number of minimal vertex covers a graph on n vertices can have is equal to the maximum number of maximal independent sets. Thus it follows that the bound on the maximum number of minimal vertex covers in graphs is the same as the bound on the maximum number of maximal independent sets in graphs.

We give the example graph that has $3^{n/3}$ minimal vertex covers in Figure 1.2.

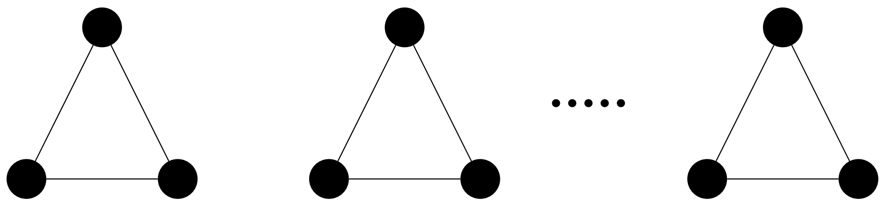


Figure 1.2: A graph having $3^{n/3}$ minimal vertex covers.

This is one of the most classical and well known examples of enumerating objects in graphs. Within a polynomial factor of the upper bound proved by Moon and Moser, this can easily extend to an algorithm that enumerates all minimal vertex covers in graphs. An example of this is the algorithm by Tsukiyama [13]. Better bounds than $3^{n/3}$ have been proved for special graph classes. One example of this is enumerating maximal independent sets in triangle-free graphs. The tight bound of $2^{n/2}$ is proved with combinatorial arguments by Hujtera and Tuza [8] and algorithmically by Byskov [3].

A *connected vertex cover* is a vertex cover U such that the subgraph induced by U is connected. From a practical view, we can imagine in our example with police officers, that each officer should be able to see another officer at all times for security reasons. The same observations about minimal and minimum vertex covers also hold for connected vertex covers. A *minimum connected vertex cover* is the connected vertex cover of smallest possible size. A *minimal connected vertex cover* is a connected vertex cover U such that no proper subset of U is a connected vertex cover. In the illustration in Figure 1.3 the red vertices in the graph to the left is an example of a minimal connected vertex cover which is not minimum. All minimum connected vertex covers must be in the set of all minimal connected vertex covers of a graph.

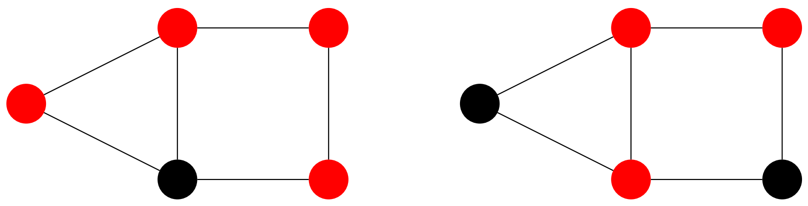


Figure 1.3: Minimal and minimum connected vertex covers of a graph.

As we saw above, enumeration of minimal vertex covers are well studied. However, the problem of enumerating minimal connected vertex covers has not been given the same attention. A recent paper by Golovach, Heggernes and Kratsch [7] studies exactly this problem. The authors give an algorithm for enumerating all minimal connected vertex covers in time $\mathcal{O}(1.8668^n)$. This also provides an upper bound of 1.8668^n on the number of minimal connected vertex covers a graph can have. They also provide a lower bound, which is a graph that has $3^{(n-1)/3} \sim 1.4422^n$ minimal connected vertex covers. This graph is shown in Figure 1.4 with a universal vertex joining $(n-1)/3$ triangles. This leaves a gap between the lower bound of 1.4422^n and the upper bound of 1.8668^n minimal connected vertex covers in general graphs.

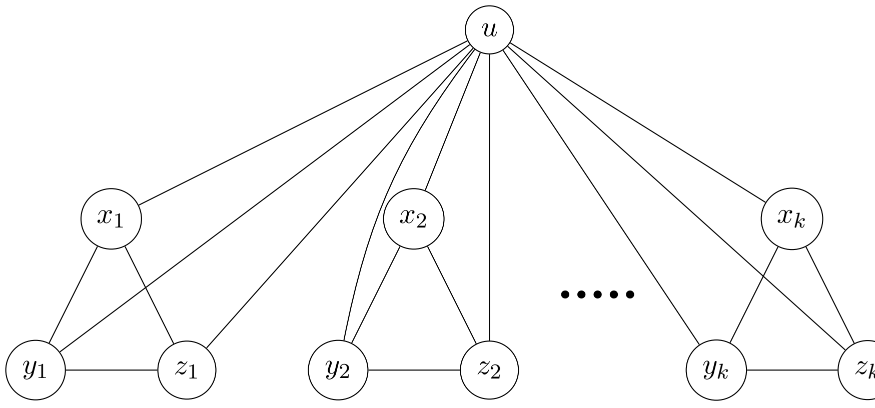


Figure 1.4: A graph having $3^{(n-1)/3}$ minimal connected vertex covers: u has to belong to every connected vertex cover, and exactly two vertices have to be selected from each triangle.

In general, a lower bound on the number of objects in a graph is given by an example graph having exactly that many objects. For the number of minimal connected vertex covers, we have seen that the lower and upper bounds are quite far apart. The main purpose of this thesis is to try to narrow this gap. We know that the real upper and lower bounds must be the same, although we do not know these. So either our upper bound is too high, or there are example of graphs with more than 1.4422^n minimal connected vertex covers. Most probably, both of these statements are true, and the bounds meet somewhere in between.

By implementing the algorithm by Golovach et al. [7] and running many tests, we hope to find better lower bound examples, and perhaps to bring down the running time of the algorithm as well.

1.3 Enumeration algorithms using branching

In theoretical computer science, we have decision problems which are problems that can be posed as yes-no questions on the input, and optimization problems where the goal is to find the best possible solution from all feasible solutions to a problem. In addition, we have extremal problems where we seek to find how many different configurations of a certain type can occur in a graph, satisfying some properties. Closely related to this, are enumeration problems. While optimization algorithms seek to find the best solution given some constraints, the goal of enumeration algorithms is to enumerate *all* solutions to a given problem.

Branching is a simple and powerful algorithmic technique, and can be used to enumerate objects. Most branching algorithms run through all possible solutions and thus have exponential running time. The method builds on the idea of backtracking, and tries to build all feasible solutions by making decisions along the way, such as deciding whether a vertex should be included in the solution or not. Such an algorithm, say $Alg(G, S)$, *branches* into several subproblems, i.e., recursive calls, $Alg(G \setminus U_1, S \cup Y_1)$, $Alg(G \setminus U_2, S \cup Y_2)$, ... , $Alg(G \setminus U_l, S \cup Y_l)$ that are solved one by one. It is important that U_i is non-empty so that the problem shrinks at every step. At each recursive call, we generate solutions that are supersets of S. In the beginning we usually call the algorithm with $Alg(G, \emptyset)$ so that all solutions are generated. Every time the algorithm branches, it generates new and smaller subproblems. Vertices that already have been branched on are not part of new instances. The execution of a branching algorithm can be viewed as a *search tree* traversed by the algorithm until a solution is discovered in one of its leaves. Each recursive call is a node in the search tree. The algorithm stops when G is empty, or when S is a solution to the problem. These are the leaves of the search tree. For a given problem, all solutions will be contained in one of the leaves in the search tree. Some subproblems are discarded, and will also be leaves in the search tree. Therefore, the number of leaves in the search tree will often be larger than the number of solutions the algorithm outputs. We measure the size of an subproblem by the number of vertices in the graph. Branching algorithms can be used to find and list all possible solutions to a problem. The solutions are obtained in the leaves of the search tree.

For an enumeration algorithm, the number of solutions that are generated go hand in hand with the running time of the algorithm, which is related to the number of nodes in the search tree. Since the total number of nodes in a tree, where every tree node branches, is at most twice the number of leaves, the number of leaves in the search tree gives both an upper bound on the number of solutions that our enumeration problem can have, and it also gives the running time of the algorithm itself. Search trees are analyzed by measuring the decrease of the instance in each branching step. We create l new subproblems at each step, where the size of the instance in the subproblems is decreased by $(c_1, c_2, ..., c_l)$, where $c_1 = |U_1|,\ c_2 = |U_2|,\ \ldots,\ c_l = |U_l|$. This is called the *branching vector*. Given this, we can create the characteristic equation $x^n = x^{c_1} + x^{c_2} + ... + x^{c_l}$. The unique positive real root to this equation gives us the upper bound on the running time of the algorithm. If α is the root, then the running time is $\alpha^n \cdot poly(n)$.

Recall that it might happen that some of the leaves in the search tree is not a solution to the problem, but rather a discarded set. As a consequence of this, the proved upper bound might be significantly higher than the maximum number of solutions that a graph can contain. Thus we search for examples of graphs on which we can find a large number of solutions. Such a graph with the proven number of solutions is called a *lower bound example*, and the proved number a *lower bound*.

Enumeration algorithms can be used to solve minimization problems, by listing all possible solutions and pick the smallest one. Although for some hard problems this is the fastest we know, usually we have a faster algorithm avoiding enumeration. In particular, as we have seen, the maximum number of minimal vertex covers a graph can have is known to be $3^{n/3} \sim 1.4422^n$. This result can easily extend to an algorithm that enumerates all minimal vertex covers in a graph within a polynomial factor of the bound. But the fastest algorithm to find a minimum vertex cover runs in time $\mathcal{O}(1.1889^n)$ [12]. We see a similar situation also for connected vertex covers: The result that we will study in this thesis shows that there are at most $\mathcal{O}(1.8668^n)$ minimal connected vertex covers in a graph. However, finding a minimum connected vertex cover can be done in time $\mathcal{O}(1.7088^n)$ [5].

Chapter 2

Enumeration of minimal connected vertex covers

In this chapter we give the details of the algorithm by Golovach et al. [7] for enumerating minimal connected vertex covers in a graph. We will also give the details of our implementation of the algorithm, how we generated graphs, and discuss how we chose to test its correctness. We did some small changes to the algorithm to improve its practical running time. We will give the details of this improvement in the last section.

2.1 The algorithm and how we implemented it

Let G be the graph whose minimal connected vertex covers we want to enumerate. The algorithm given in the paper by Golovach et al. [7] for enumerating all minimal connected vertex covers of a graph takes as input a set S of *selected* vertices and a set F of *free* vertices, where $S, F \subseteq V$, and $G = (V, E)$. The goal is to generate all minimal connected vertex covers of G that are supersets of S. The algorithm branches on a subset of *free* vertices and either *selects* some of them to be included in the minimal connected vertex cover, or forbids some of them to be selected by *discarding* them. Initially we call the algorithm with the empty set as the selected vertices and the whole vertex set V as the free vertices. In this way, all vertices of G are initially free to be selected, and all minimal connected vertex covers will be generated.

We will now describe the algorithm displayed in Algorithm 2.1 step by step. In the first step the algorithm checks whether S is a minimal connected

Algorithm 2.1 Algorithm for enumerating minimal connected vertex covers

```
1: procedure ALG($S, F$)
2:     if $S$ is a minimal connected vertex cover of $G$ then
3:         return $S$
4:     end if
5:     if $F = \emptyset$ then
6:         return
7:     end if
8:     if There are two adjacent free vertices $u, v \in F$ then
9:         ALG($S \cup \{u\}$, $F \setminus \{u\}$)
10:        ALG($S \cup N_G(u)$, $F \setminus N_G[u]$)
11:    end if
12:    if $F$ is an independent set then
13:        let $s$ be the number of components of $G[S]$
14:        for every non-empty set $X \subseteq F$ of size at most $s - 1$ do
15:            if $S \cup X$ is a minimal connected vertex cover then
16:                return $S \cup X$
17:            end if
18:        end for
19:    end if
20: end procedure
```

vertex cover of G. If so, return S and stop. This is the first rule and the base case of the algorithm. If the algorithm returns S at this step, we are at a leaf in our search tree. If S is not a minimal connected vertex cover, the algorithm continues to the second rule. The second rule of the algorithm checks whether F is an empty set. If F is in fact an empty set then stop. When F is empty, we have processed all free vertices in G, and we are at a leaf in the search tree. Note that we will not obtain a solution to our problem in this leaf and at this point the algorithm will discard S since there are no more free vertices to select. The third step, which is the only branching rule of the algorithm, branches as follows: If there exist two adjacent *free* vertices u and v in F:

- *select* u and remove it from F, i.e set $S' = S \cup \{u\}$ and $F' = F \setminus \{u\}$, and recursively call the algorithm with input S' and F'.

- *discard* u and *select* all its neighbors, i.e set $S' = S \cup N_G(u)$ and $F' = F \setminus N_G[u]$, and recursively call the algorithm with input S' and F'.

Let us explain why these two branching rules are correct. If we have two adjacent free vertices in F, S can not be a vertex cover since it does not cover all edges of G. At least one endpoint of every edge in G must belong to any vertex cover. This means that either u must be added to S, or all of the neighbors of u must be added to S. The first branching rule selects u by adding it to the set S and removing it from F. In this branch, the size of the problem shrinks by one. The second branching rule discards u by removing it from F. We know that there is at least one edge between u and another vertex in F that needs to be covered for S to be a vertex cover. This vertex is v. The algorithm therefore selects all neighbors of u by adding them to S and removing them from F. The branching rules in step 3 have a branching vector $(1, 2)$, since we remove one vertex, u, in the first branch, and at least two in the second branch, u and v.

When there are no more adjacent free vertices and F becomes an independent set, the algorithm continues in a different way. To analyze the running time, we first analyze the number of nodes in the search tree when F is an independent set. This is exactly the same number as if the algorithm would stop when F is an independent set. If the algorithm would be over at this point, then the number of leaves would be given by the $(1, 2)$ branching vector, which gives the branching number $\alpha \approx 1.61803$.

The last step of the algorithm checks whether F is an independent set. We know that S can not be a minimal connected vertex cover, or else it would have been returned in the first step of the algorithm. This means that we have to select some more vertices from F. Since F is independent, we know that all of the edges in G is covered and S is a vertex cover of G. Thus we have to select some of the vertices of F to ensure connectivity of the minimal connected vertex cover that S is a subset of. Let U be a minimal connected vertex cover of G where $U = S \cup X$ and $X \subseteq F$. Each vertex in X is a cut vertex of $G[U]$ since the set S is not connected. Since $G[S]$ has s connected components, we have to include at most $s - 1$ vertices from X to ensure the connectivity of $G[U]$, hence $|X| \leq s - 1$. Each node in the search tree where F is an independent set has a set of children that are leaves. In the paper it is computed that the total number of children is $\mathcal{O}(1.86676^n)$, where $n = |F|$.

The running time of the algorithm relies on polynomial time computations at every node in the tree. So we must show that all steps for an instance can be done in polynomial time. To verify that a set is a vertex cover can be done in polynomial time, since the vertex cover problem is shown to be NP-complete. Checking connectivity is done in linear time using breadth first search or depth first search. The only complicated step is checking minimality. We will now show that this can be done in polynomial time.

When checking whether S is a *minimal* connected vertex cover, the naive way to do this is to generate the power set of S, and for all subsets test whether they are connected vertex covers. This is a time consuming operation as the number of subsets of S is $2^{|S|}$. We prove in Lemma 2.1 that if S is a connected vertex cover, it is sufficient to check only $|S|$ subsets of S.

Lemma 2.1. *Let S be a connected vertex cover, then S is a minimal connected vertex cover if $\forall u \in S,\ S \setminus \{u\}$ is not a connected vertex cover.*

Proof. Let S be a connected vertex cover such that $\forall u \in S,\ S \setminus \{u\}$ is not a connected vertex cover. Assume for contradiction that S is not minimal. Then there must exist a set S', with $|S'| \geq 2$, such that $S \setminus S'$ is a connected vertex cover. This is illustrated in Figure 2.1. Since S' is not a part of the connected vertex cover $S \setminus S'$, all edges in S' have to be covered by $S \setminus S'$. Thus, there can not be edges between vertices in S', and S' is therefore an independent set. Let H be the graph $G \setminus S$. Then no vertices in S' can have an edge to a vertex in H, because edges between S' and H would not be covered when S' is removed.

Since both $S \setminus S'$ and S are connected vertex covers, and S' has no neighbors outside of S, all vertices of S' are necessary for connectivity of $G[S]$. However, since S' is an independent set, every vertex of S' has a neighbor in $S \setminus S'$, otherwise S would not be connected. Since $(S \setminus S') \cup \{u\}$ is a vertex cover but not a minimal connected vertex cover, u is a cut-vertex of $G[S \setminus S' \cup \{u\}]$. But then $G[S \setminus S']$ is also disconnected which gives us the desired contradiction. □

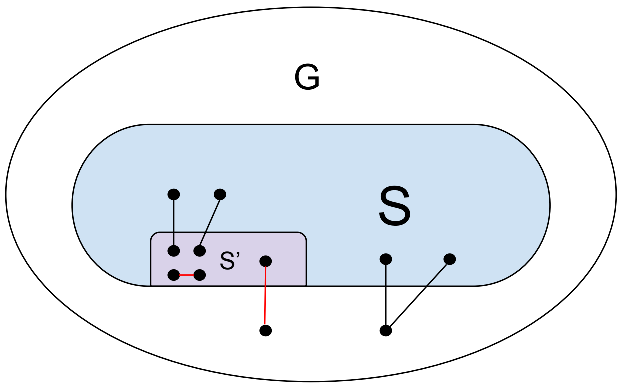


Figure 2.1: The red edges indicates edges that contradicts the assumption that S is a connected vertex cover.

2.2 Generating graphs

To test our implementation of Algorithm 2.1 we first generated graphs. We implemented an algorithm that generates all graphs up to a given size to test whether there exist graphs with a higher number of minimal connected vertex covers than the lower bound given in the paper by Golovach et al. [7]. We were able to generate all graphs on 7 vertices before running out of memory. We also implemented a second algorithm that generates connected graphs of a given size with edges distributed randomly between the vertices. This way we were able to test our implimentation of Algorithm 2.1 on larger graphs.

We will first describe the algorithm for generating connected graphs with edges distributed randomly between vertices. The algorithm is illustrated in Algorithm 2.2 and takes both the number of vertices, n, and the number of edges, e, as input. The algorithm makes sure that the graph does not contain any self-loops or multiple edges. By setting the number of edges close to the number of vertices in the graph we end up with a sparse graph, and by setting it close to the maximum number of edges, $\frac{n \cdot (n-1)}{2}$ where n is the number of vertices in the graph, we get a dense graph. The algorithm generates a graph with edges distributed randomly between the vertices. A graph with more than one component does not have any minimal connected vertex covers, and we had to ensure connectivity for the graphs we generated. Therefore, we made sure that the graps were connected before testing our implementation of Algorithm 2.1.

The first step of Algorithm 2.2 for generating graphs is to create n vertices in a list. For each of the n vertices it selects a vertex v and tries to create an edge between them. We say that an edge, (u, v), is *valid* if $u \neq v$ or the edge is not already in the list of edges. If the chosen (u, v) edge is not valid, we continue to select a random vertex v until the (u, v) edge is a valid edge, and add it as an edge in the graph. When all n vertices are processed, we are left with a connected graph G. The remaining $e - n$ edges of G are distributed as follows. Select two random vertices u and v. While u and v are the same vertex, or there already exists an edge between u and v, select two new random vertices u and v, and when the (u, v) edge is valid, add it as an edge in the graph.

We generated graphs with up to 60 vertices and 900 edges. These are not dense graphs, but based on our test results we observed that when increasing the number of edges, the number of minimal connected vertex covers and the number of discarded sets converged towards the number of vertices in the graph. This will be discussed more in the next chapter.

Algorithm 2.2 Algorithm for generating connected graphs

```
1: procedure ALG($n, e$)
2:     Create $n$ vertices as an adjacency list $A$
3:     for every vertex $u$ from $0...n-1$ do
4:         $v$ = random vertex between $0...n-1$
5:         while $u$ equals $v$ or there exists an $(u, v)$ edge do
6:             $v$ = random vertex between $0...n-1$
7:         end while
8:         Add $(u, v)$ edge to $A$
9:         $e = e - 1$
10:    end for
11:    for $i$ from 0 to $e$ do
12:        $u, v$ = random vertices between $0...n-1$
13:        while $u$ equals $v$ or there exists an $(u, v)$ edge do
14:            $u, v$ = random vertices between $0...n-1$
15:        end while
16:        Add $(u, v)$ edge to $A$
17:    end for
18:    return $A$
19: end procedure
```

As mentioned, the goal of this thesis is to try to narrow the gap between the lower and the upper bound on the number of minimal connected vertex covers in graphs. To find graphs with a higher number of minimal connected vertex covers than the existing lower bound of 1.4422^n, we generated all graphs up to 7 vertices and ran our implementation of Algorithm 2.1 on these graphs. Algorithm 2.3 generates all graphs up to a given number of vertices. The algorithm takes as input n, the number of vertices of the largest graphs to generate, and a list E of all possible pairs of the n vertices. Thus, to run the algorithm we first have to find all subsets of the n vertices of size 2. The algorithm outputs a set G_{all} of all graphs with at most n vertices. The first step of the algorithm is to add the empty set to G_{all}. It proceeds by iterating over all possible pair of vertices. For each (u, v) pair, create an edge between them and include the (u, v) edge in all previously generated graphs, and add them to G_{all}. When the algorithm terminates, we will have generated all graphs with at most n vertices.

The set of graphs returned by Algorithm 2.3 grows exponentially as n increases. There are $\frac{(n-1)n}{2}$ possible edges in a graph of n vertices that does not contain any self-loops or multiple edges. A set with $\frac{(n-1)n}{2}$ members has

Algorithm 2.3 Algorithm for generating all graphs up to size n

```
1: procedure ALG($n$, $E$)
2:     create a set $G_{all}$
3:     add the empty set to $G_{all}$,
4:     for all $(u, v)$ edges in $E$ do
5:         create a new list of subsets, $G'_{all}$
6:         for subset $S$ in $G_{all}$ do
7:             add $S$ to $G'_{all}$
8:             create a new subset $S' = S$
9:             $S' = S' \cup (u, v)$
10:            add $S'$ to $G'_{all}$
11:        end for
12:        set $G_{all} = G'_{all}$
13:    end for
14:    return $G_{all}$
15: end procedure
```

$2^{\frac{(n-1)n}{2}}$ subsets. The algorithm generates all possible subsets of the edges in E, and so there are $2^{\frac{(n-1)n}{2}}$ graphs in the set G_{all}, which the algorithm outputs. Therefore, the algorithm enumerates all graphs in time $\mathcal{O}(2^{\frac{(n-1)n}{2}})$.

The first three iterations of Algorithm 2.3 when $n = 3$ are illustrated in Figure 2.2. We chose to leave out the very first step of the algorithm when adding the empty set to the list of all graphs returned by the algorithm. Since the graphs containing just one single edge are isomorphic, the minimal connected vertex covers of the graphs will also be isomorphic, thus Algorithm 2.1 will output the same number of minimal connected vertex covers for all of the graphs. The same observation holds for the graphs that only have two edges. If we could avoid generating isomorphic graphs then the running time of the algorithm would decrease and we would not run out of memory when trying to generate all graphs up to size 8. Since the number of non-isomorphic graphs are smaller than the number of isomorphic graphs, we would be able to generate graphs larger than 7 vertices.

When we tested graphs up to 7 vertices, we discovered two graphs that we could use to create new example graphs. These example graphs have the same number of minimal connected vertex covers as the existing lower bound of $3^{(n-1)/3} \sim 1.4422^n$. We will give the details of the new example graphs in the next chapter. This observation strengthened our assumption that the existing lower bound is too low, and gave us the motivation to run tests on larger graphs.

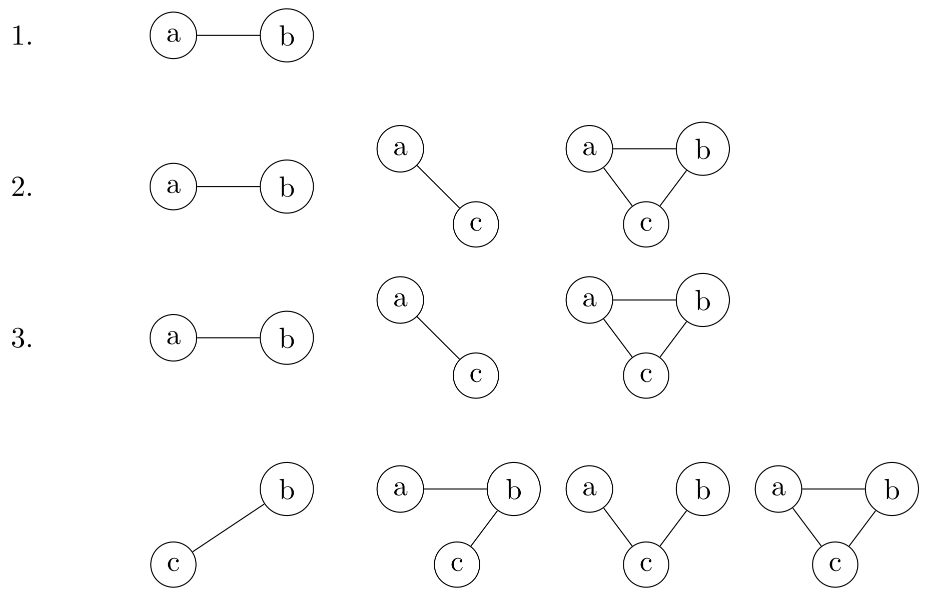


Figure 2.2: All graphs of size at most 3.

There exist tools that generate non-isomorphic graphs up to n vertices when n is small. One of them is called **geng**, and is part of a package called **gtools**, that is distributed by McKay and Piperno [9]. After testing all graphs up to 7 vertices by using Algorithm 2.3, we were curious if generating larger graphs would result in discovering graphs that could be used to prove a better lower bound. Therefore, we used **geng** to generate all graphs up to size 11. **Geng** is implemented in the programming language C, and outputs the graphs in a compressed format called **graph6**. To be able to use this data, we used a second tool from **gtools** to interpret the format, and output the graphs in a format we could read into our Java application.

According to tests run by McKay and Piperno [9], it would take 285 hours to generate all graphs up to 12 vertices, and we decided to stop at graphs of 11 vertices. In fact, as we will describe in the next chapter, with graphs of 10 vertices, we managed to construct an example graph with a higher number of minimal connected vertex covers than the lower bound of 1.4422^n proved in the paper by Golovach et al. [7]. This graph gave a theoretical basis for finding an example graph with 18 vertices, which has an even higher number of minimal connected vertex covers than the new example graph we constructed based on the graph with 10 vertices. Again, based on this graph, we theoretically proved that there exists a graph on 44 vertices that can be used to construct an example graph with an even higher lower bound. This is proved both by constructing the graph itself and by testing it with our implementation of Algorithm 2.1 in the next chapter.

2.3 Testing correctness with a trivial algorithm

Before testing our implementation of Algorithm 2.1 for enumerating minimal connected vertex covers, we had to be sure that our implementation was in fact correct. It could happen that although the algorithm in general is correct, some details are missing or are incorrect. It could also happen that during the implementation, some mistakes can occur. Such a mistake might not be easy to detect by just looking at the output. This would require manually checking that all generated sets are correct. It would be even more difficult to find out whether some sets are missing.

To overcome these issues, we implemented a very trivial, and of course slow, algorithm to generate all vertex subsets of a graph and test whether each of them is a minimal connected vertex cover.

The trivial way to enumerate all minimal connected vertex covers of a graph G is by constructing all possible subsets of G, and checking whether each of these is a minimal connected vertex cover. This algorithm has a running time of $\mathcal{O}(2^n \cdot poly(n))$ as the number of subsets for a graph on n vertices is 2^n. To test the correctness of our implementation of Algorithm 2.1 we implemented the trivial algorithm, and compared the output with the output of our implementation of Algorithm 2.1. With a running time of $\mathcal{O}(2^n \cdot poly(n))$, we were only able to test graphs up to a certain size. For graphs larger than 20 vertices, the trivial algorithm became too slow. Although, we tested our algorithm enough to be sure of its correctness.

2.4 Implementation of Algorithm 2.1

We chose to implement Algorithm 2.1 in Java. The graphs generated by Algorithm 2.2 varied from being very dense to very sparse, depending on how many edges we chose to include in the graph compared to the number of vertices. Therefore we chose to represent the graphs by adjacency-lists as opposed to adjacency-matrices. The sets S and F which the algorithm takes as input are represented as ArrayLists, where initially we call the algorithm with F being the whole vertex set of the input graph G, and S being an empty set. As opposed to arrays, when creating an ArrayList it is optional to specify its size, and ArrayLists can grow and shrink their size dynamically. This is beneficial for us, since we are adding vertices to S, and removing vertices from F. For each solution S generated by the implementation of Algorithm 2.1, we test whether it in fact is a minimal connected vertex cover. To do so we have to access the graph fast, and so we chose to represent G

as a constant field. This way G will be immutable, and can not be changed. We also have to access G when checking whether F is an independent set.

When checking whether S is a vertex cover, we iterate through all vertices of G, and for each vertex u we check whether $u \in S$ or all of its neighbors v, $v \in S$. If this holds for all vertices of G we check whether S is connected by running a depth first search. This procedure is given in Algorithm 2.4.

Algorithm 2.4 Algorithm for testing whether a set is a connected vertex cover

```
procedure Alg($S$, $G$)
    // Check whether $S$ is a vertex cover
    for each vertex $u$ in $G$ do
        for each $v$ in $N_G(u)$ do
            if $u \notin S$ and $v \notin S$ then
                return False
            end if
        end for
    end for
    //Check that $S$ is a connected set
    let $St$ be a stack
    $St$.push($S[0]$) // push the first vertex of $S$ to the stack
    while $St$ is not empty do
        $u = St$.pop()
        for each $v \in N_G(u)$ do
            if $v \in S$ and is not labeled as discovered then
                label $v$ as discovered
                $St$.push($v$)
            end if
        end for
    end while
    if there exists a vertex in $S$ which is not labeled as discovered then
        return False
    end if
    return True
end procedure
```

In Lemma 2.1 in Section 2.1 we proved that when checking minimality of a connected vertex cover it is sufficient to only check n subsets of a graph on n vertices. Checking minimality of a connected vertex cover S is done by verifying that it is not possible to remove any vertex u from S, and $S \setminus \{u\}$ being a connected vertex cover. This would result in S having a proper subset that is a connected vertex cover, and therefore S can not be minimal. We give the algorithm for checking minimality in Algorithm 2.5.

Algorithm 2.5 Algorithm for testing minimality of a connected vertex cover

```
1: procedure ALG($S$, $G$)
2:     for each $u \in S$ do
3:         $S' = S \setminus \{u\}$
4:         if $S'$ is a connected vertex cover then
5:             return False
6:         end if
7:     end for
8:     return True
9: end procedure
```

2.5 An improvement of the practical running time

We did some small adjustments to our implementation of Algorithm 2.1 to make it run faster. Instead of testing whether the set S is a minimal connected vertex cover, we first check whether S is a connected vertex cover. The problem with the original formulation in the algorithm is that if S is not a minimal connected vertex cover, the algorithm continues to work on S. However, if S is a connected vertex cover but not minimal then it can be immediately discarded at this point. If S is a connected vertex cover, we also have to check whether S is minimal. If S is minimal, we return S and stop, else we discard S and stop. This small adjustment is given in Algorithm 2.6. By implementing this adjustment, we prevent the algorithm from working on connected vertex covers that never will be minimal, and as we will see, the practical running time of the algorithm decreased.

How much time do we save by immediately discarding a connected vertex cover S after checking whether it is minimal? If a set is a connected vertex cover, but not minimal, the algorithm will not branch on S because the set F has to be an independent set and can not have any adjacent free vertices. Thus, the algorithm will eventually discard S and the size of our search tree will stay unchanged, but we still save some polynomial factor of time by implementing this small adjustment.

Let's say that S is a connected vertex cover, but not minimal. Then there can not be any adjacent free vertices in F since the edge between them would not be covered and this contradicts that S is a vertex cover. This means that the if-statement on line 8 in Algorithm 2.1 is false, and the algorithm does not branch on S. Since S is a vertex cover, F must be an independent set, otherwise there are edges in F not covered by S. This means that the if-statement on line 12 in Algorithm 2.1 is true. Recall that in Algorithm 2.1, we let s be the number of connected component in the induced subgraph $G[S]$. Since S is a connected vertex cover, the number of components of $G[S]$ is equal to one. The algorithm proceeds by constructing all non-empty subsets of S of size at most $s-1$. Since $s = 1$, $s-1$ is equal to zero and the algorithm will not construct any non-empty subsets of S. Thus at this point the algorithm eventually discards S. By discarding the connected vertex cover S immediately after we know it is not minimal, we are clearly saving some time. With this small improvement of the algorithm we prevent the algorithm to execute the following when S is a connected vertex cover but not minimal:

- Searching for non-adjacent free vertices in F
- Checking whether F is an independent set
- Counting the number of components in $G[S]$

Let us analyze the time spent by executing these three steps. Searching for adjacent vertices in F can easily be done in time $\mathcal{O}(|F|^2)$. We can in the same way check whether F is an independent set, by iterating through pair of vertices in F and making sure that there does not exist an edge between any such pairs. Counting the number of components in $G[S]$ is done by running a depth first search on the subgraph $G[S]$, which has a linear running time in the size of the graph. As proved above, all of these operations are done in a total of $\mathcal{O}(n^2)$, i.e., we save such a factor of the running time by immediately discarding S.

Algorithm 2.6 Improvement of Algorithm 2.1

```
1: procedure ALG($S, F$)
2:     if $S$ is a connected vertex cover of $G$ then
3:         if $S$ is minimal then
4:             return $S$
5:         end if
6:         Stop
7:     end if
```

In the plot given in Figure 2.3 we can see the average number of minimal connected vertex covers and the average running time for our implementation of Algorithm 2.1. The red curve is the running time before we implemented the small adjustment we mentioned above, and the brown curve is the running time of our implementation after we included the adjustment. Both of the implementations are tested on the exact same graphs, and therefore the number of minimal connected vertex covers is equal for both of the test runs. The tuples on the x-axis of the plot represent the number of vertices and edges respectively in the graphs we tested. We generated 10 graphs of each size, i.e., each fixed number of vertices and edges, using Algorithm 2.2 for generating random connected graphs, and returned the average number of minimal connected vertex cover and the average running time. The running times in the plot are scaled up by a factor of 2000.

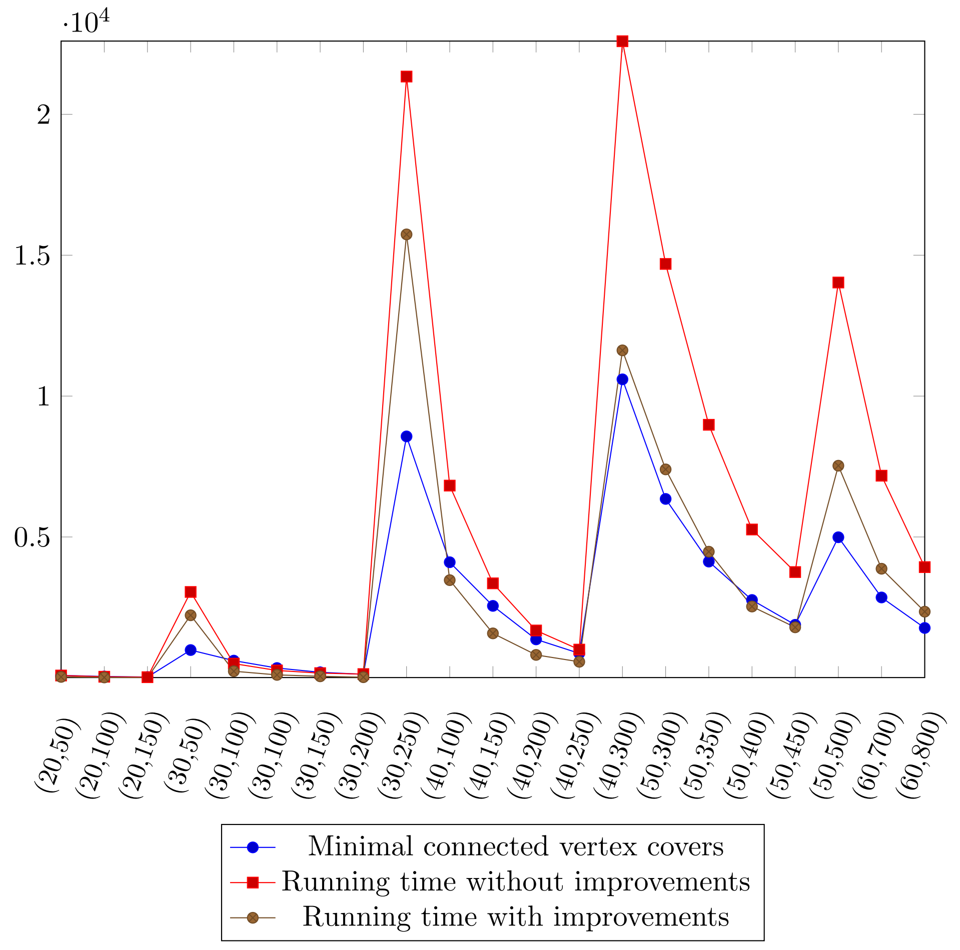


Figure 2.3: Average number of minimal connected vertex covers and running time of our implementation of Algorithm 2.1 with and without the improvemenet.

The plot in Figure 2.3 leaves a gap between the two practical running times. In fact, we can see a significant drop in the practical running time of Algorithm 2.1 after implementing the improvements. We especially see an improvement of the practical running time on sparse graphs, as these graphs have a higher number of minimal connected vertex covers than denser graphs, as we will see in Observation 4.1 in Chapter 4. The three maximal points of the plot represents the number of minimal connected vertex covers and the running time of our implementation of Algorithm 2.1 on three sparse graphs. The first one a graph with $|V| = 40$ and $|E| = 100$, the second one a graph with $|V| = 50$ and $|E| = 300$ and the last one a graph with $|V| = 60$ and $|E| = 700$.

Chapter 3

New lower bounds

In this chapter we present interesting new discoveries we made regarding the lower bound on the maximum number of minimal connected vertex covers that a graph can have. We started out with testing all graphs up to a certain size using our implementations. With practical tests we found a graph on 10 vertices with 34 minimal connected vertex covers, which we used to generate new lower bound examples with 1.4747^n minimal connected vertex covers. Even more importantly, inspired by this example, we constructed new graphs with an even higher number of minimal connected vertex covers, and on these manually constructed examples we were able to prove a lower bound of first 1.5034^n and then 1.51978^n minimal connected vertex covers.

3.1 New examples of the existing lower bound

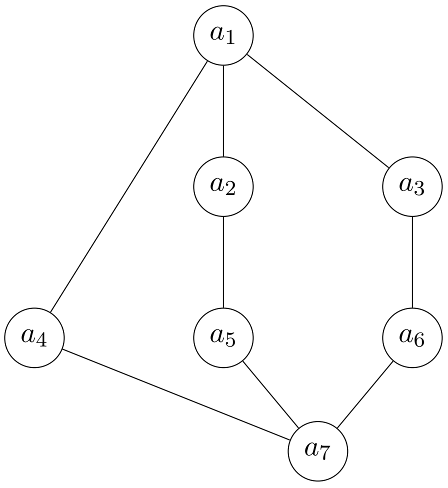
The paper by Golovach et al. [7] provides a lower bound on the maximum number of minimal connected vertex covers of a graph. They give as an example a graph that has $3^{(n-1)/3} \sim 1.4422^n$ minimal connected vertex covers. This graph is given in Figure 1.4. We did not know of any better lower bounds for the maximum number of minimal connected vertex covers on graphs in general up until we started our work. In fact, the graph given in the paper is until now the only lower bound example that has been known. The example graph G from the paper by Golovach et al. [7] consists of $k = (n-1)/3$ disjoint triangles, $\{x_1, y_1, z_1\}, \{x_2, y_2, z_2\}..., \{x_k, y_k, z_k\}$. The triangles are connected by an additional vertex u that is adjacent to every vertex in G. Every minimal connected vertex cover of G must contain u for connectivity, and exactly two vertices from each triangle. Since we have 3 ways of picking 2 vertices out of 3, each triangle has 3 minimal connected vertex covers. As the 2 vertices from each triangle are picked independently of the other triangles, G has exactly $3^{(n-1)/3}$ minimal connected vertex covers. We start by showing that there exist graphs that manage to achieve this lower bound, that are not isomorphic to the graph given in Figure 1.4.

When we generated all graphs up to 7 vertices by using Algorithm 2.3 we discovered two graphs on 7 vertices with 10 minimal connected vertex covers that we were able to construct new lower bound examples from. The two graphs given in Figure 3.1 are examples of graphs that have 10 minimal connected vertex covers each. These are listed in Table 3.1.

In general, if we want to find a new lower bound example in the same way as the graph in Figure 1.4 is constructed, we would have to find another graph on 3 vertices with 3 minimal connected vertex covers, or a graph on 6 vertices with 9 connected vertex covers, so that we could have many disjoint copies of this graph and add a new vertex for connectivity to obtain the same bound $3^{(n-1)/3} = 9^{(n-1)/6} \sim 1.4422^n$.

Let's say that a graph G has a vertex v such that v belongs to every minimal connected vertex cover of G. Then we could make a new graph G' by taking k copies of G, and glueing these copies at vertex v. In other words, G' has only one copy of v, but it has k copies of all the other vertices. Vertex v is adjacent to all copies of its neighbors, and the rest of the adjacencies in the copies of G are exactly like they are originally in G. Observe that in G', v must be in every minimal connected vertex cover to ensure connectivity. Every copy of G has exactly p minimal connected vertex covers that contain v. Thus in total, G' has p^k minimal connected vertex covers. If G has t vertices and n is the size of G', then $k = \frac{(n-1)}{(t-1)}$, and our bound is $p^{(n-1)/(t-1)}$.

(*a*)



(*b*)

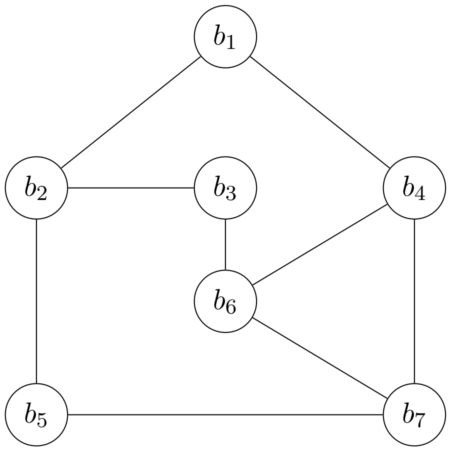


Figure 3.1: Examples of graphs on 7 vertices having 10 minimal connected vertex covers each.

When generating all graphs up to 7 vertices we did not find any graphs containing a vertex v as described above. However, each of the graphs in Figure 3.1 has 7 vertices and 10 minimal connected vertex covers, and a vertex v that is included in 9 of them. Our idea is to use this vertex v to create an infinite family of graphs with the same number of minimal connected vertex covers as the example graph given in Figure 1.4 with $\sim 1.4422^n$ minimal connected vertex covers.

We list all minimal connected vertex covers of the graphs in Figure 3.1 in Table 3.1. Observe that in graph (a), the vertices a_1 and a_7 belong to almost every minimal connected vertex cover. This is the same for vertex b_2 in graph (b). In fact, these vertices are a part of nine out of ten minimal

Graph (a)	Graph (b)
$\{a_1, a_2, a_3, a_4, a_5, a_6\}$	$\{b_1, b_2, b_3, b_5, b_7\}$
$\{a_1, a_2, a_3, a_4, a_7\}$	$\{b_1, b_2, b_3, b_6, b_7\}$
$\{a_1, a_2, a_3, a_5, a_7\}$	$\{b_1, b_2, b_4, b_5, b_6\}$
$\{a_1, a_2, a_3, a_6, a_7\}$	$\{b_1, b_2, b_4, b_6, b_7\}$
$\{a_1, a_2, a_4, a_6, a_7\}$	$\{b_1, b_2, b_5, b_6, b_7\}$
$\{a_1, a_2, a_5, a_6, a_7\}$	$\{b_1, b_3, b_4, b_5, b_6, b_7\}$
$\{a_1, a_3, a_4, a_5, a_7\}$	$\{b_2, b_3, b_4, b_5, b_6\}$
$\{a_1, a_3, a_5, a_6, a_7\}$	$\{b_2, b_3, b_4, b_5, b_7\}$
$\{a_1, a_4, a_5, a_6, a_7\}$	$\{b_2, b_3, b_4, b_6, b_7\}$
$\{a_2, a_3, a_4, a_5, a_6, a_7\}$	$\{b_2, b_4, b_5, b_6, b_7\}$

Table 3.1: List of all minimal connected vertex covers of the graphs in Figure 3.1.

connected vertex covers. Using this information, we provide a new lower bound example, which is a graph that has $9^{(n-1)/6} = 3^{(n-1)/3}$ minimal connected vertex covers. We give the new example graph, G', in Figure 3.2. G' consists of k disjoint copies of the vertices $\{a_2, a_3, a_4, a_5, a_6, a_7\}$ from graph (a) in Figure 3.1, together with a vertex a_1 which is adjacent to all copies of a_2, a_3 and a_4 in G'.

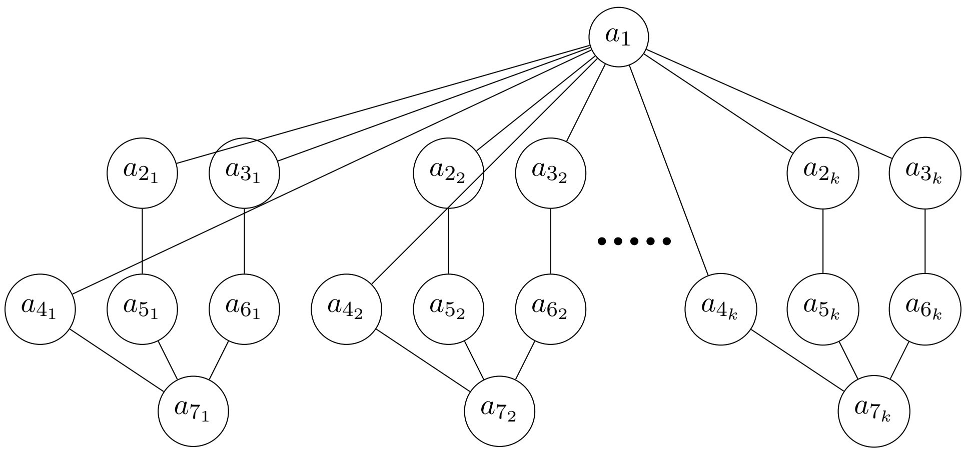


Figure 3.2: A new example graph, G', having $3^{(n-1)/3}$ minimal connected vertex covers.

Let us explain why we got this bound. The idea is similar to what we explained when G has a vertex that belongs to every minimal connected vertex cover. In our example, there is no such vertex. But still, we have a vertex that belongs to enough minimal connected vertex covers, and we can use the same idea. In particular, if a vertex v belongs to p minimal connected vertex covers of G we can use the same idea to create G' exactly as described before. In this way we will get $p^{(n-1)/(t-1)}$ minimal connected vertex covers in total, although we might lose some of the minimal connected vertex covers of each copy of G.

In our example vertex a_1 belongs to 9 out of 10 minimal connected vertex covers, so we can use the same approach to prove our bound. For connectivity, a_1 has to belong to every minimal connected vertex cover of G'. Since a_1 belongs to 9 out of 10 minimal connected vertex covers in G, we have 9 independent choices for each copy of Graph (a). Observe that the $10th$ minimal connected vertex cover of Graph (a) can not be used in this process. In fact, it is enough that only one copy uses a_1 in a minimal connected vertex cover and we could hope that we could have 10 possibilities in the remaining copies. However, as soon as a_1 is in a minimal connected vertex cover of G', the $10'th$ minimal connected vertex cover of Graph (a) is not a minimal connected vertex cover anymore.

In particular, if vertex a_1 belongs to a minimal connected vertex cover in one of the copies in G', the set $\{a_2, a_3, a_4, a_5, a_6, a_7\}$ can not belong to any minimal connected vertex cover S of G'. This is because if we remove vertex a_7, we end up with a set that in fact is a minimal connected vertex cover of G', and the minimality of the set S does not hold. Nor can the $10th$ minimal connected vertex cover of Graph (a) be included in any of the minimal connected vertex covers of G' where a_1 is not included, as this set will not be a connected set. We have now proved that each copy of Graph (a) has 9 minimal connected vertex covers including a_1, and by multiplying the number of minimal connected vertex covers in each copy by the number of copies, we see that G' has $p^k = 9^{(n-1)/6}$ minimal connected vertex covers. This is equal to the lower bound provided in the paper by Golovach et al. [7]; $3^{(n-1)/3} \sim 1.4422^n$.

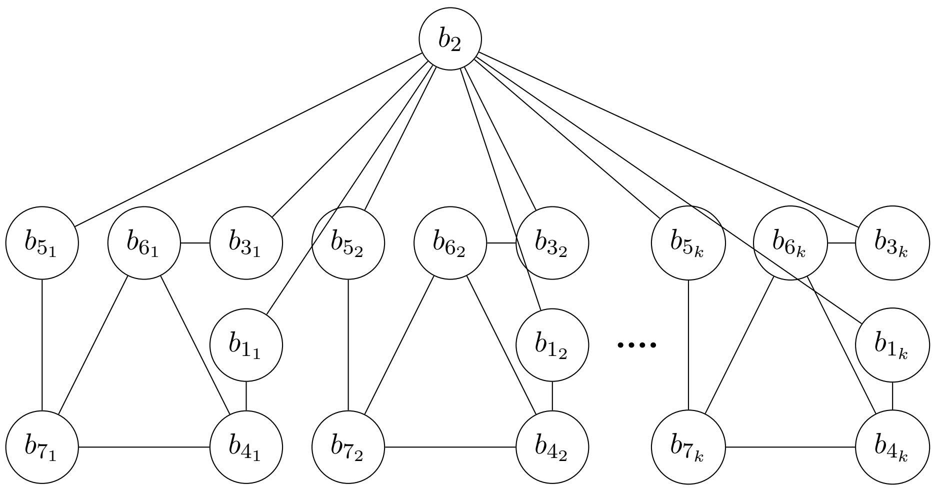


Figure 3.3: Another new example graph having $3^{(n-1)/3}$ minimal connected vertex covers.

We can use the same idea to construct a new lower bound example using Graph (b) in Figure 3.1. We give this graph in Figure 3.3. In this example, the graph is glued at vertex b_2, and each copy of Graph (b) has 9 independent choices of minimal connected vertex cover, similar to graph G' in Figure 3.2.

In this section we were able to give two new example graphs of the lower bound of $3^{(n-1)/3}$ minimal connected vertex covers on general graphs. Still, we have yet not proved that there exist graphs with a higher number of minimal connected vertex covers than the example graph given in the paper by Golovach et al. [7]. If a higher lower bound example exists, then it needs components that are larger than 7 vertices, as we up to this point tested all graphs of at most 7 vertices.

3.2 A better lower bound

The discoveries we made in the previous section made us curious if there could exist larger graphs that prove a higher lower bound on the number of minimal connected vertex covers in graphs in general. After using the graph generating tool from the package **gtools** distributed by McKay and Piperno [9] we were able to generate all graphs up to size 11. Among these graphs, we found a graph with 10 vertices and 34 minimal connected vertex covers that we could generalize to give a new and better lower bound of 1.4747^n.

These graphs gave a basis to construct even better lower bound examples. In particular we were able to modify and expand these graphs to prove a lower bound of 1.5034^n. Later in this chapter we will see that this new example graph of a better lower bound is a very sparse graph. This is not that surprising, as the graphs that prove the previously best known lower bound of 1.4422^n minimal connected vertex covers are also very sparse graphs. This section is devoted to presenting these lower bound examples. In the next section we will see that we can develop these results even further, to prove a lower bound of 1.51978^n.

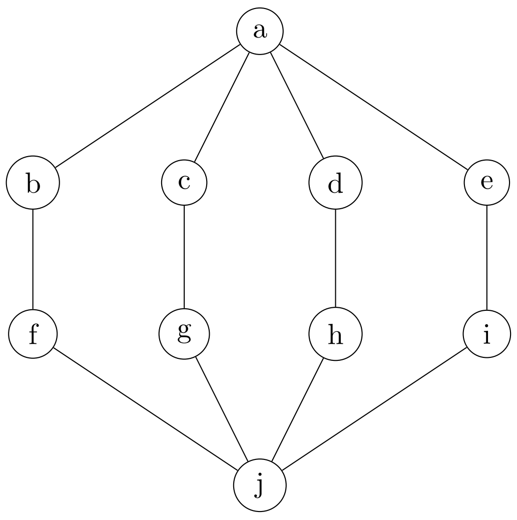


Figure 3.4: A graph on 10 vertices having 34 minimal connected vertex covers.

As already mentioned, first we discovered the graph in Figure 3.4. This is a very sparse graph. It consists of several 4-cycles glued together in the two vertices a and j. The graph has 10 vertices and 34 minimal connected vertex covers, and these are listed in Table 3.2.

$\{a,b,c,d,e,f,g,h,i\}$	$\{a,b,d,g,h,i,j\}$
$\{a,b,c,d,e,f,j\}$	$\{a,b,e,f,g,h,j\}$
$\{a,b,c,d,e,g,j\}$	$\{a,b,e,g,h,i,j\}$
$\{a,b,c,d,e,h,j\}$	$\{a,b,f,g,h,i,j\}$
$\{a,b,c,d,e,i,j\}$	$\{a,c,d,e,f,g,j\}$
$\{a,b,c,d,f,i,j\}$	$\{a,c,d,e,f,h,j\}$
$\{a,b,c,d,g,i,j\}$	$\{a,c,d,e,f,i,j\}$
$\{a,b,c,d,h,i,j\}$	$\{a,c,d,f,g,i,j\}$
$\{a,b,c,e,f,h,j\}$	$\{a,c,d,f,h,i,j\}$
$\{a,b,c,e,g,h,j\}$	$\{a,c,e,f,g,h,j\}$
$\{a,b,c,e,h,i,j\}$	$\{a,c,e,f,h,i,j\}$
$\{a,b,c,f,h,i,j\}$	$\{a,c,f,g,h,i,j\}$
$\{a,b,c,g,h,i,j\}$	$\{a,d,e,f,g,i,j\}$
$\{a,b,d,e,f,g,j\}$	$\{a,d,e,f,g,h,j\}$
$\{a,b,d,e,g,h,j\}$	$\{a,d,f,g,h,i,j\}$
$\{a,b,d,e,g,i,j\}$	$\{a,e,f,g,h,i,j\}$
$\{a,b,d,f,g,i,j\}$	$\{b,c,d,e,f,g,h,i,j\}$

Table 3.2: List of all minimal connected vertex covers of the graph in Figure 3.4.

Observe from Table 3.2 that both vertex a and vertex j each belongs to 33 out of 34 minimal connected vertex covers of this graph. Let us denote the graph of Figure 3.4 by G^*. By following the same procedure as described in the previous section, we can create a new graph G' by making k copies of G^*, and glueing each copy at vertex a. Each copy of G^* in G' has 33 minimal connected vertex covers, and G' has $33^{(n-1)/9} \sim 1.4747^n$ minimal connected vertex covers in total. This gives us a higher bound than the lower bound provided in the paper by Golovach et al. [7]; $3^{(n-1)/3} \sim 1.4422^n$.

In fact, we can generalize G^* to define a class of graphs to see if we can achieve even better lower bounds. Let us call $G(x)$ the graph given in Figure 3.5. $G(x)$ is obtained in a similar way as G^*. There are two high degree vertices in $G(x)$: a and b, with x neighbors each. $N(a) = \{a_1, a_2, ..., a_x\}$ and $N(b) = \{b_1, b_2, ..., b_x\}$. In addition to the edges between a and its neighbors, and b and its neighbors, there are exactly x edges between $N(a)$ and $N(b)$; these edges are $(a_1, b_1), (a_2, b_2), ..., (a_x, b_x)$. Observe that G^* is in fact $G(4)$. We will show that the number of minimal connected vertex covers in $G(x)$ is $x \cdot 2^{(x-1)} + 2$.

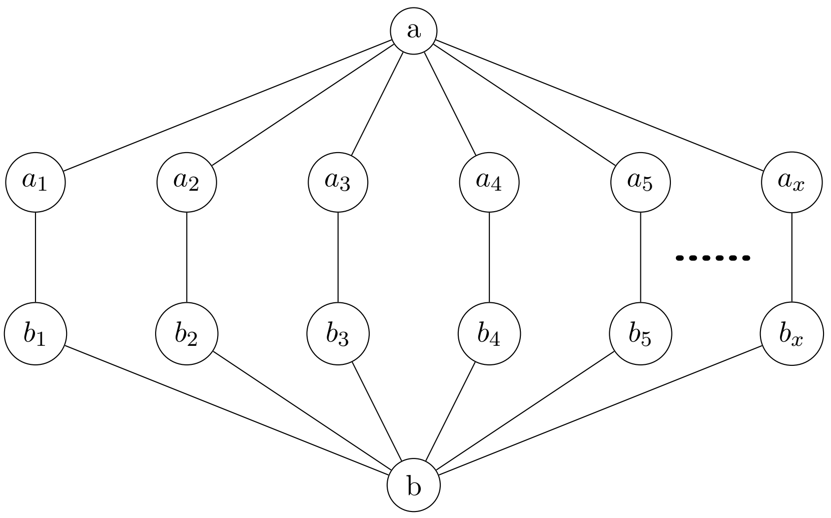


Figure 3.5: Graph $G(x)$ with an even higher number of minimal connected vertex covers.

Lemma 3.1. *$G(x)$ has $x \cdot 2^{(x-1)} + 2$ minimal connected vertex covers*

Proof. Note that there are no minimal connected vertex covers which exclude both a and b, since such a set can not be connected. Since we are looking for connected vertex covers, if both a and b are in a vertex cover then they have to be connected by a path a_1b_1, or a_2b_2, or ... or a_xb_x. We have x possibilities for this. For each of these possibilities there are $x - 1$ remaining edges that needs to be covered. Each edge can be covered by either of its endpoints. Thus we have $2^{(x-1)}$ ways to cover these edges. This gives in total $x \cdot 2^{(x-1)}$ minimal connected vertex covers in which both a and b appear. If a is not in a minimal connected vertex cover then all other vertices must be in, for connectivity and to cover all edges incident to $N(a)$. The same is true for when b is not in a minimal connected vertex cover. Thus we have in total $x \cdot 2^{(x-1)} + 2$ minimal connected vertex covers. $\square$

We can check that the number of minimal connected vertex covers of $G(4)$ is indeed $4 \cdot 2^3 + 2 = 34$.

Now, we want to see if we can achieve a better lower bound by glueing $G(x)$ at vertex a, as we did with $G(4)$. Let us call $G'(x)$ the graph which we obtain by taking a number of copies of $G(x)$ from Figure 3.5 and glueing each copy at vertex a.

Lemma 3.2. *$G'(x)$ has $(x \cdot 2^{x-1}+1)^{\frac{n-1}{2x+1}}$ minimal connected vertex covers.*

Proof. Recall that when glueing copies of $G(x)$ at vertex a, we lose one minimal connected vertex cover, namely the one that does not contain a. Thus each copy has $x \cdot 2^{(x-1)}+1$ minimal connected vertex covers. Since each copy of $G(x)$ has $2x+1$ vertices outside of a, we have in total $n = (2x+1) \cdot k + 1$ vertices in $G'(x)$, if we took k copies of $G(x)$. Thus we have $k = \frac{n-1}{2x+1}$ copies of $G(x)$. This gives a total of $(x \cdot 2^{(x-1)}+1)^{\frac{n-1}{2x+1}}$ minimal connected vertex covers. □

The number of minimal connected vertex covers in $G'(x)$ keeps increasing as x grows. However, the function $(x \cdot 2^{x-1}+1)^{\frac{1}{2x+1}}$ has a single integer maximum when $x = 8$, as we can see in the plot of the function in Figure 3.6. The graph $G'(8)$ consisting of copies of $G(8)$ glued at vertex a gives us the best bound of $(8 \cdot 2^7+1)^{\frac{n-1}{16+1}} \sim 1.5034^n$ minimal connected vertex covers. Thus, with $G'(8)$ we have been able to substantially narrow the gap between the lower and upper bounds on the number of minimal connected vertex covers in graphs.

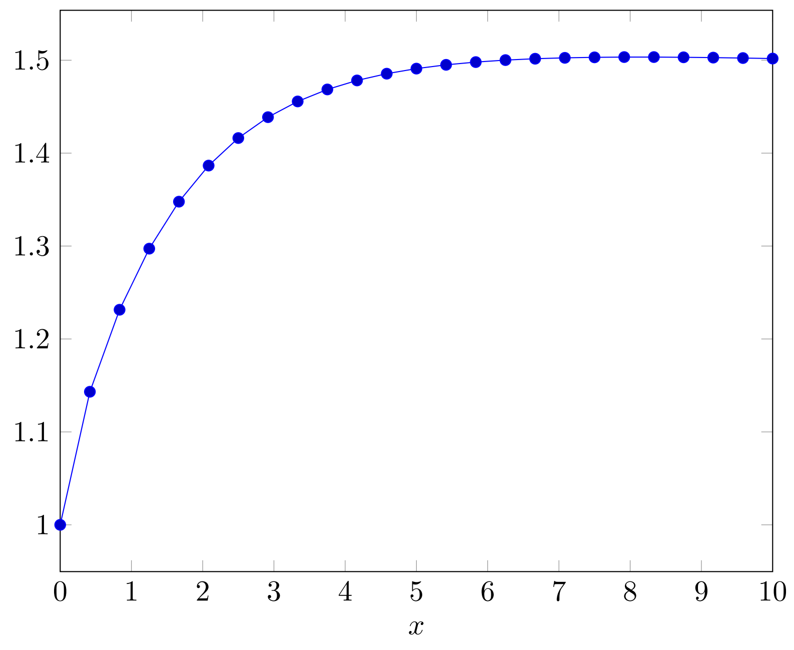


Figure 3.6: Plot of function $F(x) = (x \cdot 2^{x-1}+1)^{\frac{1}{2x+1}}$.

Corollary 3.1. *There are graphs with n vertices and 1.5034^n minimal connected vertex covers.*

Proof. $G'(8)$ is an example of such an graph. □

We have shown that by increasing the number of edges between $N_G(a)$ and $N_G(j)$ we end up with an even better lower bound than the bound we discovered with the graph $G(4)$ in Figure 3.4. By setting $x = 8$, we managed to provide a new example graph that has 1.5034^n minimal connected vertex covers. These discoveries made us curious if there are other ways we can expand the graph $G(x)$ to reach an even higher lower bound.

Let $G(x, y)$ be the graph shown in Figure 3.7. In this graph there are x vertices in the breadth, and y vertices in the depth. Observe that $G(4)$ is equivalent to $G(4, 2)$. These graphs have a higher number of minimal connected vertex covers than $G(x)$, but at the same time they have a lot more vertices. Each time we increase y by one, we add x new vertices to the graph. We will show that because of the large number of vertices, this graph will not help us prove a better lower bound than the graph $G(x)$.

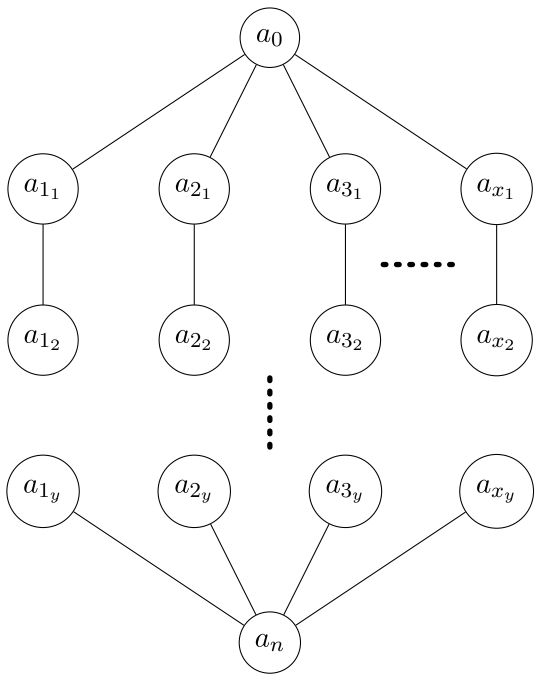


Figure 3.7: Graph $G(x, y)$.

Lemma 3.3. *$G(x, y)$ has $x \cdot y^{(x-1)} + 2$ minimal connected vertex covers*

Proof. As for $G(x)$, there are one minimal connected vertex cover which a_0 does not belong to, and one which a_n does not belong to. There are also x ways to connect vertex a_0 and a_n, for the connectivity of the minimal connected vertex cover. For the remaining vertices we also have to ensure connectivity for the minimal connected vertex covers in each path from a_0 to a_n. There are therefore y possible ways of choosing vertices on each path to be a part of the minimal connected vertex cover. Since there are $x - 1$ such paths left to cover, there are $y^{(x-1)}$ possibilities for the remaining vertices. This leaves us with $x \cdot y^{(x-1)} + 2$ minimal connected vertex covers in the graph in total. □

Let's say the graph $G(x, y)$ has p minimal connected vertex covers. Then the vertices a_0 and a_n from Figure 3.7 belong to $p - 1$ minimal connected vertex covers each. This means that we can construct a new graph $G'(x, y)$ by making k copies of the graph $G(x, y)$ and glueing each copy at vertex a_0. Each copy of $G(x, y)$ in $G'(x, y)$ contains $x \cdot y + 1$ vertices, therefore $G'(x, y)$ has $(x \cdot y^{(x-1)} + 1)^{\frac{n-1}{x \cdot y+1}}$ minimal connected vertex covers. The function has a single integer maximum when $x = 8$ and $y = 2$. But $G'(8, 2)$ is exactly $G'(8)$. This shows that we can not prove a better lower bound with the graph $G(x, y)$.

In this section we have shown that the graph $G'(x)$ gives us a better lower bound than the lower bound proved in the paper by Golovach et al. [7]; $3^{(n-1)/3} \sim 1.4422^n$. The graph $G'(8)$ has 1.5034^n minimal connected vertex covers. It turns out that there are other interesting ways to construct a new graph using the graph $G(x)$ which gives us an even better lower bound than 1.5034^n. We will see these in the next section.

3.3 An even better lower bound

We have seen that the graph $G'(x, y)$ does not give us an example of a better lower bound than the graph $G'(x)$. The problem is that each copy of $G(x, y)$ contains too many vertices, and so the exponent in the function $(x \cdot y^{(x-1)} + 1)^{\frac{n-1}{x \cdot y+1}}$ gets too small compared to the expression $x \cdot y^{(x-1)} + 1$. This is due to the dividend in the expression $\frac{n-1}{x \cdot y+1}$. But there are other ways of expanding the graph in the depth without introducing too many vertices compared to the number of minimal connected vertex covers. The graph $H(x)$ in Figure 3.8 consists of four copies of the graph $G(x)$ from Figure 3.5 glued together in the four vertices U, D, L, R. In Figure 3.8 an example is shown with $x = 4$. In the graph $H(x)$, the integer x reflects the number of edges between the neighborhoods of the four vertices U, D, L, R. In particular, each vertex U, D, L, R has $2x$ neighbors, and there are x disjoint paths of length 3 between each pair (U, L), (U, R), (D, L), (D, R).

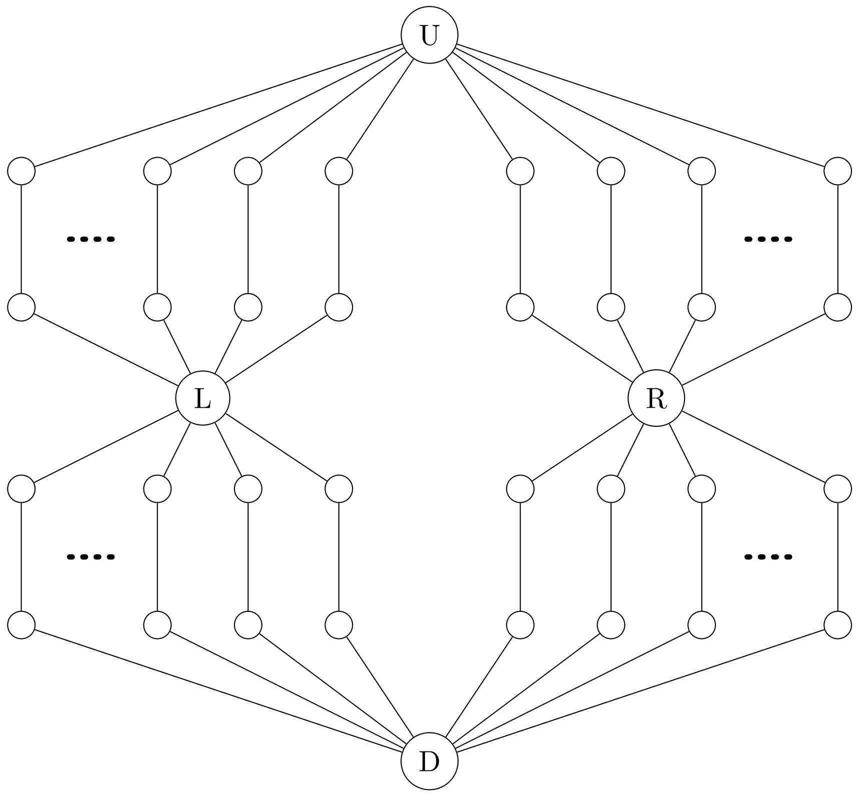


Figure 3.8: Graph $H(x)$ with a large number of minimal connected vertex covers.

Lemma 3.4. *Graph $H(x)$ has $2^{4x-1}x^3 + 2^{2x}x^2$ minimal connected vertex covers.*

Proof. Note that there is no minimal connected vertex cover which excludes both vertex U and vertex D, since such a set can not be connected. The same argument holds for excluding either vertex L or vertex R together with any of the vertices U, L, R, D.

We must count the number of minimal connected vertex covers where either U or D is missing. If U is missing, then all vertices in $N(U)$ and all vertices adjacent to a vertex in $N(U)$, except U itself, must belong to every minimal connected vertex cover. There are x possible ways to connect L to D, and x possible ways to connect R to D. These vertices have to be connected for the connectivity of all minimal connected vertex covers. For the $x-1$ edges between $N(L)$ and $N(D)$ and the $x-1$ edges between $N(R)$ and $N(D)$ there are $2^{x-1} \cdot 2^{x-1}$ possible ways to cover all these edges. The same argument holds when vertex D is missing. This gives us in total:

$$2(x \cdot x \cdot 2^{x-1} \cdot 2^{x-1})$$

If all of U, D, L and R belong to a minimal connected vertex cover together, then there are $2x \cdot x$ possible ways to connect vertex U and D. Let's say that L is a part of the path connecting U and D. Then we still have to ensure that R is connected to the rest of the minimal connected vertex cover. There are $2x$ possible ways of doing this, either choosing a path from U, or a path from D. Let's say that we chose to connect R with a path going from R to U. Then there are 2^x possible ways of covering all edges between $N_G(R)$ and $N_G(D)$. For the remaining edges in the three other component of the graph, there are $2^{(x-1)}$ possible ways to cover all the edges. This gives us in total:

$$2x \cdot x \cdot 2x \cdot 2^{x-1} \cdot 2^{x-1} \cdot 2^{x-1} \cdot 2^x$$

The last case is when both U and D belong to the minimal connected vertex cover, but either L or R is missing. There must exist a path from U to D to ensure connectivity of the minimal connected vertex cover. There are $2x$ possible ways to choose a path from U to either L or R and once that path is chosen, there are x possible ways to choose a path to D. Let's say that L is a part of this path. This means that R does not belong to any minimal connected vertex cover, and so the vertices in $N(R)$ and all vertices adjacent to $N(R)$, except R itself, must belong to every minimal connected vertex cover. For the remaining $x-1$ edges between $N(U)$ and $N(L)$ and the $x-1$ edges between $N(D)$ and $N(L)$ there are $2^{x-1} \cdot 2^{x-1}$ possible ways of covering all these edges. This gives us in total:

$$2x \cdot x \cdot 2^{x-1} \cdot 2^{x-1}$$

Summing up all these expressions, the graph $H(x)$ has:

$$2(x^2 \cdot 2^{2x-2}) + 4x^3 \cdot 2^{4x-3} + 2x^2 \cdot 2^{2x-2} = 2^{4x-1}x^3 + 2^{2x}x^2$$

minimal connected vertex covers. □

Now, again we want to see if we can achieve a better lower bound by glueing infinite many copies of $H(x)$ at a vertex which belongs to a large number of minimal connected vertex covers, as we did for the graphs in the previous section. Let us call $H'(x)$ the graph which we obtain by taking a number of copies of $H(x)$ and glueing them at vertex U.

Lemma 3.5. *$H'(x)$ has $(4^{x-1}x^2(2^{2x+1}x+3))^{n-1/8x+3}$ minimal connected vertex covers.*

Proof. Recall that when glueing copies of $H(x)$ we lose a number of minimal connected vertex covers. In Lemma 3.4 we proved that there are $x^2 \cdot 2^{(x-1)} \cdot 2^{(x-1)}$ minimal connected vertex covers which U does not belong to. Thus each copy of $H(x)$ has $x^2 \cdot 2^{(x-1)} \cdot 2^{(x-1)} + 4x^3 \cdot 2^{4x-3} + 2x^2 \cdot 2^{2x-2}$ minimal connected vertex covers. Since each copy of $H(x)$ has $8x+3$ vertices outside of U, we have in total $n = (8x+1) \cdot k + 1$ vertices in $H'(x)$ if we took k copies of $H(x)$. Thus we have $k = \frac{n-1}{8x+3}$ copies of $H(x)$. This gives a total of

$$(x^2 \cdot 2^{2x-2} + 4x^3 \cdot 2^{4x-3} + 2x^2 \cdot 2^{2x-2})^{n-1/8x+3} = (4^{x-1}x^2(2^{2x+1}x+3))^{n-1/8x+3}$$

minimal connected vertex covers. □

The function from Lemma 3.4 has a single integer maximum when $x = 5$, which we can see from the plot in Figure 3.9. This gives us 1.51978 when solving the function for $x = 5$. Thus the number of minimal connected vertex covers in $H'(5)$ is equal to 1.51978^n. This gives a better lower bound than the lower bound proved in the previous section with the graph $G'(x)$ with 1.5034^n minimal connected vertex covers.

Based on previously proved lemmas, we give the new lower bound of 1.51978^n as a theorem.

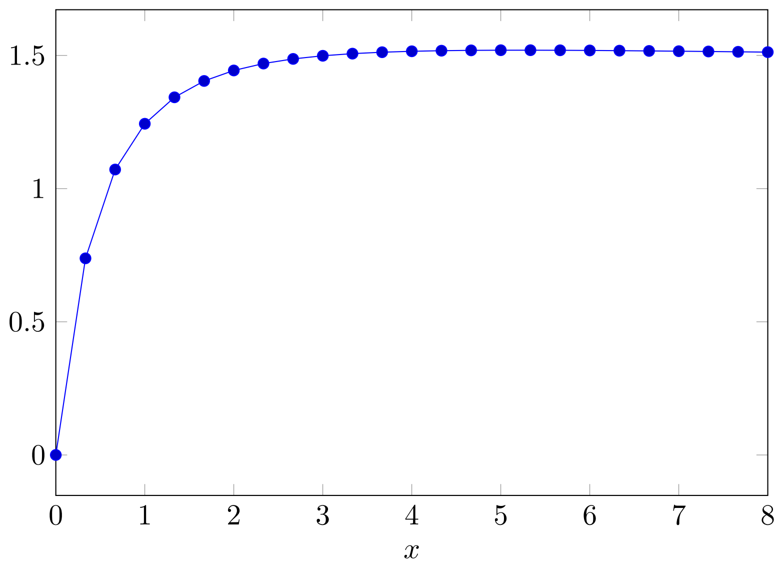


Figure 3.9: Plot of function $F(x) = (x^2 \cdot 2^{2x-2} + 4x^3 \cdot 2^{4x-3} + 2x^2 \cdot 2^{2x-2})^{1/8x+3}$.

Theorem 3.1. *There are graphs with n vertices and 1.51978^n minimal connected vertex covers.*

Proof. $H'(5)$ is an example of such a graph. □

As an illustration, the graph $H'(2)$ is given in Figure 3.10. Note that 2 is not the optimal value for x, as the graph $H'(5)$ has a larger number of minimal connected vertex covers.

We tested the graph $G(x)$ from $x = 1$ to $x = 10$, and the graph $G(x, y)$ with various values for x and y with our implementation of Algorithm 2.1. Our motivation for testing the graphs was to make sure that the algorithm returned the same number of minimal connected vertex covers as the theoretical proved functions. Also, we wanted to make sure that we have found the correct integer maximum for all of our functions and that there does not exist graphs with a higher number of minimal connected vertex covers for different values of x. The tests showed that the best value assigned to x for maximizing the function $(x \cdot 2^{(x-1)} + 1)^{\frac{n-1}{2x+1}}$ is when $x = 8$. Also, the best value for x and y in $G(x, y)$ for maximizing the function $(x \cdot y^{(x-1)} + 1)^{\frac{1}{x \cdot y+1}}$ is when $x = 8$ and $y = 2$. This graph is equivalent to the graph $G(8)$, so the graph $G(x, y)$ can not prove a better lower bound. We also tested the graph $H(x)$ with $x = 1$ to $x = 6$, which proved that the best value for x for maximizing the function $(x^2 \cdot 2^{2x-2} + 4x^3 \cdot 2^{4x-3} + 2x^2 \cdot 2^{2x-2})^{1/8x+3}$ is when

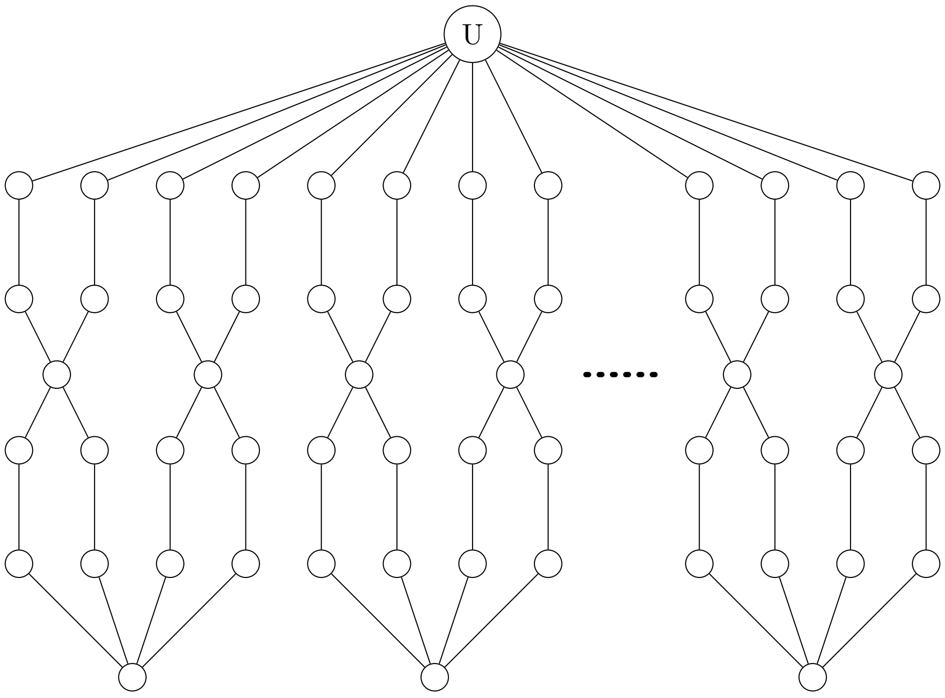


Figure 3.10: Graph $H'(2)$.

$x = 5$. The number of minimal connected vertex covers returned by the tests gave the same result as our theoretical computations and confirmed all theoretical proofs.

To conclude, in this section we managed to prove a new lower bound of 1.51978^n on the number of minimal connected vertex cover in a graph. This bound is substentially higher than the lower bound given in the paper by Golovach et al. [7] on $3^{(n-1)/3} \sim 1.4422^n$. However, it still leaves a gap between the lower bound and the upper bound of 1.8668^n minimal connected vertex covers.

Chapter 4

Further test results

In this chapter we will describe the test results we achieved when testing our implementation of Algorithm 2.1 for enumerating minimal connected vertex covers in graphs. As we saw in Chapter 3, the example graphs for proving a new and better lower bound are sparse graphs and from the plot in Figure 2.3 we can see large variations in the number of minimal connected vertex covers in sparse and dense graphs. We started by testing our implementation of Algorithm 2.1 on graphs with a fixed number of vertices and varying number of edges to see the connection between the number of minimal connected vertex covers and the number of edges in a graph. For the rest of our tests we used random graphs generated by Algorithm 2.2 due to the fact that we were able to test graphs as large as 60 vertices this way.

As described in the introduction in Chapter 1, the goal of this thesis is to try to narrow the gap between the lower and the upper bound on the number of minimal connected vertex covers in graphs. In Chapter 3 we introduced an example graph $H'(5)$ with 1.51978^n minimal connected vertex covers. With this result we have been able to raise the lower bound on the number of minimal connected vertex covers a graph can have from 1.4422^n to 1.51978^n. Still, it leaves a gap between the lower and upper bounds which is proved by the enumeration algorithm that runs in time $\mathcal{O}(1.8668^n)$ given in Algorithm 2.1 [7]. In reality, the bounds on the number of minimal connected vertex covers in graphs in general is tight, but we do not know whether the proved lower bound is too low, or the upper bound is too high. As we will show later in this chapter, some of the test results can indicate that the upper bound of 1.8668^n is too high.

The plot in Figure 4.1 shows the average number of minimal connected vertex covers, the average number of discarded sets and the average running time after testing graphs on 40 vertices. We decided to set the number of edges from 100 to 750 with an interval of 50, and for each size we generated 10 different graphs and returned the average number of minimal connected vertex covers, discarded sets and running time. The running time in the plot is scaled up by a factor of $1,000$. For the graphs with less than 300 edges the number of discarded sets are so high that we chose not to illustrate this in our plot. In fact, the average number of discarded sets for graphs with 100 edges is as high as $146,717$.

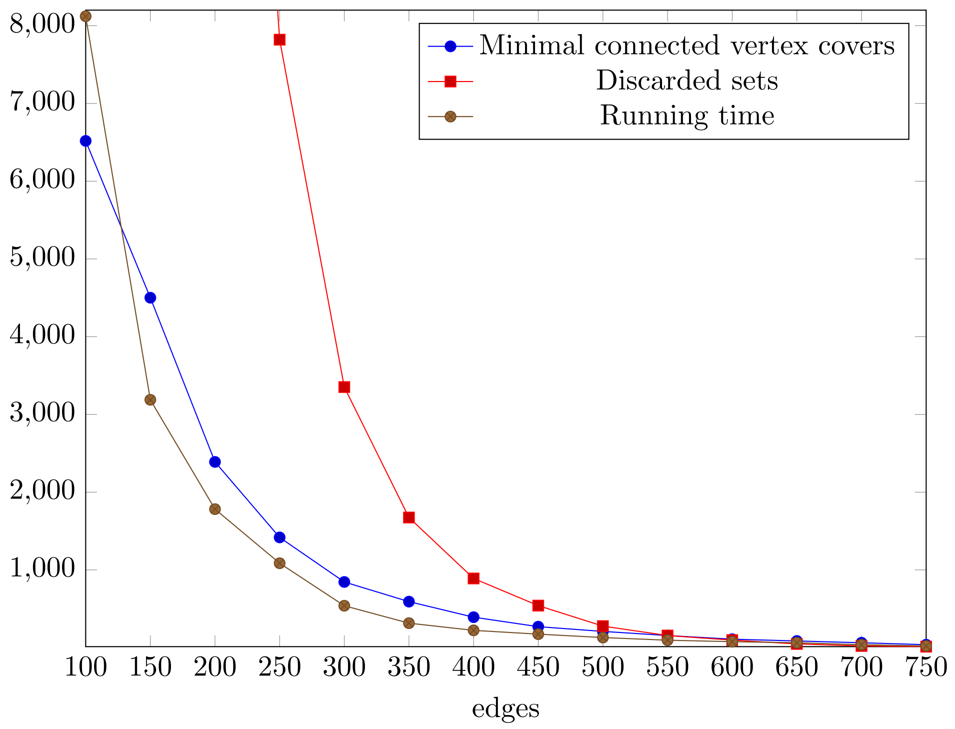


Figure 4.1: Average number of minimal connected vertex covers, discarded sets and running time of graphs on 40 vertices. The number of edges of the graphs is given on the x-axis.

In chapter 3 we saw that the lower bound examples are very sparse graphs. In fact, the example graph $H'(5)$ which proves the new lower bound of 1.51978^n consists of many 4-cycles glued together and most of the vertices in the graph are of degree 2. As we can see from the plot in Figure 4.1, the number of minimal connected vertex covers decreases drastically as we increase the number of edges. Denser graphs contain fewer minimal connected vertex cover than sparse graphs. In fact, for complete graphs the number of minimal connected vertex covers is equal to the number of vertices in the graph.

Observation 4.1. *For a complete graph with n vertices, there are n minimal connected vertex covers, each of size* $|n-1|$

Proof. For a complete graph G with n vertices and a minimal connected vertex cover U, any subset of size less than $n-1$ cannot be a vertex cover, as it would not cover edges between vertices outside of U. Since G is a complete graph there must be an edge between all pair of vertices, i.e., there must be edges between the vertices outside of U. Therefore, the set outside of U can only contain one vertex. Thus there are n minimal connected vertex covers in G, one vertex cover per vertex excluded from U. □

We tested graphs of size 40 with 100 to 750 edges without self loops or multiple edges, and from the plot in Figure 4.1 we see that the number of minimal connected vertex covers converges to n as we increase the number of edges. A graph with 40 vertices and 750 edges is a very dense graph, close to being a complete graph. This reflects in our test results where graphs with 40 vertices and 750 edges have 38 minimal connected vertex covers in average.

Although a complete graph on n vertices has n minimal connected vertex covers, and we have seen that lower bound examples are very sparse, it is interesting to see how quickly the curve representing the number of minimal connected vertex covers from the plot in Figure 4.1 starts to converge to 40 as the number of edges are increasing.

The graphs we tested to get the results we see in Figure 4.1 were all connected graphs. As we can see, the graph with 100 edges gives us the highest number of minimal connected vertex covers. Imagine that we remove some of the edges and end up with a graph on 39 edges, which is the lowest number of edges a graph on 40 vertices can have and still be connected. Then the number of minimal connected vertex covers in the graph is exactly one.

Observation 4.2. *A tree on* $n \geq 3$ *vertices has exactly one minimal connected vertex cover.*

Proof. For any tree T, we will show that there is exactly one minimal connected vertex cover S that contains all vertices, except the leaves in T. Let's say for contradiction that a leaf u is a part of a minimal connected vertex cover S. Since T has at least two edges, there is at least one more vertex x in S. Since there is exactly one unique path from u to x in T, the unique parent v of u also has to be a part of S for connectivity. Since v is a part of

S, and (u, v) is the only edge incident to u, $S \setminus u$ is also a connected vertex cover. This contradicts the minimality of S. Since this holds for every leaf in the tree, all vertices except the leaves have to belong to every minimal connected vertex cover and there is exactly one unique minimal connected vertex cover. □

Thus, a tree has only one minimal connected vertex cover, but as we can see in the plot in Figure 4.1 the number of minimal connected vertex covers is highest in the graphs with only 50 edges. This indicates that the number of minimal connected vertex covers must increase drastically when we add a few edges to a tree. In fact, we can observe that just adding one edge to a tree can increase this number from 1 to n.

Observation 4.3. *A cycle on n vertices has exactly n minimal connected vertex covers.*

Proof. For any minimal connected vertex cover S in a cycle, there is exactly one vertex u which does not belong to S. Let's say for contradiction that there is another vertex v which does not belong to S. If v is a neighbor of u, then the edge (u, v) would not be covered. If v is not a neighbor of u, then there does not exist a path between the two neighbors of u in S, thus S can not be connected. This contradicts our assumption that S is a minimal connected vertex cover. Since there are n vertices in the cycle, and there is exactly one minimal connected vertex cover per vertex we exclude, the cycle has n minimal connected vertex covers. □

With background from observations 4.1, 4.2 and 4.3 we see that the plot in Figure 4.1 gives a realistic picture of the number of minimal connected vertex covers in graphs when we increase the number of edges. However, according to observations 4.2 and 4.3, we should see an increase in the number of minimal connected vertex covers when we increase the number of edges from $n-1$ to n, whereas our curves in Figure 4.1 do not show this. This is because the figure starts from 100 edges, whereas n is only 40. It is interesting that already at $2.5n$ edges, the number of minimal connected vertex covers has started to decrease. To examine even further when the decrease starts, we also checked for edge numbers less than 100.

The plot in Figure 4.2 illustrates the average number of minimal connected vertex covers, discarded sets and running time when we tested our implementation of Algorithm 2.1 with graphs of 40 vertices and less than 100 edges. The x-axis in the plot represents the number of edges in each of the graphs. We tested 10 graphs of each size, and returned the average number of minimal connected vertex covers, discarded sets and running time. As we can see from the plot, the number of minimal connected vertex covers grows drastically when we increase the number of edges in the graphs from 50 to 65. We can also see that the curve starts to decrease gradually when the number of edges are increasing from 75.

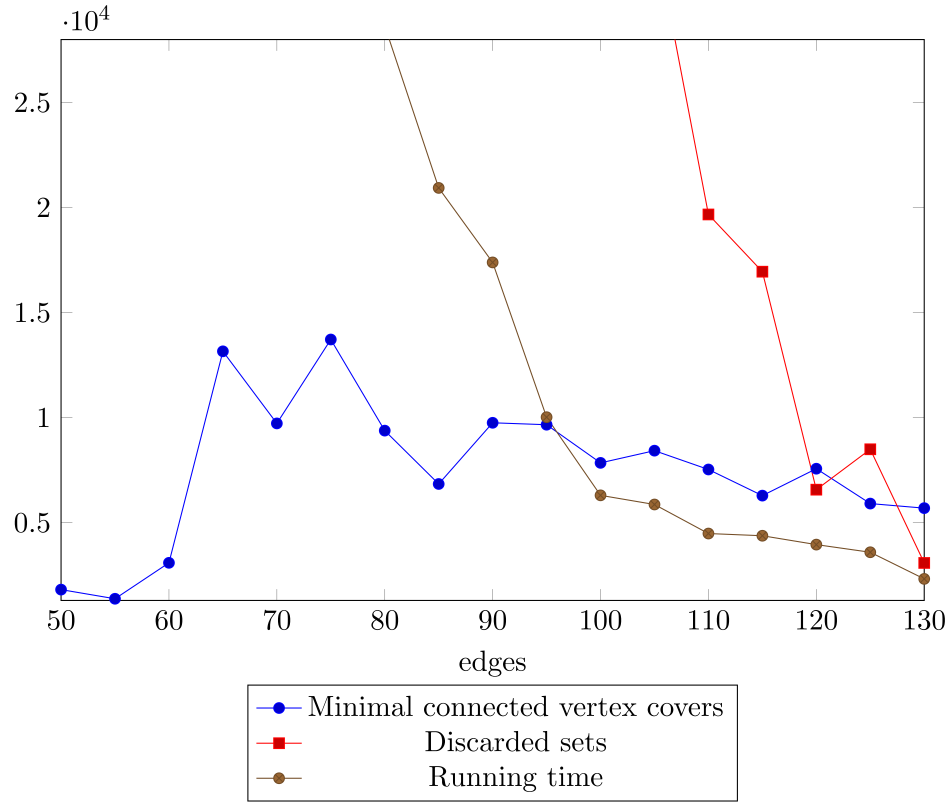


Figure 4.2: Average number of minimal connected vertex covers, discarded sets and running time of sparse graphs with 40 vertices.

We discovered large variations on the number of minimal connected vertex covers within each of the 10 tests we ran with graphs on a fixed size. We found the largest variations when testing graphs with a small number of edges, i.e., between 60 and 80 edges. However, it is clear that the number of minimal connected vertex covers are increasing from graphs with 50 edges to graphs with 75 edges, and that it is decreasing if the number of edges grows larger than 75.

We can see the difference between the smallest and largest number of minimal connected vertex covers within each of the 10 test runs for each graph of a fixed size in the plot in Figure 4.3. The brown curve in the plot is the standard deviation of the number of minimal connected vertex covers in the 10 graphs of a fixed size. The standard deviation measures the amount of variation of the number of minimal connected vertex covers. Where there is a large gap between the smallest and the largest number of minimal connected vertex covers in graphs, we see that the standard deviation is high, and that it decreases as we increase the number of edges in the graphs.

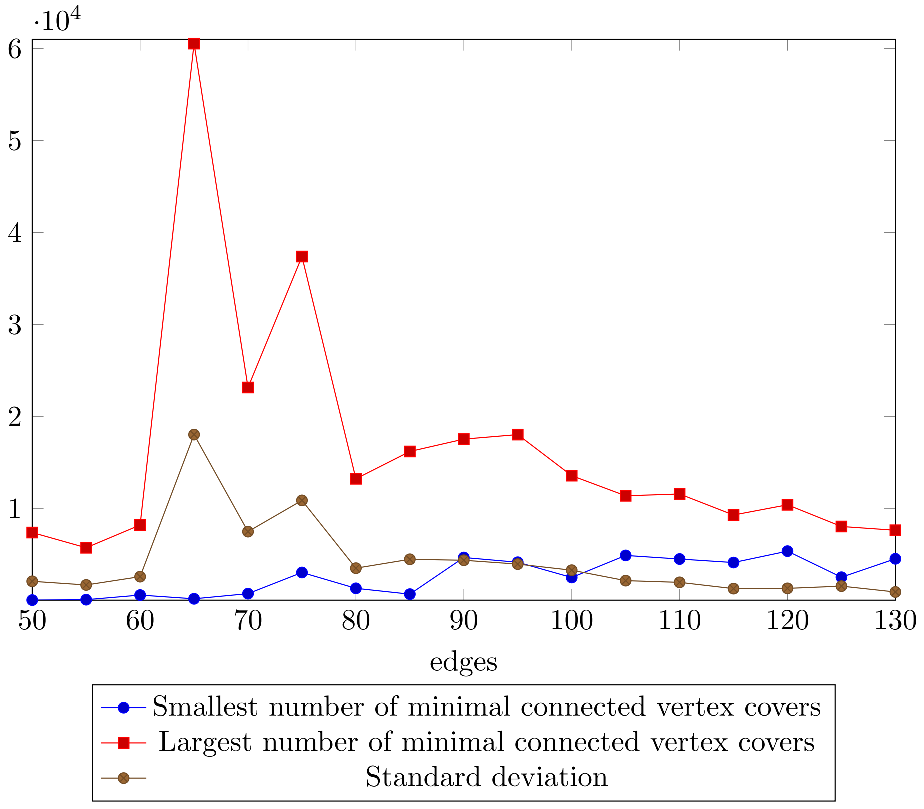


Figure 4.3: Highest and smallest number of minimal connected vertex covers in sparse graphs of 40 vertices and standard deviation of the number of minimal connected vertex covers in the 10 graphs of fixed size.

In the plot in Figure 4.4 we see the sum of the average number of minimal connected vertex covers and discarded sets, and the average running time of our implementation of Algorithm 2.1 after running it 10 times on graphs generated by the random algorithm for generating graphs given in Algorithm 2.3. Similar to the plot in Figure 4.1, we scaled the running time up with a factor of 1000. For pedagogical reasons, we would like the running time curve to be always above the number of sets in our plots, as all generation is done within the running time. This is why we try to multiply it with an appropriate number. However, this does not always result in a clear plot, so the running time curve should be considered in a sense separately from the other curves.

The tuples on the x-axis of the plot represent the number of vertices and edges in the graphs we tested. We see that the number of minimal connected vertex covers plus the number of discarded sets and the running time when testing sparse graphs is remarkable higher than for dense graphs. Both graph 9, 14 and 19 have a smaller number of edges compared to the number of vertices than the other graphs, and from the plot we see that these graphs have a large number of minimal connected vertex covers and discarded sets.

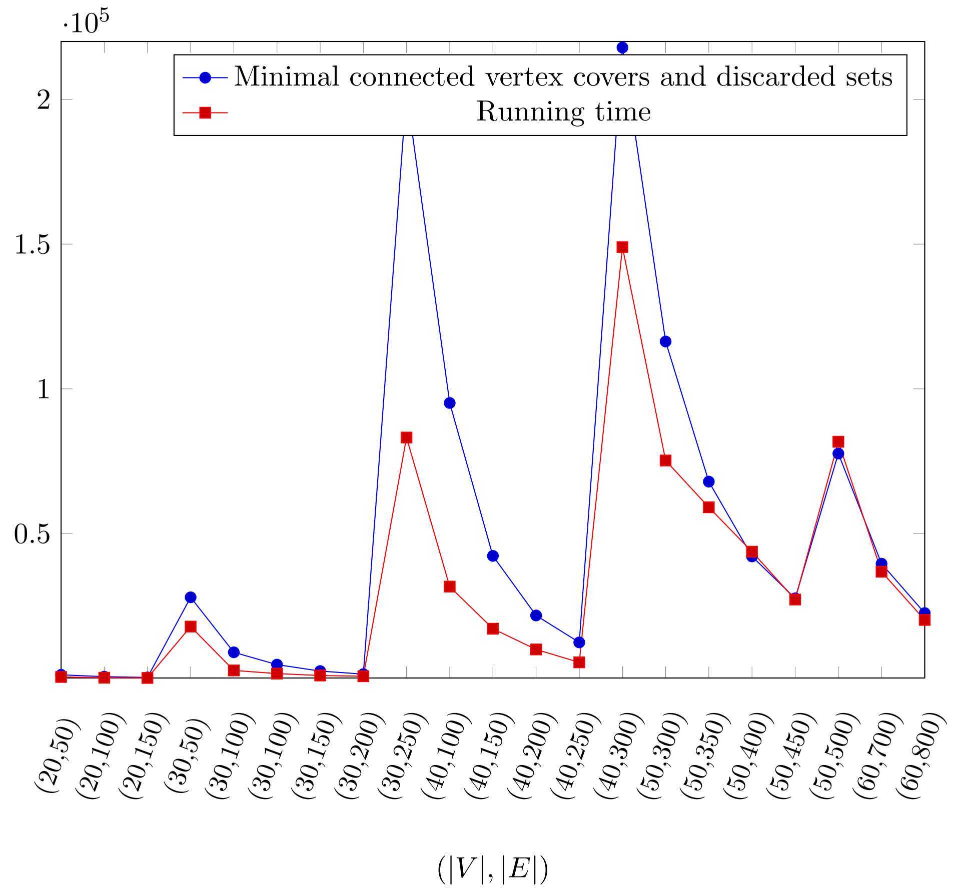


Figure 4.4: Average number of minimal connected vertex covers plus discarded sets and running time on graphs.

The curve that represents the running time of our implementation of Algorithm 2.1 follows the curve of the number of minimal connected vertex covers and discarded sets almost perfectly. Still, as we can see from the plot in Figure 4.1, the number of discarded sets is high. Based on this observation it is natural to think that the algorithm is doing some unnecessary work when evaluating sets that will never be minimal connected vertex covers, and that the upper bound on the number of minimal connected vertex covers is too high. But on the contrary, it might be that there exist a graph with more than 11 vertices and a number of minimal connected vertex covers equal to the existing upper bound of 1.8668^n. If there were to exist such a graph, the algorithm could not discard any sets and construct 1.8668^n minimal connected vertex covers.

To check this more carefully, we ran similar tests with our new lower bound graph using our implementation of Algorithm 2.1. Because of the large number of vertices, we were only able to test the graph $H(5)$, and not the graph $H'(5)$ with several copies of $H(5)$ glued together in a vertex. Following from Lemma 3.4, the number of minimal connected vertex covers in $H(5)$ is $2^{4x-1}x^3 + 2^{2x}x^2 = 65,561,600$ when $x = 5$ and we got the same result from our test. Surprisingly, the algorithm is discarding a large amount of sets when enumerating minimal connected vertex covers in this graph. In fact, a total of $46,479,146$ sets are discarded. A reason for the algorithm to discard this many sets might be that it does some unnecessary work and that the upper bound is to high. On the contrary, it might be that there exist graphs that prove a better lower bound and that the algorithm does not discard as many, or even no sets with these graphs.

Chapter 5

Conclusion

The goal of this thesis has been to try to narrow the gap between the upper and lower bounds on the maximum number of minimal connected vertex covers in graphs. To try to increase the lower bound, our approach was to run practical tests on graphs in search of a new example graph that proves a higher lower bound. Not only did we find such a graph, but we also theoretically constructed graphs to prove an even better lower bound. We also analyzed the algorithm for enumerating minimal connected vertex covers to try to prove that the algorithm does unnecessary work and that the upper bound is probably too high. In this chapter we will first give a summary of this thesis. Then we will discuss possible open questions.

5.1 Summary

In the first chapters we gave essential information on minimal vertex covers and minimal connected vertex covers. We also gave information about previous work on deciding the maximum number of such sets in general graphs. As we saw in Chapter 1, enumerating and listing all minimal connected vertex covers of a graph have not been given the same attention as minimal vertex covers. The paper "Enumeration and Maximum Number of Minimal Connected Certex Covers in Graphs" by Golovach et al. [7] gives an algorithm for enumerating all minimal connected vertex covers in a graph that runs in time $\mathcal{O}(1.8668^n)$. In Chapter 2 we gave the details of this algorithm and how we implemented it. We also gave the details about how we chose to test its correctness and present some small changes we made to give a better practical running time of the algorithm. The enumeration algorithm also proves an upper bound of 1.8668^n on the maximum number of minimal connected vertex covers that a graph can have. In spite of the results we got from improving the practical running time of the algorithm we were not able to prove any better upper bound than 1.8668^n. The paper also provides a lower bound, which is a graph that has $3^{(n-1)/3} \sim 1.4422^n$ minimal connected vertex covers. This leaves a gap between the previously best known upper and lower bounds.

In Chapter 3 we presented some new discoveries regarding the lower bound on the maximum number of minimal connected vertex covers in a graph. First we gave two new example graphs of the existing lower bound of 1.4422^n. These discoveries gave us motivation to try to generate larger graphs as we suspected that the lower bound was too low. Based on practical tests we discovered a graph $G(x)$ with a large amount of minimal connected vertex covers and from that graph we managed to construct the graph $G'(x)$ and theoretically prove that this graph gives a new and better lower bound of 1.5034^n when $x = 8$. With this graph as a basis we constructed the graph $H(x)$ which has an even higher number of minimal connected vertex covers and and which we used to construct the graph $H'(x)$ which proves an even higher and better lower bound of 1.51978^n minimal connected vertex covers. With this new example graph we have been able to increase the lower bound on the maximum number of minimal connected vertex covers from 1.4422^n to 1.51978^n. Still, it leaves a gap between the lower bound and the upper bound of 1.8668^n.

In Chapter 4 we discussed the test results from running our implementation of Algorithm 2.1 for enumerating minimal connected vertex covers in graphs. Based on these results we saw that there is a large number of minimal connected vertex covers in sparse graphs. This can imply that if there were to exist graphs that proves a better lower bound than 1.51978^n, then these graphs will most likely be sparse graphs as well. Interestingly, when sparse graphs have a large amount of minimal connected vertex covers the algorithm also discards a large amount of sets. This can be an indication that there might exist graphs that prove a better lower bound and on these graphs the algorithm will not discard as many sets. It might also be that the algorithm does unnecessary work, and that it is possible to enumerate minimal connected vertex covers with a faster algorithm and obtain a better upper bound. For this theory we also take into consideration that in our opinion the algorithm itself is quite trivial. It branches on vertices outside the minimal connected vertex cover S that is under construction, and include either the vertex itself, or its neighborhood. This procedure continues until the set is in fact a minimal connected vertex cover, or until the vertices outside form an independent set. In the latter case, the algorithm tries to construct a minimal connected vertex cover by adding a set of vertices to S, where the set is of size at most the number of components in S. To lower the upper bound, it might be that the algorithm for enumerating minimal connected vertex covers has to have more sophisticated steps.

5.2 Further work

In this thesis we have been able to raise the lower bound from 1.4422^n to 1.51978^n. Still, it leaves a gap to the upper bound of 1.8668^n, and we think there might exist even better lower bound examples. Thus two obvious paths to pursue as further work are finding better lower bound examples and improving the upper bound.

For the lower bound, as we have seen both in Chapter 3 and in Chapter 4, sparse graphs have a large amount of minimal connected vertex covers. If there were to exist graphs that prove an even higher lower bound, it might be wise to search for such graphs among sparse graphs. We tested all graphs up to 11 vertices, and even though the graph that we use as a basis to construct a new lower bound example graph has 44 vertices, it might be that there exist graphs with a number of vertices between 11 and 44 that can be used to create new example graphs that proves an even higher lower bound. It might also be that it is possible to expand the graph $G(x)$ in other ways than we have done in this thesis to find a graph with a larger number of minimal connected vertex covers.

Very recently, we learned that the graph $H(x)$ that we used as a basis to construct a graph that proves a new and better lower bound on the number of minimal connected vertex covers, can also be used to prove a better lower bound on the number of minimal independent feedback vertex sets of a graph [14]. A minimal independent feedback vertex set is a minimal independent set of vertices whose removal results in an acyclic graph. In particular, $H(6)$ has 1812087554 minimal independent feedback vertex sets, and by taking infinitely many disjoint copies of this graph we get a graph that has 1.5067^n minimal independent feedback vertex sets. These discoveries will be published in a master thesis in 2018. We find it very interesting that the graph $H(x)$ can been used as a basis to prove better lower bounds for two different problems. Could it be that this graph can be used to prove better lower bounds for other types of sets in graphs?

Although our opinion is that the lower bound on the number of minimal connected vertex covers in graphs is too low, we also think that the upper bound is too high, though we have not been able to prove this. As we saw in Chapter 4, the algorithm discards many sets when enumerating all minimal connected vertex covers in graphs. This might be an indication that the algorithm does unnecessary work, and that it is possible to construct an algorithm with more sophisticated steps that proves a better upper bound. For improving the upper bound, we think that an algorithm that goes into a detailed analysis of the vertices depending on their degree is necessary.

Although it has not been the scope of this thesis, another path to follow for further work is to study the maximum number of minimal connected vertex covers in graphs that have a special structure. In such graphs much lower bounds are possible to prove, in addition to tight bounds. For example, for chordal graphs, which are graphs that do not have induced cycles longer than 3, the upper and lower bounds match at $3^{n/3}$ minimal connected vertex covers [7]. For graphs that do not have induced cycles longer than 5, the upper bound is 1.6181^n whereas the same lower bound stands. So in this graph class, there is room for improving the upper or the lower bounds.

Bibliography

[1] B. Bollobás. *Extremal Graph Theory.* Dover Books on Mathematics. Dover Publications, 2004.

[2] B. Bollobas. *Modern Graph Theory.* Graduate Texts in Mathematics. Springer New York, 2013.

[3] J.M Byskov. Enumerating maximal independent sets with applications to graph colouring. *Operations Research Letters*, 32(6):547–556, 2004.

[4] J. Chen, I.A. Kanj, and G. Xia. *Improved Parameterized Upper Bounds for Vertex Cover*, pages 238–249. Springer Berlin Heidelberg, Berlin, Heidelberg, 2006.

[5] M. Cygan. *Deterministic Parameterized Connected Vertex Cover*, pages 95–106. Springer Berlin Heidelberg, Berlin, Heidelberg, 2012.

[6] F.V. Fomin and D. Kratsch. *Exact Exponential Algorithms.* Texts in Theoretical Computer Science. An EATCS Series. Springer Berlin Heidelberg, 2010.

[7] P.A. Golovach, P. Heggernes, and D. Kratsch. *Enumeration and Maximum Number of Minimal Connected Vertex Covers in Graphs*, pages 235–247. Springer International Publishing, Cham, 2016.

[8] M. Hujtera and Z. Tuza. The number of maximal independent sets in triangle-free graphs. *SIAM Journal on Discrete Mathematics*, 6(2):284–288, 1993.

[9] B.D. McKay and A. Piperno. Practical graph isomorphism, {II}. *Journal of Symbolic Computation*, 60(0):94 – 112, 2014.

[10] R.E. Miller and D.E. Muller. A problem of maximum consistent subsets. *IBM Research Report RC-240, J. T. Watson Research Center,New York, USA*, 1960.

[11] J.W. Moon and L. Moser. On cliques in graphs. *Israel Journal of Mathematics*, 3:23–28, 1965.

[12] J.M. Robson. Finding a maximum independent set in time o(2n/4). *Technical Report 1251-01, LaBRI, Université Bordeaux*, 2001.

[13] S. Tsukiyama, M. Ide, H. Ariyoshi, and I. Shirakawa. A new algorithm for generating all the maximal independent sets. *SIAM Journal on Computing*, 6(3):505–517, 1977.

[14] T. Wingsternes. Personal communication, 2017.