STRUCTURAL PARAMETERIZATIONS OF HARD GRAPH PROBLEMS

by

Michail Lampis

A dissertation submitted to the Graduate Faculty in Computer Science
in partial fulfillment of the requirements for the degree of
Doctor of Philosophy, The City University of New York

2011

This manuscript has been read and accepted for the Graduate Faculty in Computer Science in satisfaction of the dissertation requirements for the degree of Doctor of Philosophy.

Amotz Bar-Noy

Date | Chair of Examining Committee

Theodore Brown

Date | Executive Officer

Stathis Zachos

Noson S. Yanofsky

Gregory Gutin

Supervisory Committee

THE CITY UNIVERSITY OF NEW YORK

ABSTRACT

Structural Parameterizations of Hard Graph Problems

BY

Michail Lampis

Adviser: Amotz Bar-Noy

In traditional computational complexity we measure algorithm running times as functions of one variable, the size of the input. Though in this setting our goal is usually to design polynomial-time algorithms, most interesting graph problems are unfortunately believed to require exponential time.

Parameterized complexity theory refines this by introducing a second variable, called the parameter, which is supposed to quantify the "hardness" of each specific instance. The goal now becomes to confine the combinatorial explosion to the parameter, by designing an algorithm that runs in time polynomial in the size of the input, though inevitably exponential in the parameter. This will allow us to tackle instances where the parameter value is much more modest than the input size, which will happen often if the parameter is chosen well.

In this work we deal with parameterized versions of intractable graph problems. Our focus will be structural parameterizations, meaning that the parameters we choose will be measures which attempt to quantify the complexity of a graph. The most important such measure in the literature is treewidth, a notion which quantifies how "tree-like" a graph is. Here, we will consider parameterizations by treewidth, but also by alternative measures of graph complexity such as vertex cover, or maximum degree.

Our results move in two directions: First, we study the computational complexity of structural parameterizations of several specific problems. In this vein we show hardness results for Maximum Path Coloring parameterized by measures such as treewidth and maximum degree; we present algorithms for two parameterizations of Vertex Cover; and we present algorithms and hardness results for three different parameterizations of the modal satisfiability problem.

Second, we study the algorithmic properties of structural parameters not with respect to a specific problem, but with respect to whole classes of problems. For undirected graphs we present algorithmic meta-theorems which show that large classes of problems are tractable when parameterized by vertex cover or max-leaf number. Our results strengthen a famous theorem due to Courcelle, for these special cases. For directed graphs we give hardness results for Hamiltonian Circuit and Max Di Cut which apply to essentially all the definitions of directed treewidth which have appeared in the literature, showing that none of them is as algorithmically potent as (undirected) treewidth.

Acknowledgments

I would like to thank my advisor Amotz Bar-Noy for being a great mentor – and friend – and for the good advice he always had to offer me on all matters, whether related to research or not. I would also like to thank the other two members of my committee from CUNY, Stathis Zachos and Noson Yanofsky, as well as Gregory Gutin, who in addition to serving as my committee's outside member was kind enough to host me on two occasions in London – these visits helped me a great deal in my research.

I would also like to thank the other students (some of them no longer students) I collaborated with in the last four years: Antonis Achilleos, Panagiotis Cheilaris, Georgia Kaouri, Eun-Jung Kim.

I am also greatly indebted to Michael Fellows and Fran Rosamond for their mentorship and friendship. My meeting with Michael in Australia in 2008, including our discussions over bottles of wine, was one of the most important reasons I decided to work on parameterized complexity.

Above all, I would like to thank the one person who helped make all this not only possible, but also meaningful and worth the effort. It's hard to find the proper words to thank my classmate, co-author, and best friend, the most important person in my life, so I will simply say this: I hope that soon I will be reading the acknowledgments of your own thesis, Valia!

Contents

List of Figures

Chapter 1

Introduction

1.1 Motivation and Background

In traditional computational complexity, the running time of an algorithm solving a problem is measured as a function of the input size, and algorithms that run in polynomial time are considered efficient. However, the vast majority of interesting graph problems are intractable, that is, the best algorithms that have been found for them, unfortunately, run in time exponential in the input size, and are thus impractical. Even worse, it is now known that this cannot be improved, unless widely believed conjectures are disproved, because most graph problems are known to be NP-hard.

Parameterized complexity theory offers a different angle to this story. Here, the running time of an algorithm is measured as a function of two variables: the input size, and a second variable called the parameter. The parameter is meant to capture a given instance's "hardness" and to let us handle exactly the aspect of the problem that makes it intractable. The goal here is an algorithm that runs in time polynomial in the input size, though (inevitably) exponential in the parameter. From a practical point of view, this is much more useful than an exponential-time algorithm because

it will allow us to solve even large instances if the parameter value is moderate. If the parameter is chosen wisely, it will have moderate values in a large class of practically interesting instances.

What exactly is the parameter, though, and how can one choose it wisely? We must define a parameter value for every instance of the problem while trying to satisfy two conflicting goals: First, we would like the parameter to offer us useful algorithmic footholds into the problem, or more precisely, we would like to be able to prove that instances of the problem where the parameter has moderate value are tractable. We often say here that we want to "confine the combinatorial explosion" to the parameter, by obtaining an algorithm that runs in time exponential *only* in the value of the parameter, but polynomial in the input size. Second, we would like as many interesting instances as possible to actually have a low value for the selected parameter, to make our parameterized algorithm as useful as possible.

To make this a little more concrete, consider a problem such as GRAPH COLORING. One could consider the number of available colors as the parameter. This would satisfy our second goal, since in a lot of practical situations we need to color large graphs with a relatively small number of colors. However, it would fail the first requirement since asking whether a graph is colorable with a small number of colors (say, three) is still a hard problem. On the other hand, one could consider the number of vertices in the graph as the parameter. This would trivially satisfy the first requirement, since in a relatively small graph it's possible to brute force the problem, but not the second since the case we are generally interested in is large graphs.

Two main trends have dominated the literature on parameterizations of optimization problems on graphs. The first approach, which we might call natural parameterization, consists simply of considering the target value of the objective function as the parameter. For example, consider the VERTEX COVER problem parameterized by the size of the vertex cover we seek to find (similar parameterizations of course exist

for other such problems such as DOMINATING SET, CLIQUE, etc.). This parameterization has the potential to satisfy our second requirement, since in many practical instances we may be interested in finding a vertex cover of size much smaller than the size of the graph. The question then becomes whether we can use the moderate target value of the objective function to significant algorithmic advantage, thus also satisfying the first.

The other main trend has been to consider some measure of the input graph's complexity as the parameter, regardless of the specific problem we are trying to solve. In this work, we will call such parameterizations, which do not involve the objective function, *structural* parameterizations. By far the most important achievement of this area has been the development of the algorithmic theory of treewidth and its related graph widths. Informally, treewidth is a measure that quantifies how "tree-like" a graph is, with graphs which are closer to having a tree-like structure having lower treewidth. The reason treewidth is considered such a successful notion of graph complexity is that, to a large extent, it achieves both of our basic goals: the class of graphs of moderate treewidth is very large (and includes many well-known classes such as outerplanar and series-parallel graphs) while at the same time a large number of normally hard problems have been found to be tractable on graphs of low treewidth.

Though treewidth occupies a rare sweet spot, combining graph generality and algorithmic amenability, many other related widths have been proposed, each with its own strengths and drawbacks (e.g. clique-width, pathwidth, etc.). More recently, research has focused on *alternative* structural parameterizations, which, rather than inventing graph widths to quantify the complexity of the input graph, rely on well-known graph invariants, such as the size of a graph's minimum vertex cover, or the number of vertices that need to be deleted to make it bipartite. This work has opened the door to what could potentially become a huge research area in the future.

One of the most important advantages of structural parameterizations is the fact

that, quite often, the algorithmic ideas developed in the study of a given parameter, such as treewidth or vertex cover, can be effectively reused to attack different problems again and again. This is again exemplified by the case of treewidth, where the basic ideas of using dynamic programming on graphs of bounded treewidth can be applied to attack a vast array of graph problems.

Thus, the area of structural parameterizations of graph problems naturally leads to two directions. One is the study of a given problem parmeterized by various widths or other measures. This gives an improved understanding of what makes this problem hard and also gives us better insight into the combinatorial behavior of the problem. The other is the study of the parameters themselves, or more precisely, the study of the algorithmic properties of whole families of problems when parameterized by a given width. This is often called the area of algorithmic meta-theorems.

Algorithmic meta-theorems are general statements which prove tractability for a family of problems (often defined by expressibility in some logic) when parameterized by some measure. The most famous and celebrated result of this form is Courcelle's theorem, which states that a large family of problems (those expressible in Monadic Second Order logic) is solvable in linear time on graphs of bounded treewidth. Besides Courcelle's theorem, several other results in the same spirit have appeared in the literature and it has become an interesting topic in and of itself to discover the boundaries to what kind of meta-theorems can be proved.

1.2 Contribution

In this thesis we give several new results in the area of structural parameterizations of hard problems, related to the study of both algorithmic meta-theorems and of parameterizations of specific problems. First, we will review the formal definitions of parameterized complexity and other fundamental notions, such as treewidth, in

Chapter 2. In subsequent chapters, we state and prove our results, after reviewing additional definitions and background as needed.

In Chapter 3 we work on the area of algorithmic meta-theorems for undirected graphs and we give several strengthenings of Courcelle's famous results, but for more restricted graph classes. More specifically, our motivation here is that the "hidden constants" in the running time of the algorithm implied by Courcelle's theorem are huge (towers of exponentials) and there are lower bounds showing that this probably cannot be improved. To improve the situation we study two graph parameters which are more restricted than treewidth, namely vertex cover and max-leaf number, and show meta-theorems in those cases where the running time is vastly improved. In the case of vertex cover, we also show essentially matching lower bounds.

In Chapter 4 we switch to the study of treewidth variants for directed graphs. Unlike the case of undirected graphs, where treewidth is viewed more or less as the "right" width, several competing definitions of treewidth for digraphs have appeared in the literature. Here we will show two hardness results which imply that essentially none of them can reach the same level of success that treewidth reached for undirected graphs.

Finally, in Chapter 5 we move from the study of widths to the study of specific problems structurally parameterized. We show three sets of results:

- In Section 5.1 we study the MAXIMUM PATH COLORING problem with respect to several structural parameters: the input graph's maximum degree, the input graph's treewidth and the available number of colors. We give a number of parameterized hardness results which show that, unlike the related PATH COLORING problem, none of these parameters helps much to make the problem tractable.

- In Section 5.2 we study the problem of satisfiability for the basic modal logic K. After reviewing the basics of modal logic we show a number of tractability and

hardness results for various structural parameters which attempt to quantify how complex a modal formula is.

- Finally, in Section 5.3 we study the VERTEX COVER problem, parameterized by the number of edges one has to delete to make a graph bipartite. We show how our algorithm can be used to also obtain a tractability results for VERTEX COVER parameterized above a lower bound.

Chapter 2

Definitions and Background

2.1 General Background

Throughout this work we assume a basic background in Algorithms and Complexity as well as Graph Theory (see Kleinberg and Tardos [2006], Papadimitriou [1994], West [2001]). In this section we briefly review some basic concepts and notation.

We will be dealing with decision or optimization problems, mainly on graphs. A graph G is a pair (V, E), where V is the set of vertices and E, which is the set of edges, is a set of pairs of elements from V. Unless stated otherwise the edges are unordered and thus the graphs are undirected (We will deal with directed graphs in Chapter 4). We use n to denote the number of vertices of a graph and m to denote the number of edges. A path of length p in a graph is a sequence of distinct vertices $v_0, v_1, v_2, \ldots, v_p$ such that for all $i \in \{0, \ldots, p-1\}$ we have $(v_i, v_{i+1}) \in E$. A cycle is a path where the last vertex is the same as the first. A graph is connected if there exists a path between any two of its vertices. A graph without cycles is called a forest. A connected forest is called a tree. The neighborhood of a vertex v, denoted $N(v)$ is the set $\{w \in V \mid (v, w) \in E\}$, while $N[v]$ will denote the closed neighborhood $N(v) \cup \{v\}$. The degree of a vertex v is $|N(v)|$. We denote the maximum degree

of any vertex in a graph by Δ. For a subset $V' \subseteq V$ we write $G[V']$ for the graph obtained if we delete all the vertices in $V \setminus V'$ and all edges touching them. We call this the subgraph induced by V'. If a graph G is connected but for a set of vertices S we have that $G[V \setminus S]$ is not connected we say that S is a separator. A clique is a graph which contains all possible edges, while an independent set is a graph with no edges. A graph is bipartite if its vertex set can be partitioned into two sets which induce independent sets.

It is assumed that the reader is familiar with the basics of computational complexity theory. In order to keep the presentation clear we will be informal in the description of all our algorithms, though it should be straightforward to translate the results given here to any reasonable concrete model of computation, such as Turing machines. We treat all problems as (Yes/No) decision problems, assuming in the case of optimization problems that a target value for the objective function is given in the input. Recall that P denotes the class of decision problems solvable in time polynomial in the length of the input and NP denotes the class of problems solvable in non-deterministic polynomial time. We will often rely on the (standard) assumption that P$\neq$NP. In fact, we will sometimes prove results based on the stronger assumption that there is no $2^{o(n)}$ algorithm for 3-SAT. This is often called the Exponential-Time Hypothesis (ETH) in the literature (for more on the ETH see for example Woeginger [2003]). Besides polynomial and exponential functions we will sometimes see towers of exponentials in our running times, for which the following definitions are useful: $tow(h)$ is the function inductively defined as $tow(0) = 0$ and $tow(h+1) = 2^{tow(h)}$, while $\log^* n$ is the function inductively defined as $\log^* n = 1 + \log^*(\log n)$ if $n > 1$ and $\log^* n = 0$ otherwise.

2.2 Parameterized Complexity

2.2.1 Definitions and Classes

In this section we recall some of the basic facts of parameterized complexity theory. For more information about parameterized complexity in general see the standard monographs on the subject Flum and Grohe [2006], Niedermeier [2006], Downey and Fellows [1999].

We use the standard definitions of parameterized complexity. The input in a parameterized problem is a pair (χ, k), where χ is a string encoding the actual problem instance and k is an integer representing the parameter. There is no restriction on what the additional integer k, which we are given, is supposed to be, but generally it is expected that the value of k will somehow correspond to the hardness of the given instance, with higher values indicating a harder instance. A typical example might be a graph problem where k is the maximum degree of the input graph, or an optimization problem where k is the value of the objective function we seek to achieve. Note here that it is possible to define several different parameterized versions of the same decision problem, by making different decisions about what the parameter is. We call these alternative "parameterizations" of the problem.

A Fixed-Parameter Tractable problem is a parameterized problem which can be solved by an algorithm with running time $O(f(k) \cdot |\chi|^c)$, where f is any computable function, $|\chi|$ is the length of χ and c is a constant independent of (χ, k). The class FPT is the class of all fixed-parameter tractable problems. The class XP is the class of all parameterized problems which can be solved by an algorithm running in time polynomial in $|\chi|$ (that is, polynomial when k is constant).

As a first approximation, it is helpful to think of FPT as the parameterized analogue of the class P. In other words it is usually our first goal to achieve a running time of the form $O(f(k) \cdot |\chi|^c)$. The motivation is that, if the parameterization of

the problem is appropriate, many practically important instances will have k much smaller than $|\chi|$, so such an algorithm will be useful even if f is an exponential function. On the other hand, an $O(|\chi|^k)$ algorithm, which is typical of the class XP, can be thought of as the parameterized analogue of an exponential algorithm. Such an algorithm quickly becomes practically useless even for moderate values of k if $|\chi|$ is large. Thus, investigating whether a parameterized problem is in FPT or not is usually our first order of business, just as investigating whether a decision problem is in P or not is usually the first step one takes in traditional complexity theory.

If a problem is fixed-parameter tractable then the best way to prove this fact is usually to come up with an algorithm. But if a problem is in XP, how do we prove that it is not FPT? Though XP does capture the "bad" parameterized situation which we want to avoid, it is not the right class to prove intractability results. To make this more clear, XP is usually thought of as the parameterized analogue of EXP. However, in classical complexity the basic intractability class is NP. The role of this class is played in parameterized complexity by a hierarchy of classes called the W-hierarchy.

In short, the method used in proving parameterized intractability is similar to the theory of NP-hardness. We have a collection of classes (denoted $W[1], W[2], \ldots$) for which we know some "complete" problems. This means that if a problem we know to be $W[1]$-complete turned out to be FPT, all problems in $W[1]$ would be FPT. Using these problems as starting points we can perform parameterized reductions to show that other problems are also at least as unlikely to be FPT.

To give a more precise definition, we will say that a parameterized problem A is fpt-reducible to a problem B if there exists an algorithm which given an instance (χ, k) of A

- Produces an instance (χ', k') of B and (χ, k) is a yes-instance of A if and only if (χ', k') is a yes instance of B.
- Runs in time $O(f(k)|\chi|^c)$.

- There exists a function g such that $k' = O(g(k))$.

When A is fpt-reducible to B we write $A \leq^{fpt} B$. It is not hard to see that the notion of fpt reduction preserves FPT solvability, or in other words FPT is closed under fpt reductions. The crucial fact in the above definition is that k' must be defined in terms of k only (it must be independent of $|\chi|$). Due to this fact most classical reductions are not fpt-reductions.

Just like in classical complexity theory the main starting problem here is satisfiability. More specifically, the main problem is a parameterized version of satisfiability called Weighted SAT. Here the parameter k is the weight of the satisfying assignment, that is the number of variables set to true. More formally, Weighted SAT is: given a propositional formula and an integer k decide if the formula is satisfiable by an assignment which sets exactly k variables to true.

Clearly this problem is in XP (try out all $\binom{n}{k}$ possible assignments to the n variables). However, it is not believed to be in FPT. In fact, by the results of Chen et al. [2004] we know that if there exists even an $n^{o(k)}$ algorithm for Weighted 3-SAT then there exists a sub-exponential algorithm for (unparameterized) 3-SAT. This would disprove the Exponential Time Hypothesis.

A full definition of the W-hierarchy is beyond the scope of this work. For our purposes it is sufficient to note that Weighted 3-SAT is the prototypical W[1]-complete problem while Weighted SAT is the prototypical W[2]-hard problem.

2.2.2 How to Parameterize

As mentioned, even for a given specific decision problem there can be many alternative ways to "parameterize" it, that is, to define the meaning of the parameter k. Typically, the decision problems we care about are NP-hard, meaning that we know that the class of all instances of the problem contains at least a family of "hard" instances that requires exponential time to solve (assuming standard complexity as-

sumptions, such as the ETH). However, this does not rule out the possibility that other large families of instances are actually tractable.

Our selection of the meaning of k is supposed to reflect this situation. Our work is at its most fruitful only when k is appropriately chosen so that as many practical instances of the problem as possible have moderate values of k, while all the hard instances have large k. Thus, from the practical point of view, picking a parameterization for a problem is in fact part of our attempt to solve it by sorting out the feasible instances from the hard ones.

To make this more concrete let us mention one of the standard ways of parameterizing: natural parameterizations. Suppose we want to solve an optimization problem, such as VERTEX COVER or DOMINATING SET. In these examples we set k to be simply the size of the target vertex cover or dominating set. The motivation is that in some interesting practical cases we will have k much smaller than n, the number of vertices of the input graph. Also, it is easy to see that this parameterization of both problems is in XP: one can go through all $\binom{n}{k}$ sets of k vertices. So classifying instances by the target size of the set does at least partially sort out the good from the bad. But how good is this classification? Are these parameterizations FPT?

This style of parameterization, that is, parameterization by the objective function is probably the most popular style of parameterization found in the literature. A wealth of interesting results are known in this field. For our purposes it is important to note that DOMINATING SET is W[2]-hard in this parameterization while INDEPENDENT SET and CLIQUE are W[1]-hard. VERTEX COVER on the other hand is FPT.

Nevertheless, besides the natural parameterization which corresponds to every optimization problem, numerous others can be defined. A parameterization of an optimization problem where the parameter is independent of the objective function will be called structural. The subject of this work is such parameterizations of graph

problems. We've already mentioned one possibility in this direction (parameterization by maximum degree), but as we will see many other interesting possibilities exist.

Let us also mention here, that in many cases in the literature a problem is considered with two or more parameters at the same time. For example, taking the ideas we have discussed one step further, one may consider DOMINATING SET parameterized by *both* Δ and the target size of the set. Formally, we consider the problem as having been parameterized by the *sum* of the two values. More parameters can be added similarly.

2.3 Structural Parameters and Graph Widths

As mentioned in Section 2.2.2 we need to put considerable thought into picking the right parameter if we want to achieve the most practically meaningful results. To do this, we have to achieve a compromise between two essentially competing goals:

- As many instances as possible should have a low or moderate value of the parameter.
- As many problems as possible should be (fixed-parameter) tractable if we use the parameterization we picked.

Let us give an example to make this concrete. According to the definition it is possible to parameterize any graph problem by setting $k = n$. This would satisfy our second objective, because every decidable problem would be FPT! However, it would fall far short from achieving our first objective, since the whole point of complexity theory is dealing with large instances. Going to the other extreme, we could simply parameterize a problem by setting $k = 1$ for all instances. In this case, all instances would have a small value for the parameter, but this would not help us at all compared to trying to solve the general case of the problem.

To give a slightly less trivial example, consider the one reasonable structural graph parameter we have mentioned so far: maximum degree. Though parameterizing by Δ makes more sense than the contrived examples we just saw, it is still not entirely satisfactory: even though it can be argued that a large number of practically interesting graphs have low or moderate degree, unfortunately most of the problems we usually consider in complexity theory are still hard, in fact often even for graphs of maximum degree 3.

Thus, we arrive at one of the motivations for the theory of graph *widths*: define a complexity measure on graphs such that by looking at graphs where the measure has a low value we would be "isolating" a large class of algorithmically easy graphs. By far the most important specimen in this area is the definition of treewidth, a measure which informally tells us how much a graph looks like a tree. In the rest of this section we give the definition of treewidth and review the most important results. We also mention several other related widths and their connections to treewidth, which are summarized in Figure 2.2.

2.3.1 Treewidth

Given a graph $G(V, E)$, a tree decomposition is a tree $T(X, I)$ which has the following properties:

1. Every vertex $x \in X$ of T is a subset of V. Moreover, for every $v \in V$ there exists an $x \in X$ such that $v \in x$. Informally we say that the tree T consists of "bags" of the vertices of the original graph and every vertex of the original graph must appear in some bag.

2. For every edge $(u, v) \in E$ there exists a bag $x \in X$ such that $u \in x$ and $v \in x$.

3. For every vertex $u \in V$, the set of bags $X_u = \{x \in X \mid u \in x\}$ induces a connected graph.

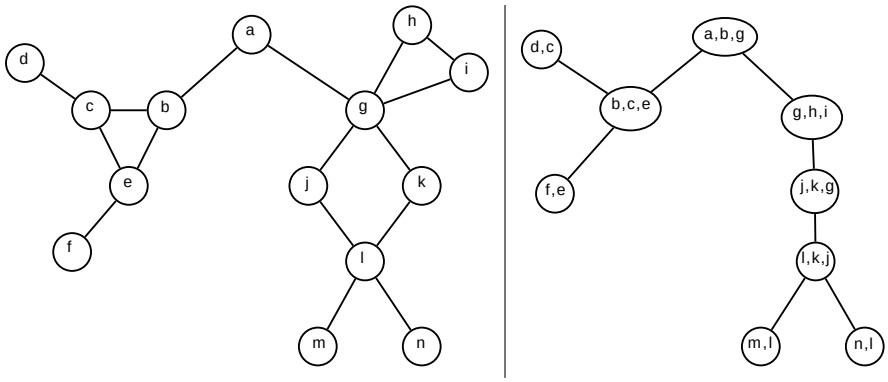


Figure 2.1: An example graph and its tree decomposition

We say that the width of a tree decomposition is $\max_{x \in X} |x| - 1$. The treewidth of a graph is the minimum width of any of its decompositions.

The definition given above is the standard one found in the literature but it is rather obscure and confusing at first. There are several other equivalent definitions for treewidth, but this one is the most important for algorithmic applications. Let us attempt to explain it.

The goal of a tree decomposition is to capture the tree-like structure of a graph (if it exists). Intuitively, a graph has a tree-like structure if we can break it down into small components which are connected in a tree. This is more or less the role played by the bags in the decomposition. More precisely, the vertices which are contained in a bag represent a border (a separator) in the graph: they separate the vertices which are to be found in the subtrees we get by removing this bag from T. In this light, properties 1 and 2 are not surprising: we want the decomposition to capture every edge and every vertex of a graph. The key is property 3: we demand that if we keep all the bags which contain a vertex u then these induce a subtree. The reason is that we need every bag to act as a separator. Another way to interpret this rule is as saying that if u is contained in two bags x, y, it is contained in all bags in the path from x to y in T. This separation property is crucial in getting treewidth-based algorithms to work.

Treewidth was first introduced in Robertson and Seymour [1986]. Besides its algorithmic applications, it has proven very interesting from a graph-theoretic perspective, one of its most important attributes being that it can be equivalently defined in many seemingly unrelated ways. For example treewidth is connected to chordal graphs, elimination schemes, partial k-trees, cops-and-robber games (Seymour and Thomas [1993], Dendris et al. [1997]), reduction rules (Arnborg et al. [1993]) and brambles (Seymour and Thomas [1993]). A great introduction to the notion of treewidth is given in Bodlaender's excellent survey papers Bodlaender [2007, 2006, 1997], Bodlaender and Koster [2008].

2.3.2 Other Widths

Treewidth is by no means the only graph width in the literature. There are many other ways to quantify the complexity of a graph, many of them with definitions related to treewidth. Here, we will briefly mention the most important ones, without giving formal definitions.

Clique-width[1] is a measure related to treewidth with one major difference: clique-width also "covers" families of dense graphs. This addresses one of the basic constraints of treewidth, the fact that only sparse graphs can have low treewidth. Several interesting connections are known between the two: first, it is known that $cw(G) \leq 3(2^{tw(G)-1})$ (Corneil and Rotics [2005], Courcelle and Olariu [2000]), meaning that if a graph has small treewidth, it must also have (relatively) small clique-width (notice though that the clique-width will in the worst case be exponentially larger). Note that it is not possible to show an inequality in the opposite direction, since cliques have small clique-width but unbounded treewidth. Therefore, in a sense, clique-width can be seen as a measure which "generalizes" treewidth.

Local treewidth, introduced in Eppstein [2000], is another generalization of treewidth,

[1] see Courcelle and Olariu [2000] for a definition

this time based on the concept of neighborhoods. In addition to treewidth, this notion generalizes the family of bounded-degree graphs and the family of planar graphs. Informally, the idea here is that a graph has small local treewidth if the local neighborhoods around all vertices induce graphs with small treewidth.

In a graph of maximum degree Δ we have that $|N_r[u]| \leq \Delta^r$ for all vertices u, where by $N_r[u]$ we denote the set of vertices at distance at most r from u. This immediately implies that graphs of bounded degree have bounded local treewidth (the neighborhood has bounded size, therefore also bounded treewidth). Furthermore, it is known that the treewidth of a planar graph is bounded by its diameter (multiplied by 3), meaning that planar graphs also have bounded local treewidth. Finally, if a graph has small treewidth all its neighborhoods must also have small treewidth.

Local treewidth is a proper generalization of treewidth, since there exist planar graphs of bounded degree (e.g. grids) with unbounded treewidth. It is incomparable to clique-width, since cliques have unbounded local treewidth and bounded degree graphs can have unbounded clique-width.

Pathwidth is a natural restriction of treewidth where the tree decomposition is required to be a path. This immediately implies that graphs with small pathwidth (that is, graphs which have a path decomposition with small width) must also have small treewidth. The converse, however, is not necessarily true. In fact, there exist trees (which by definition have the smallest possible treewidth, 1) of unbounded pathwidth. For any graph G with n vertices it is known that $tw(G) \leq pw(G) \leq tw(G) \cdot \log n$ and the inequalities are tight in the worst case. Like treewidth, pathwidth can also be defined in several equivalent ways, including a cops-and-robbers game, see Bodlaender et al. [1995].

Another measure which quantifies a graph's distance from being a tree is feedback vertex set. A feedback vertex set is a set of vertices of a graph whose removal breaks all the graph's cycles (i.e. the remaining vertices form a forest). Graphs with small

feedback vertex set also have small treewidth (by including all the vertices of the feedback set in all the bags of the decomposition), but not necessarily small pathwidth (since there exist trees with unbounded pathwidth). On the other hand graphs with small pathwidth may have unbounded feedback vertex set, the typical example being a $2 \times n$ grid.

An even more restricted measure is vertex cover. A vertex cover of a graph is a set of vertices whose removal deletes all edges. It is not hard to see that this is a parameter that is more restricted than the others we have discussed so far, since small vertex cover easily implies both small pathwidth and small feedback vertex set. On the other hand, since paths have unbounded vertex cover it is clear that the restriction is strict.

Though graphs of bounded vertex cover are very restricted, their algorithmic properties have attracted attention in the past, both in the subject of their recognition which is a flagship problem in parameterized complexity (see e.g. Chen et al. [2006]) and in the context of attacking problems which are generally hard for graphs of bounded treewidth (see e.g. Fellows et al. [2008]).

Finally, another measure which has appeared as a structural parameter in the literature is max-leaf number. We say that a connected graph G has max-leaf number at most l if no spanning tree of G has more than l leaves. The algorithmic properties of this class of graphs have been investigated in the past (Estivill-Castro et al. [2005], Fellows and Rosamond [2007], Fellows et al. [2009]). An important point is the characterization of bounded max-leaf graphs from Kleitman and West [1991] which is also heavily used in Fellows et al. [2009].

The relations between the mentioned parameters are summarized in Figure 2.2.

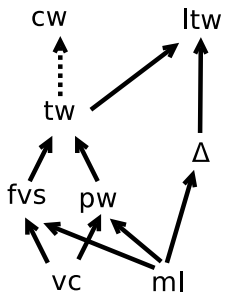


Figure 2.2: A hierarchy of graph parameters for undirected graphs. The parameters in this figure are (from top to bottom) clique-width, local treewidth, treewidth, bounded degree, feedback vertex set, pathwidth, vertex cover and max-leaf number. Arrows indicate generalization (e.g. graphs with small vertex cover also have small pathwidth, but not vice-versa). The dashed arrow indicates an exponential blow-up in the parameter.

.

2.3.3 Algorithmic Results

Now that we have established a basic grasp of the landscape of the graph parameters we will be working with we can briefly mention some results about their algorithmic properties. We invite the reader to take another look at Figure 2.2. Though all the parameters measure different things we have found connections which lead to immediate conclusions such as the fact that any problem which is easy (FPT) on graphs of small treewidth is also easy on graphs of small pathwidth, or any problem which is hard on graphs of small max-leaf number is hard on graphs of bounded degree. Informally, we could say that (positive) algorithmic results propagate downwards in this hierarchy while (negative) intractability results propagate upwards.

Treewidth is by far the most thoroughly investigated of these measures, especially with regards to its algorithmic properties. A large number of graph problems are known to be FPT when parameterized by treewidth. Algorithms for graphs of bounded treewidth generally follow a similar pattern: they perform some kind of dynamic programming on the bags of the tree decomposition. Informally, one first looks at the graph induced by the vertices of each bag of the decomposition. On each of these graphs we enumerate (i.e. list by brute force) all possible partial solutions to

the problem. Then, dynamic programming is used to merge the tables of solutions from each bag to a total solution. This is a process which takes time exponential in the size of each bag, but polynomial in the order of the graph.

Following the dynamic programming technique roughly outlined above a wide variety of problems can be solved efficiently on graphs of small treewidth. In fact, a quick survey of the literature can reveal that a large number of problems of the form "find the smallest/largest set of vertices such that some property holds" are solvable in this way when parameterized by treewidth. This includes classical problems such as MAX CUT, INDEPENDENT SET, CLIQUE, DOMINATING SET and others, which are all solvable by more or less the same technique.

This leads to the natural question, can we solve all such problems in FPT time when parameterized by treewidth? The answer, which is yes, is formalized in a famous result usually referred to as Courcelle's theorem. Courcelle's theorem states that all problems expressible in a logic language called MSO_2 (which we will formally define in Chapter 3) are solvable in linear-time on graphs of bounded treewidth. This is a far-reaching and celebrated result which covers the vast majority of NP-hard graph problems typically covered in an algorithms class today. In Chapter 3 we will mention more results in this vein, which are usually called algorithmic meta-theorems, and discuss how Courcelle's theorem can be strengthened in some special cases.

Chapter 3

Algorithmic Meta-Theorems

In this section we will show the existence of several algorithmic meta-theorems for (undirected) graphs. Recall that, as mentioned in Chapter 2 the most celebrated such metatheorem is Courcelle's theorem (see Courcelle [1990]) which states that every graph property expressible in monadic second-order (MSO_2) logic is decidable in linear time if restricted to graphs of bounded treewidth.

Here we focus on the study of algorithmic metatheorems in the spirit of Courcelle's theorem, where the class of problems we attack is defined in terms of expressibility in a logic language. More specifically, we will study the algorithmic properties of two graph classes: graphs of bounded vertex cover and graphs of bounded max-leaf number. Though Courcelle's theorem applies here, since these classes have bounded treewidth, we will show algorithms with significantly better parameter dependence than that implied by Courcelle's theorem. In the case of vertex cover, we also give an essentially matching lower bound. The results presented here have appeared in Lampis [2010] and Lampis [2011a].

3.1 Previous Work

Metatheorems have been a subject of intensive research in the last years producing a wealth of interesting results. Some representative examples of metatheorems with a flavor similar to Courcelle's can be found in the work of Frick and Grohe (Frick and Grohe [2001]), where it is shown that all properties expressible in first order (FO) logic are solvable in linear time on planar graphs, and the work of Dawar et al. (Dawar et al. [2006]), where it is shown that all FO-definable optimisation problems admit a PTAS on graphs excluding a fixed minor (see Grohe [2007] and Hlinený et al. [2008] for more results on the topic). In all these works the defining property for the problems studied is given in terms of expressibility in a logic language; in many cases metatheorems are stated with problem being defined by some other property, for example whether the problem is closed under the taking of minors. This approach, which is connected with the famous graph minor project of Robertson and Seymour (see Robertson and Seymour [1983-2004]) has also led to a wealth of significant and practical results, including the so called bi-dimensionality theory (see Demaine and Hajiaghayi [2008] for an overview and also the recent results of Bodlaender et al. [2009]).

In the case where logic is used to define the problems, many interesting extensions have followed Courcelle's seminal result: for instance, Courcelle's theorem has been extended to logics more suitable for the expression of optimisation problems (Arnborg et al. [1991]). It has also been investigated whether it's possible to obtain similar results for larger graph classes (see Courcelle et al. [2000] for a metatheorem for bounded cliquewidth graphs, Fomin et al. [2009, 2010] for corresponding hardness results and Kreutzer and Tazari [2010] for hardness results for graphs of small but unbounded treewidth). Finally, lower bound results have been shown proving that the running times predicted by Courcelle's theorem can not be improved significantly in general (Frick and Grohe [2004]).

This lower bound result is one of the main motivations of the results we will present here, because in some ways it is quite devastating. Though Courcelle's theorem shows that a vast class of problems is solvable in linear time on graphs of bounded treewidth, the "hidden constant" in this running time, that is, the running time's dependence on the input's other parameters, which are the graph's treewidth and the formula describing the problem, is in fact (in the worst case) a tower of exponentials. Unfortunately, in Frick and Grohe [2004] it is shown that this tower of exponentials is unavoidable even if we restrict ourselves to deciding FO logic on trees. For this reason, we will focus on two graph classes which do not contain the class of all trees: graphs of bounded vertex cover and graphs of bounded max-leaf number.

Though graphs of bounded vertex cover or max-leaf number are considerably more restricted than bounded treewidth graphs, these classes are still interesting from the algorithmic point of view and the complexity of hard problems parameterized by vertex cover or max-leaf number has been investigated in the past (Fellows et al. [2008], Fellows et al. [2009]). Furthermore, as mentioned, strong lower bounds are known to apply to slightly more general classes: for bounded feedback vertex set and bounded pathwidth graphs even FO logic is non-elementary, while even for binary trees (thus for graphs of bounded treewidth and max degree) FO logic is at least triply exponential (again by Frick and Grohe [2004]). Bounded vertex cover and bounded max-leaf number evade all these lower bound arguments so it's natural to ask what is exactly the complexity of FO and MSO logic for these classes of graphs.

3.2 Preliminaries

We will describe algorithmic meta-theorems, that is, general methods for solving all problems belonging in a class of problems. However, the presentation is simplified if one poses this approach as an attack on a single problem, the model checking problem.

In the model checking problem we are given a logic formula ϕ, expressing a graph property, and a graph G, and we must decide if the property described by ϕ holds in G. In that case, we write $G \models \phi$. Clearly, if we can describe an efficient algorithm for model checking for a specific logic, this will imply the existence of efficient algorithms for all problems expressible in this logic. Let us now give more details about the logics we will deal with and the graphs which will be our input instances.

Our universe of discourse will be labeled, colored graphs. Specifically, we assume that the first part of the input is an undirected graph $G(V, E)$, a set of labels L, each associated with a vertex of V and a set of subsets of V, $\mathcal{C} = \{C_1, C_2, \ldots, C_c\}$, which we refer to as color classes. Note that it could be the case that several labels are assigned to the same vertex and that some vertex belongs in several color classes. The interesting case here is unlabeled, uncolored graphs (that is, $L = \mathcal{C} = \emptyset$), but the additional generality in the definition of the problem makes it easier to describe a recursive algorithm. We include labels in our definition to allow our formulas to refer to some constant vertices of the input graph.

The formulas of FO logic are those which can be constructed using vertex variables, denoted usually by $x_i, y_i, \ldots$, vertex labels denoted by l_i, color classes denoted by C_i, the predicates $E(x_i, x_j)$, $x_i \in C_j$, $x_i = x_j$ operating on vertex variables or labels, standard propositional connectives and the quantifiers $\exists, \forall$ operating on vertex variables. The semantics are defined in the usual way, with the $E()$ predicate being true if $(x_i, x_j) \in E$ and labels being interpreted as vertex constants corresponding to the vertices of the graph they are attached to. We also sometimes extend notation slightly by using conditional quantified variables: $\exists x : \psi(x)$ $(\phi(x))$ can be read as shorthand for $\exists x(\psi(x) \wedge \phi(x))$, while $\forall x : \psi(x)$ $(\phi(x))$ is short for $\forall x(\psi(x) \rightarrow \phi(x))$.

For MSO logic the additional property is that we now introduce set variables denoted by X_i and allow the quantifiers and the $\in$ predicate to operate on them. The semantics are defined in the obvious way. If the set variables are allowed to range

over sets of vertices only, then the logic is referred to as MSO_1.

A variation is MSO_2 logic. Here, first-order variables are allowed to range over vertices or edges, and second-order variables range over sets of vertices or edges. To keep the presentation simple we will use the letters e_i, F_i for variables which range over edges and sets of edges respectively and we assume that it is clear from the context what domain each variable is quantified over. We also add the incidence predicate $I(v, e)$ which is true if edge e is incident on vertex v. Observe that in the first-order case it does not make a difference if one also allows quantification over edges or not, because any FO formula that uses edge variables can be transformed to an equivalent formula that only uses vertex variables: one simply replaces $\exists e$ with $\exists x \exists y : E(x, y)$ while also replacing $I(v, e)$ with $(v = x) \vee (v = y)$. It is known that this is not possible with MSO in general: there exist MSO_2 expressible properties which are not expressible in MSO_1. However, we will use such a transformation that works for graphs of small vertex cover.

3.2.1 Bounded Vertex Cover and neighborhood diversity

We will work extensively with graphs of bounded vertex cover, that is, graphs for which there exists a small set of vertices whose removal also removes all edges. We will usually denote the size of a graph's vertex cover by k. Note that there exist linear-time FPT algorithms for finding an optimal vertex cover in graphs where k is small (see e.g. Chen et al. [2006]). Recall that an algorithm is called fixed-parameter tractable (FPT) if it runs in time $f(k)n^{O(1)}$ for some function f of the parameter k.

In a graph of vertex cover k, the vertices outside the vertex cover can be partitioned into at most 2^k sets, such that all the vertices in each set have exactly the same neighbors outside the set and each set contains no edges inside it. Since we will not make use of any other special property of graphs of small vertex cover, we are motivated to define a new graph parameter, called neighborhood diversity, which

intuitively seems to give the largest graph family to which we can apply our method in a straightforward way.

Definition 3.1. We will say that two vertices v, v' of a graph $G(V, E)$ have the same type iff they have the same colors and $N(v) \setminus \{v'\} = N(v') \setminus \{v\}$, where $N(v)$ denotes the set of neighbors of v.

Lemma 3.2. *Having the same type is an equivalence relation on the set of vertices of a graph G.*

Proof. Obviously the relation is reflexive and symmetric, so we only need to prove that it is transitive. Suppose u and v have the same type and also that v and w have the same type. First, $N(u) \setminus \{v\} = N(v) \setminus \{u\}$ and $N(v) \setminus \{w\} = N(w) \setminus \{v\}$ from the definition. From this we have $N(u) \setminus \{v, w\} = N(v) \setminus \{u, w\} = N(w) \setminus \{u, v\}$. So, it suffices to show that if v is connected to one of u, w it is connected to the other. But if u, v are connected then u, w are also since v and w have the same type. Now, because u and v have the same type and u, w are connected then v, w are also connected.

□

Definition 3.3. A colored graph $G(V, E)$ has neighborhood diversity at most w, if there exists a partition of V into at most w sets, such that all the vertices in each set have the same type.

Lemma 3.4. *If an uncolored graph has vertex cover at most k, then it has neighborhood diversity at most $2^k + k$.*

Proof. Construct k singleton sets, one for each vertex in the vertex cover and at most 2^k additional sets, one for each subset of vertices of the vertex cover. Place each of the vertices of the independent set in one of these sets, specifically the one which corresponds to its neighborhood in the vertex cover.

□

In Section 3.7 we will show some more results about neighborhood diversity which indicate it may be an interesting parameter in its own right. However, until then our main focus will be graphs of bounded vertex cover. We will prove most of our algorithmic results in terms of neighborhood diversity and then invoke Lemma 3.4 to obtain our main objective. We will call a partition of the vertex set of a graph G into w sets such that all vertices in every set share the same type a neighborhood partition of width w. We will usually assume that a neighborhood partition of the graph is given to us, because otherwise one can easily be found in linear time by using the mentioned linear-time FPT algorithm for vertex cover and Lemma 3.4.

3.2.2 Bounded Max-Leaf Number

Recall that a connected graph G has max-leaf number at most l if no spanning tree of G has more than l leaves. The algorithmic properties of this class of graphs have been investigated in the past (Estivill-Castro et al. [2005], Fellows and Rosamond [2007], Fellows et al. [2009]). In this paper we rely heavily on a characterization of bounded max-leaf graphs by Kleitman and West [1991] which is also heavily used in Fellows et al. [2009].

Theorem 3.1. *(Kleitman and West [1991]) If a graph G has max-leaf number at most l, then G is a subdivision of a graph on $O(l)$ vertices.*

What this theorem tells us intuitively is that in a graph $G(V, E)$ with max-leaf number l there exists a set S of $O(l)$ vertices such that $G[V \setminus S]$ is a collection of $O(l^2)$ paths. Furthermore, only the endpoints of the paths can be connected to vertices of S in G.

It is well-known that a graph of max-leaf number at most l has a path decomposition of width at most $2l$. Furthermore, it must have maximum degree at most l. Bounded max-leaf number graphs are therefore a subclass of the intersection of

bounded pathwidth and bounded degree graphs (in fact, they are a proper subclass, as witnessed by the existence of say $2 \times n$ grids). Let us mention again that model checking FO formulas on binary trees has at least a triply exponential parameter dependence, so the results we present for graphs of bounded max-leaf number can also be seen as an improvement on the currently known results for FO logic on bounded degree graphs, for this more restricted case.

3.3 FO Logic for Bounded Vertex Cover

In this Section we show how any FO formula can be decided on graphs of bounded vertex cover number, with a singly exponential parameter dependence. Our main argument is that for FO logic, two vertices which have the same neighbors are essentially equivalent. We will state our results in the more general case of bounded neighborhood diversity and then show the corresponding result for bounded vertex cover as a corollary.

Lemma 3.5. *Let $G(V, E)$ be a graph and $\phi(x)$ a FO formula with one free variable. Let $v, v' \in V$ be two distinct unlabeled vertices of G that have the same type. Then $G \models \phi(v)$ iff $G \models \phi(v')$.*

Proof. Let l be a new label which is not currently used in G. Let G_1 be the labeled graph we obtain from G if we associate l with v and G_2 be the labeled graph we obtain if we associate l with v'. Then the labeled graphs G_1 and G_2 are isomorphic (meaning that there is a one-to-one correspondence between them that also respects the labels). Therefore, $G_1 \models \phi(l)$ iff $G_2 \models \phi(l)$.

□

Theorem 3.2. *Let ϕ be a FO sentence of quantifier depth q. Let $G(V, E)$ be a labeled colored graph with neighborhood diversity at most w and l labeled vertices. Then, there*

is an algorithm that decides if $G \models \phi$ in time $O((w+l+q)^q \cdot |\phi|)$, assuming that an optimal neighborhood partition is given with the input.

Proof. We will rely heavily on Lemma 3.5 and describe a recursive algorithm. If $q = 0$ the problem is trivial, so assume $q > 0$. Assume wlog that ϕ is in prenex normal form, $\phi = Qx\psi(x)$ where Q is $\exists$ or $\forall$.

Suppose that V can be partitioned into $V_1, V_2, \dots V_w$ as required by the definition of neighborhood diversity. Now, by Lemma 3.5 if $v, v' \in V_i$ for some i, and neither of the two is labeled then $G \models \psi(v)$ iff $G \models \psi(v')$. Thus, it suffices to recursively model check at most $(w+l)$ sentences of $q-1$ quantifiers to decide ϕ: we try replacing x with each of the l labeled vertices or with one arbitrarily chosen nonlabeled representative from each V_i. If x is existentially quantified we decide that $G \models \phi$ if at least one of the resulting sentences is true, while if x is universally quantified we decide that $G \models \phi$ if all of the resulting sentences are true. In the process we introduce a new label. Repeating this process constructs a computation tree with at most $\prod_{i=0}^{q-1}(w+l+i) = O\left((w+l+q)^q\right)$ leaves. The result of the computation tree can be evaluated in time linear in its size.

□

Corollary 3.6. *There exists an algorithm which, given a FO sentence ϕ with q variables and an uncolored, unlabeled graph G on n vertices with vertex cover at most k, decides if $G \models \phi$ in time $2^{O(kq+q\log q)}|\phi| + O(2^k n)$.*

Proof. The second term in the running time comes from the basic FPT algorithm for finding a vertex cover of size k. From this we can construct a neighborhood partition and invoke Theorem 3.2 and Lemma 3.4.

□

Thus, the running time is (only) singly exponential in the parameters, while a straightforward observation that bounded vertex cover graphs have bounded treewidth

and an application of Courcelle's theorem would in general have a non-elementary running time. Of course, a natural question to ask now is whether it is possible to do even better, perhaps making the exponent linear in the parameter. As we will see later on, this is not possible if we accept some standard complexity assumptions.

3.4 FO Logic for Bounded Max-Leaf Number

In this section we describe a model checking algorithm for FO logic on graphs of small max-leaf number. Because we are not going to solve MSO logic on this class of graphs, we can simplify things by assuming that our graphs only have labels and not colors (i.e. all vertices are initially uncolored). Our main tool is the mentioned observation that all but a small fraction of the vertices have degree 2, and therefore (since we assume without loss of generality that the graph is connected) induce paths. We call a maximal set of connected vertices of degree 2 a topo-edge.

Our main argument is that when a topo-edge is very long (exponentially long in the number of quantifiers of the first-order sentence we are model checking) its precise length does not matter. Readers familiar with classical results regarding Ehrenfeucht-Fraisse games and their use in proving negative results for the expressive power of FO logic on paths will recognize that the technique we use is an extension of this work to graphs of small max-leaf number (for more information on E-F games see for example Immerman [1999]).

First we define a similarity relation on graphs.

Definition 3.7. Let G_1, G_2, be two labeled graphs. For a given q we will say that G_1 and G_2 are q-similar and write $G_1 \sim_q G_2$ iff G_1 contains a topo-edge of order at least 2^{q+1} consisting of unlabeled vertices, call it P, and G_2 can be obtained from G_1 by contracting one of the edges of P. We denote the transitive closure of the relation $\sim_q$ as $\sim_q^*$.

Our main technical tool is now the following lemma.

Lemma 3.8. *Let ϕ be a FO formula with q quantifiers. Then, for any two graphs G_1, G_2 if $G_1 \sim_q G_2$ then $G_1 \models \phi$ iff $G_2 \models \phi$. Therefore, if $G_1 \sim_q^* G_2$ then $G_1 \models \phi$ iff $G_2 \models \phi$.*

Proof. We will prove the first statement by induction on q and the second statement follows directly from it. For $q = 0$ the statement is trivial because ϕ can only refer to labeled vertices and G_1, G_2 are identical with respect to these vertices.

Suppose that the statement is true for at most $q-1$ quantifiers. It suffices to show the statement for q quantifiers for a formula ϕ of the form $\exists x \psi(x)$, and the statement then easily follows for formulas which are boolean combinations of formulas of at most q quantifiers. So, suppose that $G_1 \models \exists x \psi(x)$. This means that there exists a vertex in G_1 such that if we label it with a new label l to obtain a graph G_1' (which is G_1 with the label l added) we have $G_1' \models \psi(l)$. Now we must take cases for the vertex where l is placed.

If l is placed on a vertex outside of P then it is not hard to see that $G_2 \models \phi$: we place l on the same vertex on G_2 (and obtain G_2') and now we have $G_1' \sim_{(q-1)} G_2'$ so from the inductive hypothesis $G_2' \models \psi(l)$.

Now the interesting case is when l is placed on a vertex of P. Number the vertices of P from 1 to $|P|$, starting from one of the endpoints of the path induced by P. Partition P into two parts: P_2 contains the last 2^q vertices and P_1 the rest. In G_2 we use the same numbering for the vertices of the path (of course now the numbering is from 1 to $|P| - 1$, since one edge has been contracted).

Suppose that l is placed on a vertex of P_1. We place l on the same vertex in G_2. Now, we have $G_1' \sim_{(q-1)} G_2'$, because in both graphs P has been broken into two paths P' and P''. P' has the same size on both (depending on the position where l was placed) and P'' has size at least 2^q on G_1' and one less than that on G_2'. So, by the inductive hypothesis $G_1' \models \psi(l)$ iff $G_2' \models \psi(l)$.

Finally, if l is placed on a vertex of P_2 we place l on a vertex of G_2 that has the same distance from the end of the path and two $(q-1)$-similar graphs G_1', G_2' are obtained, because the smaller part of the two into which P is broken has the same size on both graphs and the larger has size at least 2^q. So by the inductive hypothesis $G_1' \models \psi(l)$ iff $G_2' \models \psi(l)$.

The converse directions where we know that $G_2 \models \phi$ and need to show that this implies $G_1 \models \phi$ can be established with a similar argument.

□

Now we are ready to state our main result of this section.

Theorem 3.3. *Let G be a graph on n vertices with max-leaf number k and ϕ a FO formula with q quantifiers. Then, there exists an algorithm for deciding if $G \models \phi$ running in time poly$(n) + 2^{O(q^2 + q \log k)}$.*

Proof. By applying Theorem 3.1 we know that G can be partitioned into a set of at most $O(k)$ vertices of degree at least 3 and a collection of paths. By applying Lemma 3.8 we know that there exists a G' such that $G \sim_q^* G'$ and G' consists of the same $O(k)$ vertices of degree at least 3 and at most $O(k^2)$ paths whose length is at most 2^{q+1}. Of course, G' can be found in time polynomial in n.

Now, we can apply the straightforward algorithm to model check ϕ on G'. The trivial algorithm takes time $O(|V|^q) = O((k^2 2^{q+1})^q)$ giving the promised running time.

□

3.5 MSO Logic for Bounded Vertex Cover

Here we will follow a similar strategy as in Section 3.3 proving that if there is a very large number of vertices of a certain type in our graph then it is safe to delete some of

them without affecting the truth of the MSO sentence we are trying to model check. To do this we first define another kind of similarity relation on graphs.

Definition 3.9. Let G_1, G_2, be two labeled colored graphs. For given integers q_S, q_V we will say that G_1 and G_2 are (q_S, q_V)-similar and write $G_1 \sim_{(q_S,q_V)} G_2$ iff G_2 can be obtained by G_1 by deleting an unlabeled vertex u and G_1 contains at least $2^{q_S} q_V$ additional unlabeled vertices of the same type as u. We denote the transitive closure of the relation $\sim_{(q_S,q_V)}$ as $\sim^*_{(q_S,q_V)}$.

Lemma 3.10. *Let ϕ be a MSO_1 formula with q_S set quantifiers and q_V vertex quantifiers. Then, for any two graphs G_1, G_2 if $G_1 \sim_{(q_S,q_V)} G_2$ then $G_1 \models \phi$ iff $G_2 \models \phi$. Therefore, if $G_1 \sim^*_{(q_S,q_V)} G_2$ then $G_1 \models \phi$ iff $G_2 \models \phi$.*

Proof. We will prove the first statement by induction on $q_S + q_V$ and the second statement will immediately follow. For $q_V = 0$ the statement is trivial since without vertex variables the formula may only refer to the labeled vertices where G_1 and G_2 are identical so the statement is proved for $q_S + q_V = 0$.

Suppose that we have proved the statement for formulas with at most q quantified (vertex and set) variables and we are given a formula ϕ with q_S set variables and q_V vertex variables, where $q_S + q_V = q + 1$. We are also given two graphs G_1, G_2 such that $G_1 \sim_{(q_S,q_V)} G_2$. The two interesting cases are $\phi = \exists x \psi(x)$ and $\phi = \exists X \psi(X)$ (i.e. ϕ begins with an existentially quantified vertex or set variable) because the universal quantification case and boolean combinations of simpler formulas follow directly if we deal with these.

First, assume that $\phi = \exists X \psi(X)$ and that $G_1 \models \phi$. So, there exists a set S_1 of vertices of G_1 such that assigning a new color C to these, thus obtaining a new colored graph G_1', gives us $G_1' \models \psi(C)$. Let T be the type of vertices of G_1 where if we delete a vertex we obtain G_2 (recall that $|T| \geq 2^{q_S} q_V + 1$). We select a set S_2 of vertices of G_2 as follows: for every type other than T we select the same number of

vertices as S_1 has selected from this type in G_1. From T, if S_1 contains at most half the vertices of type T in G_1 we place the same number of vertices from that type of G_2 in S_2. Otherwise, we select from type T one vertex less than S_1 contains from that type in G_1. We thus obtain a graph G_2' by coloring all the vertices of S_2 with a new color C. Informally, we can say that G_1' and G_2' are the same except that one of the types of G_1' has one more vertex than the corresponding type of G_2'. Note that we have made sure that this type has at least half the vertices of T. The claim now is that $G_1' \sim_{(q_S-1,q_V)} G_2'$. To see this, observe that we can obtain G_2' from G_1' by deleting a vertex which had type T in G_1. If $|S_1 \cap T| = |S_2 \cap T|$ that vertex is one which did not receive the new color C, but this happens if at most half the vertices did, meaning its type contains at least $\lceil (2^{q_S} q_V + 1)/2 \rceil = 2^{q_S-1} q_V + 1$ vertices in G_1'. Otherwise, the vertex we can delete is one that received the new color, which means in this case its type again contains at least $2^{q_S-1} q_V + 1$ vertices. From the inductive hypothesis we now get $G_1' \models \psi(C)$ iff $G_2' \models \psi(C)$, which gives $G_1 \models \phi$ iff $G_2 \models \phi$. Similar arguments can be applied if we start with the assumption $G_2 \models \phi$.

Second, if $\phi = \exists x \psi(x)$ and $G_1 \models \phi$, there exists a vertex of G_1 such that assigning to it a new label l, thus obtaining a new graph G_1', we have $G_1' \models \psi(l)$. We assign the label l to a vertex of the same type in G_2, obtaining G_2'. Now, we have $G_1' \sim_{(q_S,q_V-1)} G_2'$, because the number of unlabeled vertices in the type where G_1 and G_2 differ has been decreased by at most one. Therefore, it is now at least $2^{q_S} q_V \geq 2^{q_S}(q_V - 1) + 1$. By inductive hypothesis we get $G_2' \models \psi(l)$ so $G_2 \models \phi$. Similar arguments can again be applied if we initially assume that $G_2 \models \phi$.

□

Theorem 3.4. *Let G be a graph on n vertices with neighborhood diversity at most w and ϕ be a MSO$_1$ formula with q_S set quantifiers and q_V vertex quantifiers. Then, given a neighborhood partition of G, there exists an algorithm which can decide if $G \models \phi$ in time $2^{O(w 2^{q_S} q_V + q_V \log q_V)}$.*

Proof. Using Lemma 3.10 we can assume that no type has more than $2^{q_S}q_V$ vertices, otherwise we can delete one vertex and get an equivalent graph. Thus, the total number of vertices is at most $w2^{q_S}q_V$.

The trivial MSO_1 model checking algorithm on a graph on n vertices would take time $O((2^n)^{q_S} \cdot n^{q_V} \cdot |\phi|)$ (try all possible cases for each set variable and each vertex variable). Using the above bound on n gives the promised running time.

□

Corollary 3.11. *There exists an algorithm which, given a MSO_1 sentence ϕ with q variables and an uncolored, unlabeled graph G with vertex cover at most k, decides if $G \models \phi$ in time $2^{2^{O(k+q)}} + O(2^k n)$.*

Again, this gives a dramatic improvement compared to Courcelle's theorem, though exponentially worse than the case of FO logic. This is an interesting point to consider because for treewidth there does not seem to be any major difference between the complexities of model checking FO and MSO_1 logic.

The natural question to ask here is once again, can we do significantly better? For example, perhaps the most natural question to ask is, is it possible to solve this problem in $2^{2^{o(k+q)}}$? As we will see later on, the answer is no, if we accept some standard complexity assumptions.

Finally, let us briefly discuss the case of MSO_2 logic. In general this logic is more powerful than MSO_1, so it is not straightforward to extend Theorem 3.4 in this case. However, if we are not interested in neighborhood diversity but just in vertex cover we can observe that all edges in a graph with vertex cover of size k have one of their endpoints in one of the k vertices of the vertex cover. Thus, any edge set X can be written as the union of k edge sets. In turn, each of these k edge sets can easily be replaced by vertex sets, without loss of information, since we already know one of the endpoints of each of these edges. Using this trick we can replace every edge set variable in an MSO_2 sentence with k vertex set variables. This leads to a $2^{2^{O(kq)}}$

algorithm for MSO_2 logic on graphs of bounded vertex cover.

Lemma 3.12. *Let ϕ be an MSO_2 sentence with q quantifiers and G be a graph of vertex cover k. Then, there exists an MSO_1 sentence ϕ' with $O(kq)$ quantifiers and a graph G' with vertex cover k and k labeled vertices such that $G \models \phi$ iff $G' \models \phi'$.*

Proof. We'll first argue that edge and edge-set variables can be removed from ϕ. G' will simply be G with k labels $l_1, \ldots, l_k$, each attached to a different vertex of the vertex cover. Suppose wlog that ϕ is in prenex normal form, and the edge-set variables which appear in ϕ are $F_1, \ldots, F_m$, while the edge variables which appear are $e_1, \ldots, e_p$. For the former, we replace their quantifications QF_i, where Q is $\exists$ or $\forall$, with k new quantified set variables for each: $QX_{i,1}QX_{i,2}\ldots QX_{i,k} : (\wedge_{1\leq j\leq k}\forall x : x \in X_{i,j}(E(x, l_j)))$. For edge variables, we replace Qe_i with $Qx_{e_i}Qy_{e_i} : E(x_{e_i}, y_{e_i})$ where x_{e_i}, y_{e_i} are new vertex variables.

We continue by replacing every occurence of $e_i \in F_j$ with the formula $\vee_{1\leq i'\leq k}(x_i = l_{i'} \wedge y_i \in X_{j,i'})$, while we replace each occurence of $I(x, e_i)$ with $(x = x_{e_i} \vee x = y_{e_i})$. Thus, we are left with an MSO_1 formula. It is not hard to see that the new formula has at most k quantifiers for every quantifier of the old formula. Its total size is also at most $O(k|\phi|)$.

Now ϕ' is equivalent to ϕ because every valuation of the edge and edge-set variables of ϕ corresponds to a valuation of the new variables of ϕ' and vice-versa.

□

Corollary 3.13. *There exists an algorithm which, given a MSO_2 sentence ϕ with q variables and an uncolored, unlabeled graph G with vertex cover at most k, decides if $G \models \phi$ in time $2^{2^{O(kq)}} + O(2^k n)$.*

3.6 Lower Bounds for Vertex Cover

In this Section we will prove some lower bound results for the model checking problems we are dealing with for vertex cover. Our proofs rely on a construction which reduces SAT to a model checking problem on a graph with small vertex cover.

Given a propositional 3-CNF formula ϕ_p with n variables and m clauses, we want to construct a graph G that encodes its structure, while having a small vertex cover. The main problem is encoding numbers up to n with graphs of small vertex cover but this can be achieved by using the binary representation of numbers. We will begin by constructing a colored graph and then briefly describe how the reduction can be strengthend to apply to uncolored graphs as well. Without loss of generality we will assume that n is a power of 2 (dummy variables can be added to ϕ_p if necessary).

We begin constucting a graph by adding $7 \log n$ vertices, call them $u_{(i,j)}, 1 \leq i \leq 7, 1 \leq j \leq \log n$. Add all edges of the form $(u_{(i,j)}, u_{(k,j)})$ (so we now have $\log n$ disjoint copies of K_7). Let $N_i = \{u_{(i,j)} \mid 1 \leq j \leq \log n\}$.

For every variable x_i in ϕ_p add a new vertex to the graph, call it v_i. Define for every number i the set $X(i) = \{j \mid \text{the j-th bit of the binary representation of i is 1}\}$. Add the edges $(v_i, u_{(1,j)}), j \in X(i)$, that is, connect every variable vertex with the vertices of N_1 that correspond to the binary representation of its index. Let $U = \{v_i \mid 1 \leq i \leq n\}$ be the vertices corresponding to variables.

For every clause c_i in ϕ_p add a new vertex to the graph, call it w_i. If the first literal in c_i is a positive variable x_k then add the edges $(w_i, u_{(2,j)}), j \in X(k)$. If the first literal is a negated variable $\neg x_k$, add the edges $(w_i, u_{(3,j)}), j \in X(k)$. Proceed in a similar way for the second and third literal, that is, if the second literal is positive connect w_i with the vertices that correspond to the binary representation of the variable in N_4, otherwise in N_5. For the third literal do the same with N_6 or N_7. Let $W = \{w_i \mid 1 \leq i \leq m\}$ be the vertices corresponding to clauses.

Finally, set the color classes to be $\{N_1, N_2, \ldots, N_7, U, W\}$.

Now, looking at the graph it is easy to see if a vertex v_i corresponds to a variable that appears positive in the clause represented by a vertex w_j. They must satisfy the formula

$$pos(v_i, w_j) = \bigvee_{k=2,4,6} \forall x : x \in N_1 \left(\exists y : y \in N_k((E(v_i, x) \leftrightarrow E(w_j, y)) \wedge E(x, y))\right)$$

It is not hard to define $neg(v_i, w_j)$ in a similar way. Now it is straight-forward to check if ϕ_p was satisfiable:

$$\begin{aligned} \phi \quad = \quad & \exists S(\forall x : x \in S(x \in U)) \wedge (\forall w : w \in W(\exists x : x \in U \\ & (((pos(x, w) \wedge x \in S) \vee (neg(x, w) \wedge x \notin S))))) \end{aligned}$$

Clearly, ϕ holds in the constructed graph iff ϕ_p is satisfiable. S corresponds to the set of variables set to true in a satisfying assignment. It is relatively easy to eliminate the colors and labels from the construction above. Colors can be reduced to labels by adding a labeled vertex for each color and connecting all the vertices that had that color to the labeled vertex. Finally, labels can also be eliminated by attaching a different FO-definable gadget to each labeled vertex. In particular, observe that all the vertices in our construction now have degree at least two. Thus, attaching a leaf to a vertex can be seen as labeling it (this can be expressed in FO logic). Similarly, we attach two leaves to the next vertex we want to label and so on. We only need a constant number of labels to simulate the constant number of color classes our construction uses. Therefore the lower bounds given below apply to the natural form of the problem.

Lemma 3.14. *$G \models \phi$ iff ϕ_p is satisfiable. Furthermore, ϕ has size $O(1)$ and G has*

a vertex cover of size $O(\log n)$.

Proof. Follows from the description of the construction.

□

Theorem 3.5. *Let* G *a graph with vertex cover* k*. Then, there exists a fixed MSO formula* ϕ *such that, unless 3-SAT can be solved in time* $2^{o(n)}$*, there is no algorithm which decides if* $G \models \phi$ *in time* $O(2^{2^{o(k)}} \cdot poly(n))$.

Proof. We have already observed that the construction we described has $k = O(\log n)$. Since the construction can clearly be performed in polynomial time, an algorithm running in time $O(2^{2^{o(k)}} \cdot poly(n))$ would imply an algorithm for SAT running in $2^{o(n)} \cdot poly(n)$.

□

Note that, since the formula used in Theorem 3.5 is fixed, it is also implied that a $O(2^{2^{o(k+q)}} \cdot poly(n))$ algorithm would also give a sub-exponential algorithm for SAT. Thus, Theorem 3.5 essentially matches the results of Corollary 3.11.

Theorem 3.6. *Let* ϕ *be a FO formula with* q_v *vertex quantifiers and* G *a graph with vertex cover* k*. Then, unless 3-SAT can be solved in time* $2^{o(n)}$*, there is no algorithm which decides if* $G \models \phi$ *in time* $O(2^{o(kq_v)} \cdot poly(n))$.

Proof. We use the same construction, but begin our reduction from Weighted 3-SAT, a well-known W[1]-hard parameterized problem. Suppose we are given a 3-CNF formula and a number w and we are asked if the formula can be satisfied by setting exactly w of its variables to true. The formula ϕ we construct is exactly the same, except that we replace the $\exists S$ with $\exists x_1 \exists x_2 \dots \exists x_w (\bigwedge_{1 \leq i < j \leq w} x_i \neq x_j)$ and all occurences of $x \in S$ with $\bigvee_{1 \leq i \leq w} x = x_i$. It is not hard to see that the informal meaning of ϕ now is to ask whether there exists a set of exactly w distinct variables such that setting them to true makes the formula true.

We now have $q_v = w + O(1)$ so an algorithm running in time $2^{o(kq_v)} \cdot poly(n)$ would imply an algorithm for Weighted 3-SAT running in $2^{o(w \log n)} \cdot poly(n) = n^{o(w)}$, and thus, by the results of Chen et al. [2004] that there exists a sub-exponential algorithm for 3-SAT.

□

3.7 Neighborhood Diversity

In this Section we give some general results on the new graph parameter we have defined, neighborhood diversity. We will use $nd(G), tw(G), cw(G)$ and $vc(G)$ to denote the neighborhood diversity, treewidth, cliquewidth and minimum vertex cover of a graph G. We will call a partition of the vertex set of a graph G into w sets such that all vertices in every set share the same type a neighborhood partition of width w.

First, some general results

Theorem 3.7. *1. Let $V_1, V_2, \ldots, V_w$ be a neighborhood partition of the vertices of a graph $G(V, E)$. Then each V_i induces either a clique or an independent set. Furthermore, for all i, j the graph either includes all possible edges from V_i to V_j or none.*

2. For every graph G we have $nd(G) \leq 2^{vc(G)} + vc(G)$ and $cw(G) \leq nd(G) + 1$. Furthermore, there exist graphs of constant treewidth and unbounded neighborhood diversity and vice-versa.

3. There exists an algorithm which runs in polynomial time and given a graph $G(V, E)$ finds a neighborhood partition of the graph with minimum width.

Proof. For the first statement, to show that every V_i induces either a clique or an independent set, we may assume that $|V_i| \geq 3$, otherwise the statement is trivial. Suppose that some V_i includes at least one edge (u, v). Consider another vertex

$w \in V_i$. The vertex w has the same type as u, therefore (w, v) must be an edge. Similarly, (w, u) must also be an edge, and generally all other vertices in V_i are connected to both u and v. Finally, if w, w' are two vertices of V_i other than u, v it must be the case that (w, w') is an edge, because (u, w') is an edge and u and w have the same type. Another way to see this observation is to say that the property of two vertices having the same type is an equivalence relation as observed in Lemma 3.2.

For the edges between V_i and V_j, suppose that there exists at least an edge (u, v) between them and let $w \in V_i$, $w' \in V_j$. v has the same type as w', therefore (u, w') must be an edge. Now, w has the same type as u so (w, w') must also be an edge, and once again this is true for any w, w'.

We have already shown the first part of the second statement. For the part with cliquewidth, we remind the reader that the graphs of cliquewidth k are those which can be constructed by repeated application of the following operations: introducing a new vertex with a label in $\{1, \ldots, k\}$, joining all vertices of label i with all vertices of label j, renaming all vertices of label i to label j and taking disjoint union of two graphs of cliquewidth at most k. We must show how to construct a graph in such a way starting from a neighborhood partition of width w, using at most $w + 1$ labels. The labels in $\{1, \ldots, w\}$ will only be used for the vertices of the corresponding set in the partition, while the extra label will be used to construct the cliques. For each V_i, if V_i is an independent set introduce $|V_i|$ new vertices with label i. If V_i is a clique repeat $|V_i|$ times: introduce a new vertex of label $w + 1$, join all vertices of label i to $w + 1$ and rename $w + 1$ to i. After all the vertices have been introduced, for all i, j for which the graph had all edges between V_i and V_j join the vertices labeled i with those labeled j.

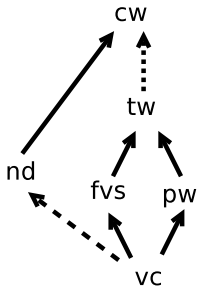
To see why treewidth is incomparable to neighborhood diversity consider the examples of a complete bipartite graph $K_{n,n}$ and a path on n vertices.

Finally, let us argue why neighborhood diversity is computable in polynomial

time. As mentioned already, the property of two vertices having the same type is an equivalence relation. The number of sets in an optimal partition is equal to the number of equivalence classes, so we simply need to determine these. It is easy to see that one can check if two vertices have the same type in polynomial time, so dividing the vertices into equivalence classes can also be done in polynomial time by checking all pairs of vertices.

□

Taking into account the observations of Theorem 3.7 we summarize what we know about the graph-theoretic and algorithmic properties of neighborhood diversity and related measures in Figure 3.1.



	FO	MSO	MSO_2
Cliquewidth	$tow(w)$	$tow(w)$	$tow(w)$
Treewidth	$tow(w)$	$tow(w)$	$tow(w)$
Vertex Cover	$2^{O(w)}$	$2^{2^{O(w)}}$	$2^{2^{O(w)}}$
Neighborhood Diversity	$poly(w)$	$2^{O(w)}$	Open

Figure 3.1: A summary of the relations between neighborhood diversity and other graph widths. Included are cliquewidth, treewidth, pathwidth, feedback vertex set and vertex cover. Arrows indicate generalization, for example bounded vertex cover is a special case of bounded feedback vertex set. Dashed arrows indicate that the generalization may increase the parameter exponentially, for example a graph of treewidth w has cliquewidth at most $O(2^w)$ and this is known to be tight. The table summarizes the best known model checking algorithm's dependence on each width for the corresponding logic.

There are several interesting points to make here. First, though this work is motivated by a specific goal, beating the lower bounds that apply to graphs of bounded treewidth by concentrating on a special case, it seems that the results which can achieved are at least somewhat better; it is possible to prove stronger meta-theorems by focusing on a class which is not necessarily smaller than bounded treewidth, only different. However, this class is a special case of another known width which gen-

eralizes treewidth as well, namely cliquewidth. Since the lower bound results which apply to treewidth apply to cliquewidth as well, this work can perhaps be viewed more appropriately as an improvement on the results of Courcelle et al. [2000] for bounded cliquewidth graphs when restricting our attention to the more special case of bounded neighborhood diversity.

Second, there is the case of MSO_2 logic. The very interesting hardness results shown in Fomin et al. [2009, 2010] demonstrate that the tractability of MSO_2 logic is in a sense the price one has to pay for the additional generality that cliquewidth provides over treewidth. It is natural to ask if these results can be strengthened to apply to neighborhood diversity or MSO_2 logic can be shown to be tractable when parameterized by neighborhood diversity.

Though we cannot yet fully answer the above question related to MSO_2, we can offer some first indications that this direction might merit further investigation. In Fomin et al. [2009] it is shown that MSO_2 model checking is not fixed-parameter tractable when the input graph's cliquewidth is the parameter by considering three specific MSO_2-expressible problems and showing that they are W-hard. The problems considered are Hamiltonian cycle, Graph Chromatic Number and Edge Dominating Set. We can show that these three problems admit FPT algorithms on graphs of small neighborhood diversity (for Hamiltonian cycle this is in fact an easy consequence of an old result from Cosmadakis and Papadimitriou [1984]). Since small neighborhood diversity is a special case of small cliquewidth, where these problems are hard, this result could be of independent interest.

Theorem 3.8. *Given an n-vertex graph G whose neighborhood diversity is w, there exist algorithms running in time $O(f(w) \cdot poly(n))$ that decide Hamiltonian cycle, Graph Chromatic Number and Edge Dominating Set.*

Proof. We will make use of an auxiliary graph G' on w vertices. Each vertex of G' corresponds to a set in an optimal neighborhood partition of G and two vertices of G'

have an edge iff the corresponding sets of the partition of G have all possible edges between them.

For Hamiltonian cycle we rely on the results of Cosmadakis and Papadimitriou [1984]. There it is shown that the TSP problem is FPT parameterized by the number of cities even when one has to visit each city a number of times given in the input. We take G' and add a self-loop to every vertex that corresponds to a clique in G. The number of times we want to visit each vertex of G' is equal to the size of the corresponding set of the neighborhood partition. We set the cost of each edge to 1 and each non-edge to 2 and solve the resulting TSP instance on G'. G is Hamiltonian iff there is a TSP tour on G' with cost n.

Let us now show how to solve graph coloring. Observe that if a set V_i of a neighborhood partition of G induces an independent set, we can delete all of its vertices but one, without affecting the graph's chromatic number, because there always exists an optimal coloring where all the vertices of V_i take the same color. So, we can assume without loss of generality that all the sets V_i of a neighborhood partition of G induce cliques (some of them of order one).

We are now going to reformulate the problem, using the fact that in any coloring of G every color class intersects each set of the neighborhood partition in at most one vertex (since all the sets of the partition induce cliques). In other words, every color class essentially coincides with an independent set of G'. Let $\mathcal{I}$ be the set of all independent sets of G' and let V_p be the set of vertices of G', each of which represents a set in the neighborhood partition of G. Consider the following ILP with variables $x_I,\ I \in \mathcal{I}$ (i.e. at most 2^w variables):

$$\min \sum_{I \in \mathcal{I}} x_I$$
$$\text{s.t. } \forall v \in V_p : \quad \sum_{I : v \in I} x_I = |V_p|$$

Intuitively, in the above ILP the variables x_I encode how many different color classes coincide with the independent set I of G' in a coloring of G.

We argue that the optimal solution to this problem is exactly the chromatic number of G. First, suppose that there exists a coloring of G with c colors. Every color class induces an independent set, so by looking at the collection of sets of the neighborhood partition that the class intersects we have that every color class corresponds to some $I \in \mathcal{I}$. Several color classes may correspond to the same set I, so we set x_I to be equal to the number of color classes that correspond to the set I. It should be easy to see then that $\sum x_I = c$. The requirement that $|V_i| = \sum_{I:i\in I} x_I$ is satisfied because every vertex belongs in exactly one color class.

For the other direction, suppose that there exist integers x_I which satisfy the ILP and $\sum x_I = s$. We can produce a coloring of G with s colors: as long as there exists an I with $x_I > 0$ select a new color and arbitrarily pick exactly one uncolored vertex from each V_i with $i \in I$. Color these vertices with the new color, and set $x_I := x_I - 1$. It is not hard to see that this algorithm will use exactly s colors. It produces a valid coloring because in the beginning we have $|V_i| = \sum_{I:i\in I} x_I$ and the equality continues to hold at each step if we only count the uncolored vertices on the left-hand side of the equation.

Thus, intuitively we have reformulated the problem as one of selecting the independent sets that will form the color classes. The main observation now is that the possible choices for the independent sets are only 2^w. Thus, we can recast the problem as an Integer Linear Program with at most 2^w variables and w constraints for the equations $|V_i| = \sum_{I:i\in I} x_I$. It follows from a seminal result of Lenstra (Lenstra Jr [1983]) that solving this can be performed in FPT time.

In the edge dominating set problem, we are asked to find a set of edges of minimum size such that all other edges share an endpoint with one of the edges we selected. This problem is equivalent to the minimum maximal matching problem, where we

are trying to find a minimum size independent set of edges that cannot be extended by picking another edge of the graph. To see why the optimal solution to the edge dominating set problem can always be transformed to a matching, suppose that we have a solution S which includes two edges $(u, v), (u, v')$. Now, if all the neighbors of v' are incident on an edge of S we can simply remove (u, v') from S and improve the size of the solution. If there is a neighbor w of v' that is not incident on an edge of S we can replace (u, v') with (w, v') in S. To see why a solution to the edge dominating set problem can always be transformed to a matching that is maximal, suppose that the matching we got was not maximal. Then there would be two unmatched vertices connected by an edge, which would imply that this edge is not dominated.

Our algorithm will proceed as follows: for every vertex cover V' of G' repeat the following (there are at most 2^w vertex covers to be considered): from V' infer a vertex cover of G by placing into the vertex cover all the vertices that belong in a type whose corresponding vertex is in V'. Also place in the vertex cover all but one (arbitrarily chosen) vertex of every vertex type that induces a clique but whose corresponding vertex is not in V'. Denote the resulting vertex cover of G by V''. Find a maximum matching on the graph induced by V'', call it M_1. Take the bipartite graph induced by the unmatched vertices of V'' and $V \setminus V''$ and find a maximum matching there, call it M_2. The solution produced is $M_1 \cup M_2$. After repeating this for all vertex covers of G', pick the smallest solution.

Now we need to argue why this solution is optimal. Let S be an optimal solution for G. We say that a set of the neighborhood partition V_i is full if all of its vertices are incident on edges of S. If we take in G' the corresponding vertices of the full sets of G, they must form a vertex cover of G', otherwise there would be two neighboring vertices with neither having any edge of S incident to it, which would mean that S is not maximal. This is a vertex cover of G' considered by our algorithm, since our algorithm considers all vertex covers of G', call it V'. Let V'' be again the vertex

cover of G our algorithm derived from V' by also including a minimal number of vertices from each remaining clique. Let V^* be the set of vertices of G incident on some edge of S, which must also be a vertex cover of G. Without loss of generality we will assume that $V'' \subseteq V^*$, because the two vertex covers of G agree on taking all vertices of the full sets and V'' takes a minimal number of vertices from every other clique. Even if V^* leaves out a different vertex from some clique because all the vertices of the clique have the same neighbors we can apply an exchanging argument and transform S appropriately without increasing its size so that both sets leave out the same vertex.

Now note that $|M_2| \leq |V''| - 2|M_1|$. So our algorithm's solution has size at most $|V''| - |M_1|$. On the other hand the optimal solution S includes some edges with both endpoints in V'', call this set S_1. Because M_1 is a maximum matching, $|S_1| \leq |M_1|$. From what we have so far, the fact that all vertices of V^* are matched by S and the fact that V'' is a vertex cover, so $V^* \setminus V''$ induces no edges we have $|V^*| = |V^* \cap V''| + |V^* \setminus V''| = |V''| + |V''| - 2|S_1| \geq 2|V''| - 2|M_1|$. This implies that $|S| \geq |V''| - |M_1|$ which concludes the proof.

□

Chapter 4

Directed Graphs

In this chapter we will present some parameterized complexity results concerning directed graphs. As mentioned in the previous chapter, much work has been devoted to the investigation of the algorithmic properties of (undirected) treewidth and its variations. Much of this work can be applied in a straightforward way to digraph problems: simply consider the treewidth of the underlying undirected graph (that is, the graph obtained if we ignore the directions of all the edges) as the parameter. This approach, however, feels less than satisfactory, since much of the structure of the input is ignored. For example, the Directed Hamiltonian Circuit problem is always easy on DAGs, regardless of the underlying graph's treewidth, which may be arbitrarily high. Thus, this motivates the question of whether one can define a width more finely tuned to digraphs, which would more accurately characterize the "easy" families of instances.

As a result, several (competing) definitions of directed treewidths have appeared in the literature. In this chapter we will present two hardness results which affect essentially all known definitions of directed treewidth: we show that Directed Hamiltonian Circuit is W-hard parameterized by the input graph's directed width and that Max Di Cut is NP-hard even for DAGs. Together with other hardness

results which have appeared in the literature more recently, these give evidence that the proposed digraph widths are unlikely to be as successful as treewidth. The results presented here have appeared in Lampis et al. [2008] and Lampis et al. [2011].

4.1 Previous Work

Several attempts have been made recently to define a treewidth for digraphs. The most notable such variations of treewidth that have been proposed in the past are probably directed treewidth (Johnson et al. [2001]), DAG-width (Obdrzálek [2006]) and Kelly-width (Hunter and Kreutzer [2007]). All these three measures can be viewed as good generalizations of treewidth in the sense that, if we take an undirected graph and replace each edge with two opposite directed edges the width of the new digraph will be the same for all three definitions and equal to the treewidth of the original graph. Directed treewidth is the most general of the three, in the sense that a graph of bounded Kelly-width or DAG-width will also have bounded directed treewidth, while the converse may not be true. Also DAG-width and Kelly-width are conjectured to be only a constant factor apart on any graph (Hunter and Kreutzer [2007]).

Thus, Kelly-width and DAG-width have the potential to provide better algorithmic properties than directed treewidth and some evidence is given in this direction in the form of an algorithm for solving a class of parity games, a problem that is open so far for directed treewidth (note though that this algorithm is not FPT, and that the problem is not believed to be NP-complete).

The most important positive result of directed treewidth (which can be extended to all the three measures) is an algorithm that solves DIRECTED HAMILTONIAN CIRCUIT in $O(n^{f(k)})$ time, k being the width of the input graph. Nevertheless, this algorithm is still far from the performance of the best treewidth-based algorithm for HAMILTONIAN CIRCUIT, which runs in fixed-parameter linear time. Unfortunately,

the reason for this distance is not addressed in Johnson et al. [2001] or in Hunter and Kreutzer [2007] where another algorithm (of similar complexity) for this problem is given. In addition, the few already known algorithmic results on these measures don't seem to indicate that they are likely to achieve a level of success comparable to treewidth, as no FPT algorithms are known for any hard digraph problems.

More recently in Kreutzer and Ordyniak [2008] the authors investigate the concepts of DAG-width and Kelly-width more closely and prove several interesting results: First, they show that the cops-and-robber games associated with both measures are non-monotone, which draws a contrast with the case of treewidth whose associated cops-and-robber games have been shown to be monotone. Second, they show that several problems which are polynomially solvable for DAGs are still NP-complete even for graphs of constant Kelly-width and DAG-width. Yet more hardness results are shown in Dankelmann et al. [2009] where the MINIMUM LEAF OUTBRANCHING problem is shown to be NP-complete even for constant width, even though it is polynomially solvable on DAGs.

A related measure is directed pathwidth. Just as pathwidth is a restriction of treewidth in the undirected case directed pathwidth is a restriction of all the previously mentioned directed measures, thus having even greater algorithmic potential. However, to the best of our knowledge no such results have been shown for directed pathwidth. In Barát [2006] it is shown that a cops-and-robber game is equivalent to directed path-width and that there always exists an (almost) optimal monotone strategy. It is worthy of note that, unlike the undirected case where treewidth and pathwidth are generalizations of different graph topologies (trees and paths respectively) in the directed case all the measures we have mentioned are based on the concept of DAGs as the simplest case.

It is also worth noting the existence of a related digraph complexity measure which is often overlooked in this discussion: cycle rank. Cycle rank was first defined in the

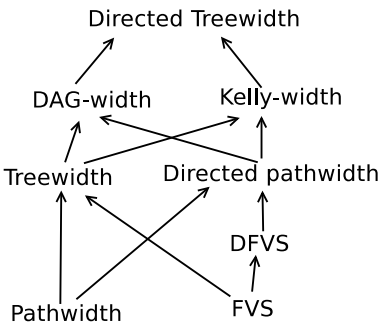


Figure 4.1: Summary of relations between digraph widths. Arrows indicate generalizations. DFVS stands for directed feedback vertex set. The undirected widths are taken with respect to the underlying undirected graph (ignoring arc directions).

1960s in Eggan [1963] and it has mainly found applications in the context of regular languages (it has been shown that the star height of a regular language is connected to the cycle rank of the NFAs which accept it). As pointed out in Gruber and Holzer [2008] cycle rank is also relevant in our discussion here, since it can be shown (Gruber and Holzer [2007]) that the directed pathwidth of a graph is upper bounded by its cycle rank.

Finally, let us point out that the current state-of-the art in the area of digraph widths is nicely summarized in Ganian et al. [2009, 2010]. In these two papers it is discussed in even more depth whether the currently proposed definitions of directed treewidth, or even some other variation thereof, could give algorithmic results comparable to those of undirected treewidth. In short, the answer seems to be no, as Ganian et al. [2010] presents evidence that any digraph width which satisfies some basic algorithmic and graph-theoretic properties must necessarily be essentially the same as (undirected) treewidth.

The relations between the mentioned digraph widths are summarized in Figure 4.1.

4.2 Preliminaries

As mentioned, the main results of this section are negative: we will present hardness results for DIRECTED HAMILTONIAN CIRCUIT and MAX DI CUT, affecting their parameterizations by directed treewidth. We will also give an extension that proves a hardness result for the related MINIMUM LEAF OUTBRANCHING problem. But first, we need to give all the necessary definitions.

First, let us give the definitions of the two problems that will be our focus.

Definition 4.1. The DIRECTED HAMILTONIAN CIRCUIT problem is that of deciding whether there exists a permutation $(v_1, v_2, \ldots, v_n)$ of the vertices of an input digraph $G(V, E)$ s.t. $\forall i \in \{1, \ldots, n-1\}$ $(v_i, v_{i+1}) \in E$ and $(v_n, v_1) \in E$.

Definition 4.2. The MINIMUM LEAF OUTBRANCHING problem is the following: given a directed graph $G(V, E)$, find an outbranching (a spanning rooted directed tree) such that the number of leaves of the tree is minimized. (see Dankelmann et al. [2009])

Definition 4.3. The MAX DI CUT problem is the following: given a digraph $G(V, E)$ and a weight function on the edges $w : E \to \mathbb{N}$, find a partition of V into two sets V_0 and V_1 so that the weight of the edge set $C = \{(u, v) \mid u \in V_0, v \in V_1\}$ is maximized. That is, the objective is to maximize $\sum_{e \in C} w(e)$.

MAX DI CUT was shown APX-hard in Papadimitriou and Yannakakis [1991]. Here we show APX-hardness for the problem's restriction to DAGs. Then we show that APX-hardness also holds for the cardinality version of the problem restricted to DAGs.

We should also recall the definitions of the two problems that will be the starting points of our reductions.

Definition 4.4. DOMINATING SET is the problem of finding a minimum cardinality

subset of vertices D of an undirected graph $G(V, E)$ s.t. any vertex in $V \setminus D$ has a neighbor in D.

When a vertex $u \in D$ is a neighbor of a vertex v, we will say that u dominates v. We will also follow the convention of saying that any vertex in D dominates itself. We will make use of the well-known result that DOMINATING SET is $W[2]$-complete when the parameter k is the size of the dominating set we are looking for (Downey and Fellows [1999]).

Definition 4.5. NAE3SAT is the problem of finding a truth assignment which, for every clause of an input 3CNF formula, assigns the value true to at least one literal, and the value false to at least one literal.

We follow the convention of saying that a clause is satisfied in the NAESAT sense, or simply satisfied, when a truth assignment assigns different truth values to two of its literals. We will mainly be concerned with the maximization version of NAE3SAT where the objective is to find a truth assignment that satisfies as many clauses as possible. This variant was shown to be APX-hard in Papadimitriou and Yannakakis [1991].

We have already mentioned that directed pathwidth can be defined in terms of a cops-and-robber game. The game's definition is the following:

Definition 4.6. The k-cop invisible-eager robber game is the game where k cops attempt to catch an invisible robber hiding in a vertex of a digraph G. The cops are stationed on vertices of G and a cop can move by removing himself from the graph and then "landing" on any other vertex. The robber can move at any time and he is allowed to follow any directed path of G, under the condition that he does not enter vertices occupied by stationary cops.

We say that k cops have a monotone strategy to win this game when they have a strategy such that the robber can never visit a vertex previously occupied by a cop.

In Barát [2006] it was shown that k cops have a monotone strategy on a graph G iff the graph has directed pathwidth k.

Kelly-width, DAG-width and directed treewidth have also been shown to be connected to similar games, restricted to monotone strategies. In fact, DAG-width is equivalent to the above game but with the robber being visible, while Kelly-width is equivalent to the above game but with the robber only being allowed to move when a cop enters his vertex. Using the approximate connection between directed treewidth and a similar game it was shown in Hunter and Kreutzer [2007] that the directed treewidth of a graph is upper-bounded by its Kelly-width multiplied by a constant.

It is not hard to infer from these results that, since the robber is stronger in the game related to directed pathwidth, a graph G will have higher pathwidth than any of the other widths. Since we are interested in proving hardness results, it will therefore suffice to show that a problem is hard for graphs of small directed pathwidth and hardness for the other widths will directly follow.

In addition to the above widths we may also wish to consider cycle rank. Cycle rank can be defined inductively as follows: if $G(V, E)$ is acyclic then $cr(G) = 0$, if G is strongly connected then $cr(G) = 1 + \min_{v \in V} cr(G - v)$ and finally if G is not strongly connected then $cr(G)$ is equal to the maximum cycle rank of any of G's strongly connected components. As mentioned, it has been shown in Gruber and Holzer [2007] that in any digraph G the cycle rank is lower bounded by the directed pathwidth (more precisely, this relation holds up to an additive constant), therefore showing a hardness result for bounded cycle rank immediately implies hardness for all the widths we have mentioned. For the sake of completeness here is another short proof of the relation between cycle rank and directed pathwidth.

Lemma 4.7. *For any digraph G, $dpw(G) \leq cr(G) + 1$, where $dpw(G)$ denotes the directed pathwidth of G and $cr(G)$ the cycle rank of G.*

Proof. By induction on $cr(G)$. If $cr(G) = 0$ then G is acyclic and $dpw(G) \leq 1$.

Suppose that the relation is true for all graphs of cycle rank up to k. Consider a graph G with $cr(G) = k+1$. If it is strongly connected then there exists a vertex v such that $cr(G-v) = k$. From the inductive hypothesis this implies $dpw(G-v) \leq k+1$, which means that $k+1$ cops have a wining monotone strategy for $G-v$. Then $k+2$ cops have a winning strategy for G: just keep the extra cop in v at all times. If G is not strongly connected there must exist an ordering of its strongly connected components, so that edges with endpoints in different components are always directed towards components later in the ordering. Applying the same argument to each component in this order we obtain a monotone winning strategy for the cops, because at any time the robber can either remain in the component he currently is (where the cops have a strategy) or move to a component later in the ordering (which means he can never come back).

□

4.3 Directed Hamiltonian Circuit

In this section we focus on the DIRECTED HAMILTONIAN CIRCUIT problem, a problem which can be solved using directed treewidth in $O(n^{f(k)})$ time (Johnson et al. [2001]). Of course this algorithm also applies to DAG-width, Kelly-width and directed pathwidth, as they are restrictions of directed treewidth. In addition, another $O(n^{f(k)})$ algorithm for this problem tailored for Kelly-width is given in Hunter and Kreutzer [2007]. Thus, a significant gap exists between the performance of treewidth, which is fixed-parameter polynomial on the corresponding undirected problem and the performance of its directed variants. We show that this is a gap that can not be bridged unless $W[2] = FPT$, by demonstrating that DIRECTED HAMILTONIAN CIRCUIT is $W[2]$-hard when the parameter is any of these widths.

The hardness proof for DIRECTED HAMILTONIAN CIRCUIT will be a parameterized reduction from the naturally parameterized version of DOMINATING SET.

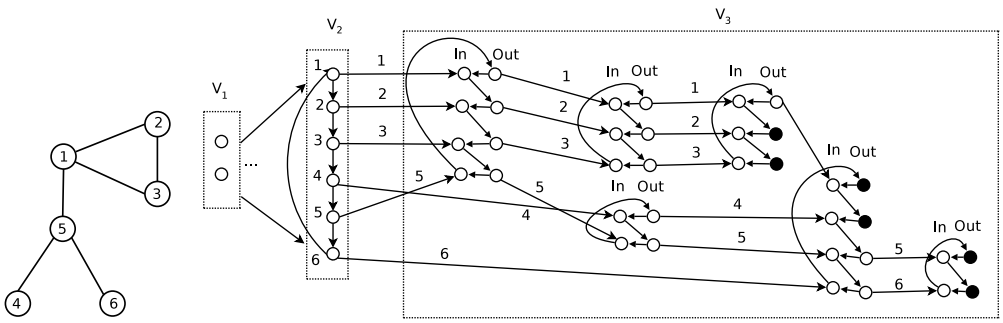


Figure 4.2: An example of our construction. The graph on the left is the original undirected graph and we are looking for a dominating set of size 2. The three parts of the produced graph are outlined. To simplify the figure some edges are not shown: the dark vertices of V_3 are the vertices which are connected to all the vertices of V_1. The gadgets C_1, C_2 and C_3 are on top in V_3, while C_4, C_5 and C_6 are below.

Theorem 4.1. *The parameterized versions of* DIRECTED HAMILTONIAN CIRCUIT*, where the parameter is the directed treewidth, Kelly-width, DAG-width, directed path-width or cycle rank of the input graph, are $W[2]$-hard.*

Proof. We will show a parameterized reduction from the naturally parameterized version of DOMINATING SET, where the parameter k is the size of the set by constructing a digraph whose cycle rank is bounded by a function of k s.t. the digraph will be Hamiltonian iff the original graph had a dominating set of size k.

Suppose we are given a graph $G(V, E)$ with $V = \{1, 2, \ldots, n\}$ and need to decide whether G has a dominating set of size k. Note that we assume that V is ordered in some way. The ordering may be arbitrary, as long as we fix it in the beginning.

Our digraph G' has vertex set $V' = V_1 \cup V_2 \cup V_3$ where

1. $V_1 = \{u_1, u_2, \ldots, u_k\}$.
2. $V_2 = \{v_1, v_2, \ldots, v_n\}$.
3. $V_3 = \{g_{(w,j,p)} \mid w \in \{1, \ldots, n\}, j \in \mathcal{N}[w], p \in \{In, Out\}\}$. Here $\mathcal{N}[w]$ denotes the neighborhood of vertex w in G including w itself (i.e. all the vertices that w dominates). $\mathcal{N}[w]$ is an ordered set according to the above mentioned ordering. It is also supplied with the constants f_w and l_w which denote the first and the

last elements in $\mathcal{N}[w]$ respectively and the operation $s_w(j)$ which outputs the element that comes after j in $\mathcal{N}[w]$ according to the ordering.

$E(G')$ consists of the following sets of directed edges

1. $E_1 = \{(u_i, v_j) \mid i \in \{1, \ldots, k\}, j \in \{1, \ldots, n\}\}$.
2. $E_2 = \{(v_i, v_{i+1}) \mid i \in \{1, \ldots, n\}\}$ (where we consider $n+1$ to be the same as 1).
3. $E_3 = \{(g_{(w,j,In)}, g_{(w,s_w(j),Out)}) \mid w \in \{1, \ldots, n\}, j \in \mathcal{N}[w]\}$, where $s_w(l_w) = f_w$.
4. $E_4 = \{(g_{(w,j,Out)}, g_{(w,j,In)}) \mid w \in \{1, \ldots, n\}, j \in \mathcal{N}[w]\}$.
5. Finally, E_5 contains the following edges: For any vertex w of the original graph G, the edge $(v_w, g_{(f_w,w,In)})$ and the edges $(g_{(j,w,Out)}, g_{(s_w(j),w,In)})$ for all $j \in \mathcal{N}[w]$ are included in E_5. Finally, the edges $(g_{(l_w,w,Out)}, u_i)$ for all $i \in \{1, \ldots, k\}$ are also included in E_5. We will call the subgraph induced by the group of vertices $g_{(w,j,p)}$ for a specific w, the gadget C_w.

Let us now discuss the basic idea behind this construction, before we get into more details. Our digraph G' consists of three parts: a constraint part V_1, a choice part V_2 and a satisfaction part V_3. V_1 functions as a constraint part because it only has k vertices and the only edges going into V_2 originate here, thus forcing us to enter the choice part exactly k times. A Hamiltonian tour will leave V_2 k times. The vertices from which it leaves V_2 must be (as we will prove) a dominating set of G, and that is why V_2 is the choice part. Finally, V_3 is arranged in such a way that it can only be traversed in a Hamiltonian way if the choice made in V_2 is indeed a dominating set.

It is clear that every gadget C_w is a directed cycle of $2 \cdot |\mathcal{N}[w]|$ vertices. Furthermore, the gadget C_i is connected to the gadget C_j iff there exists a vertex w in the original graph such that $i, j \in \mathcal{N}[w]$ and $j = s_w(i)$. Also notice that all edges between gadgets connect vertices having the same second coordinate and any vertex v_w of V_2 is only connected with vertices of the gadgets having w as the second coordinate.

Figure 4.2 gives an example of our construction and makes it clear how the edges of E_5 are placed. For example, consider the edges we place for vertex 5 of the original graph. $\mathcal{N}[5] = \{1, 4, 5, 6\}$ in the original graph. So we must have a directed edge from v_5 to C_1, from C_1 to C_4, from C_4 to C_5, from C_5 to C_6 and C_6 back to both vertices of V_1. In order to do so we connect the vertices of each gadget that correspond to 5. Such a vertex exists in every gadget C_1, C_4, C_5, C_6 according to the construction of V_3.

The crucial part of this reduction is the way the gadget C_w works. Notice that the gadget's vertices induce a directed cycle. Also, the only way to enter this cycle is through an In vertex, and the only way to leave is through an Out vertex. Suppose that a Hamiltonian tour enters a gadget C_w m times and that $X \subseteq \mathcal{N}[w]$ is the index set of the In vertices which were used. Then it must also be the index set of the Out vertices used. To see that, suppose that $X = \{j_1, j_2, \ldots, j_m\}$ in increasing order. When entering from $g_{(w,j_1,In)}$ the tour has no choice but to proceed to $g_{(w,s_w(j_1),Out)}$. Then if $s_w(j_1) \neq j_2$ the tour must move to $g_{(w,s_w(j_1),In)}$, because if it were to exit, it would be impossible to visit $g_{(w,s_w(j_1),In)}$ in the future. Using this argument again can exclude the possibility of this part of the tour exiting through any vertex other than $g_{(w,j_2,Out)}$. Similarly, the path that starts at $g_{(w,j_2,In)}$ will exit at $g_{(w,j_3,Out)}$ and so on, with $g_{(w,j_m,In)}$ exiting through $g_{(w,j_1,Out)}$. This procedure covers all the vertices of the gadget, therefore we proved that, for any set of entry vertices X, the gadget can be traversed in a way that does not exclude the existence of a Hamiltonian tour of the whole graph iff X corresponds also to the exit vertices used.

Suppose that G does not have a dominating set of size k, but that a Hamiltonian tour of G' exists. As noticed, a Hamiltonian tour will traverse V_2 in total k times. Let D be the set of choices made by the tour in V_2, i.e. the set of vertices through which the tour exits V_2. The selection of the set D in G leaves some vertex not dominated, say vertex w. Consider the gadget C_w. Since the tour is Hamiltonian, the gadget C_w

should be traversed. Suppose that the Hamiltonian tour enters C_w through vertex $g_{(w,q,In)}$. That means that q belongs in $\mathcal{N}[w]$. Combining the previously established properties of the gadgets, the Hamiltonian tour enters and exits every gadget using only vertices having second coordinates from the set D. From this we conclude that q belongs to D. Thus, we have a contradiction since D dominates w.

It remains to prove the converse, namely that a dominating set of size k implies a Hamiltonian tour. Let $D = \{d_1, d_2, \ldots, d_k\}$ be a dominating set. Informally, these will be exactly the vertices through which our tour will exit V_2. Also, because of the construction of the C_w gadgets, if such a gadget through a set of In vertices it is possible to traverse it in a Hamiltonian way exiting exactly from the same set of corresponding Out vertices. Keeping that into account we will have to show that all C_w gadgets are entered at least once, but that follows from the fact that D is a dominating set.

Let us first describe the tour outside the gadgets. Starting at u_1, move to v_{d_k+1} (once again, v_{n+1} is the same as v_1) and then follow the edges in V_2 until v_{d_1} is reached. Then we exit V_2 towards the gadgets. When we reach an *Out* gadget vertex that points to V_1 we move to u_2. From there we move to v_{d_1+1}, then to v_{d_2} and so on. This procedure makes sure that, even though we enter V_2 only k times, all of its n vertices are covered.

Let us now describe the traversal of the gadgets, starting from gadget 1. First note that $D \cap \mathcal{N}[1] \neq \emptyset$ because D is a dominating set. It is not hard to see that our tour will enter gadget C_1 through vertices $g_{(1,d_j,In)}$ for all j such that $d_j \in D \cap \mathcal{N}[1]$, since $f_j = 1$ for all these j and we leave V_2 only from exit points corresponding to D. Once inside the gadget at the vertex $g_{(1,d_j,In)}$ our tour follows the unique path to $g_{(1,d_l,Out)}$, where d_l is the next element of $D \cap \mathcal{N}[1]$ according to the ordering (or the first element of $D \cap \mathcal{N}[1]$ if d_j is the last). Note that if $|D \cap \mathcal{N}[1]| = 1$ then $d_l = d_j$. Thus, all vertices of gadget C_1 are visited exactly once and the gadget is

exited through vertices corresponding to $D \cap \mathcal{N}[1]$.

We will now inductively prove the same for all gadgets. Suppose that for all gadgets up to gadget C_i we have shown that all their vertices are visited exactly once and the gadgets are entered and exited through vertices corresponding to $D \cap \mathcal{N}[i]$. Let us now consider gadget C_{i+1}. Once again $D \cap \mathcal{N}[i+1] \neq \emptyset$ because D is a dominating set. The gadget C_{i+1} is entered only through vertices $g_{(i+1,d_j,In)}$ such that $d_j \in D \cap \mathcal{N}[i+1]$ because the only edges going into gadget C_{i+1} originate in V_2 or one of the previous gadgets for which we have assumed that they are exited through vertices corresponding to D. Once inside gadget C_{i+1} we follow a similar tour as in gadget 1; starting from vertex $g_{(i+1,d_j,In)}$ we follow the unique path to $g_{(i+1,d_l,Out)}$ and leave the gadget, where d_l is the element of $D \cap \mathcal{N}[i+1]$ which comes after d_j (or the first element of $D \cap \mathcal{N}[i+1]$ if d_j is the last). With the same reasoning as previously, all vertices of gadget C_{i+1} are visited exactly once and the gadget is exited only through vertices corresponding to $D \cap \mathcal{N}[i+1]$. This completes the proof that a Hamiltonian tour can be constructed.

Finally, what is left is to argue is that G' has low width.

First, notice that $cr(G') \leq |V_1| + cr(G' - V_1) = k + cr(G' - V_1)$. But $G' - V_1$ is not strongly connected and all its strongly connected components are directed cycles (V_2 and the gadgets C_i). Therefore, $cr(G' - V_1) \leq 1$.

□

We modify the above reduction in order to prove the following theorem

Theorem 4.2. *The parameterized version of* MINIMUM LEAF OUTBRANCHING *where the parameter is the number of the leaves of the outbranching is W[2]-hard even when restricted to graphs with constant cycle rank.*

Proof. We reduce the DOMINATING SET problem to the MINIMUM LEAF OUTBRANCHING problem. We modify the construction of the graph G' of theorem 4.1,

adding a vertex r with arcs pointing to the k vertices of set V_1 and deleting those edges of E_5 that connect the gadgets with V_1. We name the new graph G''.

Vertex r will definitely serve as the root of the outbranching since it is a source. We prove that G has a dominating set of size k iff G'' has an outbranching with k leaves.

Suppose that there is a dominating set of size k in G. From theorem 4.1 we have that G' has a hamiltonian cycle which uses k edges of E_5 that connect V_3 with V_1. Those edges are missing from G''. Therefore there are k disjoint paths from V_1 to V_3 that cover all the vertices of G'. Thus there is an outbranching with root r with k leaves.

Furthermore suppose that G'' has an outbranching T with at most k leaves. Notice that, since r is its root and there are no arcs from V_2 or V_3 to V_1, all the k arcs from r to V_1 are contained in T. Thus there are exactly k disjoint paths in T, thus exactly k leaves. Notice that if the k leaves are vertices of V_3 that connect to V_1 in G' then from T we can construct a hamiltonian circuit in G', which can help find a dominating set of size k in G (by theorem 4.1). Name these vertices of V_3 that connect to V_1 in G' black vertices. We prove that from any outbranching T with k leaves we can construct an outbranching T' with k black leaves.

First of all we can assume that T has no leaves in V_1 or V_2. If there was a leaf u_i in V_1 and the arc (v_{j-1}, v_j) is part of T then we could add the arc (u_i, v_j) and remove the arc (v_{j-1}, v_j) from T so u_i wouldn't be a leaf anymore. If there was a leaf v_j in V_2, following a similar procedure as above we could add the arc $(v_j, g_{(f_j,j,In)})$ and remove the arc $(g_{(f_j,j,Out)}, g_{(f_j,j,In)})$, so v_j wouldn't be a leaf anymore. Furthermore notice that there is no way that a vertex $g_{(i,w,In)}$ could be a leaf since vertex $g_{(s_w(i),w,Out)}$ could not be reached. So wlog we can assume that all the leaves of T are out vertices of the gadgets.

We show by induction on the gadgets that we can eliminate all non-black leaves

from T. For every gadget i starting from gadget 1 up to $n-1$ we eliminate all the black leaves from gadget i. Suppose that there is a leaf $g_{(i,w,Out)}$ in T which is not a black vertex. Then the arc $(g_{(i,w,Out)}, g_{(s_w(i),w,In)})$ is not in T. However, vertex $g_{(s_w(i),w,In)}$ is in T, thus the arc $(g_{(s_w(i),w,Out)}, g_{(s_w(i),w,In)})$ should be in T. By removing $(g_{(s_w(i),w,Out)}, g_{(s_w(i),w,In)})$ and then adding $(g_{(i,w,Out)}, g_{(s_w(i),w,In)})$ we assure that $g_{(i,w,Out)}$ is not a leaf anymore while making sure that this procedure doesn't create non-black leaves in gadgets $1 \dots i-1$. We repeat the procedure until no non-black leaf exists in gadget i. Then we continue with gadget $i+1$. Finally the last gadget n cannot have a non-black leaf since all its Out vertices are black.

Furthermore, G'' has constant cycle rank since it is not strongly connected and all the strongly connected components are cycles which have constant cycle rank.

□

4.4 Maximum Directed Cut

Let us now focus on a problem of much different nature: MAX DI CUT. Even though, as we saw in Section 4.3, no digraph complexity measure manages to provide an FPT algorithm for DIRECTED HAMILTONIAN CIRCUIT, they do succeed in providing algorithms with polynomial running times, when the width k is fixed. For MAX DI CUT the situation is much worse, as we will show that the problem is NP-hard even for $k = 1$. This creates an even larger gap with the FPT performance of treewidth than we had in the case of DIRECTED HAMILTONIAN CIRCUIT.

We will prove that MAX DI CUT is both NP and APX-hard, even when restricted to DAGs by showing a reduction from the maximization version of NAE3SAT.

Theorem 4.3. MAX DI CUT *is NP-hard and APX-hard, even when restricted to DAGs.*

Proof. We give a gap-preserving reduction from NAE3SAT to MAX DI CUT.

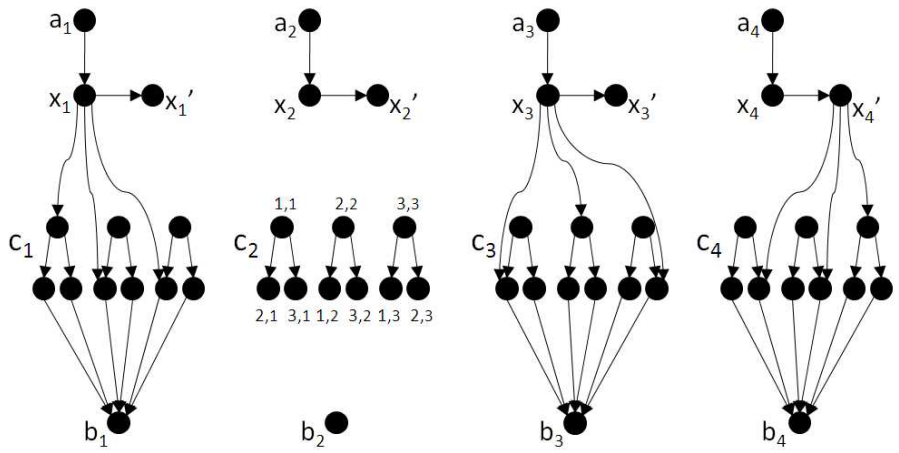


Figure 4.3: The above figure presents an example of the construction for the formula $\phi = (x_1 \vee x_2 \vee \neg x_3) \wedge (\neg x_2 \vee x_3 \vee x_4)$. From ϕ we construct $\phi' = (x_1 \vee x_2 \vee \neg x_3) \wedge (\neg x_1 \vee \neg x_2 \vee x_3) \wedge (\neg x_2 \vee x_3 \vee x_4) \wedge (x_2 \vee \neg x_3 \vee \neg x_4)$. In order for the figure to be understandable we excluded most of the edges of E_5 together with some edges of E_3.

Given a NAE3SAT formula ϕ with m clauses and n variables we construct a new NAE3SAT formula ϕ' with $2m$ clauses and n variables and show that ϕ is satisfiable iff ϕ' is satisfiable (satisfaction is in the NAESAT sense). Then from ϕ' we construct a (weighted) DAG G and show that ϕ' is satisfiable iff G has a directed cut of size $46m$. Without loss of generality, we may assume that every clause of ϕ has exactly three literals (otherwise we may repeat one).

The new formula ϕ' is constructed by taking ϕ and adding to it, for every clause the same clause with all literals complemented. If an assignment satisfies t clauses of the original formula, it must satisfy exactly $2t$ of the $2m$ clauses of ϕ'. Note that, if we denote by f_i the number of appearances of the variable x_i in ϕ, then the same variable will appear $2f_i$ times in ϕ': f_i times as x_i and f_i times as $\neg x_i$. In other words, the positive and negative appearances of each variable in ϕ' are balanced. We will make use of this fact several times. Furthermore, since every clause of ϕ has exactly three literals, we have that $\sum_i f_i = 3m$

Let us now construct the DAG $G(V, E)$. V consists of four disjoint sets of vertices

A, X, C, B. $A = \{a_1, \ldots, a_n\}$ will be a set of source vertices. $B = \{b_1, \ldots, b_{2m}\}$ will be a set of sink vertices. $X = \{x_1, x'_1, x_2, x'_2, \ldots, x_n, x'_n\}$ will be the set of vertices corresponding to literals of ϕ' while $C = \{c_{i,j,k} \mid i \in \{1, 2, \ldots, 2m\},\ j, k \in \{1, 2, 3\}\}$ will correspond to the clauses of ϕ'.

E consists of the following sets of weighted edges:

1. The set $E_1 = \{(a_i, x_i) \mid i \in \{1, \ldots, n\}\}$. Each of these edges has weight $6f_i$, where f_i is the total number of appearances of the variable x_i in ϕ.

2. The set $E_2 = \{(x_i, x'_i) \mid i \in \{1, \ldots, n\}\}$. Each of these edges also has weight $6f_i$.

3. The set $E_3 = \{(c_{i,j,k}, b_i) \mid i \in \{1, \ldots, 2m\},\ j, k \in \{1, 2, 3\}, j \neq k\}$. These have weight 1.

4. The set $E_4 = \{(c_{i,k,k}, c_{i,j,k} \mid i \in \{1, \ldots, 2m\},\ j, k \in \{1, 2, 3\}, j \neq k\}$. These also have weight 1.

5. Finally, E_5 consists of edges that connect vertices of the set X to the corresponding vertices of C. That is, we add the edges $\{(x_l, c_{i,j,k}),\ k \in \{1, 2, 3\}\}$ when the literal x_l appears in the j-th position of the i-th clause of ϕ', and the edges $(x'_l, c_{i,j,k})$ when the literal $\neg x_l$ appears in that position. These edges have weight 2.

An illustrative example of the construction is presented in figure 4.3. Vertex x'_4 is connected to $c_{4,3,1}$, $c_{4,3,2}$ and $c_{4,3,3}$ since $\neg x_4$ appears in the third position of the fourth clause. The intuition behind our construction is that the placement of the vertices of C on either side of the cut will correspond to the truth assignments for the literals. The edges inside C take care of the satisfaction. For each clause we construct three triplets of vertices. Each triplet corresponds to a different arrangement of the literals in the specific clause, where in each arrangement a different literal of the clause is

placed on top and this retains the symmetry in the clauses. Specifically, the vertex $c_{i,j,k}$ corresponds to the j-th literal of the i-th clause of ϕ' in the arrangement where the k-th literal is placed on top. In a satisfied clause one literal is false and one true and there is always an arrangement which places the true literal on top and the false one on bottom, thus contributing to the cut. However for non-satisfied clauses none of the arrangements contribute to the cut.

Suppose we are given a truth assignment that satisfies (in the NAESAT sense) t of the m clauses of ϕ. It must satisfy $2t$ of the $2m$ clauses of ϕ'. Let us partition V into V_0 and V_1. Place all vertices of A into V_0 and all vertices of B into V_1. Place the vertices of X that correspond to true literals in V_1 and the rest in V_0. Place the vertices of C that correspond to true literals in V_0 and the rest in V_1.

Let us calculate the weight of this cut. If a variable x_i is assigned the value 1 in the assignment, the edge (a_i, x_i) contributes $6f_i$ to the cut. If it is assigned 0, then x'_i is in V_1, therefore the edge (x_i, x'_i) contributes $6f_i$ to the cut. Thus, the total contribution of all edges in $E_1 \cup E_2$ is $6\sum_i f_i = 18m$. Because the appearances of each variable in ϕ' are balanced, there are as many literals that took the value true as there are literals that took the value false, in any assignment. Therefore, exactly half the edges of E_3 contribute to the cut. The number of edges in E_3 is $12m$ so, a weight of $6m$ is contributed to the cut. It is not hard to see that, for a satisfied clause C_i, the edges of E_4 incident on vertices that correspond to this clause contribute exactly 2 to the cut. On the other hand, the edges of E_4 incident on vertices of C that correspond to a clause that is not satisfied will contribute 0 to the cut, since all these vertices correspond to literals with the same truth value and are therefore on the same side of the partition. Thus, we get a total of $4t$ contributed to the cut, since $2t$ clauses are satisfied. Finally, once again because of the balancing of ϕ', exactly half of the edges of E_5 contribute to the cut: those incident on vertices of X that we placed in V_0, i.e. vertices that correspond to false literals. For each such element of X we have

in total $3f_i$ edges. Since the weight of each such edge is 2, this adds up to a total contribution of $6\sum_i f_i = 18m$.

Thus, the total size of the cut is $18m + 6m + 18m + 4t = 42m + 4t$, which is equal to $46m$ when the truth assignment satisfies every clause of ϕ.

Now for the other direction, suppose we are given a partition of V into V_0 and V_1. We will show that we can transform such a cut into a cut of the previous form, thus obtaining a truth assignment. First, observe that for any optimal cut $A \subseteq V_0$ and $B \subseteq V_1$, because it is always optimal to place a source in V_0 and a sink in V_1. Now, suppose that in the cut we are given, for some i, $x_i, x'_i \in V_0$. Then place x'_i in V_1 and this will not make the cut smaller because now the edge (x_i, x'_i) contributes to the cut and its weight is exactly as much as the weight of all other edges incident on x'_i. Also, if $x_i, x'_i \in V_1$ place x_i in V_0. This can not make the cut smaller, since the only edge lost is (a_i, x_i) and its weight is the same as that of (x_i, x'_i) which now enters the cut. Therefore, we have now made sure that for all i, x_i and x'_i are on different sides of the partition, without decreasing the size of our cut.

Consider now a vertex $c_{i,j,j}$. We know that there exists an edge $(x_i, c_{i,j,j})$ (or an edge $(x'_i, c_{i,j,j})$) of weight 2, which is as much as the weight of all other edges incident on $c_{i,j,j}$. Therefore, if x_i (resp. x'_i) is in V_0, then we can place $c_{i,j,j}$ in V_1 without decreasing the size of the cut. Otherwise, we can place $c_{i,j,j}$ in V_0, because the edge of weight 2 can not be included in the cut by changing the side of $c_{i,j,j}$ only, and therefore placing it in V_0 is not worse because this way we may also include some of the other edges in the cut. This establishes that every vertex $c_{i,j,j}$ is on a different side of the partition from its predecessor in X.

Finally, consider a vertex $c_{i,j,k},\ j \neq k$. If its predecessor in X is in V_0 we can place it in V_1 without decreasing the size of the cut, because then the edge of weight 2 is included. Otherwise we can place it in V_0, and this will include the edge $(c_{i,j,k}, b_i)$ in the cut. This does not decrease the size of the cut, since the edge of weight 2 was

not included anyway, therefore we might at most lose the other edge incident on this vertex, which also has weight 1. This establishes that each of the remaining vertices of C is also on a different side of the partition from its predecessor in X.

Now, observe that starting with any given cut, we have transformed it into a cut of a special form, without decreasing its size. From this cut we can construct a truth assignment: set to true the literals corresponding to vertices in X that we placed in V_1. This is a valid assignment, since exactly one of x_i, x'_i is in V_1. Also, if we repeat the process of the first direction of this reduction starting from this assignment we will get the same cut. Therefore, we have shown that there is a truth assignment that satisfies t of the m clauses of ϕ iff there is a cut in the DAG G of size at least $42m + 4t$. Thus,

$$\begin{aligned} OPT_{NAESAT}(\phi) = m &\Rightarrow OPT_{MDC}(G) = 46m \\ OPT_{NAESAT}(\phi) \leq (1-\epsilon)m &\Rightarrow OPT_{MDC}(G) \leq (1 - \frac{2\epsilon}{23})46m \end{aligned}$$

□

It is not hard to extend the results of the previous theorem to the cardinality version of MAX DI CUT, that is, the version where all edges have the same weight.

Theorem 4.4. *Cardinality* MAX DI CUT *is NP and APX-hard, even when restricted to DAGs.*

Proof. First, observe that all the edge weights used in the proof of Theorem 4.3 are polynomially (in fact linearly) bounded by the size of the original NAE3SAT formula. Thus, if we extend the problem's definition to include multigraphs, we can replace every edge of weight w by w parallel edges of weight 1. It is not hard to see that this does not affect the rest of the proof.

Now, let us show how to eliminate parallel edges. For each edge (u, v) introduce a directed path of length 3 u, w_1, w_2, v where w_1 and w_2 are new vertices. Observe that, if u is assigned 0 and v is assigned 1, then it is possible to include 2 of the 3 edges of the path in the cut, by assigning 0 to w_2 and 1 to w_1. However, any other assignment to u and v ensures that at most 1 of the three edges can be included in the cut, and in fact this is always possible by assigning 0 to w_1 and 1 to w_2. Thus, it is not hard to see that the reduction's arguments can now be applied with little modification.

□

Corollary 4.8. MAX DI CUT *is NP-hard and APX-hard even when restricted to graphs of bounded directed treewidth, DAG-width, Kelly-width, directed pathwidth or cycle rank.*

Proof. The proof is immediate, because DAGs have width at most 1 under the definitions of all these widths. □

Chapter 5

Structural Parameterizations for Specific Problems

The main theme of both chapters 3 and 4 was the investigation of algorithmic properties of variations of treewidth. Though treewidth has been one of the greatest success stories of parameterized complexity theory it makes sense for one to ask whether it is fruitful to consider other structural parameters, disconnected from the whole concept of "graph widths". For example, does it make sense to parameterize a problem by the input graph's maximum degree or its distance from bipartite-ness? In this section we will give examples of such parameterizations and present results for three different problems.

First, we will consider the Path Coloring problem, a well-known optimization problem with applications to optical networks. We will study several structural parameterizations of interest in practice, such as by the maximum degree of the input graph or the available number of colors.

Second, we will consider the satisfiability problem for the basic modal logic K. Modal logic is an extension of standard propositional logic with many applications (for example in AI or game theory), for which the basic satisfiability problem is known

to be PSPACE-hard. Here we investigate parameterizations of this problem by the structure of the input formula (e.g. the quantifier depth).

Finally, we will examine the VERTEX COVER problem. This is a flagship problem in parameterized complexity theory and its standard parameterization (by the size of the solution) has been widely researched. Here we will focus on a structural parameterization (by the input graph's distance from bipartiteness) and show an application to a parameterization of the same problem above a lower bound.

5.1 Path Coloring

The PATH COLORING (PC) and MAXIMUM PATH COLORING (MAXPC) problems are two well-known and widely studied combinatorial problems with applications in the field of optical networks. In PC we are given a graph representing the optical network and a set of paths on that graph and are asked to find a coloring of the paths such that any two paths which share an edge have distinct colors and the number of colors used is minimized. In the MAXPC problem on the other hand we are given a specified number of colors and must select a maximum cardinality set of paths which can be properly colored with the available colors. If the graph contains cycles we may alternatively be given the endpoints of the communication requests only, with the flexibility to choose the most suitable path for each. Then the problem is often called ROUTING AND PATH COLORING (RPC and MAXRPC). Of course, if the underlying graph is a tree the two versions of the problems are equivalent.

PC is unfortunately known to be hard to solve exactly even on very simple topologies and therefore the same holds for MAXPC. As a consequence the vast majority of research on the two problems has focused on coming up with good approximation algorithms for either minimizing the number of colors or maximizing the number of colored paths. Here, however, we investigate the complexity of solving MAXPC on

trees and tree-like graphs exactly, from the point of view of parameterized complexity theory.

The main observation we want to exploit is that MAXPC is a problem rich with reasonable parameters. For example in practical situations one may often expect that the network will have moderate maximum degree and it will be a tree or perhaps "tree-like". Furthermore, technological limitations mean that the number of available colors on each edge is also likely to be moderate. Also, as observed in Anand et al. [2003], communications networks are often built with maximum capacity in mind, meaning that typically the available resources should be enough to satisfy all or almost all requests. Interestingly, nothing prevents several of these facts from happening together. This motivates the study of the problem through a parameterized lens: one identifies a parameter (or set of parameters) expected to be small and then attempts to design an FPT algorithm for this particular parameterization or prove that none exists.

The main results we present here are the following: we study structural parameterizations, that is, parameterizations which do not involve the objective function, which is the number of requests to be satisfied. Specifically, for trees the parameters we consider are the maximum degree Δ and the number of available colors W. We show that MAXPC is W-hard when parameterized by one of these parameters, even when restricted to instances where the other is a small constant. Furthermore, we show that in general graphs, MAXPC is W-hard when parameterized by treewidth, even if both Δ and W are small constants. The results presented here will appear in Lampis [2011b].

5.1.1 Previous work

PC and MAXPC are very well-studied problems, starting from the 1980s (see Golumbic and Jamison [1985]). As mentioned, when the network graph contains cycles one may

consider either the case where requests are pre-routed or where routing is part of the problem. Furthermore, the communication network can either be assumed to be undirected or bi-directed, where in the second case every request has a direction and two requests with the same color can share an edge if they use it in opposite directions.

PC is known to be hard even in very restricted topologies, from which fact the hardness of MAXPC also follows trivially. Specifically, PC is NP-hard for undirected stars by equivalence to edge coloring in multi-graphs (Erlebach and Jansen [2001a]), undirected rings (Garey et al. [1980]) and bi-directed binary trees (Erlebach and Jansen [2001a], Kumar et al. [1997]). However, it is known to be FPT in undirected trees when parameterized by the maximum degree of the tree Δ (Erlebach and Jansen [2001a], Kumar et al. [1997]), and also to be FPT in bi-directed trees when parameterized by the maximum number of requests touching any node (Erlebach and Jansen [2001a]). A 4/3-approximation algorithm is known for PC in undirected trees (Erlebach and Jansen [2001a]) and a 5/3-approximation for bi-directed trees (Erlebach et al. [1999]).

For MAXPC a 2.22-approximation is known for bi-directed trees (Erlebach and Jansen [1998]) and a 1.58-approximation is known for undirected trees (Wan and Liu [1998]). For bi-directed trees it is also known that MAXPC is solvable in polynomial time if both the maximum degree and the number of colors are constant (Erlebach and Jansen [1998]), a result which can be extended to undirected trees in a straightforward manner. Note though that this is an XP, not an FPT algorithm (the degree of the polynomial depends on the parameter), a fact that we will return to later. For the special case where only one color is available the problem is also known as MAXIMUM EDGE DISJOINT PATHS, and is known to be NP-hard for bi-directed trees (Erlebach and Jansen [2001b]) but in P for undirected trees (Garg et al. [1997]).

5.1.2 Definitions

We will discuss the PATH COLORING problem (PC) and its corresponding maximization problem MAXPC. Our main topic is their restriction to trees. The input we are given in this case consists of an undirected tree $G(V, E)$ and a multi-set of demands $D \subseteq V \times V$. We are also given two integers W (the number of colors) and B (the number of demands we seek to satisfy). The question is whether there exist W mutually disjoint subsets $D_1, D_2, \ldots, D_W \subseteq D$ s.t. no set D_i contains two demands that share an edge and $\sum_{i=1}^{W} |D_i| \geq B$.

In other words we are asked if there exists a W-colorable set of at least B paths from the set of the given demands. This problem, where we seek to maximize B is usually called MAXPC, while PC is simply the special case when $B = |D|$. The graph G can either be considered undirected, in which case the ordering of each demand pair is irrelevant, or bi-directed, in which case two satisfied demands with the same color are allowed to use the same edge but only in opposite directions (another way to think of this is as replacing every undirected edge with two parallel arcs of opposite directions).

The problems can be generalized to graphs that contain cycles. Here, we will focus on the case where for each demand we are given the path that it must follow on the graph, but also briefly mention how our results can be extended to the case where routing is part of the problem.

We will look into different parameterizations of the same problem. The candidate parameters we are interested in are the maximum degree Δ , the input graph's treewidth t, and the available number of colors W. To keep the presentation short and concise we will use a notation where different parameterizations of MAXPC are denoted by prepending it with the list of variables we consider constant or parameters. For example, the $(p\Delta)-$MAXPC problem is the parameterized version of MAXPC when Δ is our only parameter. The reason for this notation is that we will consider

various combinations of parameters and also cases where some values are parameters and some others are fixed constants. For example $(pW, c\Delta)-$MaxPC is the special case of $(pW)-$MaxPC restricted to bounded degree trees. Observe that this is not the same as the problem $(pW, p\Delta)-$MaxPC since a hypothetical algorithm running in time say $2^W n^\Delta$ is FPT for the first problem but not for the second.

5.1.3 Structural Parameterizations

In this section we investigate parameterizations which involve the maximum degree Δ, the input graph's treewidth t and the number of available colors W. Some fixed-parameter tractability results are known in the case of PC for these cases, but unfortunately for the corresponding cases of MaxPC only XP algorithms are known and as we will show this can probably not be improved.

First, recall that in Erlebach and Jansen [1998] it was shown that MaxPC can be solved in polynomial time on bi-directed trees if both Δ and W are constant. The basic idea is a bottom-up dynamic programming technique which can be extended in a straightforward way to undirected trees also. Our first observation is that this idea can in fact be extended to graphs of bounded treewidth as well.

Theorem 5.1. $(cW, c\Delta, ct)$-MaxPC *can be solved in polynomial time for both undirected and bi-directed graphs.*

Proof. (Sketch) The algorithm is based on bottom-up dynamic programming on the tree decomposition of G. Recall that the vertices of a (non-leaf) bag of the decomposition form a separator of G. Root the tree decomposition on some arbitrary bag. The key observation now is that in any feasible solution, for any given bag B, we can only have at most $O(tW\Delta)$ satisfied demands touching the vertices of B, because no edge can have more than W satisfied demands going through it, no vertex has more than Δ edges touching it (or 2Δ for bi-directed graphs) and all bags have at most

$t+1$ vertices. This means that we can enumerate all possible sets of satisfied demands touching a bag in polynomial time (about $|D|^{O(tW\Delta)}$).

We now follow the standard treewidth techniques of calculating bottom-up for each possible local solution in a bag what is the maximum number of satisfied demands we can get for the graph induced by the vertices in the bag and those below it in the tree decomposition.

□

Theorem 5.1 essentially applies common dynamic programming techniques associated with treewidth to obtain an XP algorithm. The algorithm is likely to be extremely impractical though, even for small values of the parameters, since the exponent relies on all three. So the natural, and more important question to ask is whether any kind of fixed-parameter tractability result can be obtained.

Ideally, one would like an FPT algorithm running in time $f(W, \Delta, t) \cdot n^c$, that is, an FPT algorithm for $(pW, p\Delta, pt)$-MaxPC. Barring that, it would still be helpful if any one of the three parameters could be moved out of the exponent of n, even by itself. Unfortunately, we resolve this problem in a negative way, showing that even if any two of the parameters are small fixed constants (and are therefore allowed to appear in the exponent of n in an FPT algorithm) it is still impossible to obtain an FPT algorithm for the problem, under standard complexity assumptions. We prove this by using three parameterized reductions.

The reductions presented here will use a slightly more general problem we will call CapMaxPC. In this problem, for each edge $e \in E$ we are given an integer capacity $1 \leq c(e) \leq W$ and have the additional constraint that in a feasible solution at most $c(e)$ satisfied demands may be using e. For parameterizations not involving the objective function this problem is shown FPT-reducible to MaxPC by using a simple trick where limited edge capacity on an edge is simulated by adding an appropriate number of length 1 demands going through the edge.

We will also use another intermediate problem in our reductions, which we will call DISJOINT NEIGHBORHOODS PACKING (DNP). In DNP we are given an undirected graph $G(V, E)$ and are asked to find a maximum cardinality set $V' \subseteq V$ such that $\forall u, v \in V'$ we have $N[u] \cap N[v] = \emptyset$. The parameter we consider is the size of V'.

Overall our strategy is to start from the well-known W[1]-hard problem INDEPENDENT SET and present reductions to our problems through the two intermediate problems described above, that is, we aim to prove that IS $\leq_{fpt}$ DNP $\leq_{fpt}$ CAPMAXPC $\leq_{fpt}$ MAXPC. The trickiest step in this process will be the second reduction, where we will show three different versions, one for each parameterization of MAXPC we are interested in.

Lemma 5.1. *For both undirected and bi-directed graphs we have*

- $(pW, c\Delta)$-CAPMAXPC $\leq_{FPT}$ $(pW, c\Delta)$-MAXPC ‘
- $(cW, p\Delta)$-CAPMAXPC $\leq_{FPT}$ $(cW, p\Delta)$-MAXPC
- $(cW, c\Delta, pt)$-CAPMAXPC $\leq_{FPT}$ $(cW, c\Delta, pt)$-MAXPC

Proof. For each edge (or arc) (u, v) whose capacity is $c < W$ we add $W - c$ demands from u to v. Let A be the total number of new demands added this way. After doing this we forget about capacities and we now have an instance of MAXPC. It is not hard to see that the original CAPMAXPC instance has a coloring satisfying B demands iff the new MAXPC instance has a coloring satisfying $A + B$ demands, because there must exist some optimal solution to the new instance which uses all the newly added demands.

□

Lemma 5.2. DNP *is W[1]-hard.*

Proof. We present a reduction from the INDEPENDENT SET problem. Given a graph $G(V, E)$ and assuming without loss of generality that it has no isolated vertices and we

are looking for an independest set of size $k > 2$ in G, we will construct an equivalent instance of DNP. First, subdivide every edge of G, that is, replace each $(u, v) \in E$ with a path of length 2. Connect all newly added vertices into a clique. We will argue that the new graph has a packing of k disjoint neighborhoods iff the original graph has an independent set of size k.

If the original graph has an independent set of size k this immediately gives us a packing of the same size on the new graph by selecting the same vertices. The packing is valid since the only way two of the original vertices could have a common neighbor in the new graph is if one of the vertices introduced in the subdivisions is connected to both and that can only happen if an edge was connecting them in the original graph.

If the new graph has a packing of $k > 2$ disjoint neighborhoods given by the set of vertices V', then we can immediately infer that V' cannot include two or more of the vertices introduced in the subdivisions, since they are all connected in a clique. If V' contains one of these new vertices, say the one introduced in the subdivision of (u, v) (call that vertex w) then it cannot contain any vertices in $V \setminus \{u, v\}$ because every original vertex is connected to at least one new vertex and that vertex is connected to w. V' may also contain at most one of $\{u, v\}$, so its total size cannot be more than 2 in this case. We conclude that a packing of $k > 2$ disjoint neighborhoods must consist entirely of vertices from the original graph. To see that these form an independent set in the original graph, observe that if two were originally connected they would have a common neighbor in the new graph, violating the feasibility of the packing.

□

Theorem 5.2. $(pW, c\Delta)$-MAXPC *is W[1]-hard for both undirected and bi-directed trees.*

Proof. Given Lemma 5.1 and Lemma 5.2 the only thing left to prove is that DNP $\leq_{fpt}$ $(pW, c\Delta)$-MAXPC. Given an instance of DNP, that is a graph $G(V, E)$ and a

target size for the DNP set k, we construct a MaxPC instance as described below. We first show the reduction for undirected trees and then describe how it can be made to work for bi-directed trees as well.

First, let $|V| = n$ and we construct a "backbone", which is simply a path on $n+2$ vertices. We take $n+2$ disjoint copies of a path on n vertices and attach one of the endpoints of each to one of the vertices of the backbone so that each backbone vertex now has a path hanging from it. Label the backbone vertices $b_i, 0 \leq i \leq n+1$ and the vertices of the other paths $p_{i,j}, 0 \leq i \leq n+1, 1 \leq j \leq n$, so that the path vertex connected to b_i is called $p_{i,1}$, its other neighbor is $p_{i,2}$ and so on. Finally, for each $1 \leq i, j \leq n$ we add three vertices in the graph $v_{i,j}, u_{i,j}$ and $w_{i,j}$ and the edges $(v_{i,j}, w_{i,j}), (u_{i,j}, w_{i,j})$ and $(w_{i,j}, p_{i,j})$. In other words, we construct a path on three vertices and connect the middle vertex to $p_{i,j}$. This completes the description of the graph, which is a tree of maximum degree 3.

Now let us describe the demands. Suppose that the vertices of the original graph are numbered $\{1, 2, \ldots, n\}$. For each $i \in V$ we consider the neighborhood $N(i)$ in increasing order and let $N(i) = \{j_1, j_2, \ldots, j_{d(i)}\}$, where $d(i)$ is the degree of i. We add a demand from $p_{0,i}$ to $u_{j_1,i}$. Then, for each $l, 1 \leq l < d(i)$ we add a demand from $v_{j_l,i}$ to $u_{j_{l+1},i}$. We also add a demand from $v_{j_{d(i)},i}$ to $p_{n+1,i}$. We add all these demands for each $i \in V$ and call these demands global demands. Finally, for each $1 \leq i, j \leq n$ we add a demand from $v_{i,j}$ to $u_{i,j}$. We call these demands local.

The only thing left is to specify W, which we set to $W = k$, and the capacities. We leave all capacities unconstrained except for the edges $(b_i, p_{i,1}), 1 \leq i \leq n$, which have a capacity of 2 and the edges $(u_{i,j}, w_{i,j})$ and $(v_{i,j}, w_{i,j})$ which have a capacity of 1. The construction is now complete.

To give some intuition about this construction, notice the interaction between local and global demands. Each local demand intersects exactly two global demands in edges of capacity 1. Thus, if the local demand is satisfied the global demands are

rejected. Furthermore, if exactly one of the global demands is satisfied in a solution we can exchange it with the local demand, therefore this gadget ensures that either both global demands will be taken or both will be rejected in some optimal solution. Observe also that from all the local demands found in a branch attached to the backbone at most one will be rejected, since the edge $(b_i, p_{i,1})$ acts as a bottleneck allowing at most two global demands to go through. The idea will be that if a vertex i is in the neighborhood packing then we will select the global demand starting at $p_{0,i}$ and satisfy one after the other pairs of demands that go into branches that correspond to its neighbors, making these branches unusable for other global demands.

For a more precise argument, suppose that the original graph has a packing V' of size k, we will construct a MAXPC solution of size $n^2 + k$. Start with a solution of size n^2 by selecting all the local demands of the instance and nothing else. Now for each $i \in V'$ we will inrease the size of the solution by 1. We do this by satisfying all the global demands associated with i, that is, all demands touching a vertex $p_{i,j}$ for any j. Each time we perform this improvement step we use a new color and remove from the solution all local demands that intersect with these global demands (it is not hard to see that this gives a profit of exactly one demand). Since we are using different colors in each step the only way this process could run into a problem is in an edge where fewer that k colors can be used. For that to happen we must be trying to satisfy more than two requests going through an edge $(b_j, p_{j,1})$ but that would imply that j is a common neighbor of two vertices of the packing, violating its feasibility.

For the other direction, suppose that a solution of size n^2+k exists. As mentioned, if in a set of one local and its two intersecting global demands the solution satisfied exactly one of the global demands, we exchange it with the local demand. This means that for each edge $(b_i, p_{i,1})$ we are either satisfying two of the demands crossing it or none and furthermore that if we are satisfying two, one of them is going "left" (that is, it's other endpoint is towards b_{i-1}) and the other is going "right" (so its other

endpoint is towards b_{i+1}). Therefore, the number of satisfied requests going through each edge (b_i, b_{i+1}) is constant for all i, call this number L. We will establish that $L = k$. Pick an arbitrary satisfied demand which uses a backbone edge and delete it from the solution. This will reduce the size of the solution by one, but it will also allow us to reduce L by one, since by the same arguments used before we can make the number of satisfied demands on each backbone edge the same without affecting the size of the solution[1]. Repeat this process L times and now we have a solution which satisfies only local demands and has size $n^2 + k - L$. Since there are exactly n^2 local demands it must be the case that $k = L$. Now we can conclude that there are k vertices in the branch connected to b_0 whose demands are satisfied and all subsequent global demands associated with them are also satisfied. These give us a neighborhood packing in the original graph because if two of them had a common neighbor the solution would be exceeding some branch's bottleneck capacity of 2.

It is not hard to modify this reduction to also work for bi-directed trees. The only difference in the network is that edges $(b_i, p_{i,1})$ are given a capacity of 1, since they are intended to be traversed twice but in different directions. Other than that we remain consistent with the ordering that we have implied in our description, that is, every global demand is ordered towards the vertex that lies further to the right (the vertex closer to $p_{n+1,n}$. so to speak). We also make sure that the local demands are directed in such a way that they intersect both global demands with which they share an edge and the rest of the arguments of the reduction go through unchanged.

□

Theorem 5.3. $(cW, p\Delta)$-MAXPC *is W[1]-hard for both undirected and bi-directed trees. The result holds even for instances where all the vertices but one have degree bounded by 3.*

[1]This is implicitly relying on the fact that all global demands must intersect some local demand, which is true if the original graph had no isolated vertices

Proof. Once again we will describe a reduction from DNP, but now the produced instance will have maximum degree depending on k and a constant W. We will reuse some of the ideas of Theorem 5.2, properly adjusted. Again we will first describe a construction for undirected graphs and then discuss how it can be modified for bi-directed graphs.

Take k copies of a path on n vertices and label the vertices $S_{i,j}, 1 \leq i \leq k, 1 \leq j \leq n$. Take k more copies and label the vertices $T_{i,j}, 1 \leq i \leq k, 1 \leq j \leq n$. Add a new vertex to the graph, call it C, and connect it to all $S_{i,n}$ and $T_{i,n}$ for $1 \leq i \leq k$. Set the capacities of all edges to 1. Also, for each $i, j, 1 \leq i \leq k, 1 \leq j \leq n$ add a demand from $S(i,j)$ to $T(i,j)$.

Before we go on, let us examine the construction so far. It should be clear that the optimal solution satisfies k paths by selecting k vertices in the S branches and their corresponding vertices in the T branches. The k selected vertices will eventually encode the vertices we will pick for our neighborhood packing. What is of course missing is some machinery to ensure that our selection is indeed a packing in the original graph.

The constraints of a valid packing can be broken down as follows: for each of the $\binom{k}{2}$ pairs of vertices selected for the packing we must make sure that they do not share common neighbors. Thus, our basic tool will be a gadget that takes two of the k choices we have made and checks their compatibility. We will make $\binom{k}{2}$ copies of that gadget, attach them to C and then properly reroute the demands from S to T vertices through these gadgets.

To describe the pairwise consistency gadget, consider the instance constructed in the proof of Theorem 5.2. We modify it as follows: First, we add local requests gadgets, identical as those used in vertices $p_{i,j}, 1 \leq i, j \leq n$ to the vertices of the paths p_0 and p_{n+1}. We extend all demands which currently had an endpoint in $p_{0,j}$ or $p_{n+1,j}$ for some $j \in \{1, \ldots, n\}$ to the vertices $v_{0,j}$ and $u_{n+1,j}$ respectively, so that they

intersect the new local demands. Now we make an exact copy of the branch p_0 and all its connected gadgets (i.e. the vertices $p_{0,j}, u_{0,j}, v_{0,j}, w_{0,j}, 1 \leq j \leq n$). We call the new branch p'_0 (and the new vertices respectively $p'_{0,j}, u'_{0,j}, v'_{0,j}, w'_{0,j}, 1 \leq j \leq n$) and attach it also to b_0. We also make sure to replicate all demands that existed betweem the branch p_0 and the rest of the graph so that corresponding demands are placed between the branch p'_0 and the rest of the graph. We perform another full copy for the branch p_{n+1} producing the branch p'_{n+1} with identical vertices and demands and attach this to b_{n+1}. Now the whole gadget has $n(n+4)$ local demands overall. We set the capacities of all backbone edges to 4, all edges used by local demands to 1 and all other edges to 2.

To demonstrate the use of this gadget we will connect one such gadget on our initial construction and use it to ensure that in the optimal solution the choices encoded in the paths S_1 and S_2 (i.e. the encoding of the first two choices for the packing) are compatible. Take a gadget as described and connect its b_0 to C by an edge of capacity 4. Recall that for all $j \in \{1, \ldots, n\}$ there is a demand from $S_{1,j}$ to $T_{1,j}$. Remove these n demands and for all $j \in \{1, \ldots, n\}$ add a demand from $S_{1,j}$ to $u_{0,j}$ and a demand from $v_{n+1,j}$ to $T_{1,j}$ (in other words we are rerouting the $S_1 \to T_1$ demands through the gadget). Do the same for demands from S_2 to T_2, only reroute them through the p'_0 and p'_{n+1} branches.

A solution of size $n(n+4)+k$ can be achieved now iff the selections for active vertices in S_1 and S_2 are compatible, that is, the corresponding vertices of the initial graph have no common neighbors. This follows from the analysis of the properties of our gadget performed in Theorem 5.2.

It is now possible to complete the construction by adding more of the consistency gadgets so as to make sure that all $\binom{k}{2}$ pairs of choices are compatible. The final graph consists of the $\binom{k}{2}$ gadgets plus the $2k$ paths all attached to a single vertex of degree $\binom{k}{2} + 2k$. The total number of vertices is $O(n^2k^2)$ and a solution of size

$\binom{k}{2}n(n+4)+k$ can be achieved iff the original graph has a packing of size k.

Modifying this construction to bi-directed trees is again strightforward, since a direction was implicit in our description. Again the only major difference is that we change edges with capacities 4 and 2 to capacities 2 and 1 respectively. For the last remark of the theorem, notice that the only vertex of high degree is C. All other vertices have degree at most 3, except the b_0 vertices of the gadgets, but even this can easily be fixed since it is not necessary for the reduction to attach p'_0 and p_0 to the same vertex. We can simply subdivide the (b_0, b_1) edge and attach p'_0 there.

□

Theorem 5.4. $(cW, c\Delta, pt)$-MAXPC *and* $(cW, c\Delta, pt)$-MAXRPC *are W[1]-hard for both undirected and bi-directed graphs.*

Proof. The proof is a modification of the proof of Theorem 5.2. Informally, the only edges where we needed $W = k$ in that reduction, that is, the only edges which were meant to be traversed by k satisfied paths are those of the backbone, while in the branches numbered 1 through n we allowed up to only two satisfied demands on each edge. So the question is how to fix the backbone and first and last branch to use a constant number of colors also, using a construction of treewidth k.

To do this we replace the backbone with a $(n+2) \times k$ grid, with its vertices numbered $b_{i,j}, 0 \leq i \leq n+1, 1 \leq j \leq k$. The edges $(b_{i,j}, b_{i,j+1})$ have capacity 2, while the edges $(b_{i,j}, b_{i+1,j})$ have capacity 1. The branches $p_i, 1 \leq i \leq n$ are identical as in the reduction of Theorem 5.2 and are connected to the vertices $b_{i,k}$. For the branch p_0 we simply make k copies of it (including the demands) and connect each to a vertex $b_{0,j}, 1 \leq j \leq k$. Similarly, for the branch p_{n+1} we make k copies and connect them to $b_{n+1,j}, 1 \leq j \leq k$. We set the capacities of the edges inside these copied branches to 1.

Now the construction is complete and even though there are many ways to route the demands, it does not make a difference if we allow every possible routing (i.e.

solve MAXRPC) or specify that demands starting at the j-th copy of the branch p_0 must stay on the same level of the grid (for MAXPC). In both cases the arguments of Theorem 5.2 go through, since it is not hard to establish that for any i the total number of satisfied requests going through the edges $(b_{i,j}, b_{i+1,j})$ is the same as the total number of satisfied requests going through the edges $(b_{i,j}, b_{i-1,j})$ (the quantity L in Theorem 5.2). The modification for bi-directed graphs is also straightforward.

Finally, observe that this graph has $W = 2, \Delta = 4$ while the treewidth (in fact, the pathwidth) of the graph can be upperbounded by $k + 2$.

□

As a final note in this section, note that it is known that assuming standard complexity assumptions (specifically the Exponential Time Hypothesis which states that 3-SAT cannot be solved in $2^{o(n)}$) it is not possible to find an independent set of size k on an n-vertex graph in time $n^{o(k)}$. The reductions in Theorems 5.2 and 5.4 are linear in the parameter, meaning that assuming the ETH we know there is no $n^{o(W)}$ algorithm for MAXPC even for binary trees and there is no $n^{o(t)}$ algorithm, even when $W = 2, \Delta = 4$. The reduction in Theorem 5.3 is quadratic in the parameter, meaning that no $n^{o(\sqrt{\Delta})}$ algorithm is possible. Putting these results together tells us that no $n^{o(Wt\sqrt{\Delta})}$ algorithm is possible. Contrasting this with the algorithm of Theorem 5.1 we see that the only small gap left to close here is the complexity as a function of Δ.

5.2 Modal Satisfiability

In this section we consider the computational complexity of deciding formula satisfiability, for modal logics, focusing on the standard modal logic K. Although the complexity of satisfiability for modal logic has been studied extensively in the past, this has not been done before from a parameterized perspective. Here, several parameterized complexity results are presented for three different parameterizations of

the satisfiability problem. More specifically, we will look into parameterizations that try to quantify how "hard" an input formula is, just like treewidth tries to quantify how "hard" a graph is. The results of this section have appeared in Achilleos et al. [2010] and Achilleos et al. [2011].

5.2.1 Previous Work

Modal logic is a family of systems of formal logic where the truth value of a sentence ϕ can be qualified by modality operators, usually denoted by $\Box$ and $\Diamond$. Depending on the specific modal logic and the application one considers, $\Box\phi$ and $\Diamond\phi$ can be informally read to mean, for example, "it is necessary that ϕ", or "it is known that ϕ" for $\Box$ and "it is possible that ϕ" for $\Diamond$. The fundamental normal modal logic system is known as K, while other common variations of this logic system include T, D, $\mathsf{S4}$, $\mathsf{S5}$. Modal logic systems provide a diverse universe of logics able to fit many modern applications in computer science (for example in AI or in game theory), making modal logic a widespread topic of research. The interested reader in the recent state of modal logic and its applications is directed to Blackburn et al. [2006].

As in propositional logic, the satisfiability problem for modal logic is one of the most important and fundamental problems considered and many results are known about its (traditional) computational complexity. In Ladner [1977] it was shown that satisfiability for K, T and $\mathsf{S4}$ is PSPACE-complete, while for $\mathsf{S5}$ the problem is NP-complete. Furthermore, in Halpern [1995], it is shown that the problem remains PSPACE-complete when the formulas have at most one variable and in Chagrov and Rybakov [2002] it is shown that satisfiability for K and $\mathsf{K4}$ is PSPACE-complete even for formulas without any variables. In Halpern and Rêgo [2007], Halpern and Rêgo showed that the negative introspection axiom is in an essential way what makes the difference between normal modal logics whose satisfiability problem is in NP and those for which it is PSPACE-complete. It should be noted that the satisfiability of

propositional logic is a subcase of satisfiability for any normal modal logic, thus for any normal modal logic the problem is NP-hard. For an introduction to modal logic and its complexity see Halpern and Moses [1992], Fagin et al. [1995].

The idea of trying to isolate an "easy" family of input formulas by looking at their structure is of course not new. Perhaps the most natural way to quantify this in modal logic is to look at a formula's modal depth, that is, the nesting depth of modal operators. This complexity measure was already known in Halpern [1995] (see also Spaan [1993]) where in fact a fixed-parameter tractability result is shown when the problem is parameterized by the sum of the number of propositional variables of the formula (denoted by v) and the modality depth. However, since parameterized complexity was not well-known at the time, in Halpern [1995] it is only pointed out that the problem is solvable in linear time for fixed values of the parameters, without mentioning how different values of v and the depth affect the running time.

Here, we take the ideas of Halpern [1995] a step further. First, we take a second look at the problem parameterized by v and the modal depth and discover that the parameter dependence is, in the worst case, a tower of exponentials. Even worse, we give a lower bound argument proving that this cannot be improved by any algorithm, unless P=NP. Then, we define two other formula complexity measures, called diamond dimension and modal width. We try to use these as parameters, instead of modal depth, and show that in these cases the parameter dependence is much better (doubly exponential for diamond dimension and singly exponential for modal width).

5.2.2 Definitions

As mentioned, we will study the language of modal logic. This language contains exactly the formulas that can be constructed using propositional variables, the standard propositional operators $\wedge, \vee, \neg$ (and the operators which can be defined using these, such as $\rightarrow$, $\leftrightarrow$) and the unary modality operators ($\Box$, $\Diamond$).

More specifically, the language of modal logic is defined recursively in the following way. We have a set of propositional variables, P. Any variable in P is a formula. Furthermore, if ϕ, ψ are formulas, then $(\phi \wedge \psi), (\phi \vee \psi), \neg\phi$ and $\Box\phi, \Diamond\phi$ are formulas of the language.

It is usually assumed that P is infinite, but for our purposes we do not need to assume anything about its cardinality. However, if $P = \emptyset$, we need to include $\bot$ (or $\top$) in our language in order to be able to form formulas. It is true that we do not need so many operators in our language and that there are many choices for more succinct list of initial symbols (ex. $\wedge$ and $\neg$; $\bot$ and $\rightarrow$, etc, with either $\Box$ or $\Diamond$), but for our purposes it is convenient to include all these.

In our language we do not include the constants $\bot$ and $\top$, for false and true, but we may use them to form formulas, as they can be considered shorthand for $x \wedge \neg x$ and $x \vee \neg x$ respectively, where $x \in P$.

Standard Kripke semantics are considered here: a Kripke frame is a pair (W, R) of a set of states W and an accessibility relation R between states. A Kripke frame together with a valuation V, which is a function that defines for each propositional variable the set of states where it is true, is called a Kripke structure.

We will consider the system of modal logic usually denoted by K, where R is allowed to be an arbitrary relation between states. Other standard modal logics (e.g. T,D,S4) can be obtained by imposing various restrictions on R (e.g. if we only allow reflexive relations).

Given a Kripke structure $\mathcal{M} = (W, R, V)$, we define the relation $\models$ between states and formulas recursively on the structure of the formula:

$\mathcal{M}, s \models p$ if and only if $s \in V(p)$,

$\mathcal{M}, s \models \phi \wedge \psi$ if and only if $\mathcal{M}, s \models \phi$ and $\mathcal{M}, s \models \psi$,

$\mathcal{M}, s \models \phi \vee \psi$ if and only if $\mathcal{M}, s \models \phi$ or $\mathcal{M}, s \models \psi$,

$\mathcal{M}, s \models \neg\phi$ if and only if $\mathcal{M}, s \not\models \phi$,

$\mathcal{M}, s \models \Box\phi$ if and only if for any $v \in W$, if sRs', then $\mathcal{M}, s' \models \phi$,

$\mathcal{M}, s \models \Diamond\phi$ if and only if there is some $s' \in W$, such that sRs' and $\mathcal{M}, s' \models \phi$,

where $p \in P$, $s \in W$ and ϕ, ψ are formulas.

When $\mathcal{M}, s \models \phi$, we say that ϕ is satisfied at s, or that ϕ is true at s and that ϕ is satisfied in $\mathcal{M}$, or that $\mathcal{M}$ is a model for ϕ. A formula ϕ is valid in a structure $\mathcal{M}$, if it is satisfied at all states s of the structure and we say that ϕ is valid if ϕ is valid in all structures.

The problem studied here is *modal satisfiability* for K, that is, given a modal formula ϕ, does ϕ have model? The *modal validity* for K is the problem of determining whether a given modal formula is valid. Although we focus on satisfiability, the two problems are equivalent for modal logic, as any formula ϕ is satisfiable if and only if $\neg\phi$ is not valid.

5.2.3 Modal Depth

In this section we give the definition of modality depth. As we will see, a fixed-parameter tractability result can be obtained when satisfiability is parameterized by the number of propositional variables v and the modality depth of the input formula. This was first observed in Halpern [1995], but in this section we more precisely bound the running time (in Halpern [1995] it was simply noted that the running time is linear for constant depth and constant v with a hidden constant which "may be huge"). More importantly we show that the "huge constant" cannot be significantly improved by giving a hardness proof which shows that, if the running time of an algorithm for modal satisfiability is significantly less than an exponential tower of height equal to the modality depth, then P=NP.

Definition 5.3. The modality depth of a modal formula ϕ is defined inductively as follows:

- $\mathrm{md}(p) = 0$, if p is a propositional variable,
- $\mathrm{md}(\Diamond\phi) = \mathrm{md}(\Box\phi) = 1 + \mathrm{md}(\phi)$,
- $\mathrm{md}(\phi_1 \vee \phi_2) = \mathrm{md}(\phi_1 \wedge \phi_2) = \max\{\mathrm{md}(\phi_1), \mathrm{md}(\phi_2)\}$,
- $\mathrm{md}(\neg\phi) = \mathrm{md}(\phi)$

Note that, since for all ϕ we have $\mathrm{md}(\phi) = \mathrm{md}(\neg\phi)$ this implies that the results of this section, which we state in terms of the satisfiability problem, also apply to the validity prodblem, since deciding if some formula is valid is equivalent to deciding if its negation is satisfiable.

Theorem 5.5. *(Halpern [1995]) Modal satisfiability for the logic* K *is FPT when parameterized by* v *and* $md(\phi)$*.*

Proof. We define the d-type of a state s in a Kripke structure $\mathcal{M}$ to be the set $\{\phi \mid (\mathcal{M}, s) \models \phi \text{ and } \mathrm{md}(\phi) \leq d\}$. We will prove by induction on d that if we restrict ourselves to formulas with at most v variables then for any $d \geq 0$ there are at most $f_v(d)$ d-types, where f_v is the function recursively defined: $f_v(0) = 2^v$, $f_v(n+1) = 2^{f_v(n)+v}$.

For $d = 0$ If $\mathrm{md}(\phi) = 0$, then the formula is propositional, thus the 0-type of any state is directly defined by the set of propositional variables assigned true in the state. The number of all such possible sets of variables is $2^v = f_v(0)$.

For the case of $d+1$ The $(d+1)$-type of a state s depends on the assignment of the propositional variables in s and on the truth values of formulas of the forms $\Box\phi'$ and $\Diamond\phi'$, where $\mathrm{md}(\phi') \leq d$. Notice that these truth values depend only on

the set of d-types of the accessible states from s. Thus the number of different $(d+1)$-types on a state s is $f_v(d+1) = 2^{f_v(d)+v}$.

Now, suppose that ϕ is a satisfiable formula of modality depth $d \geq 1$. We will show how to construct a Kripke structure of about $f_v(d-1)$ states to satisfy ϕ. To achieve this, for all $i \in \{0, 1, \ldots, d-1\}$ and for all i-types we will construct a state of that i-type, thus in total we will construct $\sum_{i=0}^{d-1} f_v(i) = O(f_v(d-1))$ states. To construct the $f_v(0) = 2^v$ states that give all the different 0-types we just construct 2^v states, each with a different valuation of the propositional variables. For the subsequent levels, to construct all the states for all the different $(i+1)$-types we pick for each state a set of successor states out of the states that give us the different i-types and a valuation of the propositional variables. If ϕ is satisfiable, it must be satisfiable in this structure by adding a new state s, selecting a subset of the states that give us the different $(d-1)$-types to be its successors and a valuation of the propositional variables in s. The number of combinations of all possible subsets of successors and all variable valuations is $f_v(d)$, so the problem is solvable in $O(f_v(d) \cdot f_v^2(d-1) \cdot |\phi|)$, because the structure has $O(f_v(d-1))$ states and thus size $O(f_v^2(d-1))$ and model checking can be performed in bilinear time (linear with respect to both $|\phi|$ and the size of the model).

□

Lower Bound

Let us now proceed to the main result of this section, which is that even though modal satisfiability is fixed-parameter tractable, the exponential tower in the running time cannot be avoided. Specifically, we will show that solving modal satisfiability parameterized by modality depth, even for constant v, requires a running time which is a tower of exponentials with height linear in the modality depth. We will prove this under the assumption that P≠NP, by reducing the problem of propositional

satisfiability to our problem. Our proof follows ideas similar to those found in Frick and Grohe [2004].

Suppose that we are given a propositional CNF formula ϕ_p with variables $x_1, \ldots, x_n$ and we need to check whether there exists a satisfying assignment for it. We will encode ϕ_p into a modal formula ϕ_m (the subscripts p and m stand for propositional and modal respectively) with small depth and a constant number of variables. In order to do so we inductively define a sequence of modal formulas.

- In order to encode the variables of ϕ_p we need some formulas to encode numbers (the indices of the variables). The modal formula v_i is defined inductively as follows[2]: $v_0 := \Box\bot$ and $v_n := \left(\bigwedge_{i:n_i=1} \Diamond v_i\right) \wedge \Box\left(\bigvee_{i:n_i=1} v_i\right)$ where by n_i we denote the i-th bit of n when n is written in binary and the least significant bit is numbered 0. So, for example $v_1 = \Diamond v_0 \wedge \Box v_0$, $v_2 = \Diamond v_1 \wedge \Box v_1$, $v_5 = \Diamond v_2 \wedge \Diamond v_0 \wedge \Box(v_2 \vee v_0)$ and so on. Observe that v_0 can only be true in a state with no successor states. Also, what is important is that these formulas allow us to encode very large numbers using only a very small modality depth and no variables (or just one variable if $\bot$ is considered short for $x \wedge \neg x$).

- Next, we need to encode the literals of ϕ_p. The modal formula $\mathcal{L}(x_i)$ is defined as $\mathcal{L}(x_i) := \Diamond v_i \wedge \Box v_i$. The formula $\mathcal{L}(\neg x_i)$ is defined as $\mathcal{L}(\neg x_i) := \Diamond v_i \wedge \Diamond v_0 \wedge \Box\left(v_i \vee v_0\right)$.

- Now, to encode clauses we set $\mathcal{C}(l_1 \vee l_2 \vee \ldots \vee l_k) := \left(\bigwedge_{i=1}^{k} \Diamond \mathcal{L}(l_i)\right) \wedge \Box\left(\bigvee_{i=1}^{k} \mathcal{L}(l_i)\right)$.

- Finally, to encode the whole formula we use $\mathcal{F}(c_1 \wedge c_2 \wedge \ldots \wedge c_m) := \bigwedge_{i=1}^{m} \Diamond \mathcal{C}(c_i)$

So far we have described how to construct a modal formula $\mathcal{F}(\phi_p)$ from ϕ_p. $\mathcal{F}(\phi_p)$ encodes the structure of ϕ_p. Now we need to add two more ingredients: we must use a modal formula to describe that ϕ_p is satisfied by an assignment and that the

[2]We will use := to denote syntactic definitions of formulas and = to denote syntactic equality between formulas.

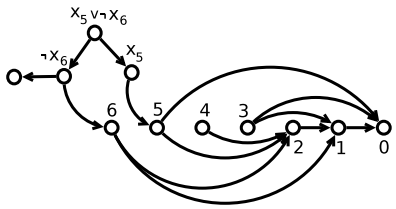


Figure 5.1: A partial example, illustrating our construction for a specific clause. For the encoding of the clause $(x_5 \vee \neg x_6)$ we build the formula $\mathcal{C}(x_5 \vee \neg x_6)$ which holds in the state at the top of the depicted model.

assignment is consistent among clauses. We give two more formulas, $\mathcal{S}$ and $\mathcal{CA}(n)$, which play the previously described roles respectively:

- $\mathcal{S} := \Box\Diamond\left[((\Diamond v_0) \to (\Box\neg y)) \wedge ((\neg\Diamond v_0) \to (\Box y))\right]$, where we have introduced a single variable y.
- $\mathcal{CA}(n) := \bigwedge_{i=1}^{n} (\Diamond\Diamond\Diamond(y \wedge v_i) \leftrightarrow \neg\Diamond\Diamond\Diamond(\neg y \wedge v_i))$

Our full construction is, given a propositional CNF formula ϕ_p with n variables named $x_1, \ldots, x_n$, we create the modal formula $\phi_m := \mathcal{F}(\phi_p) \wedge \mathcal{S} \wedge \mathcal{CA}(n)$.

Lemma 5.4. *ϕ_p is satisfiable if and only if ϕ_m is satisfiable in* K.

Proof. Suppose that ϕ_m is true at a state s of some Kripke structure. Then $\mathcal{CA}(n)$ is true at s therefore for each i we have either that $\Diamond\Diamond\Diamond(y \wedge v_i)$ is true at s or that $\Diamond\Diamond\Diamond(\neg y \wedge v_i)$ is true at s. From this we create a satisfying assignment: for those i for which the first holds we set $x_i = \top$ and for the rest $x_i = \bot$. We will show that this assignment satisfies ϕ_p.

Suppose that it does not satisfy ϕ_p, therefore there is some clause c_i which is not satisfied. However, since $\mathcal{F}(\phi_p)$ is true at s there exists a state p with sRp such that $\mathcal{C}(c_i)$ is true at p. In every successor state of p we have that $\mathcal{L}(l_j)$ is true for some literal l_j of c_i and there exists such a state for every literal of c_i. Also, in s we have that $\mathcal{S}$ is true, therefore in p we have $\Diamond\left[((\Diamond v_0) \to (\Box\neg y)) \wedge ((\neg\Diamond v_0) \to (\Box y))\right]$. Therefore, in some q such that pRq we have $((\Diamond v_0) \to (\Box\neg y)) \wedge ((\neg\Diamond v_0) \to (\Box y))$ and

we also have that $\mathcal{L}(l_j)$ is true for some literal l_j of c_i. Suppose that l_j is a negated literal, that is $l_j = \neg x_k$. Then $\mathcal{L}(l_j) = \Diamond v_k \wedge \Diamond v_0 \wedge \Box(v_k \vee v_0)$. Therefore, since $\Diamond v_0$ is true at q this means that $\Box\neg y$ is true. Because $\Diamond v_k$ and $\Box\neg y$ are both true at q there exists an r such that qRr and $v_k \wedge \neg y$ is true at r. But then $\Diamond\Diamond\Diamond(v_k \wedge \neg y)$ is true at s which implies that our assignment gives the value *false* to x_k. Since c_i contains $\neg x_k$ it must be satisfied by our assignment, a contradiction. Similarly, if $l_j = x_k$ then $\mathcal{L}(l_j) = \Diamond v_k \wedge \Box v_k$. Clearly, v_0 and v_k cannot be true at the same state for $k > 0$ therefore in q we have $\neg\Diamond v_0$ which implies $\Box y$. Therefore in some r with qRr we have $y \wedge v_k$ which implies that our assignment sets x_k to *true* and since c_i has the literal x_k it must be satisfied.

The other direction is easier. We build a Kripke structure where for each v_i there exists a state such that v_i holds in that state. We start by introducing a state without successors, in which v_0 holds. Then, for each $i \in \{1, \ldots, n\}$ we add a state with appropriate transitions to states previously introduced so that v_i holds in that state (see Figure 5.1 for an example).

Note that each time we construct a new state and place the appropriate transitions so that v_i holds in that state, we know that no other v_j with $j \neq i$ can hold in that state. The reason is that, as follows from the definition of v_i, the formula $v_i \wedge v_j$ for $i \neq j$ is unsatisfiable. This, in turn, can be established by induction: first, $v_1 \wedge v_0$ is obviously unsatisfiable. Second, if for some $i > j$ we have $v_i \wedge v_j$ is true at some state of some model, then in some state accessible from it we will have $v_k \wedge v_l$, where k is the position of the most significant bit where i and j differ and $l \neq k$ is the position of some bit of j that is set to 1. Clearly, $k < i$ and $l < j$ so this contradicts the inductive hypothesis.

Now the completion of the Kripke structure so that ϕ_m is satisfied is straightforward. For every i with $1 \leq i \leq n$ we create two more states: the first has as its only successor the state where v_i is true. The other has two successors: the state

where v_i is true and a state without successors (where v_0 holds). Thus, for each i we have a state where $\mathcal{L}(x_i)$ is true and a state where $\mathcal{L}(\neg x_i)$ is true. For every clause we create a state and for each literal l_j in the clause we add a transition to the state where $\mathcal{L}(l_j)$ is true. Therefore, for each clause c_i we have a state where $\mathcal{C}(c_i)$ is true. Finally, we add a state and transitions to all the states where some $\mathcal{C}(c_i)$ is true. Clearly, $\mathcal{F}(\phi_p)$ is true at that state, which we call the root state. Observe that $\mathcal{CA}(n)$ will also be satisfied in the root state independent of where y is true, because for every $i \in \{1, \ldots, n\}$ we have made a unique state p_i where v_i is true and p_i is at distance exactly 3 from the root.

Take a satisfying assignment; for every x_i which is true set the variable y to *true* at the state of the Kripke structure where v_i is true. Set y to *false* in every other state. Now, we must show that $\mathcal{S}$ is true at the root state. This is not hard to verify because for every clause in the original formula there is a true literal, call it l. If that literal is not negated then in the state where $\mathcal{L}(l)$ is true we have $\neg\Diamond v_0$ (because the literal is not negated) and $\Box y$ (because the literal is true, so its variable is true thus we must have set y to *true* at the variable's corresponding state). Therefore $(\neg\Diamond v_0 \to \Box y) \wedge (\Diamond v_0 \to \Box\neg y)$ is true at the literal's corresponding state and $\Diamond\left[(\neg\Diamond v_0 \to \Box y) \wedge (\Diamond v_0 \to \Box\neg y)\right]$ is true at the clause's corresponding state. Similar arguments can be made for a negated literal. Since we start with a satisfying assignment the same can be said for every clause, thus $\mathcal{S}$ is also true at the root state.

□

Now, we need to show that the produced modal formula has very small depth and the hardness result will follow in a way very similar to the results of Frick and Grohe [2004].

Recall the definition of the *tow* function from Section 2.1.

Lemma 5.5. *Suppose that ϕ_p is a propositional CNF formula with n variables. Then, if $tow(h) \geq n$ the formula $\phi_m = \mathcal{F}(\phi_p) \wedge \mathcal{S} \wedge \mathcal{CA}(n)$ has modality depth at most $4 + h$.*

Proof. First observe that the modality depth of ϕ_m is at most

$$3 + \max_{0 \leq i \leq n} \mathrm{md}(v_i).$$

Therefore, we just have to bound the modality depth of v_i.

We will use induction on h to show that $tow(h) \geq n \Rightarrow \mathrm{md}(v_n) \leq h+1$. For $h = 0$ we have $tow(h) \geq n \Rightarrow n = 0$, therefore $\mathrm{md}(v_0) = 1$ and the proposition holds.

Suppose that the proposition holds for h.

Observe that $\mathrm{md}(v_n) \leq 1 + \max_{0 \leq i \leq \log n}\{\mathrm{md}(v_i)\}$ because writing n in binary takes at most $\log n + 1$ bits. If we have $n \leq tow(h+1)$ then $\log n \leq tow(h)$. From the inductive hypothesis $\mathrm{md}(v_i) \leq h + 1$ for $i \leq \log n$. Therefore, $\mathrm{md}(v_n) \leq h + 2$ and the proposition holds.

□

Theorem 5.6. *There is no algorithm which can solve modal satisfiability in* K *for formulas with a single variable and modality depth d in time $f(d) \cdot poly(|\phi|)$ with $f(d) = O(tow(d-5))$, unless P=NP.*

Proof. Suppose that there exists an algorithm A which in time $f(d) \cdot poly(|\phi|)$ can decide if a modal formula ϕ with modality depth d and just one variable is satisfiable. We will use this algorithm to solve propositional satisfiability in polynomial time.

Given a propositional CNF formula ϕ_p we construct ϕ_m as described, and if ϕ_p has n variables let $H = \min\{h \mid n \leq tow(h)\}$. Then $\mathrm{md}(\phi_m) \leq H + 4$ and of course ϕ_m can be constructed in time polynomial in $|\phi_p|$. Now we can use the hypothetical algorithm to see if ϕ_m is satisfiable.

We have that $f(d) = O(tow(d-5))$. Therefore, running this algorithm will take time $f(H+4) \cdot poly(|\phi_m|) = O(tow(H-1) \cdot poly(|\phi_m|))$. But by the definition of H we have $tow(H-1) \leq n$, therefore this bound is polynomial in $|\phi_m|$ and therefore, also in $|\phi_p|$, which means that we can solve an NP-complete problem in polynomial

time.

□

5.2.4 Diamond Dimension

In this section we propose a structural characteristic of modal formulas called diamond dimension. This is an alternative natural formula complexity measure which intuitively bounds the size of a model required to satisfy a formula. As we will see the parameter dependence of a satisfiability algorithm for formulas of small diamond dimension is doubly exponential, immensely lower than the dependence for modal depth. However, we will also show a lower bound indicating that it is unlikely that an algorithm with singly exponential parameter dependence could exist for this measure.

Definition 5.6. Let ϕ be a modal formula in negation normal form, that is, with the $\neg$ symbol appearing only directly before propositional variables. Then its diamond dimension, denoted by $\mathrm{d}_\Diamond(\phi)$ is defined inductively as follows:

- $\mathrm{d}_\Diamond(p) = \mathrm{d}_\Diamond(\neg p) = 0$, if p is a propositional variable
- $\mathrm{d}_\Diamond(\phi_1 \wedge \phi_2) = \mathrm{d}_\Diamond(\phi_1) + \mathrm{d}_\Diamond(\phi_2)$
- $\mathrm{d}_\Diamond(\phi_1 \vee \phi_2) = \max\{\mathrm{d}_\Diamond(\phi_1), \mathrm{d}_\Diamond(\phi_2)\}$
- $\mathrm{d}_\Diamond(\Box\phi) = \mathrm{d}_\Diamond(\phi)$
- $\mathrm{d}_\Diamond(\Diamond\phi) = 1 + \mathrm{d}_\Diamond(\phi)$

Our goal with this measure is to prove that if $\mathrm{d}_\Diamond(\phi)$ is small then ϕ's satisfiability can be checked in models with few states. This is why the two properties of ϕ which can increase $\mathrm{d}_\Diamond(\phi)$ are $\Diamond$ (which requires the creation of a new state) and $\wedge$ (which requires the creation of states for both parts of the conjunction).

Lemma 5.7. *If a modal formula ϕ is satisfiable and $\mathrm{d}_\Diamond(\phi) \leq k$ then there exists a Kripke structure with $O(2^{k/2})$ states which satisfies ϕ.*

Proof. Suppose that there exists a Kripke structure which satisfies ϕ, that is there exists some state s in that structure where ϕ holds. We will construct a working set of modal formulas S which will satisfy the following properties:

(i) All formulas in S hold in s.

(ii) $(\bigwedge_{\phi_i \in S} \phi_i) \to \phi$ is a valid formula.

(iii) $\mathrm{d}_\Diamond(\phi) \geq \sum_{\phi_i \in S} \mathrm{d}_\Diamond(\phi_i)$.

We begin with $S = \{\phi\}$ which obviously satisfies the above properties. We will apply a series of transformations to S while retaining these properties until eventually we reach a point where every formula in S is simple (in a sense we will make precise later) and then we will construct a model with the promised number of states for ϕ.

While possible we apply the following rules to S:

1. If there exists a formula $\psi \in S$ such that $\psi = \psi^1 \wedge \psi^2$ then remove ψ from S and add ψ^1 and ψ^2 to S.

2. If there exists a formula $\psi \in S$ such that $\psi = \psi^1 \vee \psi^2$ then remove ψ from S. If ψ^1 is true at state s add ψ^1 to S, otherwise add ψ^2 to S.

3. If there are two formulas $\Box\psi_i$ and $\Box\psi_j$ in S then remove them and insert the formula $\Box(\psi_i \wedge \psi_j)$.

It should be clear that rule 1 does maintain the properties of S. Rule 2 also maintains the properties: property (i) is maintained because we assumed that ψ is true at state S therefore if ψ^1 is not true we add ψ^2 which must be true. The other properties are also straightforward. Finally, the third rule maintains the properties of S because of the fact that $\Box\psi_i \wedge \Box\psi_j \leftrightarrow \Box(\psi_i \wedge \psi_j)$ is a valid formula.

It should be clear that applying all the rules until none applies will take polynomial time. When we can no longer apply the rules we have that $S = \{\Box\psi, \Diamond\phi_1, \ldots, \Diamond\phi_k, l_1, \ldots, l_m\}$, where the l_i are propositional literals; in other words, we have (at most) one formula that starts with a $\Box$, say $\Box\psi$, and some number k of formulas that start with $\Diamond$, say $\Diamond\phi_i,\ 1 \leq i \leq k$.

Now we will use induction on the diamond dimension to prove the lemma. Let $s(d)$ be a function which upper bounds the number of states in the smallest model which are needed to satisfy formulas of diamond dimension d (we are going to calculate $s(d)$ recursively and prove that it is finite). First, we can say that $s(0) = 1$, because a formula with diamond dimension 0 has no diamonds. Therefore, S contains one formula that starts with a $\Box$ and some literals, for which there exists an assignment to make them all true (because of the first property of S). Clearly, a model with just one state where we pick this assignment will also make the formula that starts with $\Box$ trivially true, and by the second property of S will satisfy ϕ.

For the inductive step, suppose that all the satisfiable formulas of dimension at most $d = \mathrm{d}_\Diamond(\phi)$ need at most $s(d)$ states to be satisfied. Let's consider the diamond dimension of all the formulas in S. There are three cases: either S does not have a formula that starts with a $\Box$, or it doesn't have any formulas that start with $\Diamond$, or it has both.

If we have no formulas starting with diamonds we can easily see that the same model as in the base case suffices, since $\Box\psi$ is trivially true at a state without successors. So in this case we have just one state.

Suppose that all the formulas in S are literals or start with $\Diamond$. In this case, we have for all ϕ_i that $\mathrm{d}_\Diamond(\phi_i) \leq \mathrm{d}_\Diamond(\phi) - k$. Using the inductive hypothesis we get that the number of states to satisfy each formula ϕ_i is at most $s(\mathrm{d}_\Diamond(\phi_i))$. Clearly, we can create a model which is the union of the models for all the ϕ_i plus one state where we give an appropriate assignment to the literals and appropriate transitions so that

$\Diamond\phi_i$ is true for all i. This model has at most $1+\sum_{i=1}^{k} s(\mathrm{d}_\Diamond(\phi_i)) \leq 1+k\cdot s(\mathrm{d}_\Diamond(\phi)-k)$ states.

Finally, if we have both types of formulas in S we construct the following model: consider all the formulas $\psi\wedge\phi_i$, for all i. Clearly, they are satisfiable, because $\Box\psi\wedge\Diamond\phi_i$ is true at s. We know from the third property of S that $\mathrm{d}_\Diamond(\phi) \geq \mathrm{d}_\Diamond(\psi) + k + \sum_{i=1}^{k} \mathrm{d}_\Diamond(\phi_i)$. Therefore, $\mathrm{d}_\Diamond(\psi \wedge \phi_i) = \mathrm{d}_\Diamond(\psi) + \mathrm{d}_\Diamond(\phi_i) \leq \mathrm{d}_\Diamond(\phi) - k - \sum_{j\neq i} \mathrm{d}_\Diamond\phi_j \leq \mathrm{d}_\Diamond(\phi) - k$. Now, we take the union of the models for each $\psi \wedge \phi_i$, and each model has at most $s(d-k)$ states. We add one state and transitions to the appropriate states where $\psi \wedge \phi_i$ are true, which together with an appropriate assignment makes all formulas of S true at that state. The number of states is at most $1+k\cdot s(\mathrm{d}_\Diamond(\phi)-k)$.

From all the above cases we can upper bound $s(d)$ as $s(d) \leq \max_{1\leq k\leq d}\{1 + k \cdot s(d-k)\}$. The fastest growing of these functions, obtained for $k = 2$, is in turn upper-bounded by $O(2^{d/2})$.

□

Theorem 5.7. *Given a modal formula ϕ with v variables and diamond dimension $\mathrm{d}_\Diamond(\phi) = k$ we can solve the satisfiability problem for ϕ in time $2^{O(2^k\cdot v)}\cdot|\phi|$.*

Proof. From Lemma 5.7 it follows that if ϕ is satisfiable, this can be verified in a model of $O(2^{k/2})$ states. There are at most $2^{O(2^k)}$ Kripke frames from which we can get such models. For each we just enumerate through all possible assignments to the v variables in the $O(2^{k/2})$ states, a total of $2^{O(2^{k/2}\cdot v)}$ different assignments. Once we have fixed a model deciding if ϕ holds can be done in bilinear time.

□

Lower Bound

We will now present a lower bound argument showing that, under reasonable complexity assumptions, the results we have shown for diamond dimension cannot be improved significantly. We will once again encode a propositional 3-CNF formula ϕ_p

into a modal formula ϕ_m, this time with a goal of achieving small diamond dimension. We will also use a small number of propositional variables. We assume without loss of generality that we are given a 3-CNF formula ϕ_p with n variables, where n is a power of 2.

Let $\Box^j$ be short-hand for j consecutive repetitions of $\Box$, with $\Box^0\phi$ being equivalent to ϕ. We recursively define the formulas $F(i)$ as $F(0) := \Box\bot$ and $F(i) := \left(\Diamond(\bigwedge_{j=0}^{i-1}\Box^j b_i)\right) \wedge \left(\Diamond(\bigwedge_{j=0}^{i-1}\Box^j \neg b_i)\right) \wedge \Box F(i-1)$, where b_i are propositional variables. It is not hard to see that $\mathrm{d}_\Diamond(F(i)) = 2i$ and also that $F(i)$ can only be satisfied in a model with at least 2^i states. The model to keep in mind here is a complete binary tree of height i.

We will use the formula $F(\log n)$ to encode a 3-CNF formula with n variables and each leaf of the tree that must be constructed to satisfy it will correspond to a variable. It is now natural to encode the variables of the original formula using their binary representation. We define $B(x_k) := \bigwedge_{k_i=1} b_i \wedge \bigwedge_{k_i=0} \neg b_i$, where once again k_i denotes the i-th bit in the binary representation of k, now with the least significant bit numbered 1.

Our modal formula will also have a propositional variable y which will be true at leaves that correspond to variables of the 3-CNF formula that must be set to true. We encode a literal consisting of the variable x_k as $L_1(x_k) := \Box^{\log n}(B(x_k) \to y)$. The corresponding negated literal is $L_2(\neg x_k) := \Box^{\log n}(B(x_k) \to \neg y)$. A clause is encoded as the disjunction of the encodings of its three literals. Our final modal formula ϕ_m is a conjunction of $F(\log n)$ with the encodings of all the clauses of the propositional formula ϕ_p.

Lemma 5.8. *Given a propositional 3-CNF formula ϕ_p the modal formula ϕ_m is satisfiable in* K *iff ϕ_p is satisfiable.*

Proof. Suppose that ϕ_p is satisfiable. We construct a binary tree of height $\log n$ as our model and ϕ_m will be made true at the root. It is not hard to satisfy $F(\log n)$

at the root: simply set $b_{\log n}$ to be true on all states on one of the subtrees of height $\log n - 1$ and false in all states of the other, then proceed to satisfy $F(\log n - 1)$ at the subtrees recursively in the same manner. Every leaf of the model corresponds to a variable of ϕ_p if we read the variables b_i as encoding the binary representation of the index of the variable. We set y to be true at the leaves that correspond to variables which are true at a satisfying assignment. It is not hard to see that this satisfies the encoding of all the clauses on the assumption that we started with an assignment satisfying ϕ_p.

Now for the other direction, suppose that ϕ_m is satisfied in a state of some model. A first observation is that for all $i \in \{1, \ldots, n\}$ there must exist a state in which $B(i)$ holds and is at distance $\log n$ from the state where ϕ_m holds, as this is required for $F(\log n)$ to hold. From this we can infer that $\Box^{\log n}(B(i) \to y)$ and $\Box^{\log n}(B(i) \to \neg y)$ cannot both hold in the state where ϕ_m holds. Therefore, we can extract a consistent assignment for the variables of ϕ from the model, by setting to true the x_i for which $\Box^{\log n}(B(i) \to y)$ holds. It is not hard to see that this assignment must satisfy ϕ_p because its clauses are encoded in ϕ_m.

□

Now that we have described how to embed a 3-CNF formula into a modal formula with only logarithmically many variables and logarithmic diamond dimension we can use this fact to prove a lower bound. This time we rely on the ETH. This allows us to obtain a much sharper bound than simply assuming that P≠NP.

Theorem 5.8. *There is no algorithm which can decide the satisfiability in* K *of a modal formula* ϕ *with* v *variables and* $\mathrm{d}_\Diamond(\phi) = k$ *in time* $2^{2^{o(v+k)}} poly(|\phi|)$ *unless the Exponential Time Hypothesis (ETH) fails.*

Proof. Suppose that an algorithm running in time $2^{2^{o(v+k)}} poly(|\phi|)$ did exist. Then we could use the described construction to decide 3-CNF satisfiability for any formula

with n variables. It is not hard to see that $v + k = O(\log n)$ and that the size of the produced modal formula is polynomial in the size of the 3-CNF formula, thus this would give an algorithm running in time $2^{o(n)}$, contradicting the ETH.

□

5.2.5 Modal Width

In this section we give another structural parameter for modal formulas called modal width. We will show that satisfiability can be solved in time only singly exponential in the modal width and v. Thus, we will give an algorithm that works more efficiently for the class of modal formulas which have small width.

To give some intuition, the modal width measures how many different modal subformulas our formula contains at depth i. The idea is that the truth value of the subformulas of depth i at some state s depends only on the truth value of the subformulas of depth $i+1$ at the successors of s. If the maximum width of the formula is bounded we can exhaustively check all possible truth values for subformulas at the next level of depth and decide if some particular truth assignment to the subformulas of depth i is possible. Using this idea it is possible to obtain an algorithm with the promised running time if we use a dynamic programming technique.

First we define inductively the function $\mathrm{sub}(\phi)$ which given a modal formula returns a set of modal formulas. Intuitively, whether ϕ holds in a given state s of a Kripke structure depends on two things: the values of the propositional variables in s and the truth values of some formulas ψ_i in the successor states of s. These formulas are informally the subformulas of ϕ which appear at modal depth 1 and $\mathrm{sub}(\phi)$ gives us exactly this set of formulas.

- $\mathrm{sub}(p) = \emptyset$ if p is a propositional variable
- $\mathrm{sub}(\neg\phi) = \mathrm{sub}(\phi)$, $\mathrm{sub}(\phi_1 \vee \phi_2) = \mathrm{sub}(\phi_1 \wedge \phi_2) = \mathrm{sub}(\phi_1) \cup \mathrm{sub}(\phi_2)$

- $\text{sub}(\Box\psi) = \text{sub}(\Diamond\psi) = \{\psi\}$

Now we inductively define the set $S_i(\phi)$, which intuitively corresponds to the set of subformulas of ϕ at depth i.

- $S_1(\phi) = \text{sub}(\phi)$
- $S_{i+1}(\phi) = \bigcup_{\psi \in S_i(\phi)} \text{sub}(\psi)$

Finally, we can now define the modal width of a formula ϕ at depth i as $\text{mw}_i(\phi) = |S_i(\phi)|$ and the modal width of a formula as $\text{mw}(\phi) = \max_i \text{mw}_i(\phi)$.

Observe that, as in the case of modal depth, negations do not affect the width of a formula. Therefore, the following results, which we state in terms of the satisfiability problem, also apply to the validity problem.

The following lemma is a basic observation regarding $\text{mw}_i(\phi)$ and $\text{md}(\phi)$.

Lemma 5.9. *For all $i \geq md(\phi)$ we have $mw_i(\phi) = 0$.*

Proof. Observe that for all formulas ϕ such that $\text{md}(\phi) \geq 1$ we have $\text{md}(\phi) > \max_{\psi \in \text{sub}(\phi)} \text{md}(\psi)$. Using this fact the proof follows easily by induction on $\text{md}(\phi)$.

□

Theorem 5.9. *There exists an algorithm which decides the satisfiability of a modal formula ϕ with v variables, $md(\phi) = d$ and $mw(\phi) = w$ in time $O(2^{2v+3w} \cdot d \cdot w \cdot |\phi|)$.*

Proof. We will need to use a function $Prop(\phi)$ which, given a modal formula ϕ, returns a propositional formula which corresponds to ϕ with all modal subformulas replaced by new propositional variables. $Prop(\phi)$ can be inductively defined as follows (notice that once again we consider $\Diamond\phi$ as shorthand for $\neg\Box\neg\phi$):

- $Prop(p) = p$ if p is a propositional variable;
- $Prop(\phi_1 \vee \phi_2) = Prop(\phi_1) \vee Prop(\phi_2)$;

- $Prop(\phi_1 \wedge \phi_2) = Prop(\phi_1) \wedge Prop(\phi_2)$;
- $Prop(\neg\phi) = \neg Prop(\phi_1)$;
- $Prop(\Box\phi) = q_j$, where q_j is a new propositional variable.

Let $P = \{p_1, p_2, \ldots, p_v\}$ be the set of propositional variables appearing in ϕ. For all $i \in \{0, \ldots, d-1\}$, for all $P' \subseteq P$ and for all $S' \subseteq S_i(\phi)$ we define the formula $F(i, P', S')$,

$$F(i, P', S') = \bigwedge_{p_j \in P'} p_j \ \wedge \bigwedge_{p_j \in P \setminus P'} \neg p_j \ \wedge \bigwedge_{\psi \in S'} \psi \ \wedge \bigwedge_{\psi \in S_i(\phi) \setminus S'} \neg\psi.$$

Clearly there are at most $2^{v+w}d$ formulas $F(i, P', S')$ defined and for each one of these we will compute whether it is satisfiable or not using dynamic programming. We will use a boolean matrix $A(i, P', S')$ of size $2^{v+w}d$ to store the results.

First, we have $S_d(\phi) = \emptyset$. It is not hard to see that all formulas $F(d, P', \emptyset)$ are indeed satisfiable, so we initialize the corresponding entries in A to True. Suppose now that for some i we have filled out completely all entries $A(i+1, P', S')$. We will show how to fill out any position in row i, say position $A(i, P', S')$. The crucial part now is that if we consider the formula $Prop(F(i, P', S'))$, it will have some new variables q_i which correspond to modal subformulas which all appear in $S_{i+1}(\phi)$.

The formula $Prop(F(i, P', S'))$ has at most $v + w$ variables. It is not hard to see that if $F(i, P', S')$ is satisfiable, then $Prop(F(i, P', S'))$ is also satisfiable, so our first step is to check this. The truth assignments for the v variables are easy to infer, therefore we only need to go through the 2^w possible assignments for the new variables. For each satisfying assignment we find we then need to check if a model that satisfies $F(i, P', S')$ can be built from it.

So, suppose that Q is the set of new variables, and we have found an assignment which sets the variables of $Q' \subseteq Q$ to *true* and the rest to *false* and satis-

fies $Prop(F(i, P', S'))$. Each variable q_j of Q corresponds to a formula $\Box\phi_j$ with $\phi_j \in S_{i+1}(\phi) \cup P$. If $q_j \in Q'$ we must make sure that ϕ_j is true at all successors of the state s where $F(i, P', S')$ will hold, in the model we are building. Let $S'' \subseteq S_{i+1}(\phi) \cup P$ be the set of formulas ϕ_j which we conclude that must hold in all successors of s in this way.

If $q_j \notin Q'$ we have that $\neg\Box\phi_j$ must hold in s, thus s must have a successor where $\neg\phi_j$ is true, or equivalently ϕ_j is false. Let $S^* \subseteq S_{i+1}(\phi) \cup P$ be the set of formulas ϕ_j for which we conclude that they must be false in some successor of s in this way.

To decide if it is possible to build appropriate successors to s so that all these conditions are satisfied, we look at row $i + 1$ of A. Specifically we consider the set of entries $A(i+1, P', S')$ such that $S'' \subseteq S' \cup P'$ and $A(i+1, P', S') = T$. Informally, these correspond to formulas which are satisfiable (because the corresponding entry is set to *true*) and which also can serve as successors to s without violating the conditions of S'', that is, in any state where they hold all formulas which we need to be true at all successors of s are indeed true. Now, we simply check if for each $\phi_j \in S^*$ there exists an entry in the set we have selected so far with $\phi_j \notin S' \cup P'$. If this is the case we can conclude that $F(i, P', Q')$ is satisfiable and set the corresponding entry of A to *true*, otherwise we conclude that no satisfying model can be built from the assignment we get from Q, even though $Prop(F(i, P', S'))$ is satisfied. This whole process of computing S'' and S^* and checking through row $i+1$ of A can be performed in time $O(w \cdot 2^{v+w}|\phi|)$.

To decide if the initial formula ϕ is satisfiable, we compute $Prop(\phi)$ and perform the same process: for every satisfying assignment of $Prop(\phi)$ we look at corresponding entries of row 0 of A to see if a model for ϕ can be built. The total time for this algorithm is $O(2^{3w+2v}wd|\phi|)$, because for each of the at most $2^{v+w}d$ entries of A we need to check through at most 2^w assignments and for each we spend at most $O(w \cdot 2^{v+w}|\phi|)$.

□

Lower Bound

Intuitively, one would probably not expect that a significantly better algorithm is possible in this case, since the algorithm we have described is singly exponential in the parameter $v + w$. Indeed, it follows if one accepts the ETH that for formulas of width 0 (that is, propositional formulas) it is not possible to achieve time $2^{o(v+w)}$. Nevertheless, this kind of lower bound argument is not entirely satisfactory for our purposes, since it completely neglects the contribution of the modal width to the problem's hardness. For all we know, the best algorithm's dependence on w alone might be sub-exponential, though this would be surprising.

However, a more careful examination of the lower bound arguments we have presented for diamond dimension is useful here. The formulas constructed there have a logarithmic number of variables and linear modal width. Therefore, an algorithm which in general runs in time $2^{O(v)+o(w)}$ would in this case give an algorithm running in time $2^{o(n)}$ for propositional SAT, contradicting the ETH. In addition, even if one assumes a constant v, things cannot improve much. A second reading of the lower bound argument for modal depth shows that our construction has modal width $O(n \cdot polylog(n))$. This implies that any algorithm which runs in $2^{O(w^c)}$ for any $c < 1$ in the case of constant v would imply a $2^{o(n)}$ algorithm for SAT, again contradicting the ETH. Thus, the existence of an algorithm with significantly better dependence on w than the one presented here is unlikely.

5.3 Vertex Cover

In this section we will deal with a simpler problem: VERTEX COVER parameterized by the number of edges one needs to delete from the input graph to make it bipartite. As

we will see, it is not very hard to obtain an FPT algorithm for this problem. However, the more interesting part is that we use this algorithm as a black box to obtain a non-trivial FPT algorithm for another parameterization of VERTEX COVER. Thus, one of the points of this section is to illustrate that solutions for structural parameterizations of common problems can come in handy in many cases. The results of this section have appeared in Gutin et al. [2011a].

5.3.1 Parameterizations Above Tight Bounds

Recall that, for the natural parameterization of VERTEX COVER, that is, when the solution size k is the parameter, the problem is FPT. This can be very useful in cases where k has moderate values, much smaller than n. It is of little use, however, when k is also expected to be large.

A typical such case is when the input graph has low maximum degree. In a graph with m edges and maximum degree Δ we know that any vertex cover must have size at least $\frac{m}{\Delta}$, since no vertex can cover more than Δ edges. Therefore, it makes little sense to investigate the case where k is small since we know a lower bound on k. Rather, it was suggested in Mahajan et al. [2009] that instead one could study VERTEX COVER parameterized *above* the lower bound. In other words, we are given an input graph with maximum degree Δ and m edges and we are asked, is there a vertex cover with at most $\frac{m}{\Delta} + k$ vertices? The question now is if we can obtain an FPT algorithm when the parameter k is the distance from the known lower bound, even if we restrict Δ to be constant.

Let us mention here, that such parameterizations above or below tight bounds have been of much interest recently in the parameterized complexity community, both for their practical applications and their theoretical interest. For example, for VERTEX COVER it is now known that it is FPT parameterized by the number of vertices one is allowed to use beyond the size of the graph's maximum matching

from the results of Razgon and O'Sullivan [2009] and Mishra et al. [2007]. A few other such parameterizations for VERTEX COVER are given in Gutin et al. [2011a]. Many interesting results have also been obtained recently for parameterizations of satisfiability problems relative to tight bounds (see for example Gutin et al. [2011b, 2010]).

5.3.2 Vertex Cover on Almost Bipartite Graphs

Consider the following problem: given a graph G such that deleting a set of e of its edges the graph becomes bipartite (the set of edges is also given). Find an optimal vertex cover of G.

Obviously, in general this problem is no easier than VERTEX COVER. It may become easier though if we only consider cases where e is much smaller than the size of the graph. In other words, we would like to parameterize VERTEX COVER by the input graph's distance from being bipartite. Is this problem FPT?

Theorem 5.10. VERTEX COVER *is FPT when parameterized by the number of edges that need to be deleted to make the input graph bipartite.*

Proof. There are at most 2^e minimal covers for the e edges whose removal makes the graph bipartite: for each edge we can either select its first or its second endpoint. For each of these minimal covers repeat the following: remove the cover's vertices from the graph. Now the graph must be bipartite, since the removed vertices cover all of the e edges. Solve VERTEX COVER on the remaining graph optimally in polynomial time. In the end, select the best of the solutions computed this way.

It is not hard to see that this algorithm take $O(2^e n^{O(1)})$ time and that it will always return an optimal solution, because every feasible vertex cover must contain one of the minimal covers of the e bipartizing edges.

□

Let us also remark that Theorem 5.10 holds even if the set of edges is not given in the input. Thanks to the results of Guo et al. [2006] we can find a set of bipartizing edges of size e in time FPT parameterized by e.

Let us now move to apply Theorem 5.10 to the problem's parameterization above a tight bound. Consider the following problem: given a graph G with m edges and maximum degree Δ we are asked if there exists a vertex cover of size $\frac{m}{\Delta} + k$. We will look for an FPT algorithm when the problem is parameterized by Δ and k.

First, is it easy to determine if the answer is YES, even when $k = 0$? The answer is given by the following lemma.

Lemma 5.10. *If a graph G with m edges and maximum degree Δ has a vertex cover of size $\frac{m}{\Delta}$ then G is bipartite.*

Proof. Let V_0 be a vertex cover of the prescribed size. Observe that V_0 must induce an independent set, because if V_0 contains both endpoints of an edge then in total it covers strictly fewer than $|V_0| \cdot \Delta = m$ edges. But $V \setminus V_0$ is also an independent set, since V_0 is a vertex cover, so G is bipartite.

□

The above lemma gives us some insight into the structure of graphs where the vertex cover is almost equal to $\frac{m}{\Delta}$. It can be generalized for any k.

Lemma 5.11. *If a graph G with m edges and maximum degree Δ has a vertex cover of size $\frac{m}{\Delta} + k$ then there exists a set D of at most $k\Delta$ edges whose removal makes G bipartite.*

Proof. Again, let V_0 be a vertex cover of the prescribed size. Then, if S is the set of edges induced by V_0 we know that V_0 cannot cover more than $|V_0| \cdot \Delta - |S| = m + k\Delta - |S|$ edges. So we have $k\Delta - |S| \geq 0$. But removing S makes the graph bipartite.

□

Putting it all together, we can get an FPT algorithm for VERTEX COVER parameterized above $\frac{m}{\Delta}$. First, look for a set of bipartizing edges of size $k\Delta$ using the FPT algorithm of Guo et al. [2006]. If none exists, by Lemma 5.11 the answer is NO. If one is found, use the FPT algorithm of Theorem 5.10 to find the optimal vertex cover.

Chapter 6

Conclusions

In this work we presented several results in the theory of structural parameterizations of hard problems. This is a relatively young but rapidly evolving and advancing field of algorithmic research and hopefully the results here serve to give a small taste of the possibilities. Here, let us briefly outline a couple of future research directions.

First, there is definitely much more to be done in the area of meta-theorems for undirected graphs. This is an area which shares a lot with the field of descriptive complexity theory, though working in a slightly different setting. There is a lot of room for innovation here, both in the definition of various widths but also in the definition of new logics to capture interesting families of problems. This was for example recently explored in Pilipczuk [2011], where a meta-theorem was proved for problems expressible in some modal logic. On the other hand, there are still open problems for FO and MSO logic. To give a concrete one, the complexity of deciding MSO sentences parameterized by max-leaf is left open in this work.

Second, another topic which has been of interest in the community lately is the development of multiple-parameter theory, where one adds not only one but several parameters to a problem. In fact, the development of such a multivariate theory is the goal of a research program, as outlined in Fellows [2009]. In this work, we presented

several multi-parameter results, especially for the MAXIMUM PATH COLORING problem in Section 5.1. Much more work in this direction is possible, especially in the interplay between graph widths. One combination that could prove to be especially interesting is parameterizations by both treewidth and maximum degree.

Bibliography

Achilleos, Antonis, Michael Lampis, and Valia Mitsou. Parameterized modal satisfiability. In Abramsky, Samson, Cyril Gavoille, Claude Kirchner, Friedhelm Meyer auf der Heide, and Paul G. Spirakis, editors, *ICALP (2)*, volume 6199 of *Lecture Notes in Computer Science*, pages 369–380. Springer, 2010. ISBN 978-3-642-14161-4.

Achilleos, Antonis, Michael Lampis, and Valia Mitsou. Parameterized modal satisfiability. *Algorithmica*, pages 1–18, 2011. ISSN 0178-4617. URL `http://dx.doi.org/10.1007/s00453-011-9552-z`. 10.1007/s00453-011-9552-z.

Anand, R.S., T. Erlebach, A. Hall, and S. Stefanakos. Call control with k rejections. *Journal of Computer and System Sciences*, 67(4):707–722, 2003. ISSN 0022-0000.

Arnborg, Stefan, Jens Lagergren, and Detlef Seese. Easy problems for tree-decomposable graphs. *J. Algorithms*, 12(2):308–340, 1991.

Arnborg, Stefan, Bruno Courcelle, Andrzej Proskurowski, and Detlef Seese. An algebraic theory of graph reduction. *J. ACM*, 40(5):1134–1164, 1993.

Barát, János. Directed path-width and monotonicity in digraph searching. *Graphs and Combinatorics*, 22(2):161–172, 2006.

Blackburn, Patrick, Johan F. A. K. van Benthem, and Frank Wolter. *Handbook of Modal Logic, Volume 3 (Studies in Logic and Practical Reasoning)*. Elsevier Science Inc., New York, NY, USA, 2006. ISBN 0444516905.

Bodlaender, Hans L. Treewidth: Algorithmic techniques and results. In Prívara, Igor and Peter Ruzicka, editors, *MFCS*, volume 1295 of *Lecture Notes in Computer Science*, pages 19–36. Springer, 1997. ISBN 3-540-63437-1.

Bodlaender, Hans L. Treewidth: Structure and algorithms. In Prencipe, Giuseppe and Shmuel Zaks, editors, *SIROCCO*, volume 4474 of *Lecture Notes in Computer Science*, pages 11–25. Springer, 2007. ISBN 978-3-540-72918-1.

Bodlaender, Hans L. Treewidth: Characterizations, applications, and computations. In Fomin, Fedor V., editor, *WG*, volume 4271 of *Lecture Notes in Computer Science*, pages 1–14. Springer, 2006. ISBN 3-540-48381-0.

Bodlaender, Hans L. and Arie M. C. A. Koster. Combinatorial optimization on graphs of bounded treewidth. *Comput. J.*, 51(3):255–269, 2008.

Bodlaender, Hans L., Fedor V. Fomin, Daniel Lokshtanov, Eelko Penninkx, Saket Saurabh, and Dimitrios M. Thilikos. (meta) kernelization. In *FOCS*, pages 629–638. IEEE Computer Society, 2009. ISBN 978-0-7695-3850-1.

Bodlaender, H.L., J.R. Gilbert, H. Hafsteinsson, and T. Kloks. Approximating treewidth, pathwidth, frontsize, and shortest elimination tree. *Journal of Algorithms*, 18(2):238–255, 1995. ISSN 0196-6774.

Chagrov, Alexander V. and Mikhail N. Rybakov. How Many Variables Does One Need to Prove PSPACE-hardness of Modal Logics. In Balbiani, Philippe, Nobu-Yuki Suzuki, Frank Wolter, and Michael Zakharyaschev, editors, *Advances in Modal Logic*, pages 71–82. King's College Publications, 2002. ISBN 0-9543006-2-9.

Chen, Jianer, Xiuzhen Huang, Iyad A. Kanj, and Ge Xia. Linear FPT reductions and computational lower bounds. In Babai, László, editor, *STOC*, pages 212–221. ACM, 2004. ISBN 1-58113-852-0.

Chen, Jianer, Iyad A. Kanj, and Ge Xia. Improved parameterized upper bounds for vertex cover. In Kralovic, Rastislav and Pawel Urzyczyn, editors, *MFCS*, volume 4162 of *Lecture Notes in Computer Science*, pages 238–249. Springer, 2006. ISBN 3-540-37791-3.

Corneil, Derek G. and Udi Rotics. On the relationship between clique-width and treewidth. *SIAM J. Comput.*, 34(4):825–847, 2005.

Cosmadakis, S.S. and C.H. Papadimitriou. The traveling salesman problem with many visits to few cities. *SIAM Journal on Computing*, 13:99, 1984.

Courcelle, Bruno. The Monadic Second-Order Logic of Graphs. I. Recognizable Sets of Finite Graphs. *Inf. Comput.*, 85(1):12–75, 1990.

Courcelle, Bruno and Stephan Olariu. Upper bounds to the clique width of graphs. *Discrete Applied Mathematics*, 101(1-3):77–114, 2000.

Courcelle, Bruno, Johann A. Makowsky, and Udi Rotics. Linear time solvable optimization problems on graphs of bounded clique-width. *Theory Comput. Syst.*, 33 (2):125–150, 2000.

Dankelmann, P., G. Gutin, and E.J. Kim. On Complexity of Minimum Leaf Out-Branching Problem. *Discrete Applied Mathematics*, 157(13):3000–3004, 2009.

Dawar, Anuj, Martin Grohe, Stephan Kreutzer, and Nicole Schweikardt. Approximation schemes for first-order definable optimisation problems. In *LICS*, pages 411–420. IEEE Computer Society, 2006.

de Berg, Mark and Ulrich Meyer, editors. *Algorithms - ESA 2010, 18th Annual European Symposium, Liverpool, UK, September 6-8, 2010. Proceedings, Part I*, volume 6346 of *Lecture Notes in Computer Science*, 2010. Springer. ISBN 978-3-642-15774-5.

Demaine, Erik D. and MohammadTaghi Hajiaghayi. The bidimensionality theory and its algorithmic applications. *Comput. J.*, 51(3):292–302, 2008.

Dendris, Nick D., Lefteris M. Kirousis, and Dimitrios M. Thilikos. Fugitive-search games on graphs and related parameters. *Theor. Comput. Sci.*, 172(1-2):233–254, 1997.

Downey, R.G. and M.R. Fellows. *Parameterized complexity*. Springer New York, 1999.

Eggan, LC. Transition graphs and the star-height of regular events. *Michigan Math. J*, 10(4):385–397, 1963.

Eppstein, D. Diameter and treewidth in minor-closed graph families. *Algorithmica*, 27(3):275–291, 2000. ISSN 0178-4617.

Erlebach, T. and K. Jansen. Maximizing the Number of Connections in Optical Tree Networks. In Chwa, Kyung-Yong and Oscar H. Ibarra, editors, *ISAAC*, volume 1533 of *Lecture Notes in Computer Science*, pages 179–188. Springer, 1998. ISBN 3-540-65385-6.

Erlebach, T. and K. Jansen. The complexity of path coloring and call scheduling. *Theoretical Computer Science*, 255(1-2):33–50, 2001a.

Erlebach, T. and K. Jansen. The maximum edge-disjoint paths problem in bidirected trees. *SIAM Journal on Discrete Mathematics*, 14(3):326–355, 2001b.

Erlebach, T., K. Jansen, C. Kaklamanis, M. Mihail, and P. Persiano. Optimal wavelength routing on directed fiber trees. *Theoretical Computer Science*, 221(1-2): 119–137, 1999.

Estivill-Castro, Vladimir, Michael R. Fellows, Michael A. Langston, and Frances A. Rosamond. FPT is P-Time Extremal Structure I. In Broersma, Hajo, Matthew Johnson, and Stefan Szeider, editors, *ACiD*, volume 4 of *Texts in Algorithmics*, pages 1–41. King's College, London, 2005. ISBN 1-904987-10-9.

Fagin, Ronald, Joseph Y. Halpern, Yoram Moses, and Moshe Y. Vardi. *Reasoning About Knowledge*. The MIT Press, 1995.

Fellows, Michael R. Towards fully multivariate algorithmics: Some new results and directions in parameter ecology. In Fiala, Jirí, Jan Kratochvíl, and Mirka Miller, editors, *IWOCA*, volume 5874 of *Lecture Notes in Computer Science*, pages 2–10. Springer, 2009. ISBN 978-3-642-10216-5.

Fellows, Michael R. and Frances A. Rosamond. The Complexity Ecology of Parameters: An Illustration Using Bounded Max Leaf Number. In Cooper, S. Barry, Benedikt Löwe, and Andrea Sorbi, editors, *CiE*, volume 4497 of *Lecture Notes in Computer Science*, pages 268–277. Springer, 2007. ISBN 978-3-540-73000-2.

Fellows, Michael R., Daniel Lokshtanov, Neeldhara Misra, Frances A. Rosamond, and Saket Saurabh. Graph layout problems parameterized by vertex cover. In Hong et al. [2008], pages 294–305. ISBN 978-3-540-92181-3.

Fellows, Michael R., Daniel Lokshtanov, Neeldhara Misra, Matthias Mnich, Frances A. Rosamond, and Saket Saurabh. The Complexity Ecology of Parameters: An Illustration Using Bounded Max Leaf Number. *Theory Comput. Syst.*, 45(4):822–848, 2009.

Flum, J. and M. Grohe. *Parameterized complexity theory*. Springer-Verlag New York Inc, 2006.

Fomin, Fedor V., Petr A. Golovach, Daniel Lokshtanov, and Saket Saurabh. Clique-width: on the price of generality. In Mathieu, Claire, editor, *SODA*, pages 825–834. SIAM, 2009.

Fomin, F.V., P.A. Golovach, D. Lokshtanov, and S. Saurabh. Intractability of clique-width parameterizations. *SIAM Journal on Computing*, 39(5):1941–1956, 2010. ISSN 0097-5397.

Frick, Markus and Martin Grohe. Deciding first-order properties of locally tree-decomposable structures. *J. ACM*, 48(6):1184–1206, 2001.

Frick, Markus and Martin Grohe. The complexity of first-order and monadic second-order logic revisited. *Ann. Pure Appl. Logic*, 130(1-3):3–31, 2004.

Ganian, Robert, Petr Hlinený, Joachim Kneis, Alexander Langer, Jan Obdrzálek, and Peter Rossmanith. On digraph width measures in parameterized algorithmics. In Chen, Jianer and Fedor V. Fomin, editors, *IWPEC*, volume 5917 of *Lecture Notes in Computer Science*, pages 185–197. Springer, 2009. ISBN 978-3-642-11268-3.

Ganian, Robert, Petr Hlinený, Joachim Kneis, Daniel Meister, Jan Obdrzálek, Peter Rossmanith, and Somnath Sikdar. Are there any good digraph width measures? In Raman, Venkatesh and Saket Saurabh, editors, *IPEC*, volume 6478 of *Lecture Notes in Computer Science*, pages 135–146. Springer, 2010. ISBN 978-3-642-17492-6.

Garey, M.R., D.S. Johnson, GL Miller, and C.H. Papadimitriou. The complexity of coloring circular arcs and chords. *SIAM Journal on Algebraic and Discrete Methods*, 1:216, 1980.

Garg, N., V.V. Vazirani, and M. Yannakakis. Primal-dual approximation algorithms for integral flow and multicut in trees. *Algorithmica*, 18(1):3–20, 1997.

Golumbic, Martin Charles and Robert E. Jamison. Edge and vertex intersection of paths in a tree. *Discrete Mathematics*, 55(2):151–159, 1985. ISSN 0012-365X.

Grohe, Martin. Logic, graphs, and algorithms. *Electronic Colloquium on Computational Complexity (ECCC)*, 14(091), 2007.

Gruber, Hermann and Markus Holzer. Finite automata, digraph connectivity, and regular expression size. In Aceto, Luca, Ivan Damgård, Leslie Ann Goldberg, Magnús M. Halldórsson, Anna Ingólfsdóttir, and Igor Walukiewicz, editors, *ICALP (2)*, volume 5126 of *Lecture Notes in Computer Science*, pages 39–50. Springer, 2008. ISBN 978-3-540-70582-6.

Gruber, Hermann and Markus Holzer. Finite automata, digraph connectivity, and regular expression size. Technical Report `http://drehscheibe.in.tum.de/forschung/pub/reports/2007/TUM-I0725.pdf.gz`, 2007.

Guo, Jiong, Jens Gramm, Falk Hüffner, Rolf Niedermeier, and Sebastian Wernicke. Compression-based fixed-parameter algorithms for feedback vertex set and edge bipartization. *J. Comput. Syst. Sci.*, 72(8):1386–1396, 2006.

Gutin, Gregory, Leo van Iersel, Matthias Mnich, and Anders Yeo. All ternary permutation constraint satisfaction problems parameterized above average have kernels with quadratic numbers of variables. In de Berg and Meyer [2010], pages 326–337. ISBN 978-3-642-15774-5.

Gutin, Gregory, Eun Jung Kim, Michael Lampis, and Valia Mitsou. Vertex cover problem parameterized above and below tight bounds. *Theory Comput. Syst.*, 48 (2):402–410, 2011a.

Gutin, Gregory, Eun Jung Kim, Stefan Szeider, and Anders Yeo. A probabilistic approach to problems parameterized above or below tight bounds. *J. Comput. Syst. Sci.*, 77(2):422–429, 2011b.

Halpern, Joseph Y. The effect of bounding the number of primitive propositions and the depth of nesting on the complexity of modal logic. *Artif. Intell.*, 75(2):361–372, 1995.

Halpern, Joseph Y. and Yoram Moses. A guide to completeness and complexity for modal logics of knowledge and belief. *Artif. Intell.*, 54(3):319–379, 1992. ISSN 0004-3702. doi: http://dx.doi.org/10.1016/0004-3702(92)90049-4.

Halpern, J.Y. and L.C. Rêgo. Characterizing the NP-PSPACE gap in the satisfiability problem for modal logic. *Journal of Logic and Computation*, 17(4):795, 2007.

Hlinený, Petr, Sang il Oum, Detlef Seese, and Georg Gottlob. Width parameters beyond tree-width and their applications. *Comput. J.*, 51(3):326–362, 2008.

Hong, Seok-Hee, Hiroshi Nagamochi, and Takuro Fukunaga, editors. *Algorithms and Computation, 19th International Symposium, ISAAC 2008, Gold Coast, Australia, December 15-17, 2008. Proceedings*, volume 5369 of *Lecture Notes in Computer Science*, 2008. Springer. ISBN 978-3-540-92181-3.

Hunter, Paul and Stephan Kreutzer. Digraph measures: Kelly decompositions, games, and orderings. In Bansal, Nikhil, Kirk Pruhs, and Clifford Stein, editors, *SODA*, pages 637–644. SIAM, 2007. ISBN 978-0-898716-24-5.

Immerman, N. *Descriptive complexity*. Springer Verlag, 1999. ISBN 0387986006.

Johnson, Thor, Neil Robertson, Paul D. Seymour, and Robin Thomas. Directed tree-width. *J. Comb. Theory, Ser. B*, 82(1):138–154, 2001.

Kleinberg, J. and É. Tardos. *Algorithm Design*. Pearson Education, 2006.

Kleitman, D.J. and D.B. West. Spanning trees with many leaves. *SIAM Journal on Discrete Mathematics*, 4:99, 1991.

Kreutzer, Stephan and Sebastian Ordyniak. Digraph decompositions and monotonicity in digraph searching. In Broersma, Hajo, Thomas Erlebach, Tom Friedetzky, and Daniël Paulusma, editors, *WG*, volume 5344 of *Lecture Notes in Computer Science*, pages 336–347, 2008. ISBN 978-3-540-92247-6.

Kreutzer, Stephan and Siamak Tazari. On brambles, grid-like minors, and parameterized intractability of monadic second-order logic. In Charikar, Moses, editor, *SODA*, pages 354–364. SIAM, 2010.

Kumar, S.R., R. Panigrahy, A. Russell, and R. Sundaram. A note on optical routing on trees. *Information Processing Letters*, 62(6):295–300, 1997.

Ladner, Richard E. The computational complexity of provability in systems of modal propositional logic. *SIAM J. Comput.*, 6(3):467–480, 1977.

Lampis, Michael. Algorithmic meta-theorems for restrictions of treewidth. In de Berg and Meyer [2010], pages 549–560. ISBN 978-3-642-15774-5.

Lampis, Michael. Algorithmic meta-theorems for restrictions of treewidth. *Algorithmica*, pages 1–19, 2011a. ISSN 0178-4617. URL `http://dx.doi.org/10.1007/s00453-011-9554-x`. 10.1007/s00453-011-9554-x.

Lampis, Michael. Parameterized maximum path coloring. *6th International Symposium on Parameterized and Exact Computation (IPEC) – to appear*, 2011b.

Lampis, Michael, Georgia Kaouri, and Valia Mitsou. On the algorithmic effectiveness of digraph decompositions and complexity measures. In Hong et al. [2008], pages 220–231. ISBN 978-3-540-92181-3.

Lampis, Michael, Georgia Kaouri, and Valia Mitsou. On the algorithmic effectiveness of digraph decompositions and complexity measures. *Discrete Optimization*, 8(1): 129–138, 2011.

Lenstra Jr, HW. Integer programming with a fixed number of variables. *Mathematics of operations research*, pages 538–548, 1983.

Mahajan, M., V. Raman, and S. Sikdar. Parameterizing above or below guaranteed values. *Journal of Computer and System Sciences*, 75(2):137–153, 2009.

Mishra, S., V. Raman, S. Saurabh, S. Sikdar, and C. Subramanian. The complexity of finding subgraphs whose matching number equals the vertex cover number. *Algorithms and Computation*, pages 268–279, 2007.

Niedermeier, R. *Invitation to fixed-parameter algorithms*. Oxford University Press, USA, 2006.

Obdrzálek, Jan. Dag-width: connectivity measure for directed graphs. In *SODA*, pages 814–821. ACM Press, 2006. ISBN 0-89871-605-5.

Papadimitriou, C.H. *Computational complexity*. Addison-Wesley, 1994.

Papadimitriou, Christos H. and Mihalis Yannakakis. Optimization, approximation, and complexity classes. *J. Comput. Syst. Sci.*, 43(3):425–440, 1991.

Pilipczuk, Michal. Problems parameterized by treewidth tractable in single exponential time: a logical approach. *CoRR*, abs/1104.3057, 2011.

Razgon, I. and B. O'Sullivan. Almost 2-sat is fixed-parameter tractable. *Journal of Computer and System Sciences*, 75(8):435–450, 2009.

Robertson, Neil and Paul D. Seymour. Graph minors. II. Algorithmic Aspects of Tree-Width. *J. Algorithms*, 7(3):309–322, 1986.

Robertson, Neil and Paul D. Seymour. Graph minors. I-XXIII. *J. Comb. Theory, Ser. B*, 1983-2004.

Seymour, Paul D. and Robin Thomas. Graph searching and a min-max theorem for tree-width. *J. Comb. Theory, Ser. B*, 58(1):22–33, 1993.

Spaan, E. *Complexity of modal logics*. PhD thesis, University of Amsterdam, 1993.

Wan, P.J. and L. Liu. Maximal throughput in wavelength-routed optical networks. *Multichannel Optical Networks: Theory and Practice*, 46:15–26, 1998.

West, D.B. *Introduction to graph theory*, volume 1. Prentice Hall Upper Saddle River, NJ, 2001.

Woeginger, G. Exact algorithms for NP-hard problems: A survey. In *Combinatorial Optimization-Eureka, You Shrink!*, volume 2570 of *Lecture Notes in Computer Science*, pages 185–207. Springer, 2003.