Jagiellonian University
Department of Theoretical Computer Science

Piotr Kubaty

Implementation of exact algorithms for minimum dominating set

Bachelor Thesis

Supervisor: dr hab. Krzysztof Turowski

July 2023

Contents

Chapter 1

Introduction

1.1 Introductory definitions

In the following, a graph means an undirected, simple (i.e. there can be at most one edge between two vertices) graph. In the context of graph G, the set of its vertices is denoted by $V(G)$ and the set of its edges by $E(G)$. The definitions of this section or equivalent ones can be found in [2] or in [9]

Definition 1 (cardinality)**.** *The cardinality of a set is the number of its elements.*

For a set X, one denotes its cardinality with $|X|$.

Definition 2 (frequency)**.** *Given a family $\mathcal{S}$ of sets, the frequency of an element $u \in \bigcup \mathcal{S}$ is the number of sets $S \in \mathcal{S}$ satisfying $u \in S$.*

Definition 3 (closed neighborhood)**.** *In a graph G, a closed neighborhood of $v \in V(G)$ (denoted by $N_G[v]$ or simply $N[v]$ in case G is evident) is the union of the set of vertices adjacent to a vertex v and $\{v\}$.*

1.2 Problem statement

Definition 4 (dominating set)**.** *Given a graph G, a dominating set D in G is a subset of $V(G)$, which satisfies the property that every vertex $v \in V(G) \setminus D$ is adjacent to at least one $u \in D$.*

The property of being dominating set may be beneficial. For instance, one can consider a network of devices. Then the dominating set could be the set of transmitters, and each device would be a transmitter or have a direct connection with a transmitter [4]. It would be convenient if the number of transmitters was as low as possible. In the scenario of an ad hoc computer network, connected dominating sets can be utilized as virtual backbones [1]. Some other applications of dominating set or its variants are summarized in [19]

Definition 5 (Dominating Set Problem)**.** *When given an instance consisting of a graph G and a natural number k, the problem of deciding whether there exists a dominating set of cardinality at most k in G is called the Dominating Set Problem [12]*

As $V(G)$ is clearly a dominating set in G, one may consider the following optimization problem instead.

Definition 6 (Minimum Dominating Set Problem)**.** *Given a graph G, the Minimum Dominating Set Problem asks for a dominating set of minimum cardinality.*

The above problem will be abbreviated to MDS in the following.

Definition 7 (Vertex Cover Problem)**.** *When given an instance consisting of a graph G and a natural number k, the problem of deciding whether there exists a subset C of $V(G)$, such that every edge is adjacent to at least one $u \in C$ is called the Vertex Cover Problem [12]*

As mentioned in Garey and Johnson [12], the Dominating Set Problem is NP-complete due to the reduction from the Vertex Cover Problem. The latter was proven to be NP-complete in [17]. Having a polynomial-time algorithm for MDS would give a solution to the Dominating Set Problem in polynomial time, which would imply P = NP. Nevertheless, there are other techniques to approach this problem.

1. **Parameterized Algorithms.** For instance, when in graph G the shortest cycle is of length at least 5, there is the FPT (fixed parameter tractable) algorithm, parameterized by the cardinality parameter k of an instance (G, k) of the Dominating Set Problem [3]. Authors consider polynomial-time kernelization into an equivalent instance (G', k) with $|V(G')| = O(k^3)$. Afterwards, one can proceed with any exact algorithm described in the following sections. This means that for small k and large n, such instances can be solved in $2^{O(k^3)}$. Despite that, in general, the Dominating Set Problem is a W[2]-complete problem [5].

2. **Approximation Algorithms.** One may want to obtain an approximation for MDS. There is a $\big(\ln(\Delta(G) + 1) + 1\big)$-approximation algorithm for MDS [18], where $\Delta(G)$ is the maximum degree of a vertex in G. In worst-case $\Delta(G) = |V(G)| - 1$. The idea of the algorithm is that in each iteration, choose a vertex with the most adjacent vertices that are not dominated. However, there is no ϵ for which there would exist an $(1 - \epsilon)\ln(|V(G)|)$-approximation algorithm unless NP $\subseteq$ DTIME$(|V(G)|^{O(\log\log(|V(G)|))})$ [18].

3. One may be unsatisfied with $\big(\ln(\Delta(G) + 1) + 1\big)$-approximation or just need an exact solution. One may seek algorithms that are faster than exhaustive (brute-force) algorithms, as well as consider worst-case complexity bounds for each such algorithm. Having an $2^{o(V(G))}$ algorithm for

MDS would imply that SNP $\subseteq$ SUBEXP [11] or, equivalently, *Exponential Time Hypothesis* fails [15]. Due to the latter, some researchers are focused on providing exact algorithms for MDS running in time $O^*(c^{|V(G)|})$ with the smallest possible $c > 1$.

Now we proceed to a related covering problem:

Definition 8 (set cover)**.** *Set cover $\mathcal{C}$ of a set $\mathcal{U}$ is any family of sets satisfying $\bigcup_{S\in\mathcal{C}} S = \mathcal{U}$*

Definition 9 (Set Cover Problem)**.** *When given an instance consisting of $\mathcal{U}$, $\mathcal{S}$, such that $\mathcal{U} = \bigcup_{S\in\mathcal{S}} S$ and an integer k, the problem of deciding whether there exists a set cover $\mathcal{C} \subseteq \mathcal{S}$ with $|\mathcal{C}| \leq k$ (minimum cover in [12]).*

Definition 10 (Minimum Set Cover Problem)**.** *Given an instance consisting of $\mathcal{U}$, $\mathcal{S}$, such that $\mathcal{U} = \bigcup_{S\in\mathcal{S}} S$, Minimum Set Cover Problem asks to find a set cover of minimum cardinality.*

The above problem will be abbreviated to MSC in the following. MSC is well defined because $\mathcal{S}$ is a valid set cover. In the process of refining their MDS algorithms one may decide to generalize the original problem or transform it into another problem in order to achieve a better complexity bound for the algorithm. In the case of MDS, one of such ways to do so is by considering equivalent MSC instances. One may observe that an instance of MDS can be transformed to an instance of MSC by substituting $\mathcal{U} = V(G)$, $\mathcal{S} = \{N[v] : v \in V(G)\}$ [24]. Observe that a subset D of $V(G)$ is a dominating set of G if and only if $\bigcup_{v\in D} N[v] = V(G)$. Moreover, if D is the minimum dominating set then D and $\{N[v] : v \in D\}$ have the same cardinalities. The other way a solution to MSC in which each $S \in \mathcal{S}$ is of the form $\{N[v] : v \in V(G)\}$ can be translated into a corresponding MDS solution of equal cardinality. That means one may transform any instance of MDS into an instance of MSC, solve this broader problem, and given a minimum set cover, use association between sets of MSC with corresponding vertices of MDS to obtain the construction of a dominating set of minimum cardinality.

Algorithm 1 Minimum dominating set algorithm

```
1: procedure MDS(G)
2:     MSC(V(G), {N[v] : v ∈ V(G)})
3: end procedure
```

Although MSC is NP-complete in general [17], it can be solved in polynomial time in the case when each $S \in \mathcal{S}$ is of cardinality 2. Then one can associate each set in $\mathcal{S}$ with an edge in the graph.

Definition 11 (edge cover)**.** *Given a graph G, an edge cover E' is a subset of $E(G)$ such that for any vertex $u \in V(G)$ there exists an edge $e \in E'$ containing u [12].*

Definition 12 (Minimum Edge Cover Problem)**.** *Given a graph G containing no isolated vertices, the Minimum Edge Cover Problem asks for an edge cover of minimum cardinality.*

The Minimum Edge Cover Problem can be solved by computing a maximum matching $\mathcal{M}$ of cardinality m. The latter has a polynomial algorithm for an arbitrary graph G [6]. Then, for each yet-uncovered vertex, choose an arbitrary edge adjacent to it [24]. Observe that the cardinality of this edge cover equals $m+(|V(G)|-2m) = |V(G)|-m$ and is the minimum possible. Indeed, suppose conversely that there exists an edge cover $\mathcal{R}$ of cardinality $r < |V(G)| - m$. Assign to each $v \in V(G)$ an edge e_v that covers v. Since each edge was assigned to at most 2 vertices, there are at least $|V(G)| - r$ edges that were assigned to exactly 2 vertices. These form a matching of cardinality greater than m, a contradiction.

In order to provide faster algorithms for computationally hard (exponential) problems like MDS, researchers started to seek patterns in a given instance to avoid checking all possible candidates for a solution. In the following part, exponential time, polynomial space algorithms are presented, together with their analysis. In the next section, the Measure and Conquer technique is introduced, along with Branch and Reduce algorithms, which can be analyzed with it. In the succeeding section, other techniques are used: one algorithm of Schiermeyer and one of Fomin, Kratsch, Woeginger is described. In the last section, some results of the implementations' executions are illustrated. The following table presents an overview of known exact algorithms. Algorithms that have been implemented are in bold. In addition, an exhaustive algorithm (running in $O^*(2^n)$) was implemented. Worth noting is that the best bound was obtained by Iwata, who used the Potential Method in the analysis of his algorithm.

Refs	**Authors**	**Year**	**Time**
[11]	**Fomin, Kratsch, Woeginger**	**2004**	$O(1.9379^n)$
[14, 13]	**Grandoni**	**2004/2006**	$O(1.9053^n)$
[20, 23]	**Schiermeyer**	**2004/2008**	$O(1.8899^n)$
[10]	Fomin, Grandoni, Pyatkin, Stepanov	2005	$O(1.7697^n)$
[9, 8, 7]	**Fomin, Kratsch, Grandoni**	**2005/2009**	$O(1.5263^n)$
[22]	van Rooij, Bodlander	2008	$O(1.5134^n)$
[24]	**van Rooij, Bodlander**	**2011**	$O(1.4969^n)$
[16]	Iwata	2011	$O(1.4864^n)$

Table 1.1: Known Minimum Dominating Set algorithms, which use polynomial space. The implemented algorithms are presented in bold.

Chapter 2

Algorithms

2.1 Branch and reduce algorithms

2.1.1 Branch and reduce

One approach of solving a problem in exponential time is to use a recursive algorithm. The Branch and Reduce algorithm takes advantage of recursion and consists of two kinds of rules, described in [9]. The idea is as follows:

- A branching rule is a procedure that, for a given instance, generates in polynomial time two or more sub-instances (solved recursively). Having solutions to all of them allows one to retrieve an optimal solution to the instance. Usually, each branch makes some assumption about the solution that is disjoint from the ones in the other branches, and at least one of these assumptions is true.

- A reduction rule is a procedure that looks for a property of an instance. This search consumes polynomial time in terms of the size of the instance. If the property is found, the procedure transforms in polynomial time the instance into an equivalent sub-instance (smaller in the sense of some natural parameters), which means that by having the optimal solution to the smaller instance, one can retrieve the optimal solution for the instance.

The algorithm halts in the base case when it is known that the solution can be computed in polynomial time. Otherwise, the algorithm sequentially tries to apply each of the reduction rules. If one applies, it calls recursively on the instance obtained by the reduction and afterwards retrieves the solution from the result of the call. If none of the rules apply, it chooses from a set of branching rules to apply. It calls on each of the sub-instances recursively and compares the results to choose the one from which the optimal solution is retrieved.

In order to analyze such an algorithm, one can represent its execution with a tree, with the nodes of the tree being sub-calls of the recursive algorithm [9]. In the previous, the root of the tree corresponds to the main instance. Assuming

that the depth of the recurrence is polynomial in terms of the size of the input instance, in order to analyze the worst-case O^* time complexity, one needs a function bounding number of leaves in the corresponding tree [24].

2.1.2 Measure and Conquer

Measure and Conquer is a technique for analyzing the complexity of the Branch and Reduce algorithm by introducing the concept of measure, a function taking an instance and returning a non-negative real number. When measure can be bounded in terms of the instance size, one can evaluate complexity in terms of the instance size by evaluating complexity in terms of the measure. To achieve this, any sub-instance obtained by a reduction or branching rule must have a smaller measure.

In a complexity proof using Measure and Conquer, one introduces a set of recursive inequalities for each branching rule (These are similar to recursive inequalities in an analysis depending on some natural parameter of the instance such as the number of vertices for MDS). For an instance I of measure k for which a branching rule generates sub-instances $I_1, I_2 \ldots, I_s$, the corresponding inequality is in the form

$$N_I(k) \leq N_I(k - \Delta_1) + N_I(k - \Delta_2) + \cdots + N_I(k - \Delta_s) \tag{2.1}$$

where $N_I(m)$ denotes the maximum number of leaves in the tree associated with the execution of the algorithm on any sub-instance of measure not greater than m and for each $j \in \{1, \ldots, s\}$ measure of instance I_j is not greater than $k - \Delta_j$ [9] [24]. It is also possible, in a refined analysis, that a greater Δ'_j instead of Δ_j can be obtained when it can be inferred that a reduction rule can be immediately applied after branching. Hence, one would like to find a positive real root of the equation

$$\alpha^k = \alpha^{k-\Delta_1} + \cdots + \alpha^{k-\Delta_s} \tag{2.2}$$

[9]. Then for $c \geq \alpha$ and $f(x) = N_I(0) \cdot c^x$

$$f(k) \geq f(k - \Delta_1) + f(k - \Delta_2) + \cdots + f(k - \Delta_s).$$

$$\forall_i \quad f(k - \Delta_i) \geq N_I(k - \Delta_i) \implies f(k) \geq N_I(k)$$

.

If there exists some $f(x) = O(c^x)$ such that any instance I of measure k satisfies $N_I(k) \leq f(k)$, then the algorithm finishes in $O^*(c^{m(n)})$ where function $m(n)$ is an upper bound on the measure of the instance of size n. The analysis using the Measure and Conquer technique is more detailed than the one based directly on the instance size. That results from the fact that each branching rule generates more recursive inequalities than there would be in the standard analysis expressing complexity from a single parameter of the size of an instance. With a good choice of measure, the resulting complexity bounds are tighter since the measure is optimized to improve on the most pessimistic recurrences.

2.1.3 Set Cover Algorithm of Grandoni

The algorithm is described in [13]. It applies the Branch and Reduce method. The author uses analysis directly based on the number of vertices in the graph.

Algorithm 2 Minimum Set Cover

1: **procedure** MSC($\mathcal{U}, \mathcal{S}$) ▷ $\bigcup \mathcal{S} = \mathcal{U}$
2: if $\mathcal{S} = \emptyset$ return $\emptyset$ ▷ base case
3: if there exist $Q, R \in \mathcal{S}$ satisfying $Q \subseteq R$, discard Q and call recursively
4: if there exists $u \in \mathcal{U}$ and a unique $R \in \mathcal{S}$, satisfying $u \in R$, take R to the solution, remove all $R's$ elements from $\mathcal{U}$, and call recursively
5: if none of the above applies, let $S \in \mathcal{S}$ be a set of maximum cardinality
6: in one branch discard S and let $C_{discard}$ be the solution of recursive call
7: in the other branch, remove all $S's$ elements from $\mathcal{U}$ and let C_{take} be the solution of recursive call
8: from $C_{discard}$, $(C_{take} \cup \{S\})$ return one of smaller cardinality
9: **end procedure**

Correctness. To prove that Algorithm 2 returns a proper set cover, it is sufficient to note that

- The reduction on line 3 of Algorithm 2 will be called `R-subset` in the following. When $Q \subseteq R$, for any MSC solution containing Q one has an equivalent solution taking R instead.
- The reduction on line 4 will be called `R-unique` in the following. When R is unique such that $u \in R$, then any solution contains R.
- Branching: the optimal solution either contains S or not.

Complexity analysis. Consider instance $I = (\mathcal{U}, \mathcal{S})$ and let $d = |\mathcal{U}| + |\mathcal{S}|$ be its dimension. Let one define $N(k)$ to be number of leaves of the tree associated with the execution of the algorithm, as described in subsections 2.1.1 and 2.1.2. Following holds:

$$N(d) \leq N(d-1) + N(d-4),$$

Indeed, let s be the cardinality of set S, considered in the branching. If $s \geq 3$, then in a branch discarding S, the dimension is decreased by 1. In a branch removing all elements of S, the dimension decreases by at least 1+3, hence the above inequality. Corresponding inequality (2.2) gives some root α satisfying $\alpha < 1.3803$. Therefore given graph G with $n = V(G)$ $d = 2n$ and the composition of Algorithm 1 and Algorithm 2 gives $O^*(1.3803^{2n}) = O^*(1.9053^n)$ time complexity for MDS.

2.1.4 Set Cover Algorithm of Fomin-Grandoni-Kratsch

The algorithm is described in [7]. Algorithm 3 is a minor refinement of Algorithm 2. Standard analysis using the dimension of the set cover instance gives

the same $N(d) \leq N(d-1)+N(d-4)$ recurrence and thus the same complexity. Authors take advantage of the Measure and Conquer technique to improve the complexity bound.

Algorithm 3 Minimum Set Cover

1: **procedure** MSC($\mathcal{U}, \mathcal{S}$) ▷ $\bigcup \mathcal{S} = \mathcal{U}$
2: if $\mathcal{S} = \emptyset$ return $\emptyset$ ▷ base case
3: if `R-subset` discard Q and call recursively
4: if `R-unique` take unique R to the solution and call recursively
5: if all sets have cardinality at most 2 return Edge-Cover($\mathcal{U}, \mathcal{S}$)
6: if none of the above applies, let $S \in \mathcal{S}$ be a set of maximum cardinality
7: in one branch discard S and let $C_{discard}$ be the solution of recursive call
8: in the other branch remove all $S's$ elements from $\mathcal{U}$ and let C_{take} be the solution of recursive call
9: from $C_{discard}$, $(C_{take} \cup \{S\})$ return one of smaller cardinality
10: **end procedure**

As one can see, the only difference is line 5. Here, since all sets have cardinality exactly 2. Sets of cardinality 1 are not considered because of `R-subset` and `R-unique` rules. Hence, these sets are straightforward edges in a graph $G = (\mathcal{U}, \mathcal{S})$. Thus, correctness is shown.

Complexity analysis. The considered measure is defined as

$$k = k(\mathcal{U}, \mathcal{S}) = \sum_{u \in \mathcal{U}} v(f(u)) + \sum_{S \in \mathcal{S}} w(|S|) \tag{2.3}$$

for some weight functions $v, w : \mathbb{N}_+ \to \mathbb{R}_+$ and $f(u)$ denotes the frequency of u in $\mathcal{S}$.

Some intuitions. There are several key observations:

1. If two elements $u_1, u_2 \in \mathcal{U}$ of the same frequency contribute the same amount to the first sum, no matter to which sets they belong. However, these elements may influence the second sum differently, depending on the cardinalities of the sets u_1, u_2 belong to.

2. When any rule is applied, the measure decreases by the amount dependent on the pattern it is focused on, not the whole instance. In addition, the weight functions are bounded, and their images are finite sets. This way, one can obtain a finite set of recursive inequalities to satisfy.

3. The weight functions are non-decreasing functions. This is important whenever a reduction rule removes a set or element from the instance. In such situation, the measure always decreases. The dual argument applies when an element of $\mathcal{U}$ is removed from instance: the cardinalities of some sets decline by 1, and the measure decreases as well.

4. $v(1) = w(1) = 0$. This comes from the fact that when there is an element of frequency 1, the `R-unique` applies, and if the latter does not hold but there is a set of cardinality 1, the `R-subset` applies.

For an instance of measure equal to k, one would like to bound the number of base instances considered during the execution (number of leaves in the tree2.1.1). For this purpose, define function N as in section 2.1.2. Let S be the set (of maximum cardinality) on which branching decides. Each recursive inequality is of the form

$$N(k) \leq N(k - \Delta k_{take}) + N(k - \Delta k_{discard})$$

Δk_{take} is a lower bound for the decrease of the measure in the branch for which the algorithm takes S into the solution. That consists of a direct decrease from taking S, as well as an additional decrease in a situation when some reduction rules can be inferred to happen after branching. Analogously, $\Delta k_{discard}$ for the other branch. To obtain good bounds, one needs to consider all possible cases of how taking and discarding S influences the evaluation of weight functions v, w (defined below) on each element of $\mathcal{U}, \mathcal{S}$, respectively. Introduce functions $\Delta v, \Delta w$ such that they satisfy

$$\Delta v(i) = v(i) - v(i-1), \quad \Delta w(i) = w(i) - w(i-1) \quad \text{for } i \geq 2.$$

and w is concave

$$\Delta w(i-1) \geq \Delta w(i) \quad \text{for } i \geq 3 \tag{2.4}$$

Exact values of v, w remain to be optimized in order to achieve the best complexity bound.

The above concepts are extremely useful in the simplification of the following analysis. For instance, discarding S decreases the frequency of any of its elements by 1. For this simple fact, one adds for each $u \in S$ $\Delta v(f(u))$ to $\Delta_{discard}$. Let r_i denote the number of elements of frequency i in S. Then $\sum_{u \in S} \Delta v(f(u))$ is equivalent to $\sum_{i=2}^{\infty} r_i \Delta v(i)$. Inequality 2.4 is used when taking S. Then each $u \in S$ is removed from the instance, thus $f(u)$ times decreasing cardinalities of sets in which it occurred. This can be bound by $\Delta w(|S|) \sum_{i=2}^{\infty} r_i(i-1)$.

$$\Delta k_{take} = w(|S|) + \sum_{i=2}^{\infty} r_i v(i) + \Delta w(|S|) \sum_{i=2}^{\infty} r_i(i-1)$$

$$\Delta k_{discard} = w(|S|) + \sum_{i=2}^{\infty} r_i \Delta v(i) + r_2 \cdot v(2) + [r_2 > 0] v(2)$$

i	1	2	3	4	5	>5
$v(i)$	0.0000	0.3996	0.7677	0.9300	0.9856	1.0000
$w(i)$	0.0000	0.3774	0.7548	0.9095	0.9764	1.0000

Table 2.1: optimized choice of weight functions, rounded to 4 digits

Given graph G with $n = V(G)$, $k \leq 2n$ and composition of Algorithm 1 and Algorithm 2 gives $O^*(1.2353^{2n}) = O^*(1.5263^n)$ time complexity for MDS. van

Rooij and Bodlaender [24] present analysis for this algorithm with almost the same $k_{discard}$ inequality but with different weights. In particular, for $i \geq 7$ $v(i)$ is approximately 0.6612 instead of 1 and for $i \geq 7$ $w(i) = 1$. With this advantage, van Rooij and Bodlaender obtain $O(1.28505^{1.661244n}) = O(1.51685^n)$ complexity for this algorithm.

2.1.5 Algorithm of van Rooij and Bodlaender

The algorithm is described in [24]. It is an extension of the previous algorithm, adding three additional rules.

Algorithm 4 Minimum Set Cover

1: **procedure** MSC($\mathcal{U}, \mathcal{S}$) ▷ $\bigcup \mathcal{S} = \mathcal{U}$
2: if $\mathcal{S} = \emptyset$ return $\emptyset$ ▷ base case
3: if `R-unique` take unique R to the solution and call recursively
4: if `R-subset` discard Q and call recursively
5: if there exist $q, r \in \mathcal{U}$ such that any set containing r contains q as well, remove q from $\mathcal{U}$ and call recursively
6: if there exists set R with $|\bigcup_{u \in R, f(u)=2} R_u \backslash R| < |e \in R : f(e) = 2|$, where for each R_u is the second set containing u, then take R to the solution and call recursively
7: if there exists $R = \{e_1, e_2\}$, $f(e_1) = f(e_2) = 2$, $e_1 \in R_1 \neq R$, $e_2 in R_2 \neq R$, then replace sets R, R_1, R_1 with $Q := (R_1 \cup R_2) \setminus R$. In a recursive call, if Q belongs to the sub-solution, return the sub-solution augmented with R_1, R_2 instead of Q. If Q is not in the sub-solution, return sub-solution augmented with R
8: if all sets have cardinality at most 2, return Edge-Cover($\mathcal{U}, \mathcal{S}$)
9: if none of the above applies, let $S \in \mathcal{S}$ be a set of maximum cardinality
10: in one branch discard S and let $C_{discard}$ be the solution of recursive call
11: in the other branch remove all $S's$ elements from $\mathcal{U}$ and let C_{take} be the solution of recursive call
12: from $C_{discard}$, $(C_{take} \cup \{S\})$ return one of smaller cardinality
13: **end procedure**

Correctness One can see that each rule transforms set cover instance into a smaller one. This holds for *size two set with only frequency two elements rule* (introduced below) as well, since each reduction rule decreases $|\mathcal{U}| + |\mathcal{S}|$ by at least 1. Moreover, one needs to check if the newly added reduction rules provide minimal set cover solutions.

1. The reduction on line 5 of Algorithm 4 is dual to the `R-subset`. Suppose that there exist elements q, r such that $\{S : q \in S\} \supseteq \{S : r \in S\}$. In other words, any set containing r contains q as well. In such a situation, one does not need to check whether a candidate to set cover solution covers q as long as it is checked that r is covered.

2. The reduction on line 6 of Algorithm 4. Given a set $S \in \mathcal{S}$, define the function $f_2(S) = \{u \in S : \text{ frequency of } u \text{ is } 2\}$. For a given R each $u \in f_2(R)$ occurs in R and just one other set, name it R_u. A crucial observation in the following is that, due to the subsumption rule applied before, all sets $R_u s$ are pairwise distinct. (Parenthetically, if frequency of u were >2 this trick would not work.) Consider any hypothetical set cover solution $\mathcal{C} \subseteq \mathcal{S}$ that does not contain R. $\mathcal{C}$ must cover each element $u \in f_2(R)$, meaning $R_u \in \mathcal{C}$. Therefore to cover all elements of $f_2(R)$, one needs $|f_2(R)|$ sets. Consider set $T := \bigcup_{u \in f_2(R)} R_u \setminus R$. If $|T| < |f_2(R)|$, then for each $u \in T$ fix T_u to be any set containing u. Observe that one can consider alternative solution $\mathcal{C}'$ defined by removing from $\mathcal{C}$ all R_us for $u \in f_2(R)$, adding set R and all elements of $\mathcal{T} := \{T_u : u \in T\}$. Since the number of added sets is $1+|\mathcal{T}| \leq 1+|T| \leq |f_2(R)|$, there exists a set cover solution containing R with cardinality not greater than any solution that does not contain R. In conclusion, if there exists R such that $|T| < |f_2(R)|$ one can reduce instance $(\mathcal{U}, \mathcal{S})$ by taking R to the solution.

3. Reduction on line 7 is *size two set with only frequency two elements rule*. It applies if there exists $R = \{e_1, e_2\}$ such that the frequencies of e_1, e_2 are both 2. It means that e_1 occurs in some set, say $R_1 \neq R$ and analogously R_2 for e_2. Define $Q := R_1 \cup R_2 \setminus R$. Then one can reduce an instance $(\mathcal{U}, \mathcal{S})$ to an instance $(\mathcal{U}', \mathcal{S}')$, such that $\mathcal{U}' := \mathcal{U} \setminus R$, $\mathcal{S}' := \mathcal{S} \setminus \{R, R_1, R_2\} \cup \{Q\}$. In other words, one obtains a reduced instance by removing elements e_1, e_2 and then merging R_1, R_2 into one set. Let $\mathcal{R} \subseteq \mathcal{S}'$ be a minimum cardinality solution in a reduced instance. Now suppose one has solution $\mathcal{C}$ to the original problem. If $R \in \mathcal{C}$ then $\mathcal{C} \setminus \{R\}$ is a solution to the reduced problem. In that case, $|\mathcal{C}| - 1 \geq |\mathcal{R}|$. If $R \notin \mathcal{C}$, then $R_1, R_2 \in \mathcal{C}$, as e_1, e_2 must be both covered. $\mathcal{C} \setminus \{R_1, R_2\} \cup \{R\}$ is then solution to reduced problem and $|\mathcal{C}| - 2 + 1 \geq |\mathcal{R}|$. That proves the minimum cardinality solution to the original problem is at least one greater than the one in the reduced problem. The following is a construction of solution one bigger than $\mathcal{R}$. In case $Q \in \mathcal{R}$, one retrieves a solution to the original instance of cardinality $|\mathcal{R}| + 1$ by taking R_1, R_2 instead of Q into solution. It is so because R_1, R_2 cover elements covered by Q and additionally e_1, e_2. In case $R \notin \mathcal{R}$, one retrieves the solution to the original instance of cardinality $|\mathcal{R}| + 1$ by adding R.

Complexity analysis. Given a graph G, let $n = |V(G)|$. Algorithm 1 composed with Algorithm 4 achieves $O(1.4969^n)$ complexity. For this purpose, measure is defined as in (2.3), but different functions v, w are chosen:

i	1	2	3	4	5	6	7	>7
$v(i)$	0.00000	0.01118	0.37948	0.52608	0.57380	0.59111	0.59572	0.59572
$w(i)$	0.00000	0.35301	0.70602	0.86689	0.94395	0.98128	0.99706	1.00000

Table 2.2: Optimized choice of weight functions, rounded to 5 digits

As mentioned in the previous subsection, reduction rules cannot increase the measure. As *the size two set with only frequency two elements rule* adds a new set to the instance, one needs to verify it. Note that when applied, some sets R, R_1, R_2 with $|R| = 2$ are removed and a set $Q = R_1 \cup R_2 \setminus R$ is introduced. By this, no $u \in \mathcal{U}$ increases its frequency, so it suffices that $w(i + j - 2) \leq w(2) + w(i) + w(j)$ holds for any $i, j \geq 2$, what can be easily checked.

It remains to consider the behaviour of the measure when the branching rule is applied. With extensive sub-casing analysis van Rooij and Bodlaender achieve following bounds:

$$\begin{aligned}\Delta k_{take} \geq\ & w(|S|) + \sum_{i=2}^{\infty} r_i v(i) + \Delta w(|S|) \sum_{i=2}^{\infty} r_i (i-1) \\ & + (r_{f3} + r_{f\geq 4})(w(2) - \Delta w(|S|)) + r_{s3}(\Delta w(3) - \Delta w(|S|)) \\ & + [r_{f3} > 0]\min(v(3), r_{f3}\Delta v(3)) + [r_{\text{rule 7}}](2v(2) + w(2))\end{aligned}$$

$$\begin{aligned}\Delta k_{discard} \geq\ & w(|S|) + \sum_{i=2}^{\infty} r_i \Delta v(i) \\ & + r_2(v(2) + w(2)) + r_{s3}\Delta w(3) + r_{s\geq 4}(w(4) - w(2)) \\ & + [r_{f3} > 0]\Delta v(3) + [r_{f\geq 4} > 0](v(4) - v(2)) \\ & + \min\Big(q_{\vec{r}}\Delta w(|S|), \lfloor \frac{q_{\vec{r}}}{2} \rfloor w(2) + (q_{\vec{r}} \bmod 2)\Delta w(|S|)\Big)\end{aligned}$$

As one can see, right hand sides of these inequalities are extensions of corresponding expressions in the previous algorithm's analysis. Additional summands are obtained for each of the added rules. This complex inequalities evaluate inequalities for each possible structure of S. When $S = \{e_1, \ldots, e_{|S|}\}$ of frequencies $f(e_1), \ldots, f(e_{|S|})$, one can substitute suitable values for each term in the above inequalities, obtaining associated recurrence. The bounding recurrence result from sets S is described below. For each of them, all elements of S have the same frequency: One may notice that for each $i = 3, \ldots 8$ there is a case

cardinality of S	3	3	4	4	5	5	6	6	7	7	8
frequency of any element of S	3	4	2	5	5	6	6	7	7	8	8

Table 2.3: Bounding cases

in which S has cardinality i. Intuitively, if it was not the case for some i, then since $\Delta w(i+1) = w(i+1) - w(i)$, one would decrease $w(i)$ thus improving some bounding recurrences and obtaining better complexity.

With the above bounding cases van Rooij and Bodlaender obtain $N(k) \leq 1.28759^k$ and since $k(\mathcal{U}, \mathcal{S}) = \sum_{u \in \mathcal{U}} v(f(u)) + \sum_{S \in \mathcal{S}} w(|S|) \leq (0.59572 + 1)n$, this yields $O(1.28759^{1.59572n}) = O(1.4969^n)$

2.2 Algorithms for the minimum optional dominating set

Domination in the graph can be generalised with the concept of optional domination [23].

Definition 13 (optional dominating set)**.** *Given graph G and a partition of $V(G)$ into disjoint sets V_F, V_B, V_R a set $D \subseteq V(G)$ is called an optional dominating set if $V_R \subseteq D$ and $V_B \subseteq N[D]$.*

Definition 14 (Minimum Optional Dominating Set Problem)**.** *Given a graph G the Minimum Optional Dominating Set Problem asks for an optional dominating set of minimum cardinality. (abbreviated to MODS)*

V_F, V_B, V_R stand for free vertices, bound vertices, and required vertices, respectively. When $V_F, V_R = \emptyset$, the above is equivalent to dominating set.

Definition 15 (induced subgraph)**.** *Given a graph G and $V' \subseteq V(G)$, (V', E') is a subgraph induced by V' if $E' \subseteq E(G)$ and E' contains all edges of G that join two vertices in V'. It is denoted by $G[V']$ [2].*

2.2.1 Algorithm of Fomin-Kratsch-Woeginger

The algorithm is described in [11].

In 1996, Reed proved the following result in combinatorics: *Graph G of minimum degree at least three contains a dominating set D of size at most $\frac{3}{8}|V(G)|$* [21].

The following algorithm behaves like Branch and Reduce on vertices of degree 1 or 2. The difference is when eventually the minimum degree in the graph is ≥ 3 (or $V = \emptyset$). Then it exhaustively checks all sets of cardinality $\leq \frac{3}{8}|V(G)|$ (what is done in exponential time, unlike for the branch and reduce algorithms).

In Algorithm 5 *adding x to the solution* means moving x to V_R and moving $N(v) \cap V_B$ to V_F.

Correctness. Let's consider branching only (lines 4-5), as the rest is trivial. There is some minimum cardinality solution D. Then one can observe that there must exist a minimum cardinality solution D' which satisfies either

1. $v \notin D'$ and $u_1 \in D'$
2. $v \in D'$ and $u_1, u_2 \notin D'$
3. $v, u_1 \notin D'$

If $v \notin D$, then D satisfies 1. or 3. Otherwise, either D satisfies 2., or v and (at least) one of its neighbors is in D. Let D' contain the other neighbor instead of v. Then $|D'| = |D|$ and D' is a solution. Now for $i = 1, 2, 3$ $i - th$ branch on lines $4 - 5$ works under the assumption that there exists D' satisfying the $i - th$ condition from above.

Algorithm 5 Minimum Optional Dominating Set

procedure MODS($V_F, V_B, V_R, E(G)$) ▷ $V(G) = V_F \cup V_B \cup V_R$

if there exists $u \in V_F$ with a unique neighbor v, remove u, and call recursively

if there exists $u \in V_B$ with a unique neighbor v, add v to the solution, remove u, and call recursively

if there exists $v \in V_F$ with two neighbors u_1, u_2, consider three branches:

1. add u_1 to the solution and remove v
2. add v to the solution and remove u_1, u_2
3. remove v

and then call recursively

if there exists $v \in V_B$ with two neighbors u_1, u_2, consider three branches

1. add u_1 to the solution and remove v
2. add v to the solution and remove u_1, u_2
3. remove v and add u_2 to the solution

and then call recursively

let $I \subseteq V_B$ be the set of that are isolated (of degree 0). Move I to V_R.

the minimum degree in $G[V_F \cup V_B]$ is 3. Among all $S \subseteq V_F \cup V_B$ such that $|S| \leq \frac{3}{8}|V_F \cup V_B|$ choose the minimum optional dominating set S^* in $G[V_F \cup V_B]$ and return $S^* \cup V_R$

end procedure

Complexity analysis For graphs of minimum degree 3, the algorithm tests $O^*(\binom{n}{\lfloor \frac{3n}{8} \rfloor}) = O^*(1.93782^n)$ subsets. The analysis of branching rules uses recursive inequalities, similarly to Measure and Conquer algorithms. Let $T(n)$ denote total number of subsets tested in all sub-calls of the algorithm. As mentioned, for leaf sub-instance $T(n) = O^*(1.93782^n)$. Branching on line 4 gives the following inequality

$$T(n) \leq T(n-2) + T(n-3) + T(n-1)$$

as $n-2, n-3, n-1$ are the number of vertices in the first, second and third branch, respectively. This inequality corresponds to some $\alpha_1 \approx 1.8393$ (inequality (2.2)) Similarly, branching on line 5 gives the following inequality

$$T(n) \leq T(n-2) + T(n-3) + T(n-2)$$

This inequality corresponds to some $\alpha_2 \approx 1.6180$ (inequality (2.2)).

Putting both recurrences together, we get $\alpha_1, \alpha_2 < 1.93782$, thus the algorithm finishes in $O^*(1.93782^n)$.

2.2.2 Algorithm of Schiermeyer

The algorithm is described in [23]. Its part is a pre-processing algorithm that mutates an instance of $MODS$, obtaining equivalent one (with same cardinality of minimum optional dominating set), which satisfies some useful properties.

Algorithm 6 Core pre-processing

```
1: procedure CORE(V_F, V_B, V_R, E(G))                    ▷ V(G) = V_F ∪ V_B ∪ V_R
2:     do
3:         remove from V_B all isolated vertices and add them to V_R
4:         remove V_B ∩ N[V_R] from V_B and add to them to V_F
5:         remove all edges with both endpoints in V_F or one endpoint in V_R
6:         remove all vertices in V_F, which have at most one neighbor in V_B
7:         while there exists a vertex u in V_B, with a unique neighbor v do
   shift v to V_R
8:         end while
9:     while anything applied
10:    return changed sets V_F, V_B, V_R, E(G)
11: end procedure
```

One can observe the following properties about the core instance:

- during pre-processing, any vertex can be moved only in the following ways: from V_B to V_F or from V_B to V_R or from V_F to V_R
- $v \in V_F \cup V_B \implies \deg(v) \geq 2$. Moreover, V_F is an independent set, so $v \in V_F \implies |N[v] \cap V_B| \geq 2$

- if $u \in V_F \cup V_B, v \in V_R$, then u, v are in different connected components. $MODS$ can be solved for each connected component separately, so one can move whole V_R to a set D and remove them from considerations: $V_R = \emptyset$.

Suppose that S is a subset of $V_F \cup V_B$, such that $3|S| \leq |N[S]|$ and $\forall_{v \notin S}$ $3|S \cup \{v\}| > |N[S \cup \{v\}]|$. Then the following hold:

- $v \notin N[S] \implies v$ has at most one neighbor in $V \setminus N[S]$.
- $T', U', S', E' = \text{CORE}(V_F \setminus S, V_B \setminus S, S, E(G))$, then $U' \subseteq V \setminus N[S]$. Thus, any connected component of $G[U']$ is an isolated vertex or two connected vertices.
- $v \in T' \implies v$ has at most two neighbors in U'. Since $|N[v] \cap V_B| \geq 2$, $|N[v] \cap V_B| = 2$

The following algorithm returns minimum cardinality optional dominating set.

Algorithm 7 Minimum Optional Dominating Set

1: **procedure** MODS($V_F, V_B, V_R, E(G)$) ▷ $V(G) = V_F \cup V_B \cup V_R$
2: start with $D = \emptyset$
3: $V_F, V_B, V_R, E(G) \leftarrow \text{CORE}(V_F, V_B, V_R, E(G))$
4: $D \leftarrow V_R$, $V_R \leftarrow \emptyset$
5: if $V_B = \emptyset$, return D
6: check whether there exists $S \subseteq V_F \cup V_B$ such that $3|S| \leq |V_F \cup V_B|$ and S is optional dominating set in $G[V_B \cup V_F]$. If one exists, choose one with minimum cardinality and return $D \cup S$
7: **for each:** $S \subseteq V_F \cup V_B$, such that $3|S| \leq |N[S]|$ and $\forall_{v \notin S}$ $3|S \cup \{v\}| > |N[S \cup \{v\}]|$ **do**
8: $T', U', S', E' = \text{CORE}(V_F \setminus S, V_B \setminus S, S, E(G))$
9: let $H = G[U']$. For all $u \in T', v_1, v_2 \in U'$ such that $\{u, v_1\}, \{u, v_2\} \in E', \{v_1, v_2\} \notin E'$ add $\{v_1, v_2\}$ to $E(H)$.
10: compute the maximum matching M in H (with added edges)
11: let set D_S consist of vertices not covered by M, and for each $e \in M$ add one $v \in T' \cup U'$, such that both endpoints of e are its neighbors in $G' = (T' \cup U', E')$.
12: **end for**
13: out of all S considered on line 7 return $D \cup S' \cup D_S$ with minimum cardinality
14: **end procedure**

Correctness. Core pre-processing returns an instance with the same cardinality of optimal solution because, for lines 6,7 of Algorithm 6 if u, v with $N[u] \cap V_B \subseteq N[v] \cap V_B$ it is always fine to take v instead of u to the solution. If Algorithm 7 finishes before line 7, it checks all possibilities of core instance. If it continues, then there exists some optimal solution D^* with

$D \subseteq D^*$ and $3|D^* \setminus D| > |V_F \cup V_B|$. Therefore $3|D^* \setminus D| > |N[D^* \setminus D]|$. Let $S^* \subseteq D^* \setminus D$ be a (greedily computed) set such that $3|S| \leq |N[S]|$ and $\forall_{v \notin S^*}$ $3|S^* \cup \{v\}| > |N[S^* \cup \{v\}]|$. S^* is then one of the sets considered on line 7 and for S^* algorithm returns $D \cup S^{*\prime} \cup D_{S^*}$. Since core pre-processing is correct, there exists optimal $D^{*\prime}$, such that $S^{*\prime} \subseteq D^{*\prime}$. It remains $|D^{*\prime} \cap (T' \cup U')| = |D_{S^*}|$. This is true because $v \in T' \cup U' \implies |N[v] \cap U'| \leq 2$. Thus, the minimum dominating set is equivalent to minimum edge cover 12, which is de facto what the algorithm constructs.

Complexity analysis. Core pre-processing runs in polynomial time because, in each iteration, either an edge or a vertex is deleted or a vertex is moved in the direction $V_B \to V_F \to V_R$. On line 6 of Algorithm 7 there are $O^*(\binom{n}{\lfloor \frac{n}{3} \rfloor}) = O^*(1.8899^n)$ sets considered. On line 7 condition $3|S| \leq |N[S]|$ implies $|S| \leq \frac{n}{3}$, there are at most $O^*(1.8899^n)$ such sets. All other computations can be performed in polynomial time, which gives $O^*(1.8899^n)$ time complexity.

Chapter 3

Implementation

The algorithms were implemented in C++. *NetworKit* and *Boost* frameworks were used for data structures (e.g. graph) and algorithms (e.g. for random graph generation and finding maximum matching). The implementation's source code was added to the **koala-networkit** repository and is available at `https://github.com/krzysztof-turowski/k`
Classes BranchAndReduceMDS⟨GrandoniMSC⟩, BranchAndReduceMDS⟨FominGrandoniKratschMSC⟩, BranchAndReduceMDS⟨RooijBodlaenderMSC⟩, FominKratschWoegingerMDS, SchiermeyerMDS implement algorithms of GRANDONI, Fomin-Grandoni-Kratsch, van Rooij-Bodlaender, Fomin-Kratsch-Woeginger, Schiermeyer, respectively.

3.1 Implementation decisions

3.1.1 Measure and Conquer algorithms

1. Due to the fact that implementation is expected to run on instances that contain several dozen vertices, the asymptotic time of execution is not the only factor one should take care of. To enhance implementation's running time, one may consider avoiding some operations that are slow in practice. For instance, copying and allocating memory for the structures defining the correspondence of elements belonging to the sets. The alternative is to modify the structures before each call to the sub-problem and retrieve them whenever it returns.

2. The family of sets in the sub-problems does not contain empty sets. These, however, may be present in memory representations of the structures used. They are not counted into the solution's cardinality and do not influence any high-level logic of the algorithm.

3.1.2 Algorithms for the minimum optional dominating set

1. Similarly as for measure and conquer copying structures used in recursive calls are avoided, instead, these are modified and then restored properly after a recursive call.

2. In the parts of the algorithms, where small dominating set candidates are tested (of cardinality up to $\frac{1}{3}|V(G)|$ and $\frac{3}{8}|V(G)|$ in the algorithms of Schiermeyer, Fomin-Kratsch-Woeginger, respectively), search is conducted in ascending cardinality order.

3.2 Data sets

Considered graphs were:

- all (non-isomorphic) graphs of cardinality up to 10. These graphs were sourced from a site `https://users.cecs.anu.edu.au/~bdm/data/graphs.html`.

- because the number of different graphs grows too quickly, bigger graphs were generated randomly using *NetworKit*, with each pair of vertices considered independently, with equal probability of being connected by an edge.

3.2.1 Correctness

For each implemented algorithm, the implementation's correctness was tested against the exhaustive algorithm on all small graphs. Moreover, implemented algorithms were checked against themselves on bigger random graphs.

3.3 Experiments

Considered graph groups were of average degree 3, 6 and $(|V(G)| - 1)/2$. For each of them and each integer $n = |V(G)|$ from 11 to 64, 100 random graphs were created. Each algorithm was executed on the same set of graphs. The average and worst (among these 100 graphs) execution times are presented in the following figures.

- graphs with average degree 3 can be solved very fast, especially with branch and reduce algorithms (although when $n > 32$, the execution time experiences high variance (Figure 3.4). It is due to the fact that many reduction rules are focused on sets of cardinality 2. The Fomin-Kratsch-Woeginger algorithm also solves this type of graphs quickly because, in the beginning, it branches of vertices of degree one and two. The algorithm of

Schiermeyer is the slowest one. In such graphs, dominating sets are relatively large, and it cannot find one with cardinality $\leq \frac{n}{3}$, thus performing worse.

- small graphs with average degree 6 can be solved quicker with the Schiermeyer and Fomin-Kratsch-Woeginger algorithms. They are quite dense in this case. However, for larger (quite sparse) graphs, Branch and Reduce algorithms outperform them.
- the algorithms of Grandoni, Fomin-Grandoni-Kratsch, and van Rooij-Bodlaender have similar execution times. In fact, this is understandable: in [24] it was proved that the running times are $O(1.5709^n)$, $O(1.5169^n)$, $O(1.4969^n)$, respectively. It is unknown, whether these bounds are tight. It is possible that asymptotically these differ more or less that it follows from the measure and conquer analysis. It is also possible that the hardest graphs for these algorithms are completely different from the ones considered.
- when graphs are very dense (average degree $\approx (n-1)/2$), the algorithms of Schiermeyer and Fomin-Kratsch-Woeginger seem to be better. It is due to the fact, that minimum dominating sets are usually small in such scenario, possibly $O(\log(n))$.
- in the Figure 3.11 small graphs were benchmarked. All algorithms visibly outperform the exhaustive algorithm for input graphs with just 10 vertices.

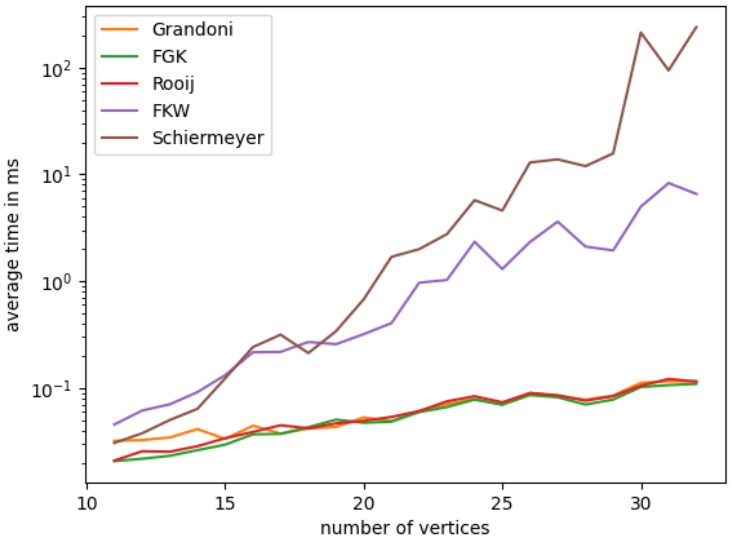

Figure 3.1: average degree = 3

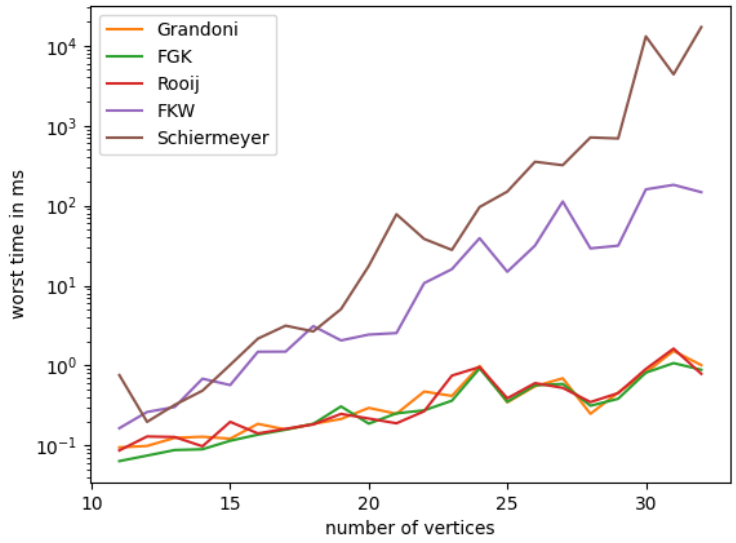

Figure 3.2: average degree = 3

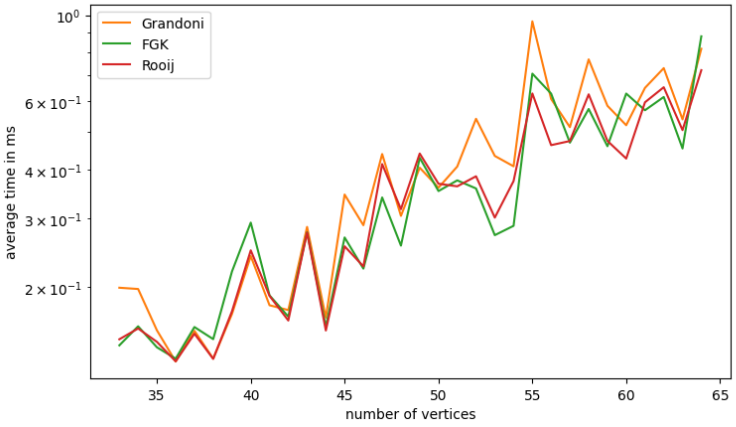

Figure 3.3: average degree = 3

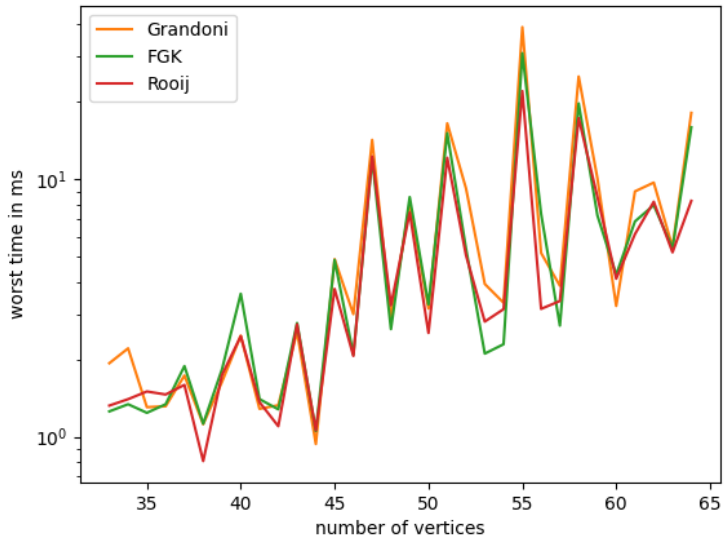

Figure 3.4: average degree = 3

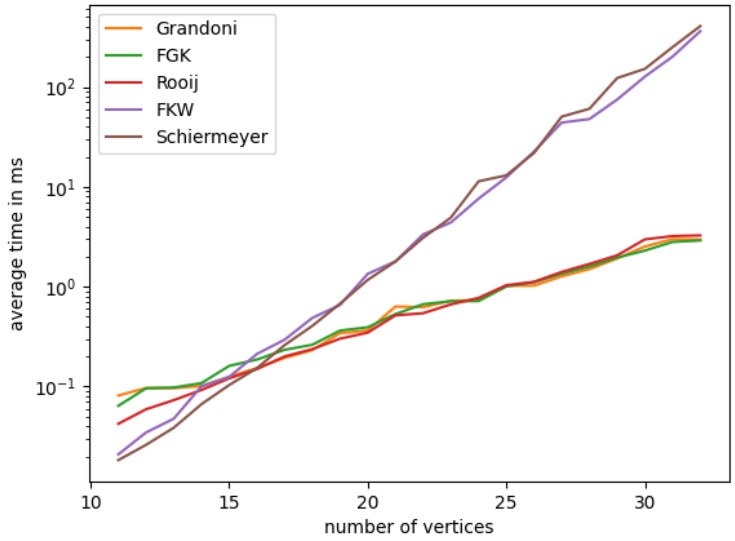

Figure 3.5: average degree = 6

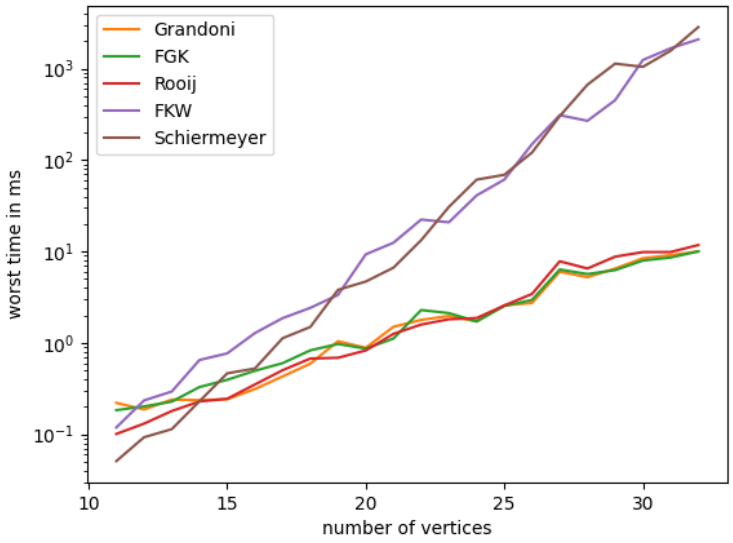

Figure 3.6: average degree = 6

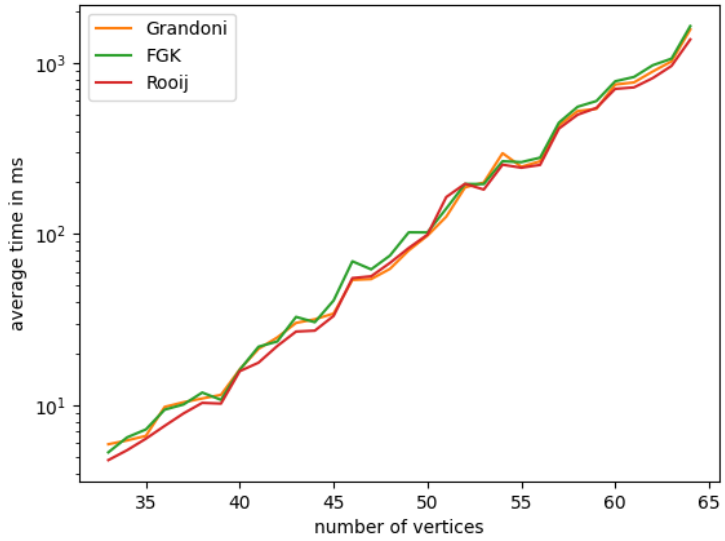

Figure 3.7: average degree = 6

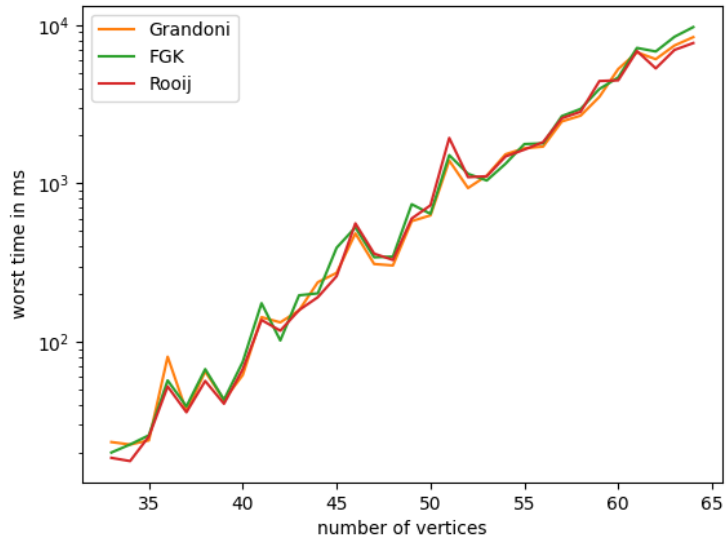

Figure 3.8: average degree = 6

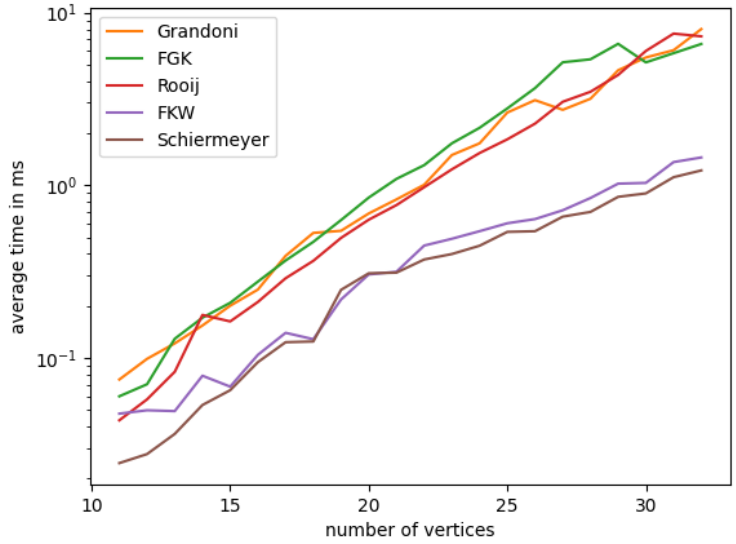

Figure 3.9: average degree = $\frac{n-1}{2}$

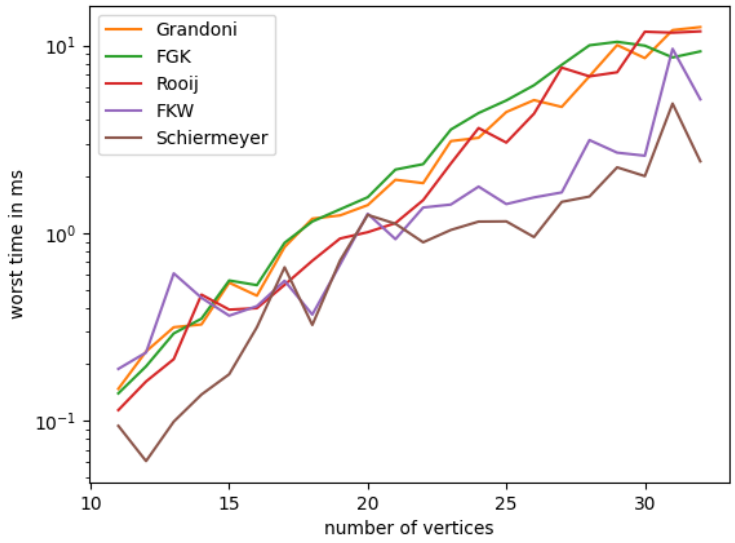

Figure 3.10: average degree = $\frac{n-1}{2}$

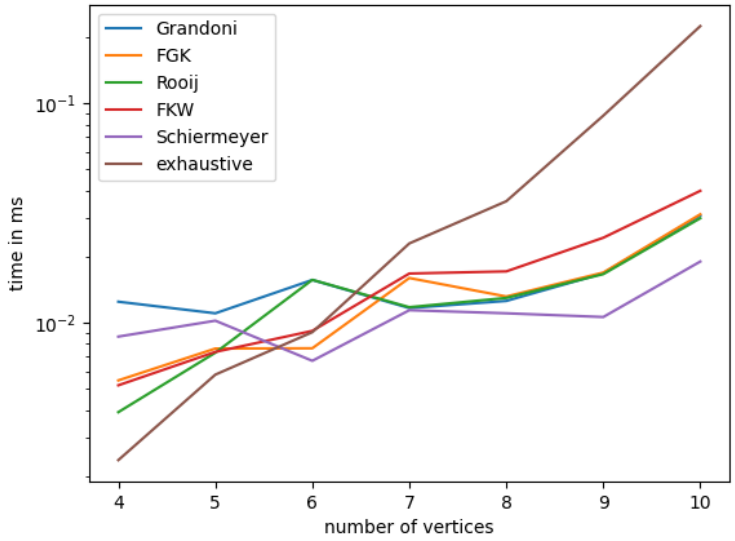

Figure 3.11: all graphs between 4 and 10 vertices, average time

Bibliography

[1] Xin Bai et al. "Minimum connected dominating sets in heterogeneous 3D wireless ad hoc networks". In: *Ad Hoc Networks* 97 (2020), p. 102023. ISSN: 1570-8705. DOI: https://doi.org/10.1016/j.adhoc.2019.102023. URL: https://www.sciencedirect.com/science/article/pii/S1570870518304074.

[2] Béla Bollobás. *Modern graph theory*. Vol. 184. Springer Science & Business Media, 1998.

[3] Marek Cygan et al. *Parameterized Algorithms*. Springer International Publishing, 2015. ISBN: 9783319212753. URL: https://books.google.pl/books?id=FrgOCgAAQBAJ.

[4] Lukas Dijkstra, Andrei Gagarin, and Vadim Zverovich. "Weighted domination models and randomized heuristics". In: *arXiv preprint arXiv:2203.00799* (2022).

[5] Rodney G. Downey and Michael R. Fellows. "Fixed-parameter tractability and completeness". In: *Complexity Theory: Current Research*. 1992, pp. 191–225.

[6] Jack Edmonds. "Paths, Trees, and Flowers". In: *Canadian Journal of Mathematics* 17 (1965), pp. 449–467. DOI: 10.4153/CJM-1965-045-4.

[7] Fedor V. Fomin, Fabrizio Grandoni, and Dieter Kratsch. "A Measure & Conquer Approach for the Analysis of Exact Algorithms". In: *J. ACM* 56.5 (Aug. 2009). ISSN: 0004-5411. DOI: 10.1145/1552285.1552286. URL: https://doi.org/10.1145/1552285.1552286.

[8] Fedor V. Fomin, Fabrizio Grandoni, and Dieter Kratsch. "Measure and Conquer: Domination – A Case Study". In: *Automata, Languages and Programming*. Ed. by Luís Caires, Giuseppe F. Italiano, Luís Monteiro, Catuscia Palamidessi, and Moti Yung. Berlin, Heidelberg: Springer Berlin Heidelberg, 2005, pp. 191–203. ISBN: 978-3-540-31691-6.

[9] Fedor V. Fomin, Fabrizio Grandoni, and Dieter Kratsch. "Some New Techniques in Design and Analysis of Exact (Exponential) Algorithms". In: *Bull. EATCS* 87 (2005), pp. 47–77.

[10] Fedor V. Fomin, Fabrizio Grandoni, Artem V. Pyatkin, and Alexey A. Stepanov. "Bounding the Number of Minimal Dominating Sets: A Measure and Conquer Approach". In: *Algorithms and Computation*. Ed. by Xiaotie Deng and Ding-Zhu Du. Berlin, Heidelberg: Springer Berlin Heidelberg, 2005, pp. 573–582. ISBN: 978-3-540-32426-3.

[11] Fedor V. Fomin, Dieter Kratsch, and Gerhard J. Woeginger. "Exact (Exponential) Algorithms for the Dominating Set Problem". In: *Graph-Theoretic Concepts in Computer Science*. Ed. by Juraj Hromkovič, Manfred Nagl, and Bernhard Westfechtel. Berlin, Heidelberg: Springer Berlin Heidelberg, 2005, pp. 245–256. ISBN: 978-3-540-30559-0.

[12] Michael R. Garey and David S. Johnson. *Computers and Intractability; A Guide to the Theory of NP-Completeness*. USA: W. H. Freeman & Co., 1990. ISBN: 0716710455.

[13] Fabrizio Grandoni. "A note on the complexity of minimum dominating set". In: *Journal of Discrete Algorithms* 4.2 (2006), pp. 209–214. ISSN: 1570-8667. DOI: https://doi.org/10.1016/j.jda.2005.03.002. URL: https://www.sciencedirect.com/science/article/pii/S1570866705000225.

[14] Fabrizio Grandoni. *Exact Algorithms for Hard Graph Problems*. https://people.idsia.ch/~grandoni/Pubblicazioni/G04phd.pdf. [Online; accessed 5-July-2023]. 2004.

[15] Russell Impagliazzo and Ramamohan Paturi. "On the Complexity of K-SAT". In: *J. Comput. Syst. Sci.* 62.2 (Mar. 2001), pp. 367–375. ISSN: 0022-0000. DOI: 10.1006/jcss.2000.1727. URL: https://doi.org/10.1006/jcss.2000.1727.

[16] Yoichi Iwata. "A Faster Algorithm for Dominating Set Analyzed by the Potential Method". In: *Parameterized and Exact Computation*. Ed. by Dániel Marx and Peter Rossmanith. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 41–54. ISBN: 978-3-642-28050-4.

[17] Richard M. Karp. "Reducibility among Combinatorial Problems". In: *Complexity of Computer Computations: Proceedings of a symposium on the Complexity of Computer Computations, held March 20–22, 1972, at the IBM Thomas J. Watson Research Center, Yorktown Heights, New York, and sponsored by the Office of Naval Research, Mathematics Program, IBM World Trade Corporation, and the IBM Research Mathematical Sciences Department*. Ed. by Raymond E. Miller, James W. Thatcher, and Jean D. Bohlinger. Boston, MA: Springer US, 1972, pp. 85–103. ISBN: 978-1-4684-2001-2. DOI: 10.1007/978-1-4684-2001-2_9. URL: https://doi.org/10.1007/978-1-4684-2001-2_9.

[18] Ralf Klasing and Christian Laforest. "Hardness results and approximation algorithms of k-tuple domination in graphs". In: *Information Processing Letters* 89.2 (2004), pp. 75–83. ISSN: 0020-0190. DOI: https://doi.org/10.1016/j.ipl.2003.10.004. URL: https://www.sciencedirect.com/science/article/pii/S0020019003004642.

[19] Mark Sh. Levin. "On combinatorial optimization for dominating sets (literature survey, new models)". In: *arXiv preprint arXiv:2009.09288* (2020).

[20] Bert Randerath and Ingo Schiermeyer. *Exact algorithms for MINIMUM DOMINATING SET*. https://www.researchgate.net/publication/239724498_Exact_algorithms_for_MINIMUM_DOMINATING_SET. [Online; accessed 5-July-2023]. 2004.

[21] Bruce Reed. "Paths, Stars and the Number Three". In: *Combinatorics, Probability and Computing* 5.3 (1996), pp. 277–295. DOI: 10.1017/S0963548300002042.

[22] Johan M. M. Van Rooij and Hans L. Bodlaender. *Design by Measure and Conquer, A Faster Exact Algorithm for Dominating Set*. 2008. arXiv: 0802.2827 [cs.DS].

[23] Ingo Schiermeyer. "Efficiency in exponential time for domination-type problems". In: *Discrete Applied Mathematics* 156.17 (2008). Cologne/Twente Workshop on Graphs and Combinatorial Optimization, pp. 3291–3297. ISSN: 0166-218X. DOI: https://doi.org/10.1016/j.dam.2008.05.035. URL: https://www.sciencedirect.com/science/article/pii/S0166218X08002795.

[24] Johan M.M. van Rooij and Hans L. Bodlaender. "Exact algorithms for dominating set". In: *Discrete Applied Mathematics* 159.17 (2011), pp. 2147–2164. ISSN: 0166-218X. DOI: https://doi.org/10.1016/j.dam.2011.07.001. URL: https://www.sciencedirect.com/science/article/pii/S0166218X11002393.