Broadcast Domination with flexible powers

DIPLOMARBEIT

zur Erlangung des akademischen Grades
Diplom-Informatiker

FRIEDRICH-SCHILLER-UNIVERSITÄT JENA
Fakultät für Mathematik und Informatik

eingereicht von Michael Schnupp
geb. am 17.03.1978 in Werdau

Betreuer: Jiong Guo
Prof. Dr. Rolf Niedermeier

Jena, 28.04.2006

Zusammenfassung

In der Graphentheorie sind Dominierungsprobleme eine wichtige Gruppe von Problemen. Das wohl bekannteste und meist untersuchte Problem aus dieser Gruppe ist Dominating Set. Auch Probleme wie Roman Domination und Broadcast Domination sind bekannte Dominierungsprobleme, welche alle eine gewisse Ähnlichkeit besitzen. Bei allen diesen Problemen müssen die Knoten eines Graphens dominiert werden. Um dies zu erreichen, werden Knoten aus dem Graphen ausgewählt und diesen eine gewisse ,,Dominierungsstärke" zugewiesen.

In dieser Arbeit werden wir das Problem General Broadcast Domination vorstellen. Unter anderem ergeben sich dann die oben genannten Probleme als Spezialfälle dieses Problems, indem verschiedene Dominierungsstärken zugelassen werden und diesen Stärken verschiedene Kostenfunktionen zugeordnet werden.

Wir werden danach zeigen, dass alle so entstehenden Probleme auf allgemeinen Graphen NP-vollständig sind, falls die zugelassene Dominierungsstärke nach oben beschränkt ist. Dies gilt sogar weiterhin, falls der Eingabegraph planar, bipartit oder chordal ist. Weiterhin werden wir für diese Art Probleme einen Algorithmus angeben, welcher auf Graphen mit beschränkter Baumweite in polynomieller Zeit eine Lösung berechnet.

Zuletzt werden wir zeigen, dass, wenn wir beliebig große Dominierungsstärken zulassen, einige Probleme selbst auf allgemeinen Graphen effizient lösbar sind. Wir werden Verfahren skizzieren, die bei der Lösung dieser Probleme behilflich sind und offene Probleme vorstellen, bei denen diese Verfahren versagen.

Abstract

In this thesis we examine problems similar to Dominating Set and Broadcast Domination, which ask to dominate a graph by assigning different powers to some vertices of the graph. We will introduce a new problem called General Broadcast Domination and show that Dominating Set and Broadcast Domination are among others special cases of General Broadcast Domination.

We will prove that all these problems are NP-complete on general graphs and even on planar, bipartite, and chordal graphs, if there is an upper bound for the allowed power. We further give a polynomial time algorithm which solves all those problems for graphs with bounded treewidth.

At the end we will show how the situation changes, if arbitrary large powers are allowed. We show ways to solve some of these problems efficiently and give examples of similar problems with still unknown complexity.

Contents

1 Introduction **8**

2 Preliminaries **10**
2.1 Notation 10
2.2 Definition of some problems 11
2.3 Graph classes 13
2.4 Tree Decomposition 14
2.5 Handling of NP-complete problems 16

3 General Broadcast Domination **17**
3.1 Motivation 17
3.2 Formal definition 19
3.3 Special cases 20

4 NP-completeness for problems with bounded power set **23**
4.1 NP-completeness of Roman Domination 24
4.2 NP-completeness of (a, b)-Domination 28
4.3 NP-completeness of r-Dominating Set (and other problems) . 30

5 Algorithms for problems with bounded power set **32**
5.1 Induced power 32
5.2 An algorithm for trees 33
5.2.1 Notation 33
5.2.2 Algorithm 35
5.3 An algorithm for graphs with bounded treewidth 38
5.3.1 Preconditions 38
5.3.2 The idea 38
5.3.3 The algorithm 41
5.3.4 Time complexity 47

6 Problems with unbounded power set **49**
6.1 Cases with an efficient solution 49
6.2 Cases with a radial solution 51
6.3 Cases with a linear solution 52
6.4 Cases with unknown complexity 52

7 Conclusion **54**

Bibliography **55**

1 Introduction

Algorithmic graph theory is a very important area of theoretical computer science ([5], [8]). Many real world problems can be modeled as graph problems, since many scenarios can be represented by graphs. A graph consists of a set V of vertices and a set E of edges between these vertices. For example a road map consists of cities and roads between them, or a social network may consist of persons and relations between them. Taking the latter example, we can use graphs as the Figure 1.1 to illustrate social networks by letting the graph vertices represent different people and edges mean that these people know each other.

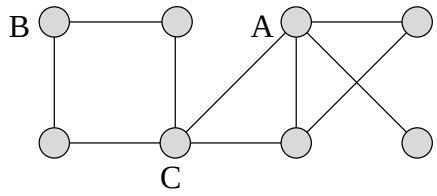

Figure 1.1: A graph representing a social network. The vertices (gray circles) are people which know each other iff there is an edge (line) between them.

With this intuitive model one can ask different questions. One question could be: How many people do we need to know, such that we know everyone either directly or indirectly via a second person?

Looking at the figure we can find knowing A and B is already enough. In this case it is quite simple to see the solution, but given a more complicated graph we usually cannot see the solution at once. We can even show that we cannot hope for an efficient solving strategy for this problem, since this problem is equivalent to a well-known problem called Dominating Set, which is known to be NP-complete [12], which means we cannot hope for an efficient solution strategy.

We can then get similar derived problems by changing the problem a little bit: Maybe we are already pleased if we know everyone "over two corners", i.e., we always know someone who knows someone knowing that person. Hence we can reach all people within distance two from our friends. In the given example in Figure 1.1 we see that we need only C in that case.

But if we want, for example, to pass a message to everyone it may be much more time-consuming for a person to see the neighbors of the neighbors, too. Therefore we might be tempted to give those cases different costs and trying to minimize the total amount. And there the story goes.

In this thesis we examine a class of problems similar to all those mentioned above. We are going to define a new class of problems, called General Broadcast Domination, which allows different allowed distances, called powers, and different cost functions and thereby subsumes amongst others the problems Dominating Set, Roman Domination, and Broadcast Domination, since they will turn out to be special cases of General Broadcast Domination. We will examine the complexity of some of these problems and try to find results which hold for a class of problems as large as possible.

In particular, we will divide all General Broadcast Domination problems in problems with bounded power and with unbounded power. We are going to show, that a bounded power will always make the problem NP-complete on general graphs and some graph classes. We further prove that those problems can however always be solved efficiently if the input graph has a bounded treewidth. In the last part we will examine problems with unbounded powers. Some of these problems can be solved efficiently even on general graphs, but the complexity of some others is still open.

2 Preliminaries

2.1 Notation

Let $G = (V, E), E \subseteq \{\{u, v\} \mid u, v \in V\}$ be a graph. We always assume that graphs are simple (no loops or multiple edges), undirected, unweighted, and connected. We write $V(G)$ to denote the vertex set V of G and we write $E(G)$ to denote the edge set E of G. We will refer to Figure 2.1 as an example for the following definitions.

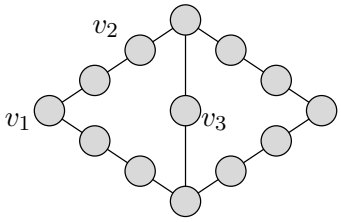

Figure 2.1: An example graph G for illustrating the definitions.

The *distance* between two vertices u, v in G, denoted by $d_G(u, v)$, is the minimum number of edges on a path between u and v. (We usually drop the index G writing $d(u, v)$.) The distance between v_1 and v_3 in Figure 2.1 is therefore $d(v_1, v_3) = 4$.

The *eccentricity* of a vertex v, denoted by $e(v)$, is the largest distance from v to any vertex of G. In Figure 2.1, $e(v_1) = 6$, $e(v_2) = 5$, and $e(v_3) = 4$.

The *radius* of G, denoted by $rad(G)$, is the smallest eccentricity in G, while the *diameter* of G, denoted by $diam(G)$, is the largest eccentricity in G. In Figure 2.1 $rad(G) = 4$ and $diam(G) = 6$.

A vertex v with $e(v) = rad(G)$ is called a *center vertex*. Vertex v_3 and its two neighbors are center vertices in the example.

For a vertex set $S \in V$ the *induced subgraph* is $G[S] = (S, \{\{u, v\} \in E \mid u, v \in S\})$.

The *neighborhood* of a vertex v is the set $N(v) = \{w \in V \mid \{v, w\} \in E\}$. Furthermore for a constant k we define $N_k(v) = \{w \in V \mid d(w, v) \leq k\}$ and for a function $f : V \to \mathbb{N}$ we define $N_f(v) = \{w \in V \mid d(w, v) \leq f(v)\}$.

The neighborhood of a set of vertices is the union of the neighborhoods of the members.

For a *partition* $V = V_1 \oplus V_2 \oplus V_3 \oplus \ldots$, it holds $\bigcup V_i = V$ and $V_i \cap V_j = \emptyset$ for all $i \neq j$.

The symbol $\mathbb{N}$ denotes the natural numbers including zero.

A P_k is a path consisting of k vertices.

Further basics on algorithmic graph theory and the associated notation can be found in standard literature on graphs [8] and algorithms [7].

2.2 Definition of some problems

In this thesis we will frequently refer to some well-known problems. Many of these are described in [12]. We give now the formal definitions of the problems handled in this thesis.

Vertex Cover

> INPUT: A graph $G = (V, E)$ and an integer k.
> QUESTION: Is there a subset $S \subseteq V$, $|S| \leq k$, such that $\forall \{u, v\} \in E : u \in S \vee v \in S$?

We could also think of every vertex covering the incident edges and all edges have to be covered.

Dominating Set

INPUT: A graph $G = (V, E)$ and an integer k.
QUESTION: Is there a subset $S \subseteq V$ with $|V| \leq k$ and $\forall v \in V \setminus S \exists s \in S : \{v, s\} \in E$?

Alternatively we can say that a vertex in S *dominates* itself and its neighbors and it is asked to dominate the whole graph.

Roman Domination

INPUT: A graph $G = (V, E)$ and an integer k.
QUESTION: Is there a set $S_1 \subseteq V$ and a set $S_2 \subseteq V \setminus S_1$ with $|S_1| + 2|S_2| \leq k$ and $\forall v \in V \setminus S_1 \exists s \in S_2 : \{v, s\} \in E$?

Alternatively we can say that a vertex in S_1 dominates itself and a vertex in S_2 dominates itself and each neighbor and it is asked to dominate the whole graph.

Broadcast Domination

INPUT: A graph $G = (V, E)$ and an integer k.
QUESTION: Is there a function $f : V \to \mathbb{N}$ so that $\sum_{v \in V} f(v) < k$ and every vertex is within distance $f(v)$ from some vertex v?

In this case each vertex v with $f(v) > 0$ dominates all vertices within distance $f(v)$ and it is again asked to dominate the whole graph.

For each problem there is also a minimizing version asking for the smallest k for which the corresponding question is true. In the following we will always refer to the minimizing version.

2.3 Graph classes

Many problems can be proved to be NP-complete on general graphs, but they can be solved in polynomial time if the input graph has special properties. Graphs with special properties can be categorized in different graph classes. Each class is defined by a set of constraints for the graph. We give a definition of the classes we are going to use in this thesis.

First we need to introduce two basic definitions: A *clique* of a graph $G = (V, E)$ is a subset $C \in V$ where every pair of vertices from C is directly connected by an edge in G. An *independent set* of a graph is a subset $C \in V$ where no pair of vertices from C is directly connected by an edge in G.

Now we can easily define the needed graph classes. A graph is a *planar* graph if it can be embedded in the plane (drawn with points for vertices and curves for edges) without crossing edges. A graph is a *bipartite* graph if there is a partition of its vertex set into two independent sets. A graph is a *chordal* graph if every cycle of length at least 4 has a chord. A graph is a *split* graph if there is a partition of its vertex set into a clique and an independent set.

Figure 2.2 gives an example of a split and a bipartite graph.

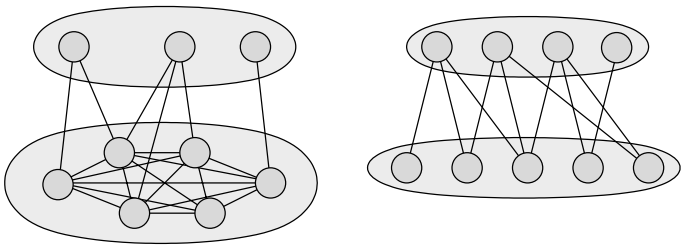

Figure 2.2: A split graph on the left and a bipartite graph on the right. The clique and the independent sets are marked.

A more detailed survey on graph classes can be found in [5]. A pretty complete overview over literally all the known classes with definitions, relations, and references to proofs of important properties can be found online at the "Information System on Graph Class Inclusions"(ISGCI) [4].

Since the properties of some classes restrict those of others, some classes include

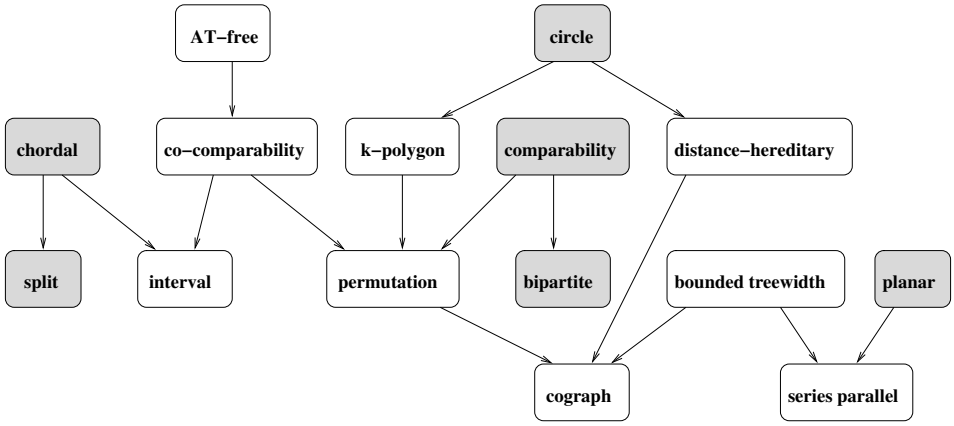

Figure 2.3: Some graph classes and their inclusion. An arrow means that the first class is a superclass of the second. The gray classes are the ones on which Dominating Set is known to be NP-complete. Even many other considered problems will turn out to be NP-complete on these classes.

others. Figure 2.3 shows some classes and their relationships. If a problem can be proved to be NP-complete on a class, it will obviously be NP-complete on any superclass, too. If an algorithm works for a class, it will clearly also work for each subclass.

2.4 Tree Decomposition

Some graphs have similar properties like trees, but obviously are no trees. Figure 2.4 shows such a graph. It has many cycles and is obviously no tree, but nevertheless it looks a bit like a tree, since it is at no place "wider" than two vertices. Such a graph is said to have a bounded treewidth.

Formally we define the treewidth by a *tree decomposition*:

A tree decomposition $\mathcal{X}$ of G is a pair $\langle \{X_i \mid i \in V(T)\}, T \rangle$, where each X_i is a subset of V, called a *bag*, and T is a tree with elements of $V(T)$ as nodes. The following three properties must hold:

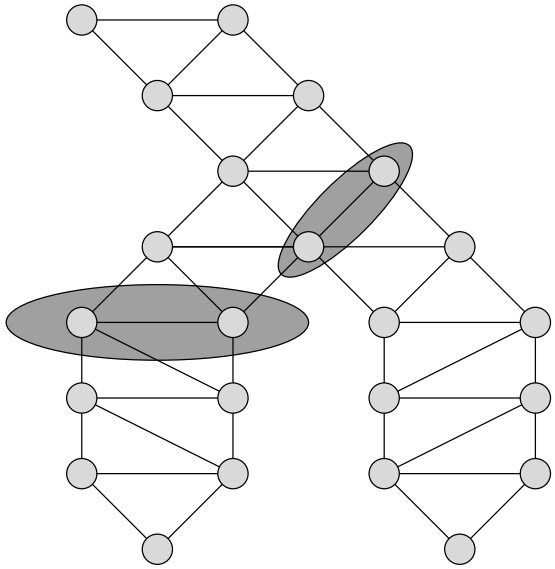

Figure 2.4: A graph with a treewidth of two. We can cut the graph into two parts by deleting bags of two vertices.

- $\bigcup_{i \in V(T)} X_i = V$,
- for every edge $\{u, v\} \in E$, there is an $i \in V(T)$ such that $\{u, v\} \subseteq X_i$,
- for all $i, j, k \in V(T)$, if j lies on the path between i and k in T, then $X_i \cap X_k \subseteq X_j$.

The *treewidth* $tw(\mathcal{X})$ of $\mathcal{X}$ is defined to be the size of the largest bag minus one, i.e.,

$$tw(\mathcal{X}) = \max\{|X_i| \mid i \in V(T)\} - 1$$

A special kind of tree decomposition we are going to use later is called *nice tree decomposition*:

A tree decomposition $\langle \{X_i \mid i \in V(T)\}, T \rangle$ is called a *nice tree decomposition* if the tree T is rooted and the following conditions are satisfied:

- Every node of the tree T has at most two children.

- If a node i has two children j and k, then $X_i = X_j = X_k$ (in this case i is called a *JOIN NODE*).

- If a node i has one child j, then either
(a) $|X_i| = |X_j| + 1$ and $X_j \subsetneq X_i$ (in this case i is called a *INTRODUCE NODE*).
(b) $|X_i| = |X_j| - 1$ and $X_i \subsetneq X_j$ (in this case i is called a *FORGET NODE*).

Each tree decomposition can easily be transformed into a nice tree decomposition[17]. More about tree decomposition can be found in [1],[3], and [21].

2.5 Handling of NP-complete problems

In this thesis we will most of the time only try decide if there is a polynomial time algorithm for a given problem or if it is NP-complete. If the problem turns out to be NP-complete, this means we cannot hope for an polynomial time algorithm to solve this problem. Every algorithm will have an exponential running time in the worst case, and it will therefore usually simply last too long.

If one has to solve such an NP-complete problem anyway there are nevertheless some approach to solve it. We will in this thesis not handle those approaches, but will now point out some literature for the interested reader.

One way could be that we actually do not need the very exact solution and a good approximation is enough. There is much literature on polynomial time approximation algorithms as [2] and [22].

If we want to compute the exact solution anyhow, there is also the possibility that we can find a parameter such that the exponential complexity of the problem can be shifted from the complete input size to the parameter which may be much smaller. Such a parameter can e.g. be the solution size or the treewidth of the input graph. More on fixed-parameter algorithms can be found in [9] and [20].

3 General Broadcast Domination

In this chapter we are going to define the General Broadcast Domination problem, which will be the foundation of this thesis.

3.1 Motivation

Dominating Set [14][15] is considered to be among the most important graph problems. Herein, one is asked to find a subset $D \subseteq V$ dominating all vertices, given that each vertex in D dominates itself and its neighbors.

In the literature many domination-like problems have been considered. In some of these problems new domination rules are added, as e.g. in Power Dominating set[1] [13], where a dominated vertex with all but one neighbors dominated dominates the remaining neighbor, too. Others add restrictions on valid solutions, as Connected Dominating Set [23] where the set D has to induce a connected subgraph of G. A third class of problems results by simply allowing different powers of the domination. This is the class of problems we are going to discuss in this thesis.

With the usual domination mechanism a vertex dominates only its direct neighbors. In Figure 3.1 we therefore need three vertices to dominate the whole graph. The left graph in Figure 3.2 illustrates such a solution. By assigning a power to each vertex in the domination set, we extend the domination range to all vertices within a distance at most equal to the given power. In the example we can now just give the center vertex a power of two and it will dominate the whole graph as illustrated in the middle graph of Figure 3.2. This new domination mechanism comes from the problem Broadcast Domination [16] where

[1] In Power Dominating set the power refers to electrical networks, which suggest special rules. It is independent of the power we are going to assign to vertices.

arbitrary positive powers can be assigned and it is asked to minimize the total power assigned. This problem of Broadcast Domination may arise if a radio station wants to reach all cities in a given area and therefore wants to place broadcast station (with possibly different transmitting power) in some of those cities. A higher power can reach more cities but has a higher cost, too.

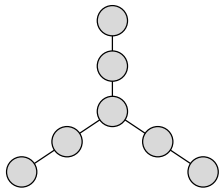

Figure 3.1: A very simple example graph.

There is further a classical problem called Roman Domination [11]. In its original formulation one is asked to protect the Roman empire by placing roman legions in different areas of the empire, assuming that a single legion can protect the region in which it was placed, and a region with two legions can protect the neighbor regions, too; it can afford sending one legion out without leaving the region unprotected.

This problem can be seen as being able to assign two different powers. A power of one enables a vertex to dominate all vertices of distance one, which are obviously the neighbors, and a power of zero, which dominates all vertices with distance zero which is only the vertex itself. In this case we can dominate the example graph by giving the center vertex a power of one while giving the three still undominated vertices a power of zero. This is shown in the right graph of Figure 3.2.

In all problems it is asked to find an optimal solution. Therefore we need also a cost function, which gives the cost of a vertex with power p. In Broadcast Domination the cost is just the power itself while in Roman Domination the cost is the number of legions which is one more than the power.

Figure 3.2 shows again the above described solutions. These solutions are indeed optimal, there is no other solution with a lower cost.

Despite the different solutions, the problems are all quiet similar regarding the way the domination works. Therefore we are going to define the problem Gen-

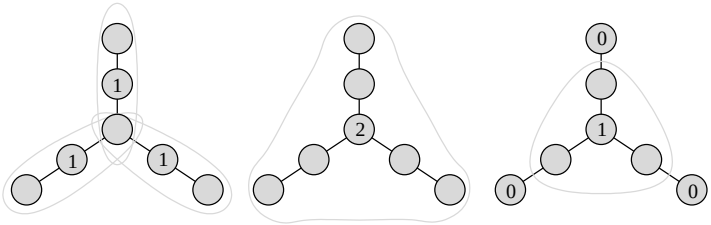

Figure 3.2: An optimal solution for Dominating Set, Broadcast Domination and Roman Domination: The vertices containing numbers form the dominating set. The number itself is the power of that vertex.

eral Broadcast Domination, as a generalization of the above problems.

3.2 Formal definition

To formally define the problem General Broadcast Domination, we need the following definitions.

Let $G = (V, E)$ be the graph we want to dominate and $P \subseteq \mathbb{N}$ the *power set*[2], which determines the allowed powers. Whenever we choose a subset $D \subseteq V$ of *dominating vertices* and a *power function* $p : D \to P$, which assigns each dominating vertex an allowed power, we call the pair (D, p) a *partial domination*. A dominating vertex v with power $p(v)$ dominates all vertices up to distance $p(v)$. A partial domination dominates all vertices $v \in V$ which are dominated by any dominating vertex, i.e., $\{v \mid \exists w \in D : v \in N_p(w)\}$. An example is shown in Figure 3.3.

In addition, a *cost function* $c : P \to \mathbb{N}$ assigns each dominating vertex a cost that depends on its power. Then the cost of the partial domination is simply the sum of the costs of the dominating vertices:

$$c(D, p) = \sum_{v \in D} c(p(v))$$

[2]Here we really mean a set of different powers, not the set of subsets of a set.

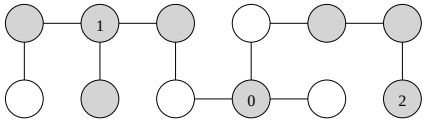

Figure 3.3: An example of a partial domination. The numbered vertices are the dominating vertices. The number itself is the associated power. Therefore, all gray vertices are dominated.

A partial domination which dominates the whole graph with minimum cost is called a *minimum domination* or the *solution* of the General Broadcast Domination problem specified by P and c.

The General Broadcast Domination problem is now defined as follows:

> INPUT: A power set P and a cost function c. A graph $G = (V, E)$.
> QUESTION: What is the cost of a minimum domination?

The problem is similar to Broadcast Domination but we have generalized it in that we can restrict the allowed powers and use different cost functions.

3.3 Special cases

Using the definition of General Broadcast Domination, we can now define more special problems $GBD(P, c)$ by simply specifying P and c. This is done in Table 3.1, which defines well known problems as well as new problems, which can be seen as generalizations of the known problems.

The parameter r in r-Dominating Set and Restricted Broadcast Domination is an arbitrary constant. For $r = 1$ the problems obviously become equivalent to Dominating Set, therefore $r \geq 2$ is the only new case.

It is also easy to see that the cost function should never assign a cost of zero to any power. Otherwise, we have always a cost-zero-solution. It is also not helpful to have a not strictly monotonic increasing cost function, since whenever $c(a) \leq c(b)$ with $b < a$ power b is never needed. Therefore we will only consider strictly monotonic increasing cost functions which assign only positive costs.

Problem	P	$c(p)$
DS (Dominating Set)	$\{1\}$	1
r-Dominating Set ($r \geq 2$)	$\{r\}$	1
RD (Roman Domination)	$\{0, 1\}$	$p + 1$
BD (Broadcast Domination)	$\{1, 2, \ldots, n\}$	p
RBD (Restricted BD)($r \geq 2$)	$\{1, 2, \ldots, r\}$	p
MBD (Modified BD)	$\{0, 1, 2, \ldots, n\}$	$p + 1$
ABD ((a, b)-Domination)	$\{a, \ldots, b\}, a < b$	p

Table 3.1: Some problems seen as special cases of General Broadcast Domination

With $P = \{1\}$ and $c(1) = 1$ we end up with Dominating Set, since each dominating vertex dominates its neighbors and doing so will have the cost of one, which means that the overall cost will be the number of dominating vertices.

In the case of Roman Domination a power of zero (corresponding to a single legion) dominates only the vertex itself and has a cost of one. Whereas a power of one (corresponding to two legions) has a cost of two while dominating the neighbor vertices, too.

While in Roman Domination we cannot assign powers greater than one, in Broadcast Domination any positive power is possible. Since these problems will turn out to have very different complexities, we define the Restricted Broadcast Domination problem in the way that we allow arbitrary powers up to a parameter r. We also define the Modified Broadcast Domination problem, which is almost the same as Broadcast Domination, but by letting the cost be one higher than in usual Broadcast Domination. Therefore we can now allow a power of zero. Because power zero is sometimes not allowed, we can also consider (a, b)-Domination, where only powers between a and b are allowed.

Depending on P and c we get pretty different solutions. Figure 3.4 shows three ways to dominate a path of length nine: We can use three vertices with power one, one with power four or two with power two. Under the restrictions of Broadcast Domination clearly the first way is the cheapest, since the three vertices have cost one each, resulting in a total cost of three while in the other cases the total cost is four. This totally changes if we assume the rules

of Modified Broadcast Domination. Now the cost of each dominating vertex is one higher than in Broadcast Domination. And so the first and the third way will have a total cost of six, while the second one only costs five. When restricting the possible powers, by disallowing powers greater than two the second way becomes invalid and the others have the same cost.

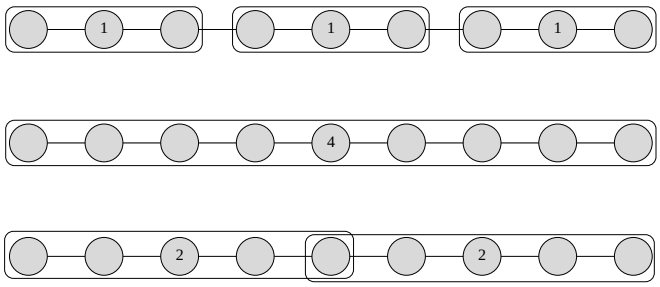

Figure 3.4: Three possible ways to dominate a path of length nine: three dominating vertices with power one, one with power four or two with power two.

We see how different values for P and c change the kind of solution. This is the reason why we might need different strategies to find an appropriate solution. Indeed we will now see that in some cases there is an efficient algorithm while other cases are NP-complete.

4 NP-completeness for problems with bounded power set

While some problems can *always* be solved efficiently, others turn out to be NP-complete on general graphs. Most of them even stay NP-complete when restricted to various graph classes.

Table 4.1 gives again the definition of the problems we are dealing with.

Problem	P	$c(p), p \in P$
Dominating Set	$\{1\}$	1
r-Dominating Set	$\{r\}$	1
Roman Domination	$\{0, 1\}$	$p + 1$
(a, b)-Domination (ABD)	$\{a, \ldots, b\}, a < b$	p
Restricted Broadcast Domination	$\{1, 2, \ldots, r\}$	p

Table 4.1: Allowed powers and cost function, defining the considered problems.

In this section we will prove NP-completeness of some problems on general graphs as well as on selected graph classes. We are going to give a reduction from Dominating Set to r-Dominating Set, from Vertex Cover to Roman Domination and from Roman Domination to (a, b)-Domination. The complexity results for (a, b)-Domination also hold for Restricted Broadcast Domination since it is a special case of (a, b)-Domination namely $(1, r)$-Domination.

Table 4.2 gives the complexity results for the problems we are dealing with.

Class	Recognition	VC	DS	RD	ABD
general	-	NP-c	NP-c	NP-c	NP-c
planar	lin	NP-c	NP-c	NP-c	NP-c
bipartite	lin	pol	NP-c	NP-c	NP-c
comparability	pol	pol	NP-c	NP-c	NP-c
split	lin	lin	NP-c	NP-c	lin
chordal	lin	lin	NP-c	NP-c	NP-c

Table 4.2: An overview of some problems and their complexity on some graph classes. The proofs for recognition (i.e., determining if a given graph belongs to that graph class,) Vertex Cover and Dominating Set are given in [4]. The gray entries in RD(Roman Domination) and ABD((a, b)-Domination) follow from inclusions, while the remaining entries are proved in this chapter.

4.1 NP-completeness of Roman Domination

Roman Domination and many of its properties have been introduced in [6]. The complexity of Roman Domination over some graph classes is studied in [19] and some parameterized results are presented recently in [11].

The NP-completeness of Roman Domination on the considered classes is stated in [19], but we did not find formal proofs in any literature. This is why we show Roman Domination being NP-complete on planar, split, and bipartite graphs by reducing from Vertex Cover, which is NP-complete even on planar graphs[12]. The proofs are very similar, therefore we give them together. Before we begin with the actual reduction, the following definition and lemma will be useful.

A *tent* (Figure 4.1) is a complete bipartite graph $K_{2,3}$, consisting of two *base* vertices and three *top* vertices.

Lemma 1. *Every minimum domination (D, p) for* Roman Domination *on a tent never contains top vertices, and therefore one of the base vertices in assigned a power of one.*

Proof. Assume there would be a minimum domination (D, p), which does not assign power one to a base vertex. Then, the top vertices are not dominated by

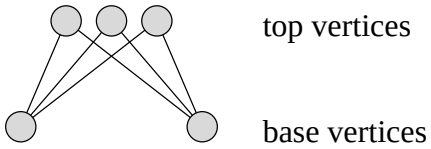

Figure 4.1: A tent consists of two base vertices and tree top vertices

a neighbor and p has to assign at least power zero to all the three top vertices, which will result in cost three. This would be more than the cost of two, which would result by assigning power one to one base vertex. Therefore (D, p) cannot be a minimum domination which would contradict the assumption. □

Theorem 1. ROMAN DOMINATION *is NP-complete on planar, split and bipartite Graphs. There is a reduction from (planar)* VERTEX COVER.

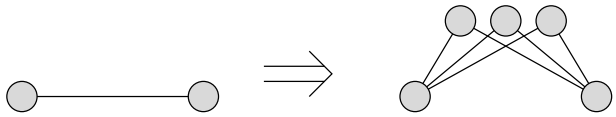

Figure 4.2: Substituting an edge with three new vertices. We thereby create a tent.

Proof. Given a graph $G = (V, E)$ we construct a new graph G' by leaving it unchanged besides substituting each edge with three new vertices (Figure 4.2). By doing so we create a tent for each former edge.

For planar graphs we also add the edges from E. (For planar graphs the resulting graph is still planar.) For split graphs we connect each pair of V by an edge. (The vertices of V now form a clique, while the added vertices are clearly an independent set, hence, the graph is a split graph.) For bipartite graphs we further add a single vertex z and connect it to all the vertices from V as well as to three more new degree one vertices. (This forces z to dominate the three vertices as well as all vertices in V and the graph will still be bipartite.)

A small example of the thereby created graphs is given in Figure 4.3.

Now assume we constructed the new graph G' as described above from the original graph G. First, it is easy to see, that in the bipartite case, each

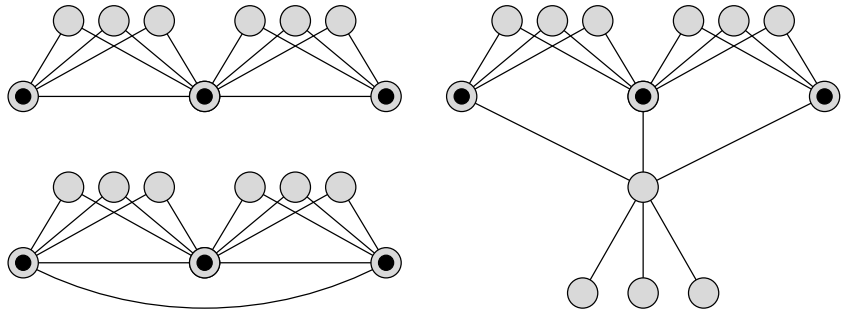

Figure 4.3: The resulting graphs after applying the described transformation on a P_2. The marked vertices are the ones corresponding to the original P_2. At the left are the planar and the split graph; at the right is the bipartite graph.

Roman Domination solution gives vertex z a power of one. (Otherwise all three neighbor vertices of degree one have to be given at least power zero, which would result in a higher cost than a power one for z.)

We are going to show: Graph G has a Vertex Cover solution of size k iff graph G' has a Roman Domination solution of cost $2k$, respectively $2k + 2$ in the bipartite case. The higher cost of the Roman Domination solution in the bipartite case comes from vertex z which is needed to dominate all vertices in V.

First assume graph G has a Vertex Cover solution of size k. Then we get the Roman Domination solution by simply assigning each vertex of the Vertex Cover solution a power of one. Since a Vertex Cover solution contains at least one end vertex for each edge in G, all the top vertices are dominated by those power one vertices. The second base vertex of each tent is dominated by the first one or, in the bipartite case, by vertex z.

Now we study a Roman Domination solution (D, p) on Graph G'. Because of Lemma 1 no top vertex is contained in D and for each tent a base vertex is assigned power one. This means choosing exactly those power one vertices (without the z-vertex) will result in a vertex cover of graph G.

Therefore a Vertex Cover solution on G is essentially a Roman Domination solution on G' and vice versa. □

A stronger reduction from Minimum Set Cover for bipartite graphs

A reduction from Vertex Cover is not very strong since Vertex Cover can for example easily 2-approximated. Therefore we now give a second reduction from Minimum Set Cover, which is much harder to approximate [10].

Theorem 2. Roman Domination *on bipartite graphs is NP-complete. There is a reduction from* Minimum Set Cover.

Proof. Let $(C = \{S_1, S_2, \ldots, S_m\}, k), S_i \subseteq U = \{u_1, u_2, \ldots, u_n\}$ be an instance of Minimum Set Cover.

We now construct a graph $G = (V, E)$ with $V = U' \cup C \cup \{z\} \cup H$ where $U' = \{u_{11}, u_{12}, u_{13}, u_{21}, u_{22}, u_{23}, \ldots, u_{n1}, u_{n2}, u_{n3}\}$ consists of three vertices for each element in U. (They will be the vertices to be dominated.) Vertex z and $H = \{h_1, h_2, h_3\}$ are new help vertices. (They will make sure, the vertices in C are already dominated.) Now let $E = \{\{z, v\} \mid v \in H \cup C\} \cup \{\{S, u_{ij}\} \mid S \in C, u_i \in S, 1 \leq j \leq 3, S \in C\}$. (i.e., we connect H to z, z to C, and C to U' if an element is in the corresponding set.)

Figure 4.4 shows an example of a resulting graph.

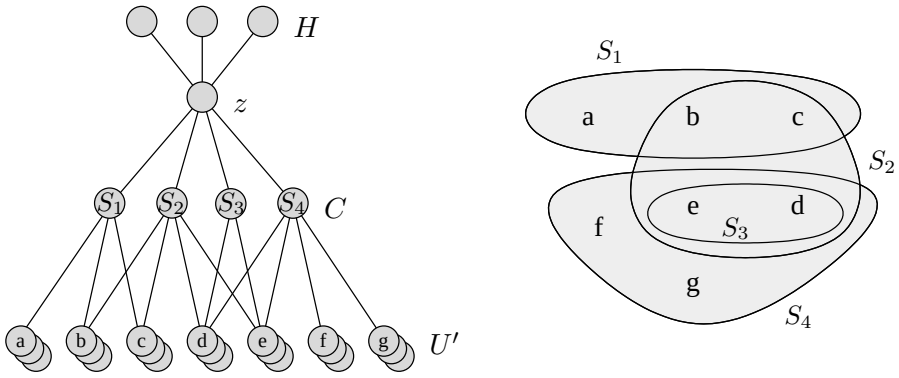

Figure 4.4: The bipartite graph constructed in the proof.

By adding the three leaves h_1, h_2, h_3 we force power one for z in every optimal solution of Roman Domination. The vertices from C are also dominated by vertex z.

Hence we only have to dominate the vertices from U'. Since we have three vertices for each element from U again any optimal solution will never contain vertices from U'. So we have to find a minimum subset of C dominating all the vertices of U' (by letting them have a power of one). This is exactly the problem Minimum Set Cover.

Each Roman Domination solution therefore implies a Minimum Set Cover solution by choosing the power one vertices of C, while we can construct a Roman Domination solution from a Minimum Set Cover solution by simply assigning the corresponding C vertices (as well as vertex z) a power of one.

Clearly, there is a solution of Minimum Set Cover of size k, iff there is a solution of Roman Domination of cost $2k + 2$. □

4.2 NP-completeness of (a, b)-Domination

In this section we deal with (a, b)-Domination, a special case of General Broadcast Domination, which only assigns powers $P = \{a, \ldots, b\}, a < b$ and costs $c(p) = p$.

First we need the following definition and lemma.

An (n, m)-*spider* is a graph consisting of m copies of a path P_n (*legs*) each of them with one end vertex connected by an edge to a central vertex, called *head*. The other end vertices of the legs we call *feet*. An example is shown in Figure 4.5.

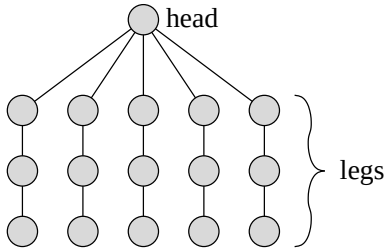

Figure 4.5: A (3,5)-spider

Lemma 2. *An optimal* (a, b)-Domination *solution of a* $(b-1, b)$*-spider never uses a leg vertex as a dominating vertex.*

Proof. Assume we are given an (a, b)-Domination solution of a $(b-1, b)$-spider. If a single dominating vertex dominates more than one foot of the spider, it will dominate all feet and thus the whole spider. If this dominating vertex is a leg vertex, it has to be a neighbor of the head vertex with power b. In this case we could dominate the spider with lower cost by giving the head vertex power $b - 1$, which would contradict the optimality of the solution.

If no two feet are dominated by the same domination vertex, each foot has to be dominated by its own dominating vertex. Since each dominating vertex has at least cost 1, the total cost would again be at least b, which would again be more than giving the head vertex power $b - 1$. □

This lemma will help us constructing a reduction proving the following theorem.

Theorem 3. (a, b)-Domination *is NP-complete on planar, bipartite, and chordal graphs.*

Proof. Since a given solution candidate can easily be verified in polynomial time, (a, b)-Domination is clearly in NP. To show NP-hardness, we give a reduction from Roman Domination.

Given a Roman Domination instance $G = (V, E)$, we construct an (a, b)-Domination instance G' by adding b P_{b-1} to each vertex, i.e., for each vertex $v \in V$ we add b new copies of a P_{b-1} and connect one end vertex of each path to v.

We have thereby created in G' a $(b - 1, b)$-spider for each vertex it V and we will show that G has a Roman Domination solution (D, p) of cost s iff G' has an (a, b)-Domination solution of cost $s + (b - 1)|D|$.

Given a Roman Domination solution (D, p) we can obviously get an (a, b)-Domination solution (D, p') by adding $b - 1$ to each power, i.e., $p'(v) = p(v) + b - 1$ for each $v \in D$.

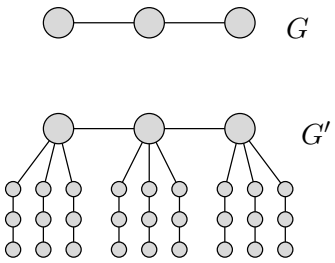

Figure 4.6: The original graph G and the graph G' constructed in the proof.

Given an (a, b)-Domination solution (D, p'), by Lemma 2, set D can not contain any newly added vertices. Since the new vertices have to be dominated, too, p' only assigns the powers b and $b-1$ and we get a valid Roman Domination solution (D, p) by letting $p(v) = p'(v) - (b-1)$.

It is easy to see that the above construction preserves the properties of planar, bipartite, and chordal graphs. Hence, the NP-completeness of (a, b)-Domination follows from the NP-completeness of Roman Domination for these graph classes. □

4.3 NP-completeness of r-Dominating Set (and other problems)

The preceding NP-completeness proof for (a, b)-Domination needs two powers b and $b-1$. That is why we defined the problem with $a < b$ disallowing the case of $a = b$. But even if we have a problem, which allows only a single power r, the problem is also NP-complete.

Theorem 4. *r-Dominating Set is NP-complete on planar, bipartite, chordal graphs.*

Proof. Again, a solution candidate can easily be verified in polynomial time. Therefore r-Dominating Set clearly is in NP. To show the hardness we give a reduction from Dominating Set using the same principle as in the proof of Theorem 3, constructing a graph G' by attaching r P_{r-1} to each vertex of the Dominating Set instance G.

If we have a DOMINATING SET solution (D, p) we can obviously get an r-DOMINATING SET solution (D, p') by adding $r-1$ to each power, i.e., $p'(v) = r$ for each $v \in D$.

Given an r-DOMINATING SET solution (D', p') for G' we will now build a new set D which will be a solution of DOMINATING SET on G. Let $v \in D'$ be a dominating vertex of the r-DOMINATING SET solution.

If $v \in V$, then it is the head of a spider and dominates due to power r all neighbor spiders, too. Therefore we simply choose v in our new dominating set D. If $v \notin V$, then it can only be a leg vertex next to a head vertex h. (Otherwise it would dominate only one feet, which cannot be optimal.) In this case replacing v by h leads to another valid solution, since it dominates (at least) the same vertices and has the same cost. Therefore we add the corresponding head vertex h to D. In this way we get a set D from D' with $|D| = |D'|$, leading to the same cost.

Therefore G has a DOMINATING SET solution of cost x iff G' has an r-DOMINATING SET solution of cost x. □

Looking at the above proof we can observe that it also works for problems where powers $< r-1$ are allowed. Therefore we can state the following corollary.

Corollary 1. *Each* GENERAL BROADCAST DOMINATION *problem with cost function $c(p) = p$ and restricted power set P (i.e., $\exists_k \forall_{p \in P} : p \leq k$) is NP-complete on planar, bipartite, chordal graphs.*

Proof. Let r be the highest allowed power in P. Depending on whether $r-1 \in P$ we use either the proof for Theorem 3 or Theorem 4. Since we always constructed $(r-1, r)$-spiders, smaller powers will never be used. This is why those proofs still work for problems which allow smaller powers. □

5 Algorithms for problems with bounded power set

In chapter 4 we showed that all bounded problems are NP-complete even when restricted to some graph classes. There is however always an efficient algorithm when the input graph has a bounded treewidth. We will prove this now by giving a dynamic programming algorithm for that case.

Let us assume a problem $GBD(P, c)$ with a power set P bounded by a parameter r and an arbitrary cost function c. W.l.o.g. we assume in the following a continuous power set, i.e., each power up to r is allowed. (If one is not allowed we could simply allow it, giving it the same cost as the next higher allowed power.) Also w.l.o.g we assume a monotonically increasing cost function.

We are now going to define the central concept of "induced powers" and will thereafter present an algorithm to solve such a problem on trees and graphs with bounded treewidth.

5.1 Induced power

First we need the concept of induced powers. Whenever a vertex v is given power p, it dominates each vertex w for which there is a path from v to w of length at most p. We call such a path a *domination path*. Each vertex x on such a path can be seen as having an *induced power* of $p' := p - d(v, x)$, since the power of vertex v *guarantees* the p'-neighborhood of x to be dominated. We call v a *source* of the induced power.

In Figure 5.1 a power of five on vertex A will e.g. result in an induced power of one in vertex E. An induced power of one in vertex E, could come from a power five vertex A as well as from a power three vertex C. Only knowing the induced

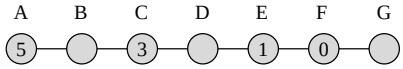

Figure 5.1: If vertex A has power five, vertex C will have an induced power of three, E an induced power of one, and F an induced power of zero.

power of a vertex therefore does not allow us to say exactly which vertex has induced it, but we can be sure that the induced power p (like a real power) will dominate all the p-neighborhood. Knowing in Figure 5.1 that vertex E has an (induced) power of one, we can guarantee vertex F to be dominated while we cannot for vertex G. Note that a source of an induced power can also be the induced power of another vertex. This is why we can think of A inducing C and C inducing E. Hence an induced power has usually many sources and one of these is always a neighbor vertex.

5.2 An algorithm for trees

Trees are an especially simple case of a graph with bounded treewidth. Understanding the algorithm for trees will help to understand the algorithm for general graphs with bounded treewidth.

5.2.1 Notation

We want to solve problem $GBD(P, c)$ with power set P bounded by a constant k.

We are given a rooted tree $T = (V, E)$ with root r and want to compute the minimum cost to dominate the whole tree. We use a dynamic programming algorithm to compute for each vertex $v \in V$ the cost A_v of the subtree T_v rooted at v.

The problem is, that vertices inside a subtree can dominate vertices outside the subtree and vice versa (See Figure 5.2). To deal with this problem, we first see, that each (domination) path between a vertex in the subtree T_v and a vertex not in that subtree has to contain the vertex v. Therefore the dominating vertex

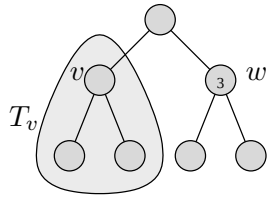

Figure 5.2: A small tree with a vertex v and the corresponding subtree T_v. Giving vertex w a power of three will induce power one in vertex v.

will induce a power between 0 and r in vertex v. Hence we can distinguish $2r$ different cases which we will denote as $\mathcal{C} = \{-r, \ldots, -0, +0 \ldots, +r\}$.

On the one hand it may be that in the solution vertex v is already dominated from a vertex outside the considered subtree T_v. In this case it is induced a power x between 0 and r[1] and we do not have to dominate the x-neighborhood of v, since it is already dominated by the induced power of vertex v. We denote this case as $-x$.

If, on the other hand, vertex v is not dominated from outside the subtree, then this vertex has to be dominated by a vertex inside the subtree. This vertex will again induce a power x between 0 and r in vertex v. It will this time dominate vertices outside the subtree which are within distance x of v. This case is denoted by $+x$.

To simplify the presentation we will also define an ordering on the cases of $\mathcal{C}$:

$$-r < -(r-1) < \cdots < -1 < -0 < +0 < +1 < \cdots < +(r-1) < +r$$

In our algorithm we will compute $A_v(c)$ for each vertex v and each case $c \in \mathcal{C}$.

$$A_v(-x) = \min\{c(D,p) \mid (D,p) \text{ dominates } V(T_v) \setminus N_x(v)\}$$
$$A_v(+x) = \min\{c(D,p) \mid (D,p) \text{ dominates } T_v \text{ and induces power } x \text{ in } v\}$$

Thereby we can observe a monotonicity in A_v. Clearly the cost $A_v(-x)$ cannot be greater than $A_v(-y)$ for any $y \leq x$, since the vertices to be dominated in

[1] A power of r actually cannot be induced from outside, we nevertheless compute this value, since we can use this value in the algorithm.

$A_v(-x)$ form a subset of those in $A_v(-y)$. An example is given in Figure 5.3. For the same reason $A_v(+y)$ cannot be greater than $A_v(+x)$ for any $y \leq x$. It should also be clear that always $A_v(-0) \leq A_v(+0)$, since we have to dominate one vertex more in the $+0$ case, namely v. Hence we can conclude:

$$A_v(c_1) \leq A_v(c_2) \text{ for all } c_1 \leq c_2$$

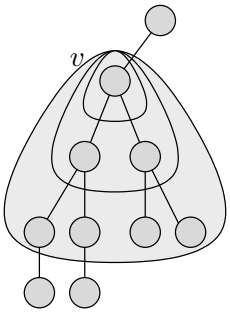

Figure 5.3: A part of a tree. If vertex v is induced power zero, one, or two from vertices outside the subtree T_v the marked vertices are already dominated and the remaining vertices for a larger induced power are clearly a subset of the remaining vertices of a smaller induced power.

5.2.2 Algorithm

With this notation we can describe the algorithm. We use a bottom up approach to compute A_v for each tree vertex v, i.e., we first compute A_v for the leaves and further process a vertex v for which all the child vertices have already been processed.

Leaves

Computing A_l is very simple for a leaf l. For the negative cases the leaf is already dominated from vertices outside the subtree and we see from the definition of $A_v(-x)$ that there are no vertices of T_v left to dominate. Therefore the cost is clearly zero.

$$A_l(-x) = 0$$

For the positive case $+x$ the subtree has to guarantee an (induced) power of x in l. Since we are considering a leaf, the subtree consists of only a single vertex and we have to give it the requested power.

$$A_l(+x) = c(x)$$

Inner vertices

After the leaves are processed, we will choose an inner vertex v, whose child vertices $ch(v)$ are already processed. Computing $A_v(+r)$ is especially easy, since the requested power cannot be induced, it can only result from vertex v being assigned power r directly. Therefore we need cost $c(r)$ for that vertex but we can guarantee the neighbors of v an induced power of $r-1$. We can now exploit the monotony and simply use $A_w(-(r-1))$ to determine the cost of the subtrees of a child w.

$$A_v(+r) = c(r) + \sum_{w \in ch(v)} A_w(-(r-1)))$$

If smaller powers are requested (case $+x$ for $0 < x < r$) there are two possibilities. The power of x can either be assigned, as in the $+r$ case, or it can be induced by some child vertex. Therefore we have to try all cases and choose the minimum cost.

$$A_v(+x) = \min\Bigg(c(x) + \sum_{w \in ch(v)} A_w(-(x-1)), \min_{w \in ch(v)} (A_w(+(x+1)) + \sum_{u \in ch(v) \setminus \{w\}} A_u(-(x-1)))\Bigg)$$

The computing for the $+0$ case works exactly in the same way, but the last term would result in a $A_w(-(-1))$ or $A_u(-(-1))$ which has to be changed to

$A_w(+0)$ or $A_u(+0)$ respectively. This is because $+0$ means, the vertex is just managed to be dominated, but does not dominate other vertices. Hence we have to make sure that the other subtrees are also dominated by themselves by requiring $+0$.

To compute the negative cases is again quiet simple. If we are about to compute $A_v(-x)$ we can be sure that v is already dominated and has an induced power of x. Therefore we can again guarantee all the neighbor vertices an induced power of $x-1$. In the -0 case we can again not guarantee the neighbors an induced power and we have to choose $+0$: $A_v(-0) = \sum_{w \in ch(v)}(A_w(+0))$ and $A_v(-x) = \sum_{w \in ch(v)}(A_w(-(x-1)))$for $1 \leq x \leq r$.

In a case as shown in Figure 5.4 where one child dominates another child the above computation could give a cost that is too high. However such a case would induce a power in vertex v. Therefore the correct value will be in the corresponding positive case. This is why we need a postprocessing, where we let $A_v(c) := A_v(c')$ if $c' > c \wedge A_v(c) := A_v(c')$. This fixes the problem and can be justified by the mentioned monotony.

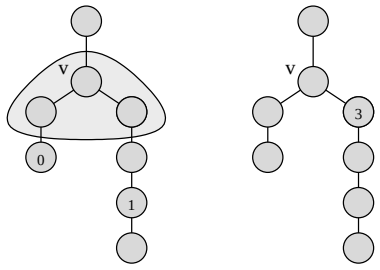

Figure 5.4: If we compute a negative case (-1) and the cost function assigns $c(0) = c(1) = c(2) = c(3) = 1$, treating the two child subtrees separately gives an suboptimal result. Treating separately as in the left case the cost is $c(0)+c(1) = 2$ while treating them together results in cost $c(3) = 1$.

Result

After all vertices are processed, the root r of the tree is also processed. The cost of the complete tree is the value of $A_r(+0)$ since the subtree T_r is the complete tree and $+0$ requires even the root to be dominated from vertices inside the tree.

5.3 An algorithm for graphs with bounded treewidth

We will now use similar concepts as in the tree algorithm to build an algorithm for general bounded treewidth graphs.

5.3.1 Preconditions

We are given an input graph $G = (V, E)$ with a treewidth of k and an associated nice tree decomposition $< X_i, T >$. We further assume a problem $GBD(P, c)$ with a power set P bounded by r, i.e., no power greater than r is allowed. The cost function c may be arbitrary, but w.l.o.g. we still assume a monotonically increasing cost function. We also assume w.l.o.g. a continuous power set, i.e., each power up to r is allowed. (If one is not allowed we could simply allow it, giving it the same cost as the next higher allowed power.) Now we are going to compute the minimum cost to dominate the whole graph.

5.3.2 The idea

To achieve this, we process the nodes in the nice tree decomposition from the leaves to the root. For each processed bag X_i we call the already processed vertices (i.e., $\bigcup\{X_j \mid j \text{ is a descendant from } i \text{ in } T\}$) the *inside* I of the graph. The remaining vertices in V ($V \setminus I$) are called the *outside* O. An example is shown in Figure 5.5.

While computing the cost of the respective inside we encounter the same problem as in the tree case. Vertices of the inside can dominate outside vertices, and vice versa, while inducing a power between 1 and r in some bag vertex. Therefore we use the same notation as in the tree case, but we have to apply it to all vertices in the bag. We are going to color the vertices in the bags by elements from the partial ordered set $\mathcal{C} = \{-r, \ldots, -0, +0, \ldots, +r\}$. We later write $+x$ to denote a positive color and $-x$ for a negative one.

The coloring of the bag is based on "guarantees". We are going to make guarantees for the induced power of a positively colored vertex, while we can use the guarantees made by the negatively colored vertices.

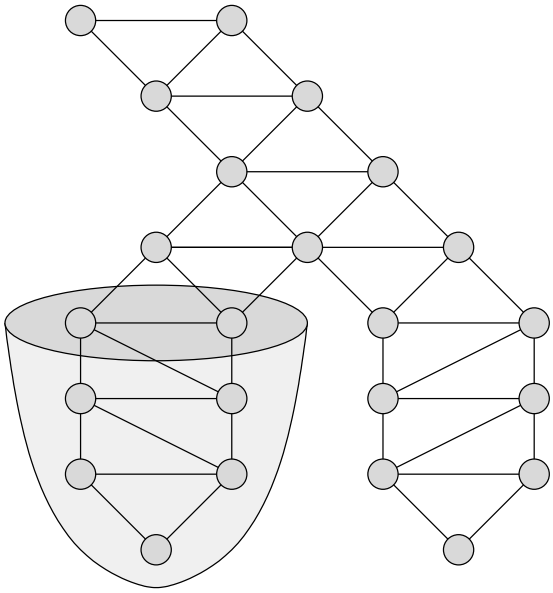

Figure 5.5: A graph with bounded treewidth. The dark vertex set represents a bag. Then the vertices below form the inside.

A color of $+x$ means that we have to guarantee that the corresponding vertex has an (induced) power of (at least) x with a source in the inside, while a color of $-x$ means we can already assume an induced power of x. (We have to justify this later.)

Therefore we can for example guarantee a vertex a power of x if it has a neighbor which is guaranteed a power of $x+1$. Now let $A_i(c_1, \ldots, c_{n_i})$ be the minimum cost of dominating the inside-graph $G[I]$ assuming the given colors.

The formal definition is a bit complex. Since we have to express the given guarantees by assigning powers which have no cost. This is done by explicitly requiring the power, but subtracting the cost afterwards:

$$\begin{aligned} PC_C = \{(D,p) \mid & (\forall_{v \in I} v \text{ is dominated }) \\ & \wedge (c_l = -x \Rightarrow v_l \in D \wedge p(v_l) = x) \\ & \wedge (c_l = +x \Rightarrow v_l \text{ has an induced power of } x)\} \end{aligned}$$

,

$$A_i(c_1, \ldots, c_{n_i}) = \min_{(D,p) \in PC_C} c(D,p) - \sum_{1 \leq j \leq n_i} \begin{cases} c(x) & \text{if } c_j = -x \\ 0 & \text{otherwise} \end{cases}$$

We further define for a bag of size of n a *color graph* C_n:

$$C_n = (\mathcal{C}^n, \{((c_1, \ldots, c_t, \ldots, c_n), (c_1, \ldots, c'_t, \ldots, c_n)) \mid c_t = c'_t + 1\})$$

That is, the vertices are all possible colorings of the bag, and an edge from one coloring to another exists iff both colorings differ only in a single position and in that position the first color has a value one greater than the second color. An example of such a color graph is given in Figure 5.6.

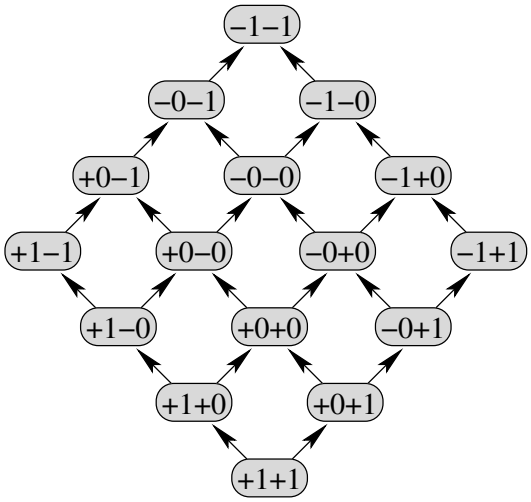

Figure 5.6: A color graph for $\{-1, -0, +0, +1\}^2$.

The same monotony as in the tree case holds for A_i here, since for every edge in the color graph the vertices to be dominated in the first coloring form a superset of the ones to be dominated in the second coloring. We will make use of this fact frequently. We can in many situations with a clear conscience consider only one coloring, since we can be sure that all other possible colorings cannot have lower costs. For example, requiring a power of four would clearly also dominate the 3-neighborhood but it obviously cannot be cheaper. We say larger colors have larger *requirements* and therefore (possibly) larger costs.

The algorithm now computes A_i for all possible colorings of all bags in the nice tree decomposition from the leaves to the root. The cost of the whole graph then is $A_r(+0, \ldots, +0)$, since the root r of the nice tree decomposition is the last one processed which means that the inside spans the whole graph, and each vertex now is dominated.

5.3.3 The algorithm

The main algorithm goes as following:

Algorithm 1.
forall nodes i in T (from leaves to root) do
    forall colorings C do
        compute $A_i(C)$
    od
od
output $A_r(+0, \ldots, +0)$

We now compute the A_i depending on the type of node:

LEAVES

W.l.o.g. we assume that leave-bags consist of only one vertex.

For negative colors we can assume the vertex to be dominated, therefore we have no cost for now: $A_i(-x) = 0$.

Now we want to assign a positive color. Since this is the only vertex (and also the only vertex in the inside) we have to guarantee the required power by assigning power x to the vertex, resulting in cost $c(x)$: $A_i(+x) = c(x)$.

INTRODUCE

We are now going to process the introduce node i with child j, whereas $X_i = \{v_1, \ldots, v_{n_j}, v\}$ and $X_j = \{v_1, \ldots, v_{n_j}\}$. Since we process T from the leaves to the root, the child node is already processed and therefore A_j already known.

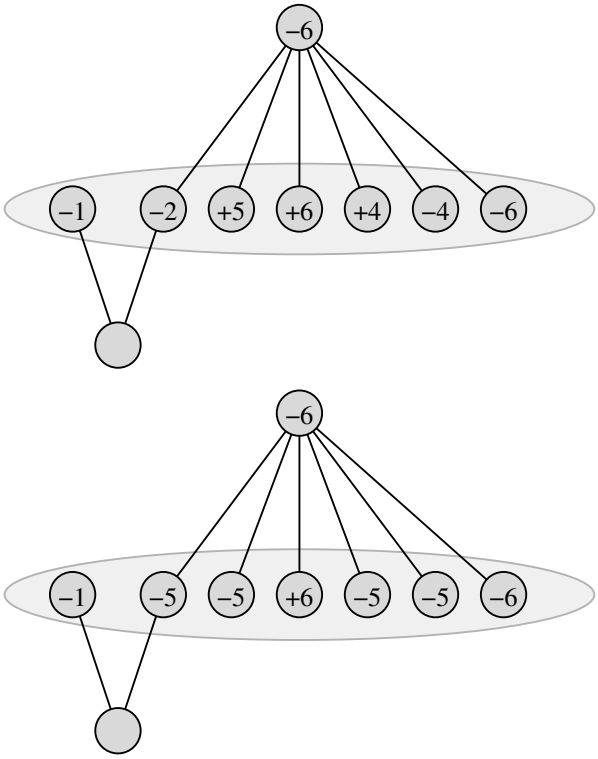

Figure 5.7: The marked vertices are the bag X_j. The upper vertex is to be introduced. When computing the cost for the first coloring, we lookup the second coloring.

Introduce nodes are the most complex ones, almost all of the work has to be done here. We will first distinguish between positive and negative colors of the introduced vertex v.

Negative colors

A negative color $-x$ of a vertex guarantees the vertex to be dominated and an (induced) power of x. Obviously adding a negative color vertex to the bag cannot raise the cost, but it can sometimes lower it, since the new vertex is capable of inducing powers in the already processed vertices.

Since the introduced vertex v is guaranteed to have power x, we can guarantee all the neighbors of v a power of $x - 1$, as they will be induced

by v. Therefore we have to find an appropriate coloring C' which we will use to find the correct value in A_j and can therefore color the neighbor vertices of v with color $-x$, unless they have higher requirements in the coloring C to be computed.

Figure 5.7 shows an example. The first picture shows the coloring C for X_i to be computed, the second one shows the coloring C' of X_j which we will lookup. We can replace -4 by -5 since we can guarantee a power of five and a power of five will dominate at least the same vertices as a power of four. We can replace color $+5$ by -5, because the added vertex will guarantee this vertex the needed power of five. The colors $+6$ and -6 have to be preserved, since they cannot be justified by the new vertex. The new vertex can also induce powers to bag vertices, which are no direct neighbors, for example the very left vertex in Figure 5.7. We do not have to care about them, since they will also be induced by a neighbor bag vertices, which was handled while computing the child node.

Therefore we compute the new values as follows:

$$A_i(c_1, \ldots, c_{n_j}, -x) = A_j(c_1^x, \ldots, c_{n_j}^x)$$

$$c_a^x := \begin{cases} -(x-1) & \text{if } |c_a| \leq (x-1), \\ c_a & \textit{otherwise.} \end{cases}$$

Positive colors

Now assume the case, that we want to give the new vertex v a positive color $+x$, i.e., we want to compute $A_i(c_1, \ldots, c_{n_j}, +x)$. A color $+x$ means, that we have to guarantee the vertex an (induced) power of x.

The power x of vertex v can either be *induced* by another vertex $w \in I$, or it must be *assigned* to v. If it is already induced, we are done, since we know, that the cost is the same as without the new vertex. If that power is not already induced, we have to assign the required power to v. Assigning that power will result in a higher cost, but it will also induce other vertices, as in the case of the negative colors.

Now the problem is to decide, whether the power is already induced or must be assigned. To decide this, we use another observation: Whenever a vertex $w \in I$ induces a power of x in v, it will also induce power $x+1$ to a neighbor of v. (See Figure 5.8.)

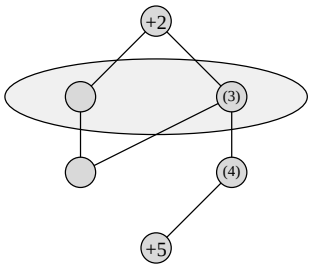

Figure 5.8: If the power five vertex induces a power of two to the introduced vertex v, there has to be a neighbor of v with induced power three.

Therefore we can check, whether the power is induced by recoloring the neighbor vertices of v: If a neighbor vertex $u \in N(v) \cap I$ is induced a power of $x+1$ anyway, we can color this vertex $+(x+1)$ and the cost A_j will not change. If the cost changes we can be sure, power $x+1$ was not induced in u.

Now we try this for every neighbor $u \in N(v) \cap I$. If we find a neighbor which does not change the cost, the power is induced. If the power changes for each neighbor, the power is not induced and we have to assign the power.

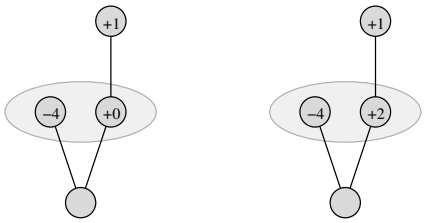

Figure 5.9: A part of a graph with two different colorings. The marked area is the current bag the upper vertex is to be introduced.

Figure 5.9 gives another example. Since the left bag vertex is guaranteed a power of four, the right bag vertex is guaranteed to have an induced power of two, (and therefore the power of one is guaranteed to v,) no matter what color the right bag vertex is. If the coloring only requires

the right bag vertex a power of zero through a color of +0, we cannot decide, whether the required power of v is induced by simply examine the colors. We have to check the cost. If we recolor the neighbor vertex as in the right part of the example, we will find the same cost and therefore know, that the power is already induced.

Formally we can say $+x$ is induced iff

$$\exists v_t \in X_j : A_j(c_1 \dots c_{n_j}) = A_j(c_1, \dots, c_{t-1}, +(x+1), c_{t+1}, \dots, c_{n_j}).$$

.

Now we can compute the cost as follows:

$$A_i(c_1, \dots, c_{n_j}, +x) = \begin{cases} A_j(c_1 \dots c_{n_j}) & \text{if } +x \text{ is induced,} \\ c(x) + A_i(c_1, \dots, c_{n_j}, -x) & \text{otherwise.} \end{cases}$$

FORGET

Now we are going to process a forget node i ($X_i = \{v_1, \dots, v_{n_i}\}$) with its child node j ($X_j = \{v_1, \dots, v_{n_i}, v\}$).

Since the vertex v should be forgotten, we must make sure it is dominated, but we do not have to require it to dominate any neighbor vertices. This is exactly the situation that corresponds to +0.

Therefore we do not need to compute any minimum and can simply write:

$$A_i(c_1, \dots, c_{n_i}) = A_j(c_1, \dots c_{n_i}, +0).$$

JOIN Now i is a join node with the two children j_1 and j_2. ($X_i = X_{j_1} = X_{j_2} = \{v_1, \dots, v_{n_i}\}$)

While computing A_{j_1} and A_{j_2} we computed the cost of the respective inside, which can be seen as a single *branch* of the graph. The join node now joins those two branches to a single branch, as shown in Figure 5.10.

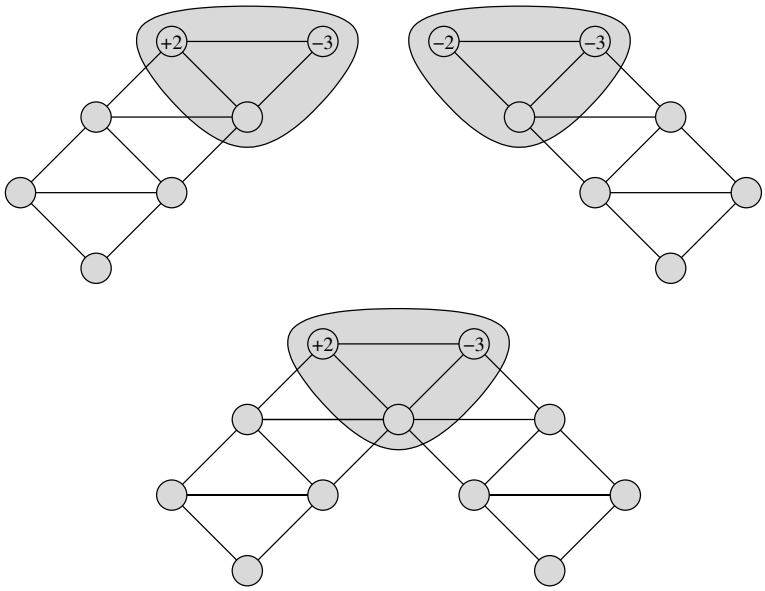

Figure 5.10: Two branches of the graph will be joined to a single branch. The marked vertices are the bags, which consist of exactly the same vertices. If a vertex is dominated from outside (-3) both branches are. If a vertex has to dominate the outside (+2), one branch has to do the domination, while the other is dominated by the first one (-2).

This means we have to add the costs of the two branches in such a way to assure the required colors.

We will do this by finding "dividing" colorings $C' = c'_1, \ldots, c'_{n_i}$ and $C'' = c''_1, \ldots, c''_{n_i}$ for the coloring $C = c_1, \ldots, c_{n_i}$ to be computed. After that we simply need to add the appropriate costs and take the minimum.

$$A_i(C) = \min_{C', C'' \text{dividing } C} (A_{j_1}(C') + A_{j_2}(C''))$$

$$C', C'' \text{ divide } C \text{ iff}: \begin{cases} c_t = -x & \Rightarrow c'_t = c''_t = c_t \\ c_t = +x & \Rightarrow (c'_t = +x \wedge c''_t = -x) \vee (c'_t = -x \wedge c''_t = +x) \end{cases}$$

We made sure that dividing colors assure the requested values in coloring

C and took always the lowest possible requirements, which has to be optimal because of the effective monotony.

If a vertex is dominated from outside, both branches are dominated. Therefore we can guarantee this power to both branches. If the vertex has to dominate the outside, one branch has to do the domination. The other one then is dominated by the first one.

5.3.4 Time complexity

The computation for the forget node was as follows:

$$A_i(c_1, \ldots, c_{n_i}) = A_j(c_1, \ldots c_{n_i}, +0)$$

This can obviously be processed in $O(1)$ time.

The negative colors of the introduce node was computed as follows:

$$A_i(c_1, \ldots, c_{n_j}, -x) = A_j(c_1^x, \ldots, c_{n_j}^x)$$
$$c_a^x := \begin{cases} -(x-1) & \text{if } |c_a| \leq (x-1) \\ c_a & otherwise \end{cases}$$

Since we have to test each color in the bag and we assumed a treewidth of k this can be done in $O(k)$ time.

For the positive colors of the introduce node we have to decide, whether the required power is already induced. This is done by testing

$$\exists v_t \in X_j : A_j(c_1 \ldots c_{n_j}) = A_j(c_1, \ldots, c_{t-1}, +(x+1), c_{t+1}, \ldots, c_{n_j})$$

which will also work in $O(k)$ time.

For the join nodes, A_i is computed as

$$A_i(C) = \min_{C', C'' \text{dividing } C} (A_{j_1}(C') + A_{j_2}(C'')).$$

Therefore the running time in this case is determined by the number of values over which the minimum is taken.

Here is again the definition of "divide":

$$C', C'' \text{ divide } C \text{ iff}: \begin{cases} c_t = -x & \Rightarrow c'_t = c''_t = c_t \\ c_t = +x & \Rightarrow (c'_t = +x \wedge c''_t = -x) \vee (c'_t = -x \wedge c''_t = +x). \end{cases}$$

We see, that the resulting number of pairs mainly depends on the number of positive colors in the coloring. Having z positive colors in the coloring will result in 2^z different cases.

There are $\binom{n_i}{z}$ places for the positive values and, since the powers are bounded by r, there are both $r+1$ positive and $r+1$ negative colors. Hence there are $\binom{n_i}{z}(r+1)^{n_i}$ different colorings with z positive colors.

Taking the sum over all possible colorings results in:

$$\sum_{z=0}^{n_i} 2^z \binom{n_i}{z} (r+1)^{n_i} = (3r+3)^{n_i}$$

The computation for the join nodes is the most complex one. Since the input graph has treewidth k the bag size cannot be greater than $k+1$. Hence, the time complexity on a nice tree decomposition with N nodes is $O((3r+3)^{k+1}N)$.

Because of this algorithm we can conclude:

Theorem 5. *Each problem $GBD(P, c)$ with P bounded by a constant r can be solved on a graph with treewidth k in time $O((3r+3)^{k+1}N)$.*

6 Problems with unbounded power set

We already showed in this thesis that all problems $GBD(P, c)$ with a bounded power set P are NP-complete on general graphs. This changes if arbitrary large powers are allowed. For many such problems there is an efficient algorithm, while for many others the complexity is still unknown.

The crucial difference lies in the fact, that arbitrarily large powers in many cases allow us to replace two dominating vertices by a single one with correspondingly higher power. This can sometimes greatly simplify the structure of a solution, which makes such a solution much easier to find.

We are going to examine different problems $GBD(P, c)$ and will sometimes refer to the special cases Broadcast Domination and General Broadcast Domination. The definitions of those problems are again given in Table 6.1.

Problem	P	$c(p), p \in P$
Broadcast Domination	$\{1, 2, \dots\}$	p
Modified Broadcast Domination	$\{0, 1, 2, \dots, r\}$	$p + 1$

Table 6.1: The definitions of Broadcast Domination and Modified Broadcast Domination.

6.1 Cases with an efficient solution

We call a partial domination (D, p) *efficient* if every vertex v in G is dominated by exactly one dominating vertex $d \in D$.

For an efficient domination (D, p) we will similar to [16] define the domination

graph $G_{D,p} = (D, \{\{u, v\} \mid N_{p(u)+1}(u) \cap N_{p(v}(v) \neq \emptyset\})$. That is a graph consisting of the dominating vertices, which are connected, if the vertex sets they dominate in G are connected by some edge. An example is given in Figure 6.1.

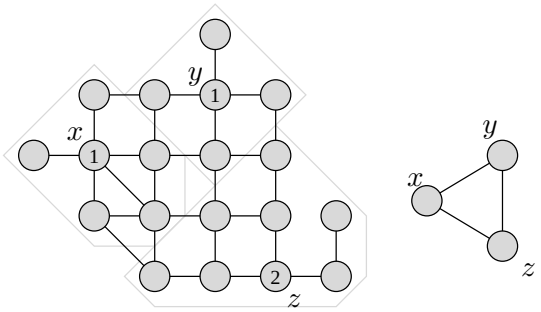

Figure 6.1: A graph with an efficient domination and the resulting domination graph.

Lemma 3. *For a problem $GBD(P, c)$ with $P = \mathbb{N}$ and a cost function that holds $c(a) + c(b) \geq c(a + b)$, there is always an efficient solution.*

Proof. Assume we have a solution (D, p) which is not efficient. Then there is a vertex v which is dominated by two dominating vertices u and w, as in the example in Figure 6.2. Since they both dominate vertex v, there is a path between u and w with length at most $p(u) + p(w)$. Now let v' be the vertex which is on this path with distance $p(w)$ from vertex u. This vertex has distance $p(w)$ from u and a distance of at most $p(u)$ from w. Assigning this vertex a power of $p(u) + p(w)$ will therefore induce the original powers in u and w. We can therefore replace u and w by the single domination vertex v', which will, because of the required restriction on the cost function, result in no higher cost than the original solution. □

Obviously the restriction holds for both Broadcast Domination and Modified Broadcast Domination. Therefore we know there is always an efficient solution for those problems. But the structure of such a solution can be proved to be even simpler, as shown next.

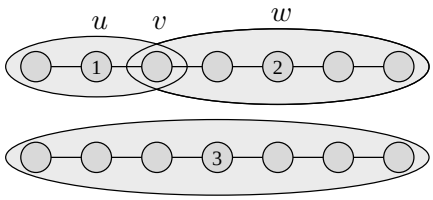

Figure 6.2: A vertex is dominated by two different dominating vertices. We can dominate all the vertices by a single by a single dominating vertex.

6.2 Cases with a radial solution

If we require stronger restrictions we can even more simplify the resulting structure of the solution.

Lemma 4. *If $c(a) + c(b) \geq c(a + b) + 1$ there is always a radial solution, i.e., as solution consisting of a single dominating vertex.*

Proof. Since the required restriction is stronger than the restriction in Lemma 3, there is always a efficient solution. Now assume two dominating vertices u and w which are connected in the domination graph. Figure 4 shows an example of this case. We can now temporarily raise the power of vertex u by one. Then it will dominate a vertex which is already dominated by w and we can therefore replace it by a single domination vertex of power $p(u) + p(w)$ as shown in the proof of Lemma 3. The total cost will now result in $c(p(u) + p(w))$ which will by the required restriction have no higher cost than $c(p(u)) + c(p(w))$ which was the original cost. □

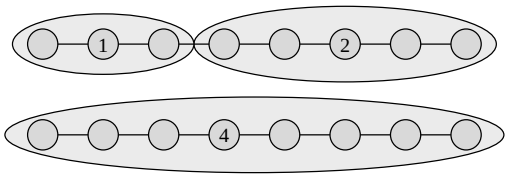

Figure 6.3: Two domination vertices can be replaced by a single one if the cost function allows it.

The restriction of Lemma 4 holds for example for Modified Broadcast

DOMINATION. Therefore all we need to solve this problem is finding a central vertex in G and assigning it power $rad(G)$.

6.3 Cases with a linear solution

With a problem like BROADCAST DOMINATION, the requirements of Lemma 4 are not fulfilled and there is generally no radial solution. But even with this problem we can find a simpler structure. Whenever the domination graph has a vertex with degree of at least three, we can replace the effected dominating vertices by a single domination vertex. (See Figure 6.4.)

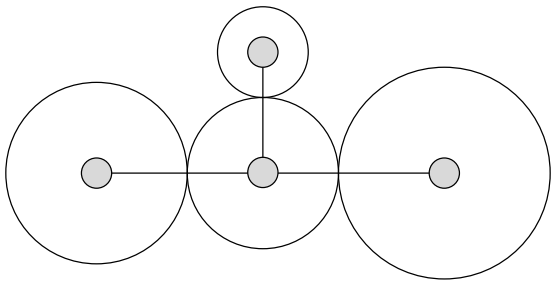

Figure 6.4: A domination graph with the dominated vertices adumbrated by the circles. If the domination graph has a vertex of degree at least three, we can again replace the effected vertices by a single one.

The resulting domination graph will therefore consist only of vertices of degree at most two, which can only be a path or a circle. Such a solution can be found in polynomial time. Details about the replacement and the algorithm can be found in [16].

6.4 Cases with unknown complexity

As shown above in some cases there is an algorithm to solve a problem $GBD(P, c)$ efficiently. But to other cases the above reasoning can not be applied.

This can either be due to the power set or the cost function. If for example only

square numbers are allowed in the power set, we generally cannot substitute the domination vertices with the required power and would need to use a higher power which again will result in a higher cost.

The problem can also be the cost function. Even if we allow arbitrary powers, we cannot easily substitute dominating vertices if the cost function does not meet the needed requirements. An example are functions like $c(x) = x^2$ or even $c(x) = x - 1$.

The complexity of those problems is still unknown and will be subject of further research.

7 Conclusion

In this thesis we examined a couple of problems in the domain of General Broadcast Domination.

The complexity is well understood for problems with a bounded power set. On planar, chordal, and bipartite graphs these problems are NP-complete. On graphs with bounded treewidth they can be solved efficiently. On still other graph classes, the complexity depends on the exact parameters of the problem. For example on split graphs Dominating Set and Roman Domination are NP-complete, but there is a trivial solution if a power of two is allowed.

One direction for research could be to examine other graph classes or to study those problems from the viewpoint of approximation or parameterized complexity. A parameterized approach for Roman Domination has recently been presented in [11].

Problems with an unbounded power set seem to be much more irregular. Some problems are NP-complete, while others have trivial solutions. Even problems, which only slightly differ in the cost function turn out to have quite different complexity. For example a cost function $c(x) = x + 1$ results in a trivial solution, with a cost function $c(x) = x$ we need a sophisticated algorithm and the complexity with a cost function of $c(x) = x - 1$ is still unknown.

Recently, Heggernes and Lokshtanov [16] have reported some progress in this area, but many questions are still open. Even the complexity of some practically motivated variants like a cost function of $c(x) = x^2$ is still unknown.

Further research should try to find a stronger criterion to recognize NP-complete cases and to develop new techniques to solve other cases.

Bibliography

[1] Jochen Alber. *Exact Algorithms for NP-hard Problems on Networks: Design, Analysis, and Implementation.* PhD thesis, Universität Tübingen, 2002. 16

[2] Giorgio Ausiello, Pierluigi Crescenzi, Giorgio Gambosi, Viggo Kann, Alberto Marchetti-Spaccamela, and Marco Protasi. *Complexity and Approximation: Combinatorial Optimization Problems and Their Approximability.* Springer, 1999. 16

[3] Hans L. Bodlaender. A partial k-arboretum of graphs with bounded treewidth. *Theor. Comput. Sci.*, 209(1-2):1–45, 1998. 16

[4] Andreas Brandstädt. Information system on graph class inclusions, 2006. http://wwwteo.informatik.uni-rostock.de/isgci/. 13, 24

[5] Andreas Brandstädt, Van Bang Le, and Jeremy P. Spinrad. *Graph Classes: a Survey.* Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 1999. 8, 13

[6] Ernest J. Cockayne, Paul A. Dreyer Jr., Sandra Mitchell Hedetniemi, and Stephen T. Hedetniemi. Roman domination in graphs. *Discrete Mathematics*, 278(1-3):11–22, 2004. 24

[7] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms, Second Edition.* The MIT Press and McGraw-Hill, 2001. 11

[8] Reinhard Diestel. *Graph Theory.* Springer, 2005. 8, 11

[9] R. G. Downey and M. R. Fellows. *Parameterized Complexity*. Springer, 1999. 16

[10] Uriel Feige. A threshold of ln n for approximating set cover. *J. ACM*, 45(4):634–652, 1998. 27

[11] Henning Fernau. Roman domination: A parameterized perspective. In Jirí Wiedermann, Gerard Tel, Jaroslav Pokorný, Mária Bieliková, and Julius Stuller, editors, *SOFSEM*, volume 3831 of *Lecture Notes in Computer Science*, pages 262–271. Springer, 2006. 18, 24, 54

[12] M.R. Garey and D.S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, 1979. 8, 11, 24

[13] Jiong Guo, Rolf Niedermeier, and Daniel Raible. Improved algorithms and complexity results for power domination in graphs. In Maciej Liskiewicz and Rüdiger Reischuk, editors, *FCT*, volume 3623 of *Lecture Notes in Computer Science*, pages 172–184. Springer, 2005. 17

[14] Teresa W. Haynes, Stephen T. Hedetniemi, and Peter J. Slater, editors. *Domination in Graphs: Advanced Topics*, volume 209 of *Pure and Applied Mathematics*. Marcel Dekker, 1998. 17

[15] Teresa W. Haynes, Stephen T. Hedetniemi, and Peter J. Slater. *Fundamentals of Domination in Graphs*, volume 208 of *Pure and Applied Mathematics*. Marcel Dekker, 1998. 17

[16] Pinar Heggernes and Daniel Lokshtanov. Optimal broadcast domination of arbitrary graphs in polynomial time. In Kratsch [18], pages 187–198. 17, 49, 52, 54

[17] Ton Kloks. *Treewidth, Computations and Approximations*, volume 842 of *Lecture Notes in Computer Science*. Springer, 1994. 16

[18] Dieter Kratsch, editor. *Graph-Theoretic Concepts in Computer Science, 31st International Workshop, WG 2005, Metz, France, June 23-25, 2005, Revised Selected Papers*, volume 3787 of *Lecture Notes in Computer Science*. Springer, 2005. 56

[19] Mathieu Liedloff, Ton Kloks, Jiping Liu, and Sheng-Lung Peng. Roman domination over some graph classes. In Kratsch [18], pages 103–114. 24

[20] Rolf Niedermeier. *Invitation to Fixed-Parameter Algorithms*. Oxford University Press, 2006. 16

[21] Neil Robertson and Paul D. Seymour. Graph minors. ii. algorithmic aspects of tree-width. *J. Algorithms*, 7(3):309–322, 1986. 16

[22] Vijay V. Vazirani. *Approximation Algorithms*. Springer, 2001. 16

[23] Jie Wu and Hailan Li. On calculating connected dominating set for efficient routing in ad hoc wireless networks. In *DIAL-M*, pages 7–14. ACM, 1999. 17

Danksagung

Ich danke meinen Betreuern Jiong Guo und Rolf Niedermeier für die mir gegenüber aufgebrachte Geduld und die vielen Tipps und Hinweise, die mir bei der Erstellung dieser Arbeit sehr geholfen haben.

Außerdem danke ich meiner Mutter, Günter und meinen Großeltern, die mich während des Studiums und insbesondere während der Diplomarbeit auf jede mögliche Art unterstützten.

Selbständigkeitserklärung

Ich erkläre, dass ich die vorliegende Arbeit selbständig und nur unter Verwendung der angegebenen Quellen und Hilfsmittel angefertigt habe.

_____

Jena, den 28.04.2006 (Michael Schnupp)