National Technical University of Athens

School of Applied Mathematics and Physical Sciences

Graduate Thesis

Parameterized Complexity and Model Checking on Bounded Families of Graphs

Author:
Anastasiadi Ellie

Supervisor:
Dr. Pagourtzis Aris

September 2018

Abstract

In this Thesis we are studying the Model Checking Problem in a Parameterized Framework. The main objective is to determine the impact of an increment in our descriptive capabilities to the complexity of the Model Checking Problem over Graphs. There are some results in the field pointing to an inversely proportional relation but only as an intuitive notion not properly analyzed. This relation is presented through the different parameters that must be utilized to classify the Model Checking Problem for a logic as FPT. From a graph theoretic approach we aim to express the boundary between instances that can be checked quickly for a property in contrast with the ones cannot. To prove this relation we are using graph decompositions and properties of parameters to establish either a hierarchy between them or to derive a measurement of the cardinality of the bounded instances. From the Computability side, parameterized circuits are used to predict such relations between logics of different expressive power.

Declaration

I declare that..

Acknowledgements

I want to thank...

Contents

List of Figures

Chapter 1

Introduction

In this Chapter i will try to summarize the reasons that lead me to this approach among with some basic arguments on the purposes of this attempt.

1.1 Idea Overview

The main objective of defining and studying the Model Checking problem is not to derive a "fast" algorithm that solves or describe a technique of generating such algorithms.At least not in this work.The Model Checking Problem gives us a way of formulating groups of problems (or properties) and deriving through its study information about their structure. Recently it has been studied excessively due to the applications it finds in formal verifications.

On the other end Theoretical Computer Science has developed a very big interest on descriptive frameworks. Upper bounds on the time an algorithm requires to check a property are derived from the logic required to define the property. A millstone in this direction was given by R.Fagin in 1973 proving that each property expressible in $\exists$ SO is checkable in NP-time.

The Model Checking as a formulation is very simple and does not require any particular effort towards that end. In the Classical Framework it is defined as:

φ-Model Checking
Input: A Model G and a Property φ.
Output: "Yes" iff G $\models \varphi$, "No" otherwise.

In a intuitive manner we expect the computational complexity of the Model Checking of a property to increase as the logical symbols required to express the property, become more complex. Of course that doesn't mean that expressing a property using more complex tools will increase the corresponding complexity. What i am interested in here is the computational

cost of properties when they are expressed in the *least* complex logical framework. A way to express this is through the Model Checking Problem (from now on refereed as MCP).

As you can see the MCP has two inputs and therefore its complexity will be somehow depending on both.It is obvious that the complexity as as function will be increasing as some part of the input does.My focus is to try and keep the complexity *low* as i increase one parameter and keep the other one bounded. The resulting complexity will depend on both the size of the model and the characteristics of the input property. In a closer look other characteristics of the Model might also be better for determining behavior towards complexity. This is why a parameterized framework would be more suitable for our study.

Parameterized Algorithms and Complexity are not new in the Computer Science community either. The initial work begun by Downey and Fellows in the early 90's. They initially proposed that the study of the complexity of various problems is often not realistic to happen in only terms of size of the input.From that begun a huge wave of research in the parameterized framework they proposed that resulted in very usefully results in both Algorithmic and Theoretical directions. The MC problem which i will be studding in this framework is defined as

PARAMETERIZED φ-MODEL CHECKING
Input: A Model G and a Property φ.
Parameter: l=$|\varphi|$
Output: "Yes" iff G $\models \varphi$, "No" otherwise.

Note here that the characteristics of the property or the model are left vague on purpose. We will expand on that when the theoretical background has been established.

To formulate the above correctly i am going to utilize knowledge from various fields. Mathematical Logic and Model Theory are used to categorize properties in the corresponding levels of expression power. Algorithmic schemes and Reductions will be given in a parameterized framework.For a problem to be characterized computable it will need to belong in the class FPT(Fixed Parameter Tractable) which is the parameterized equivalent of *easy* problems. Additionally i will try to assimilate all results with existing knowledge from Computability Theory.
The ultimate purpose is to derive a set of rules describing the above behavior in a way that could be utilized for the design of algorithms. My approach will try to mathematically relitivize the parameter that classifies a problem as FTP with the class of logic required to express the corresponding property. An example of such an accomplishment could be of the form ;

If a property ϕ cannot be expressed in the Logic Class A then

the parameter that classifies ϕ as FPT is an upper bound for each parameter classifying to FPT a property expressible in A.

Of course there is some amount or pre-existing work on the Parameterized Complexity of the MC Problem supporting the notion of a such corelation but not in a uniersal manner. Such are the work of Brunno Courcelle on the Model Checking for MSO and of Seesse on the expresibiliti of FPT decidable properties. !!!!!!!. Up until First Order Logic there is no need for bounds on the Size of the model since the whole Class can be clasified as FPT parameterized by the !!!!!! of the Properties. As we advance in classes such as Monadic Second Order , $\exists$ Second Order , $\forall$ Second Order or Second Order there is need for bounds to be applied in both parameters in order to remain FPT. I will try to interprete these works in the framework i am presenting. In order to make all the above more well defined i will try to put them in an algorithmic context.Anyone with even the slightest familiarity with Computer Sciense will have some characheristic problems and the coresponding algorithm tat solves it. How is though the problem and the solution assosiated with formal properties?

A Property is a mathematical expresion that uses logical symbols to express an idea. When those symbols are interpeted in some model the search for the property becomes the definition of the problem.

For instance given a Set α of numbers and the relation $\leq$ the property of P:

$$x \in A : \forall y \in A, x \leq y$$

would be interpeted as the property of x being the smallest number in A. The corresponding problem would be find the minimum element of A.

This is a very simple example though. As properties become more complex the notion of expressing them withought "hand-waving" and phisical language becomes more demanding. Sometimes the logical language doesent seem enough to express o property. But how do we know that we have reached the edge of exresiblity for the curent set of symbols? This is where the erenfrout Frasee!!!!!! games begun to answer this question fo t First order logic and triggered a similar movement for higher level logics. These will be explaind appropriately in the coresponding section.

Assuming now we have managed to categorize all properties we can think for Models such as Sets of Numebrs , Graphs , Funtions , Matrixes !!! , Matroids ,Latices ectettera can we derive from this characterization some sort of the Time a Machine wil need to recognise given property? Intuition would push us to say that the harder it is to express e paroperty the more time demanding would be its recognition. In deed all results this far have ton contradicted this statement but each one independent of the others withought a notion of larger scale behaviur. My work will focus on convincing

that such a behaviur exists and that there are tools that can give s some information about it.

1.2 Why Parameterized?

To give a full argument of that we will need of course a much more detailed introduction to the basic notions of classical and parameterized Complexity Theory. Nevertheless historically there is some valuable knowledge that can be utilized here.

The Complexity required to recognize sets of properties has been studied this far in both Frameworks. Early on in the history of computer Science the language theoretic approach for Computable properties was one of the most promising ones. Of course here there is a very large gap between the use of Logics or Grammars as the tool for expression. With Grammars and the separation of Languages into Context Free , Context Sensitive , Regular etcetera brought results that connected the corresponding language to the computational model able to recognise the hole group .INSERT HERE A TABLE THAT SUMMARISES From the side of Logic though the search for the limits of computation has had a more difficult road. The correspondence between the levels of the Polynomial Hierarchy and the Σ hierarchy over properties expressed with specific quantifier alterations is one of the most important reasons to pursue the notion of a strict correlation of the fields. The main meaning of this correspondence is that based of the alterations of quantifiers needed to express a property we can acquire an upper bound for the Checking Complexity. The converse would be that a known complexity for the checking of a property also provides an argument for its expressibility in one of the levels of the Σ hierarchy. The proof of this is not constructive and therefore does not provide algorithms for each case. Therefore, although extremely interesting the above are more of theoretic value. In reality those results are very difficult to utilize. For instance:

> A Programmer, Joe is trying to design an effective algorithm to recognize a new property in graphs. On his hands he has a rough drought of the algorithm and with the help of a Theoretical Computer Scientist he has also defined the property in a Logical Framework.He can use our theory and understand form the formulation of the property and check his algorithm is under the upper bound we provide. Or in the converse direction he can use the algorithm to check if the formulation is using minimal quantifier alterations (in the case is algorithm is optimal) . In both cases he cannot know if his algorithm is indeed *fast*

What was the problem in the above example then? The secret lies in the notion that both directions of the equivalence produce upper bounds. And

in addition the produce those upper bound in a non constuctive way. What Joe would want is a way to know that his algorithm cannot become more efficient. He wants a lower bound.

Attempts to produce more tight results have been made. As mentioned above by Fagin $\exists SO$ captures NP which means that every property expresible in $\exists SO$ will be checkable in NP-time and every property checkable in NP time will be expresible in $\exists SO$. Is that an improvement for Joe? We are still not providing him with any lower bounds. In deed the given upper bound is more tight and more elegant but it would have worned him that a competitor is able to improve the result significantly.

Of course a programmer reading all the above would probably think that in any case the proof that theory can provide for the existence of an algorithm is in no way close to the actual discovery. And in many ways he would be right. A good programmer when facing a problem is going to solve it by taking into consideration many variables. He can event try to minimize variables that the above theory can't even recognise such as number or proccesors , space , a spesific data structure and many more. How can a theorietic approach based on input length predict such detailed work on a spesific problem when the studies are takng place on groups of problems?

Don't rush to shout IT CAN'T!

But if you already did then you are not entirely mistaken. In order to provide Realistic Lower Bounds one must try to capture all this extra information. Many aproaches have arrised form such ideas. Propabilistic frameworks expore the Computational power of Traditional Models if allowed to give answers that will not always be correct. Aproximation algorithms also provide fastest results but sacrifice the optimal solution. The parameterized approach tries to constuct algorithms that wil be fast in a majority of the cases. This happens by utilizing spesific structural characteristics of a problem. By considericng these characteristics *bound by a constant irellevant of the size of the input* we are able to constuct prosedures that remain efficient even for NP-hard problems. The formal definition of this study aims to bring closer the design of algorithms and the exloration of stuctural parameters that have an impact on their complexity.

A very good way for someone to get familiar with the parametirized aproach is to imagine everything through models. In the most intimate form one can imagine models as graphs. There are many onter kinds of models but espessialy in Computer Sciense almost everything is expresible in a graph framewirk. In many cases our results will be spesificaly for graphs but this in not a general rule. Nevertheless the problem of searching a graph to see if it has a property is always way easier to compehend. In addition, defining families is also more intuitive since we can corespond a group in a spesific *image*. The defining of families of graphs with formal tools wil be analised later.

Under this notion our search for reasurment for Joe takes another turn. Now theoretical proofs over this framework can actualy interfere with other aspects of a problem rather than its formal definition. There are many exremely interesting results to support this such ass SEESE , COURCELLE , OUM , DEMAIN and many more. These results not only encurage the intuitive notion of this thoery but give appliable examples for aglorithm constuction. To expand i will need a background on some theoretic consepts.

While reading from now on keep in mind the 5 important questions of Computer Sciense Theory.

1. What is the purpose of this appoach?
2. Is this tool appropriate for the study of this subject?
3. What kind of assumtions are needed?
4. Are these assumptions realistic?
5. Can we do better?

Those will not only help someone understand the this work but in my opinion help in the general scientific procedure in this field.

Finaly it is appropriate to mention here that in the above list of questions one could maybe argue that maybe the approximation framework or the probabilistic one are more appropriate for such a study. Now of course advances in both of those areas are very helpful for the society of computer science and also provide powerful tools to confront intractability. However at least from my perspective they lack the characteristic of being very close to the intuition of the way we understand structures and graphs. These intuitive notions are much harder to be transmitted trough said frameworks.

This doesn't mean however that no interest exists there. In the final chapter of this thesis i will describe ways to both interpret results in those framework or reproduce some of the procedures described later but on classes described through other frameworks and not logic.

1.3 Historical Notes

Early in the 80's research pointed out that classical complexity might not be the ideal framework for some specific problems and corresponding algorithms. In particular, Vardi pointed out that the input for database-query evaluation consists of two components, query and database. For first-order queries, query evaluation is P-SPACE-complete, and for fix-point query it is EXPTIME-complete, but, if you fix the query, the complexity goes down to LOGSPACE and PTIME correspondingly. In particular, the size of the database was not the right complexity for database-query complexity and the size of parameter counted.[37]

> Those works initiated the notion the time needed for NP-Complete problems although exponential might accept improvements in the way the exponential will present itself.[36]

Parameterized complexity and algorithms have developed rapidly during the last three decades. Since the fundamental work of Downey and Fellows in[32, 33, 34] with a series of papers aiming to establish a global framework and present their previous work, parameterized complexity theory introduced numerous innovative ideas in algorithmic design and offered insightful results in almost all disciplines of theoretical computer science.

The first monograph in this field was the book:[R.G. Downey and M.R. Fellows.Parameterized complexity. Monographs in Computer Science. Springer-Verlag, New York,1999.] The next two monographs in the field appeared during 2006. One was the book: [Rolf Niedermeier.Invitation to fixed-parameter algorithms , volume 31 of Oxford Lecture Series in Mathematics and its Applications. Oxford University Press, Oxford, 2006] and the other was the book: [J. Flum and M. Grohe.Parameterized complexity theory . Texts in Theoretical Computer Science. An EATCS Series. Springer-Verlag, Berlin,2006]

In the same time and ever earlier completely different people studied independently the time complexity of recognizing parameters in graphs. The oldest relevant is of course treewidth for which the very first result that this thesis will utilize was presented by Bodlaender in a 1996 paper [31]. The problem of deciding if a graph has treewidth k is NP-complete if k is allowed to vary, was discussed a decade later in Arnborg, Corneil,and Proskurowski and in 1987 [25], those authors also showed that there was an $O(n^{k+2})$ for treewidth-k graphs. Robertson and Seymour [9] gave the first FPT (it was $O(n^2)$) algorithm in 1995. Their algorithm was based upon the minor well-quasi-ordering theorem, and thus was highly nonconstructive and non-elementary and had huge constants.

Such works initiated a very large wave of researchers to turn to the parameterized approach.The study of decidability results for monadic second-order theories of classes of combinatorial objects and their connections with automata has a long and interesting history, as can be found in Büchi [1]. Probably the first result is the one of Büchi who proved that a language is regular iff it is definable by some formula in monadic second-order logic (of strings). The famous hallmark result here is the decidability result of Rabin [10].

Detleff Seese was the first to conjecture that classes of graphs with decidable theories were "similar to trees" [23]. Seese was able to prove that if a class of planar graphs with maximal degree ≤ 3 avoids certain graphs as in-

duced subgraphs, then it has a very nice parse tree. Since Rabin had proven the decidability of the monadic second-order logic of trees, Seese was able to deduce that these theories were decidable. Of course, Rabin's method is not very feasible. Seese continued to pursue his conjecture that *tree-like* graphs were the boundary of decidability.

Chapter 2

Parameterized Complexity

This chapter is dedicated to present all of the tools i will be using and explain the most important notions of each theory.

2.1 Definitions

To begin we will present the notions of Parameterized Complexity in corespondance with Classical Complexity theory.

We are from now on concerned with problems(languages) of two inputs. One is the instance and one the parameter. When given a representation of an istance we count on our understunding of the idea it represents to understund it. Similarily when given a number that represents a parameter and its relation to the instance we are able to inetrpet it as a part of the problem. And above all the parameter is considered bounded and small.

Definition 2.1.1 (Parameterized Problem)**.** A *parameterization* of Σ^* is a *recursive* function $k : \Sigma^* \to \Sigma^*$. A *parameterized problem* is a tuple (L, k), where $L \subseteq \Sigma^*$ and k is a parameterization of Σ^*.

Our parameter k is defined by this recusrive function in order to demonstrate its relation with each instance. For readability purposes and withought loss of generality we will consider it to be and integer and therefore our Languages become of the form, $k : \Sigma^* \to \mathbb{N}$

The above definition is very close to the clasical complexity definiton of what we call a Language or Problem. Following that we have to define what is considered a parameterized *fast* algorithm.

Definition 2.1.2 (The Class FPT)**.** The class of parameterized problems that can be solved in time

$$O(f(k) * n^c)$$

, where f(k) is computable.

Note. The above definition might alert someone in the sence that it allows extremely large grouth functions regarding the parameter.

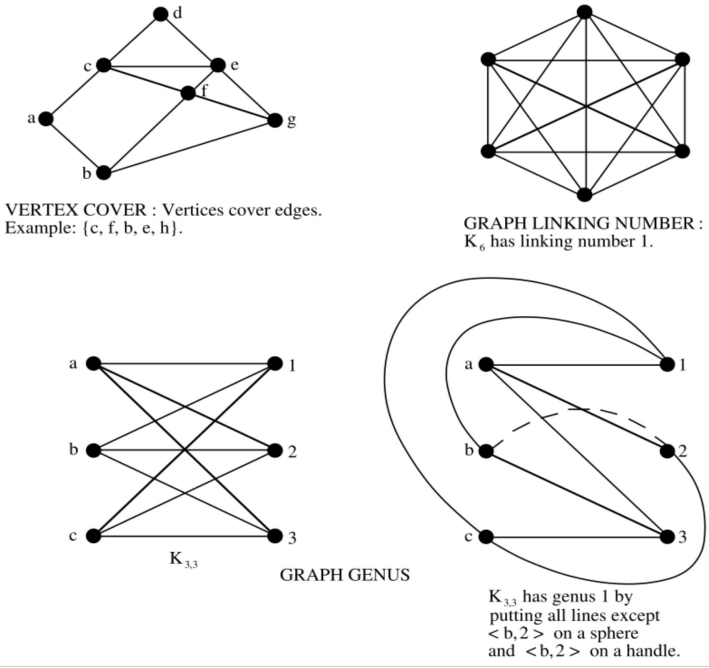
That is true but in the sense that the parameter is small these increments wil remain bounded by a constant. Another thing the reader might notice is that we are using the algorithm in a Turing-Machine sence although we haven't beed very spesific on th definition of it. To design algoriithms that are in agreement within the current standards of programing we will be using turing machines in the classes that have to do with algorithms.

Now is a good time to give some examples of parameterized problems. In this framework a problem always comes pared with a parameter. Paired with another parameter a problem might display a completely different complexity behaviour. For example, consider VERTEX COVER parameterized by k, defined as follows.

k- VERTEX COVER
Input: A Graph G=(V,E)
Parameter: An integer k
Output: "Yes" if G has a VC of size k , "No" otherwise.

As you can see we have used here the size of the solution to define our parameter. Here are some other examples for graph problmes that can also be parameterized similarly.

Figure 2.1: Other problems parameterized by the size of the solution



If we consider the clasical versions of the above problems where k is not fixed, they are all NP-Hard. However each one exhibits some form of parameterized tractability. Note here that there are two versions of Parameterized Tractability. Uniform and non-Uniform. The difference is that in the uniform case there exists an algorithm of the appropriate running time that solves the problem for each k. In the non- uniform case for each k there exists an FPT algorithm of the appropriate running time but there is no way to describe the prosedure for all k. This is a descrimination that arrises from the attempt to solve spesific NP-hard problems. In this approach the descrimination is not of great importance.

2.2 Parameters

It is already obvious that parameters can be of different *power*. An easy example is the minimun and maximum edge degree of a graph. Since always

$$Max_d \geq Min_d$$

we can assume that if a problems is FPT parameterized by Min_d it will be FPT parameterized by Max_d. Of course the relationship between parameters will be described way more explicitely later on but this already gives an image of how can someone approach a Problem from a paameterized point of view. If say it seems completely imposible to find an FPT algorithm for a problmem one can start thinking of more *strong* parameters.

By doing so someone must be carefull not to end up with a solution only because he choose an extremely stong parameterization . Although the formal definition does not forbid parameters that are comparable with the input something like this can be very easily proved to have no scientific interest. Say for instance that someone picks a parameter that can bound the total input . This is not hard to imagine but lets say that someone parameterizes a problem by n-1 where n is the size of the input.

Theorem 2.2.1. *If for a parameter k $\exists$ an increasing recursive function $g(k)$ such that $n \leq g(k)$ then all problems of classical complexity $\mathcal{O}(f(n))$, f:computable are FPT parameterized by k.*

Proof. Pick a problem P of classical complexity $\mathcal{O}(f(n))$. We know that $f(n)$ is increasing since it is a complexity function.Also $n \leq g(k) \rightarrow f(n) \leq f(g(k))$ therefore the Complexity of P is upper bound by $\mathcal{O}(f(g(k)))$ and hence in can be solved in such time.

Since f,g are computable and increasing $\rightarrow$ $f \circ g$ is computable and increasing. The above complexity is of the form

$$O(f(k) * n^c)$$

and therefore P is FPT . □

The correct usage of parameters for each problem is nt at all trivial and has presented a very wide arrea for research.

There is a very large amount of work focusing on new algorithmic techniques and tools that utilize all of the above notions and have been very fruitful this far. Advances in parameterized algorithms have produced a great amount of results that all are extremely interesting. However my interest is less problem and therefore are not presented here.

2.3 Hierarchy

Returning to the familiar for us examples we could attempt to parameterize the Graph Genus problem by the vertex cover of the input graph. Would that result in a fixed parameter problem? For spesific parameters and some spesific problems this question can be ansered easily. For problems in P by default all parameterizations result in FPT algorithms. But in the general case this question is a very complex and interesting one. The base of course for all arguments towards answering start from the definition of parameterized reductions.

Definition 2.3.1 (Parameterized Reduction)**.** For two parameterized problems (L, k) , (L', k'). We say that (L, k) reduces to (L', k') through an FPT-reduction (noted . $L \leq_{FPT} L'$) if there exists an alforithm R such that

1. For eatch $x \in \Sigma^*$, $x \in L \Leftrightarrow R(x) \in L'$
2. R is computed by an FPT-algorithm.
3. $k' = g(k)$, where $g : \mathbb{N} \to \mathbb{N}$ is computable.

If $A \leq_{FPT} B$ and $B \leq_{FPT} A$, we say that A, B are FPT-equivalent (noted. $A \equiv_{FPT} B$).

Using the above definition we can start building a complexity theory. Parameterized problems in FPT are grouped together on the notion that you can decide the yes-insances of each one independently withought exiding the time limits of the definition. Of course for each problem in FPT the algorithm might be of completely different complexity. And the parameters that reduce each one to FPT are not nececarily computaly bound by each other. The Turing machine model is not able to provide easily a hardness theory. This will be explained later.

Our complexity theory begins by defining classes of hard problems and describing relations between them. Of course as mentioned before one problem can be FPT on one parameterization and not FPT on another. Our parameterized reduction captures this by requesting the parameter of the new parameterized problem to be at least computably bound by the old one.

Looking at the classical complexity theory we will try to define the TURING MACHINE ACCEPTANCE in a parameterized framework in order to build up from there equivalences between problems

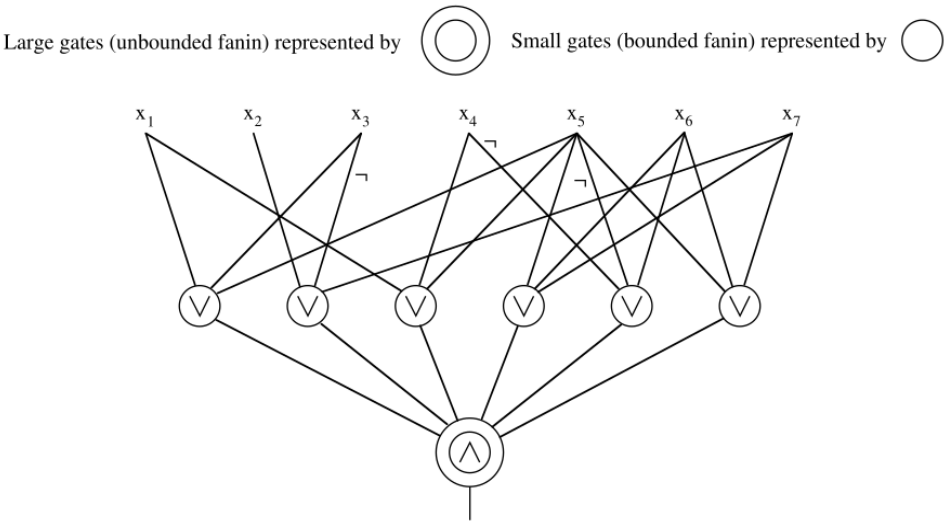
SHORT TURING MACHINE ACCEPTANCE

Input:	A nondeterministic Turing machine M and a string x.
Parameter:	An integer k
Output:	"Yes" if M has a computation path accepting x in $\leq$ k steps, "No" otherwise

It seems to us that if one accepts the philosophical argument that TURING MACHINE ACCEPTANCE is intractable, then the same reasoning would suggest that SHORT TURING MACHINE ACCEPTANCE is fixed-parameter intractable. We will soon establish that there are a large number of problems of the same fixed-parameter complexity as SHORT TURING MACHINE ACCEPTANCE . We believe that the existence of such a large number of problems of the same fixed-parameter complexity as SHORT TURING MACHINE ACCEPTANCE gives further weight to the thesis that SHORT TURING MACHINE ACCEPTANCE is not fixed-parameter tractable.

The main theorem is an analog of Cook's theorem. We will first establish some preliminary results. We will need some definitions. It is convenient to consider a 3CNF formula as a (boolean) circuit. Thus, a 3CNF formula is considered as a circuit consisting of one input (of unbounded fanout) for each variable, possibly inverters below the variable, and structurally a large and of small or's (of size 3) with a single output line.

Figure 2.2: A 3CNF fornula is a **large** AND of **small** ORs



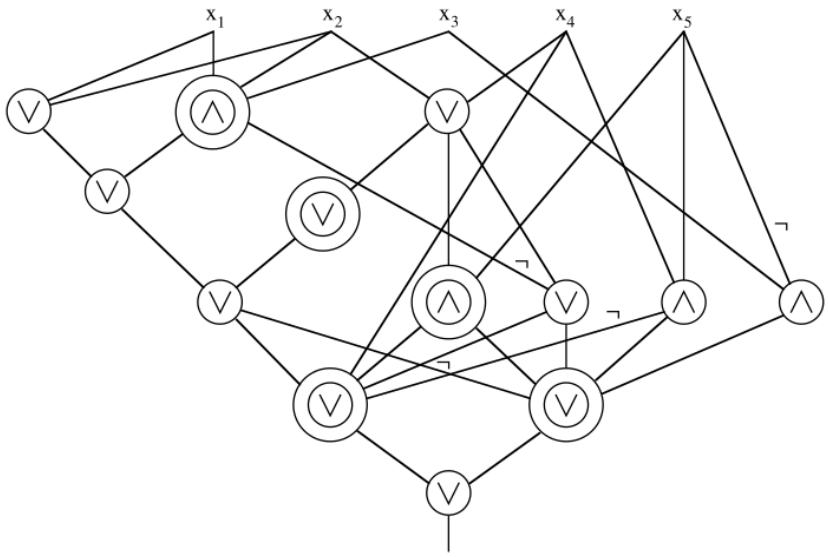
The reader should refer to Fig. 2.3. We can similarly consider a 4CNF formula to be a large and of small or's, where *small* is defined to be 4. More generally, it is convenient to consider the model of a decision circuit. This is a circuit consisting of large and small gates with a single output line, and no restriction on the fanout of gates. For such a circuit, the depth is the maximum number of gates on any path from the input variables to the output line, and the weft is the *large gate depth*. More formally, the weft of a circuit is defined as follows:

Definition 2.3.2. Let C be a decision circuit. The ***weft*** of C is defined to be the maximum number of large gates on any path from the input variables to the output line. (A gate is called large if its fan-in exceeds some prearranged bound. The reader should note that none of our results depend upon what the bound actually is.)

Let $F = \{C_1, ..., C_n, ...\}$ be a family of decision circuits. Associated with F is a basic parameterized language:

$$L_F = \{\langle C_i, k\rangle : C_1 \text{ has a weight k satisfying assignment } \}$$

Figure 2.3: A *weft* 2 *depth* 5 decision circuit.



For instance, if F is the family of boolean circuits corresponding to propositional formulas in 3CNF form, then L F corresponds to WEIGHTED 3CNF SATISFIABILITY . A generalization of the class of circuits corresponding to 3CNF formulas is provided by the following:

WEIGHTED WEFT t DEPTH h CIRCUIT SATISFIABILITY (WCS(t, h))

Input:	A weft t depth h decision circuit C
Parameter:	A positive integer k
Output:	"Yes" if Does C have a weight k satisfying assignment , "No" otherwise

Definition 2.3.3. Notation will be as follows:

- We will denote by $L_F(t, h)$ the parameterized language associated with the family of weft t depth h decision circuits.
- **(Basic hardness class)** We define a language L to be in the class $W[t]$ iff L is fixed-parameter reducible to $L_F(t, h)$ for some h

With the reduction matching the one given in definition 2.3.1

The final theorem of ths section will be an analog of cooks theorem that the readen can find the proof of in [35]. The theorem is given nevertheless because most of the proofs of chapter 5 are given in a circuit framework and thus it is important for the reader to know the relation between circuits and Turing machines.

Theorem 2.3.1. *(Analog of Cook's Theorem (I))*
The following are complete for $W[1]$*:*

- *WEIGHTED n-SATISFIABILITY for any fixed* $n \geq 2$*.*
- *SHORT TURING MACHINE ACCEPTANCE .*

So to summarize the class FPT is associated with circuits of 0 weft and a a constant k to bound the fan-in of the gates. $W[1]$ would be of weft 1 and so on. The current state of the parameterized hierarchy is given in the following schema:

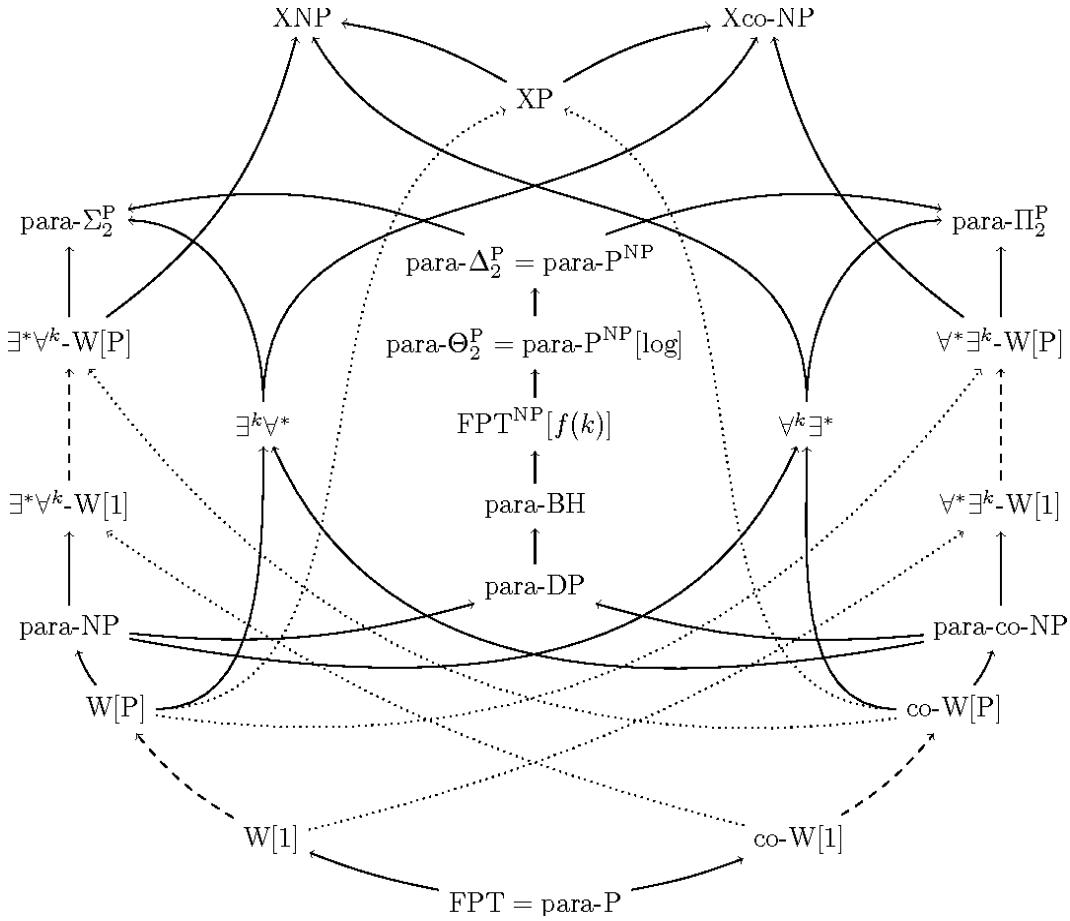


Figure 2.4: Parameterized Hierarchy

2.4 Parameterized Model Checking

One of the main themes in descriptive complexity theory is to study the complexity of problems of the following type:

> Given a finite structure $\boldsymbol{A}$ and a sentence ϕ of some logic L,
> decide if $\boldsymbol{A}$ satisfies ϕ

This problem , which will be called the *model-checking problem for L*, has several variants. For example, given a structure $\boldsymbol{A}$ and a formula $\phi(\overrightarrow{\chi})$ we may want to compute the set of all tuples $\overrightarrow{\alpha} \in A$ such that $\boldsymbol{A}$ satisfies $\phi(\overrightarrow{\alpha})$, or we may just want to count the number of such tuples. Often, we fix the sentence ϕ in advance and consider the problem :

> Given a structure $\boldsymbol{A}$ decide if $\boldsymbol{A}$ satisfies ϕ

Model-checking problems and their variants show up very naturally in various of applications in computer science such as Database Query Evaluation , Model-checking in Computer-Aided verification and Constraint Satisfaction Problems .

It is obvious already that proving that the Model-checking problems is FPT or FLP (= Fixed-Parameter Linear) seems to be a very meaningful statement in comparison to the notion that the Model-checking problem being in PTIME.

Now as one might predict there are two possible parameters in this quest. The sentence ϕ and the structure $\boldsymbol{A}$. Depending on which of these are considered parameters of the problems and which are fixed we get tree different versions of the MCP of logic L.

Complexity theory defines its main concepts via acceptance of string languages by computational devices such as Tuuring Machines. To talk about complexity of logics on finite structures, we need to encode finite structures and logical formulae os strings. For formulae we shall assume some natural encoding: for example $enc(\phi)$, the encoding of a formula could be its syntactic tree (represented as a string).

As for structures there are several different ways to encode them. They are all of course equivalent but given a specific one will make some proofs easier and will some times provide useful information on the running time of a procedure. Suppose we have a structure $\boldsymbol{A}$ over a vocabulary σ .

Let $\boldsymbol{A} = \{a_1, \ldots, a_n\}$. For encoding a structure we always assume an orderig on the universe. In some cases the ordering relation is part of the vocabulary and thus we use it. In others we just shoose and arbitrary one. The order in this case will have no effect on the result of algorithms and properties but we need it to be able to talk of computability and complexity.

We choose for now an order of the universe say $a_1 \leq a_2 \ldots \leq a_n$. Each k-ary relation R^A will be encoded by a $n^k - bit$ string $= enc(R^A)$ as follows:

- Consider an enumeration of all k-tuples over $\boldsymbol{A}$ in the lexicographic order.
- Let $\overrightarrow{a_j}$ be the jth tuple in this enumeration.
- the jth bit of $enc(R^A)$ is 1 if $\overrightarrow{a_j} \in R^A$ and 0 if $\overrightarrow{a_j} \notin R^A$
- σ contains only relations symbols since a constant can be encoded as a unary relation containing one element.

If $\sigma = \{R_1, \ldots R_p\}$ then the basic encoding of a structure is the concatination of then encodings of said relations . In the circuits computational model the the length of the input is a parameter of the model and thus the $|\boldsymbol{A}|$ can be easily calculated from the basic encoding but for turring machines the size of the input must be known by the device. For this reason we include in the encoding of $\boldsymbol{A}$ the size of the model as follows:

$$enc(\boldsymbol{A}) = 0^n * 1 * enc(R_1^A) * \ldots * enc(R_pA). \tag{2.1}$$

The length of the above string is denoted by $\| \boldsymbol{A} \|$ and is

$$\| \boldsymbol{A} \| = (n+1) + \sum_{i=1}^{p} n^{arrity(R_i)}$$

Now that we have an encoding of $\boldsymbol{A}$ we can talk about the complexity of the model checking problem.

Definition 2.4.1. Let K be a complexity class and L a logic.

1. The *data* complexity of L is K if for every sentence ϕ in L, the language

   $$\{enc(\boldsymbol{A}) | \boldsymbol{A} \models \phi\}$$

   belongs to K.

2. The *expression* complexity of L is K if for every finite structure $\boldsymbol{A}$ the language

   $$\{enc(\phi) | \boldsymbol{A} \models \phi\}$$

   belongs to K.

3. The *combined* complexity of L is K if the language

   $$\{enc(\phi), enc(\boldsymbol{A}) | \boldsymbol{A} \models \phi\}$$

   belongs to K.

Note here that data complexity refers to the complexity of finding all the properties that hold for a specific structure (size of structured is considered constant and ignored) while the expression complexity is the problem of recognizing the models that satisfy a specific property(this is the more common algorithm design notion that the reader is probably familiar with) . The combined complexity represents the general model problem with which i am focusing on.

The formalization i will follow from now on is :

k -ϕ MODEL CHECKING FOR ϕ
Input: A Graph G=(V,E) and ϕ
Parameter: Parameter k + $|\phi|$
Output: "Yes" if G $\models \phi$, "No" otherwise

As you see here the general term of structure changes to *Graph*. Our size N here denotes the number of nodes and the edge relation is the one and only member of σ

From now on all result are given for graphs sine i will be taking into account specific graph parameters. However through easy transformation processes that can be fount in [38] all structures and relations can be transformed to graphs with only the edge relation. What i will be looking for is to derive some rules that correlate the parameter k with the logic L as to end up with the above problem being in FPT.

A more detailed introduction in pamameterized complexity in Greek can be found in [44].

2.5 Graph Metrics

In this section i will give the basic definition and results one the graph metrics that will be necessary for later on. Mainly i will present only metrics that correspond to specific problems and theorems presented later. The same study can be done for a variety of others though if they appear to give promises towards descriptive results.

2.5.1 Treewidth

Treewidth is the most popular and old of our different parameters. It was introduced in the early 70's from independent researchers but it took the form that we study today in 1984 by Neil Robertson and Paul Seymour [6] in a series of papers on graph minors. After that it has been excessively used both in the design of algorithms and in the theoretical computability theory.

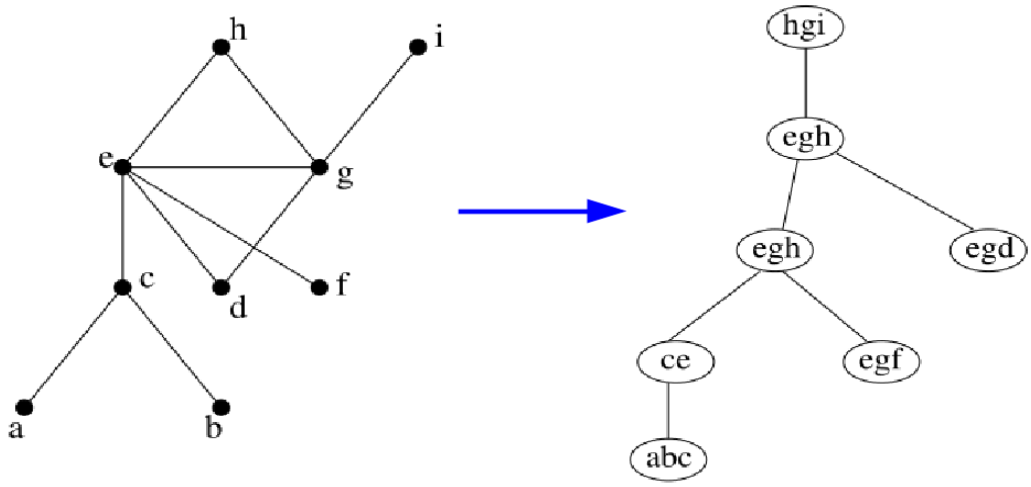
Treewidth is tightly bound to the notion of a tree decomposition. So:

Definition 2.5.1. (Tree Decomposition - Treewidth)

1. A **Tree decomposition** of a graph G=(V,E) is a tree T together with a collection of subsets T_x (called bags) of V labeled with the vertices x of T such that $\cup T_x = V$ and the following hold
   - For every edge uv of G there is a some x such that u,v $\in T_x$
   - If y is a vertex of on the unique path in T from x to z then $T_x \cap T_z \subseteq T_y$.
2. The width of a tree decomposition is the maximum value of $|T_x|$ -1 over all the vertices of the tree T of the decomposition.
3. The **treewidth** of Graph G is the minimum width of all thee decompositions of G.

There is a very large amount of results that were produced in the attempt to make the construction of tree decompositions easier . Here are some that would appear useful to be considered by the reader in order to understand

Figure 2.5: A Tree Decomposition of treewith 2



better the rest of this thesis. For e graph $G = (V, E)$ and treewith k we know:

1. $k \geq$maximum clique size of G

2. In every graph G, there exists a vertex of degree at most $tw(G)$.

Both of these proofs are quite easy and will be given in the appendix. Computing treewith is NP-complete in the classical framework as shown in 1987 from Arnborg, Corneil Proskurowski [25]. It was proved to be FPT on the clasical parameterization on size of the solution by Bodlaender and Kloks [27]in 1996.

Determining if a problem is FPT is usually done by excluding a specific set of minors as mentioned before in section 2.4 .In the case of families of bounded treewidth it is quite difficult ti give a specific selection of graphs but there are some results.

For every finite value of k, the graphs of treewidth at most k may be characterized by a **finite** set of forbidden minors. (That is, any graph of treewidth >k includes one of the graphs in the set as a minor.) Each of these sets of forbidden minors includes at least one planar graph.

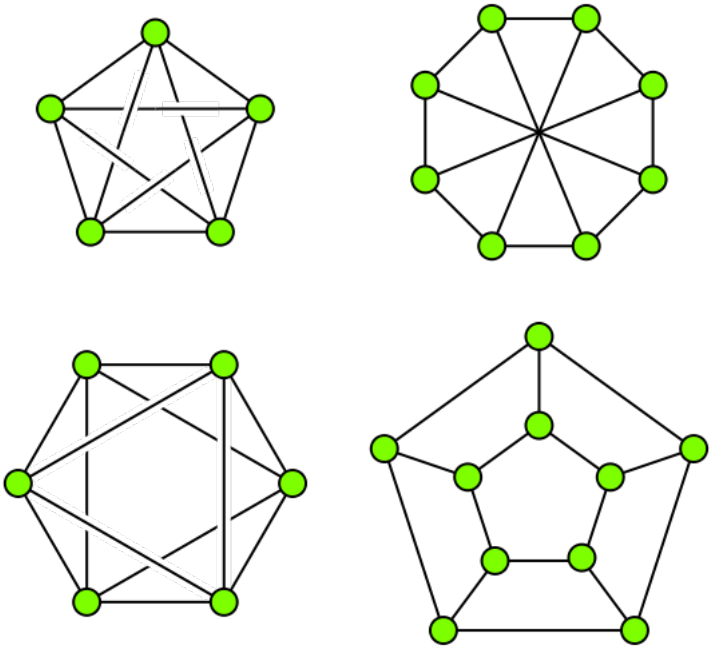
- For $k = 1$, the unique forbidden minor is a 3-vertex cycle graph.

- For $k = 2$, the unique forbidden minor is the 4-vertex complete graph K_4.

- For $k = 3$, there are four forbidden minors: K_5, the graph of the octahedron, the pentagonal prism graph, and the Wagner graph. Of these, the two polyhedral graphs are planar.

For larger values of k, the number of forbidden minors grows at least as quickly as the exponential of the square root of k. However, known upper bounds on the size and number of forbidden minors are much higher than this lower bound.

Figure 2.6: Forbidden minors for $k = 3$



At the beginning of the 1970s, it was observed that a large class of combinatorial optimization problems defined on graphs could be efficiently solved by non serial dynamic programming as long as the graph had a bounded dimension, a parameter shown to be equivalent to treewidth by Bodlaender (1998). Later, several authors independently observed at the end of the 1980s that many algorithmic problems that are NP-complete for arbitrary graphs may be solved efficiently by dynamic programming for graphs of bounded treewidth, using the tree-decompositions of these graphs.

As an example, the problem of coloring graph of treewidth k may be solved by using a dynamic programming algorithm on a tree decomposition of the graph. For each set X_i of the tree decomposition, and each partition of the vertices of Xi into color classes, the algorithm determines whether that coloring is valid and can be extended to all descendant nodes in the tree decomposition, by combining information of a similar type computed

and stored at those nodes. The resulting algorithm finds an optimal coloring of an n-vertex graph in time $O(k^{k+O(1)} * n)$, a time bound that makes this problem fixed-parameter tractable.

2.5.2 Cliquewidth

In graph theory, the clique-width of a graph G is a parameter that describes the structural complexity of the graph. it is closely related to treewidth, but unlike treewidth it can be bounded even for dense graphs.This is quite useful since it appears there are specific problems that are easy for graphs of bounded treewidth and for dense graphs. It is defined as the minimum number of labels needed to construct G by means of the following 4 operations :

- Creation of a new vertex v with color i (noted $i(v)$)
- Disjoint union of two colored graphs G and H (denoted $G \oplus H$)
- Joining by an edge every vertex labeled i to every vertex labeled j (denoted join(i, j)), where $i \neq j$
- (i $\rightarrow$ j): recolor all vertices i to color j

Definition 2.5.2. The smallest number of colors needed to construct G through the above operations (i.e. as a colored graph classically isomorphic to G) is called the cliquewidth of G, cw(G).

Graphs of bounded clique-width include the cographs and distance-hereditary graphs. Although it is NP-hard to compute the clique-width when it is unbounded, and unknown whether it can be computed in polynomial time when it is bounded, efficient approximation algorithms for the clique-width are known. Based on these algorithms and on Courcelle's theorem, many graph optimization problems that are NP-hard for arbitrary graphs can be solved or approximated quickly on the graphs of bounded clique-width.

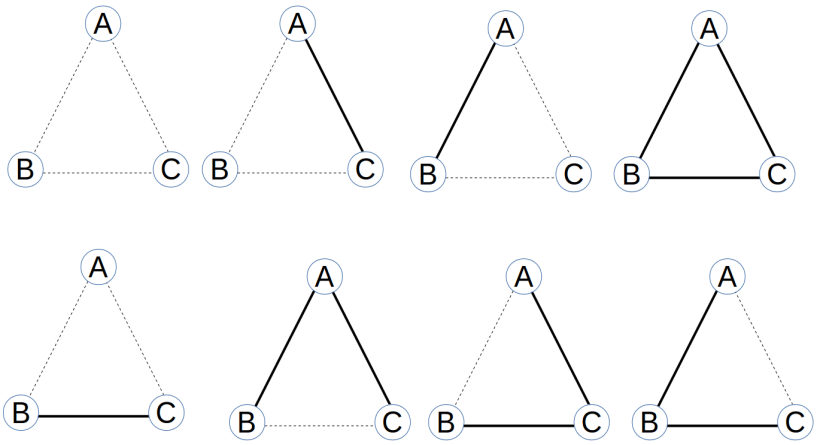
The construction sequences underlying the concept of clique-width were formulated by Courcelle, Engelfriet, and Rozenberg in 1990 [28] and by Wanke (1994). The name "clique-width" was used for a different concept by Chlebíková (1992). By 1993, the term already had its present meaning.

Easy results about cliquewidth that may be utilized later are:

1. Cliques have cliquewidth 2.
2. If G has bounded treewidth then it has bounded cliquewidth.
3. The complement graph of a graph of clique-width k has clique-width at most $2k$

In classical complexity cliquewidth is NP-complete to compute but

Figure 2.7: All possible graphs of cliquewidth 3 involving vetrices A,B,C



2.5.3 Branchwidth

In graph theory, a branch-decomposition of an undirected graph G is a hierarchical clustering of the edges of G, represented by an unrooted binary tree T with the edges of G as its leaves. Removing any edge from T partitions the edges of G into two subgraphs, and the width of the decomposition is the maximum number of shared vertices of any pair of subgraphs formed in this way. The branchwidth of G is the minimum width of any branch-decomposition of G.

Officially then:

Definition 2.5.3. Let G be a graph on n vertices.

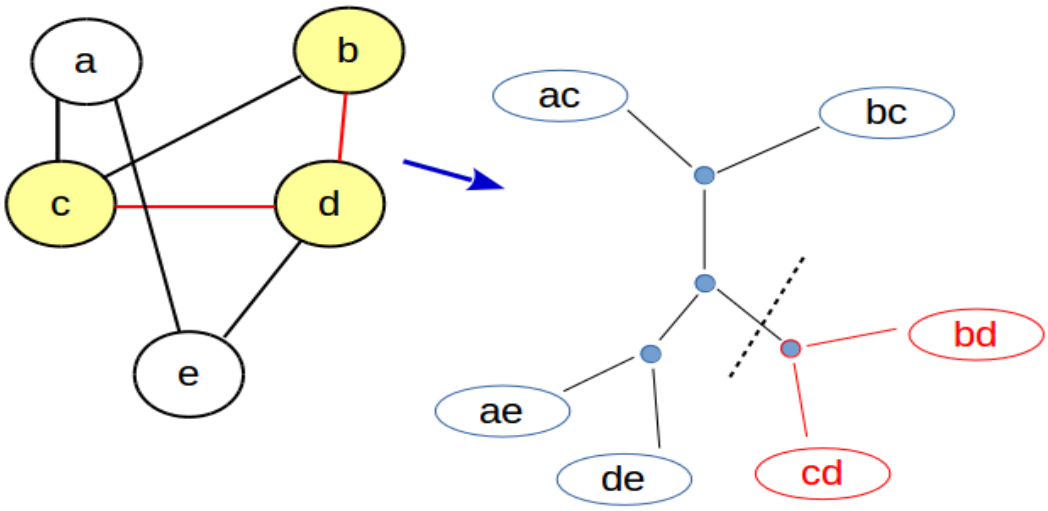
- A *branch decomposition* of G is a pair (T, τ), where T is a tree with vertices of degree 1 or 3 and τ is a bijection from the set of leaves of T to the edges of G. If e is any edge of the tree T, then removing e from T partitions it into two subtrees T_1 and T_2.

- This partition of T into subtrees induces a partition of the edges associated with the leaves of T into two subgraphs G_1 and G_2 of G. This partition of G into two subgraphs is called an *e-separation*.

- The *width* of an e-separation is the number of vertices of G that are incident both to an edge of E_1 and to an edge of E_2; that is, it is the number of vertices that are shared by the two subgraphs G_1 and G_2.

- The width of the branch-decomposition is the maximum width of any of its e-separations. The *branchwidth* of G is the minimum width of a branch-decomposition of G.

Figure 2.8: A branch decomposition showing an e-separation. The separation, the decomposition, and the graph all have width three



It is NP-complete to determine whether a graph G has a branch-decomposition of width at most k, when G and k are both considered as inputs to the problem.[11] However, the graphs with branchwidth at most k form a minor-closed family of graphs,[8] from which it follows that computing the branchwidth is **fixed-parameter tractable**: there is an algorithm for computing optimal branch-decompositions whose running time, on graphs of branchwidth k for any fixed constant k, is linear in the size of the input graph.[29]

For *planar* graphs, the branchwidth can be computed exactly in polynomial time. This in contrast to treewidth for which the complexity on planar graphs is a well known open problem. The original algorithm for planar branchwidth, by Paul Seymour and Robin Thomas, took time $O(n^2)$ on graphs with n vertices, and their algorithm for constructing a branch decomposition of this width took time $O(n^4)$.This was later sped up to $O(n^3)$

As with treewidth, branchwidth can be used as the basis of dynamic programming algorithms for many NP-hard optimization problems, using an amount of time that is exponential in the width of the input graph or matroid.For instance, Cook Seymour (2003) apply branchwidth-based dynamic programming to a problem of merging multiple partial solutions to the traveling salesman problem into a single global solution, by forming a sparse graph from the union of the partial solutions, using a spectral clustering heuristic to find a good branch-decomposition of this graph, and

applying dynamic programming to the decomposition.

Fomin & Thilikos (2006) argue that branchwidth works better than treewidth in the development of fixed-parameter-tractable algorithms on planar graphs, for multiple reasons: branchwidth may be more tightly bounded by a function of the parameter of interest than the bounds on treewidth, it can be computed exactly in polynomial time rather than merely approximated, and the algorithm for computing it has no large hidden constants.

Forbidden Minors By the Robertson–Seymour theorem, the graphs of branchwidth k can be characterized by a finite set of forbidden minors.

1. graphs of branchwidth 0 are the matchings. The minimal forbidden minors are a two-edge path graph and a triangle graph.
2. The graphs of branchwidth 1 are the graphs in which each connected component is a star. The minimal forbidden minors for branchwidth 1 are the triangle graph and the three-edge path graph.
3. The graphs of branchwidth 2 are the graphs in which each biconnected component is a series-parallel graph. The only minimal forbidden minor is the complete graph K4 on four vertices.
4. The graphs of branchwidth 2 are the graphs that don't have as minors the graph of the octahedron, the complete graph K5, the Wagner graph and the cube graph

2.5.4 Pathwidth

In the first of their famous series of papers on graph minors, Neil Robertson and Paul Seymour (1983) define a path-decomposition of a graph G to be a sequence of subsets X_i of vertices of G, with two properties:

- For each edge of G, there exists an i such that both endpoints of the edge belong to subset X_i
- For every three indices $i \leq j \leq k, X_i \cap X_k \; Xj$

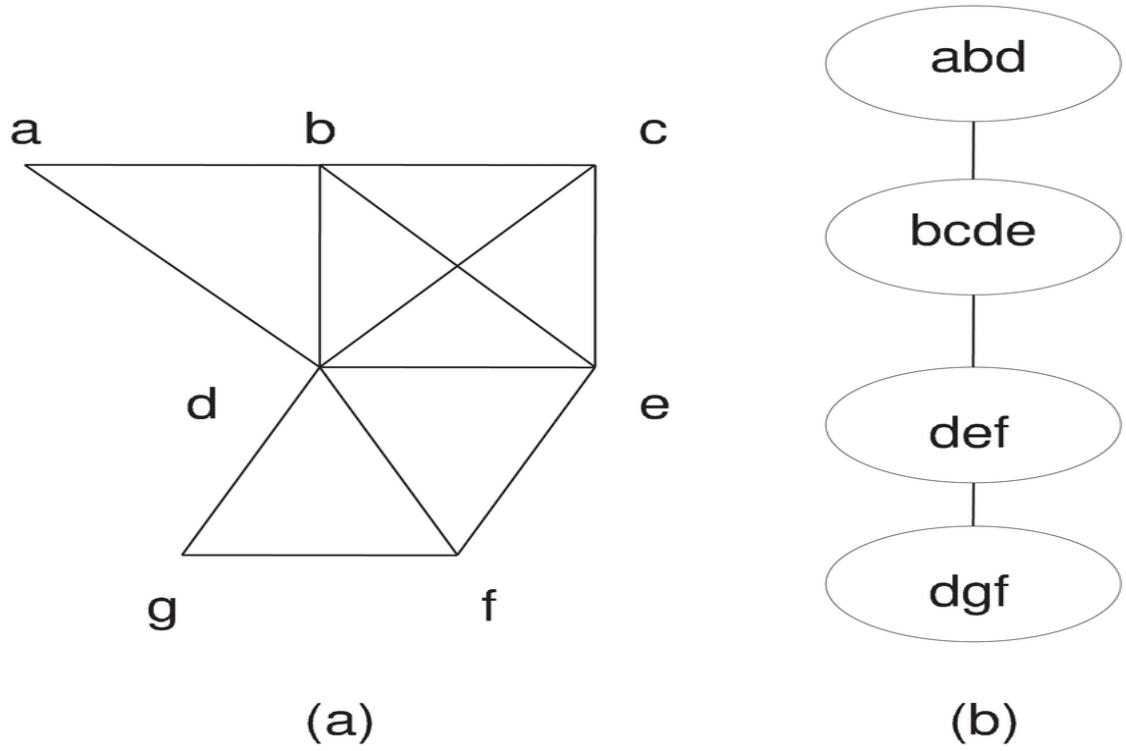
The second of these two properties is equivalent to requiring that the subsets containing any particular vertex form a contiguous subsequence of the whole sequence. In the language of the later papers in Robertson and Seymour's graph minor series, a path-decomposition is a tree decomposition (X, T) in which the underlying tree T of the decomposition is a path graph.

Definition 2.5.4. The width of a path-decomposition is defined in the same way as for tree-decompositions, as $max(i)|X_i|-1$, and the *pathwidth* of G is the minimum width of any path-decomposition of G.

The subtraction of one from the size of X_i in this definition makes little difference in most applications of pathwidth, but is used to make the pathwidth of a path graph be equal to one.

Although very similar to treewidth note here that **only** path graphs and not trees have a pathwidth of 1.

Figure 2.9: A path decomposition of Pathwidth 3



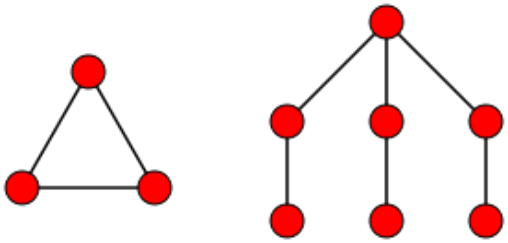
It is NP-hard to find the pathwidth of arbitrary graphs, or even to approximate it accurately.[30] However, the problem is fixed-parameter tractable: testing whether a graph has pathwidth k can be solved in an amount of time that depends linearly on the size of the graph. Additionally, for several special classes of graphs, such as trees, the pathwidth may be computed in polynomial time without dependence on k. Many problems in graph algorithms may be solved efficiently on graphs of bounded pathwidth, by using dynamic programming on a path-decomposition of the graph[2].

Forbidden Minors The property of having pathwidth at most p is, itself, closed under taking minors: if G has a path-decomposition with width at most p, then the same path-decomposition remains valid if any edge is

removed from G, and any vertex can be removed from G and from its path-decomposition without increasing the width. Contraction of an edge, also, can be accomplished without increasing the width of the decomposition, by merging the sub-paths representing the two endpoints of the contracted edge. Therefore, the graphs of pathwidth at most p can be characterized by a set X_p of excluded minors.

Although X_p necessarily includes at least one forest, it is not true that all graphs in X_p are forests: for instance in figure 2.10, X1 consists of two graphs, a seven-vertex tree and the triangle K3.

Figure 2.10: Forbidden Minors of Pathwidth 1



However, the set of trees in X_p may be precisely characterized: these trees are exactly the trees that can be formed from three trees in$X_p{-}1$ by connecting a new root vertex by an edge to an arbitrarily chosen vertex in each of the three smaller trees. For instance, the seven-vertex tree in X_1 is formed in this way from the two-vertex tree (a single edge) in X_0. Based on this construction, the number of forbidden minors in Xp can be shown to be at least $(p!)^2$.The complete setX_2 of forbidden minors for pathwidth-2 graphs has been computed and it contains 110 different graphs.

Chapter 3

Graph minors

3.1 Definitions

In this course, a graph is given by a set V, whose elements are called **vertices**, and a set E whose elements, called **edges** of the graphs, are distinct subsets of size 2 of V. According to the usual vocabulary, this means that our graph will always be simple and without loops. Unless specified, V will always be a finite set. For a graph G, V(G) will always denote its set of vertices, E(G) its set of edges. Very often we will write xy instead of x, y for an edge of G.

A vertex v is a neighbor of a vertex u if uv E(G). The neighborhood of u, denoted N(u) is the set of neighbors of u. Its degree, denoted d(u) is the cardinality of its neighborhood. The maximum degree of a graph is usually denoted . A graph with no edges will be called a stable set, or independent set, and a graph will all possible edges between its vertices a clique, or complete graph. The complete graph on n vertices is usually denoted K_n . The path P k is a graph with $V(P_k) = x_1, x_2, ..., x_k$, with edges E = $x_i, x_i + 1, 1 \leq i \leq$ k $-$ 1 .

The vertices x_1 and x_k are called the endpoints of the path. If we add the edge x_k x_1 to P k then the resulting graph is the **circle** on k vertices, denoted C_k .

3.2 Tree Algorithmic Problems

Consider the following problem of connectivity.

k-DISJOINT PATH

Input: A graph G, an integer k and two subsets of vertices A and B of size k

Output: TRUE if there exists k vertex disjoint paths from A to B

This problem is a very classical one, and Ford-Fulkerson Algorithm tells

us that this is solvable in time O((k|E(G)|) (classical Ford-Fulkerson Algorithm is for edge disjoint path in the directed case, but it is easy to reduce our case to this one). The maximum value k corresponds to a minimum vertex cut separating A and B and is a classical result of Menger.

Theorem 3.2.1. *([3]) Let x and y be distinct vertices of a graph G. Then the minimum number of vertices whose deletion separates x from y is equal to the maximum number of internally disjoint paths between x and y.*

Now consider the smilingly similar problem. Problem : k-disjoint rooted path problem Input : A graph G, an integer k, and two subsets of vertices X = x 1 , x 2 , . . . , x k and Y = y 1 , y 2 , . . . , y k Output : TRUE if there exists disjoint paths P 1 , P 2 , . . . , P k , such that P i is a path from x i to y i . This kind of problem in a more general form is know as commodity flow problem and has many applications. With k part of the input, this problem is NP-complete, even restricted to the class of planar graphs. Nevertheless, in the Graph Minor series of papers, Robertson and Seymour proved a polynomial algorithm for fixed k. This result is extremely difficult and relies and techniques and notions that will be illustrated in this course.

Theorem 3.2.2. *The k-disjoint path problem can be solved in time O (f (k).n 3*

(Robertson-Seymour, [19])

The result has been improved to quadratic time by Kawabayarashi, Kobashi and Reed ([]). Let us see an algorithmic consequence of this result related to topological minor detection.

Definition 3.2.1. A graph H is topological minor of a graph G if there exists a injective mapping f from V(H) to V(G) such that for each edge uv of H, there exists in G a path P uv connecting f (u) and f (v) in G with the property that all these path are internally disjoint.

Example. Describe the graphs that do not contain the following graphs as topological minors : K 3 , K 1,3 , K 1,4 . A natural algorithmic problem is then the following. Problem : Topological H-minor detection Input : A graph G and a graph H. Output : TRUE if H is a topological minor of G, FALSE otherwise. The problem is NP-complete if H is part of the input, but if H but if H is fixed, then this problem was proven to be polynomial by Robertson and Seymour.

Theorem 3.2.3. *Let H be a fixed graph. There exists a polynomial time algorithm to decide whether H is a topological minor of a given graph G.*

Proof. Let f : V(H) $\rightarrow$ V(G) be an injection (note there are polynomially many such objects), we want to decide if there exists disjoint paths in G

between the f (v) corresponding to edges of H. To do that, we replace each vertex f (v) by d H (v) copies of f (v) (having the same neighbours). Now, for k = |E(H)|, solving the k-Rooted Disjoint Path Problem for these sources clearly solves the desired question. □

The complexity of this algorithm is hence O(f (k)) n k , where k is the size of H, and n the size of G. It is therefore polynomial for every fixed k. In 2010, Grohe, Kawabarayashi, Marx, and Wollan proved a stronger result, that this can be done in O(f (k)) n 3 which is FPT . In particular, the previous theorem implies that any family of graphs that is defined with forbidding a FINITE family of graphs as topological minors is polynomially testable. One such family is very well known, it is the family of planar graphs, as was proven by Kuratowski in 1930.

Theorem 3.2.4. *(Kuratowski,1930) A graph G is planar if and only if it does not contain K 5 or K 3,3 as a topological minor.*

The crucial fact here is not that planar graphs are defined by a certain list of forbidden topological minors, this is easy (why?), it is the finiteness of this list that is non trivial. The central result of Robertson and Seymour theory is that many different graph properties can be characterized by a finite list of forbidden substructures, and hence get polynomial time recognition algorithm. Of course, one does not need the difficult of Robertson and Seymour to prove that planar graphs are polynomially recognisable, there even exists linear time algorithm to do that. Nevertheless, we will see later that there are instances of such recognition problem for which the only proof of polyniomiality was obtained through their results.

3.3 Minors

Minors are defined through three operations on a graph G (at the end of each line the notation for the resulting graph).

1. Remove a vertex v (and all its incident edges) : $G\ v$
2. Remove an edge e (but not its end vertices) :$G\ e$
3. Contract an edge $e = xy$, which means remove x and y, add a new vertex z whose neighborhood is the union of the neighborhoods of x and y (without putting any loop on z) : G/e.

A contraction G/e is topological if one of the endpoints of e has degree 2. Its inverse is the subdivision operation which consists in removing an edge xy, adding a new vertex z, and adding the edges xz and zy.

Definition 3.3.1. Let G and H be two graphs.

- H is an induced subgraph of G if H is obtained from G by the repeated use of rule 1.
- H is a subgraph of G if H is obtained from G by the repeated use of rule 1 and 2.
- H is a spanning subgraph of G if H is obtained from G by the repeated use of rule 2.
- H is a minor of G if H is obtained from G by the repeated use of rule 1,2 and 3.
- H is a topological minor of G is H is a minor of G and every contraction used was topological.

Recall that the largest integer k such that G has a complete graph (resp. independent set) on k vertices as an (induced) subgraph, is called the clique number (resp. independence number) of G, denoted c.n(G) (resp. i.n(G)). Due to a classical result of Karp ([15]), deciding if a graph has c.n(G) k or not (similarly for i.n(G)) is an NP-hard problem. The following definition gives an alternate form of minors that is often useful.

Definition 3.3.2. Let G and H be two graphs, and denote $V(H) = \{v_1, \ldots, v_p\}$. Then H is a minor of G if and only if there exists p connected and disjoint subgraphs $G_1, \ldots, G_p$ of G such that for every edge $(v_i v_j)$ of H, there exists an edge between G_i and G_j .

Let us mention here that high density implies the existence of a large minor. There exist theorems with better bounds, but we are only interested here in the fact that such a bounds exists.

Theorem 3.3.1. *Every graph with average degree at least 2^{r-2} contains K_r as a minor.,*

Proof. By induction on r. Let G be a graph of average degree at least 2^{r-2} . There- fore $|E(G)|/|V(G)| \geq 2^{r-3}$. Let H be minimal amongst all minors of G such that $|E(H)|/|V(H)| \geq 2^{r-3}$. It implies that when one contracts an edge in H, one must loose at least 2^{r-3} edges (otherwise the inequality would still be satisfied, and H would not be minor minimal). Hence, for any xy edge of H, x and y have at least 2^{r-3} common neighbors. In other words, if x is a vertex in H, then the minimum degree in its neighborhood is at least 2^{r-3} , so by induction it contains a K_{r-1} minor, which yields with x the desired K r minor. □

Let us discuss now the difference between minors and topological minors. Topological minors are special kind of minors but of course the converse is

not true : a graph G can contain H as a minor, but not as a topological minor. However When H is of small maximum degree, this is true.

Another very useful result is of course the following by Robertson and Seymour [7]

3.4 Wagner's Conjecture

Definition 3.4.1. A class of graphs C is said to be **minor closed** if for every graph $G \in C$ and every minor H of $G, H \in C$.

Definition 3.4.2. If C is a minor closed class of graphs, a graph G is a bound for C if G is not in C but every strict minor of G is.

Theorem 3.4.1. *Let C be a minor closed class, and X be its (possibly infinite) set of bounds. Then, $G \in C \Leftrightarrow G$ does not contain any graph of X as a minor*

We are now ready to state the celebrated conjecture formulated by Wagner in 1937:

Theorem 3.4.2. *A minor closed class of graph is defined by a finite list of forbidden minors.*

The above means that for a minor closed class you can easily check if a graph is in it by checking all of the graphs minors for one of the given finite list of forbidden ones. Now this is extremely important for most of the theorems that will be mentioned in chapter 7.

Let us mention here that high density implies the existence of a large minor; there exists theorems with better bounds, but we are only interested here in the fact that such a bounds exists.

Definition 3.4.3. The set of forbidden minors(F_C) for a class of graphs Cwill be called ***obstructions***. A graph belongs to C if and only if it does not contain (as a minor) any graph in F_C

Let us discuss here what can these obstructions be. For a given minor closed class C, a graph H is said to be a bound if G is not in C but every strict minor of G is. Note that if H is a bound, since it is not in C it must contain any obstruction so it must be itself an obstruction. The following easy theorem tells us that the set of bounds is in fact a sufficient set of obstructions.

Theorem 3.4.3. *Let C be a minor closed class, and B be its (possibly infinite) set of bounds. Then $G \in C$ if and only if G does not contain any graph of X as a minor.*

Proof. If H not in the class then either it is minimal or it contains H_0 not in the class.We can repeat the argument, and since there exists no infinitely decreasing sequence of graphs, every graph not in the class admits one of the bound as a minor. □

(This uses the fact that there exists no infinite decreasing sequence of graphs for the minor order - such partial orders are called well founded.)

As said before, it implies that testing if a certain graph G belongs to C is exactly testing if G contains one of the minor-minimal graphs with respect to C.

Therefore, testing if a certain graph G belongs to C is exactly testing if G contains one of the minor-minimal graphs with respect to C.

Here is a table describing the set of minor-minimal graphs for certain classes.

Graph Class	Minor Minimal Graphs
Forests	Triangle
Union of paths	Triangle , Claw
Planar	K_5 & $K_{3,3}$
Toric	$\geq$ 16629 (but finite)

So another way of stating Wagner conjecture would be to say : Every minor closed class of graphs has a finite set of bounds. Note that by definition one bound cannot be the minor of another. Using the terminology of partially ordered sets, they form an anti-chain : a set of pairwise not comparable elements. So a way to prove Wagner conjecture would be to prove that there exists no infinite anti-chain for the minor relation on graphs. In fact this equivalent as we will show now.

Definition 3.4.4. A partial order 4 defined on a set X is a **well quasi order** (WQO) if there is no infinite decreasing sequence and no any infinite anti-chain.

A infinite sequence that is either decreasing or an anti-chain will always be called a bad sequence. A wqo is hence defined as a partial order with no bad sequences. Note that in the case of graphs, there cannot be an infinite decreasing sequence, so the only possible bad sequence would be an infinite anti-chain.

Theorem 3.4.4. *Wagner's conjecture is equivalent to say that the class of all graphs with the minor relation is a wqo.*

Proof. Assume that the minor relation is a wqo. consider a class C that is minor closed. Let F_C be the class of graphs minimally (for the minor relation) not in C. Then F_C is an anti-chain, so it is finite, and it is easy to see that G is in C if and only if G does not contain any graph in F_C as a

minor. Now assume that Wagner's conjecture is true. Assume there exists a bad sequence of graphs G_n . Let C be the class that do not contain any of the G n as a minor. It is minor closed, hence there exists a finite list (H_i) such that G is in C iff it does not contain any of the H i as a minor. So every G_i must contain one of these graphs as a minor. By pigeonhole principle, there exists G_i and G_j that contain the same H_k . But conversely, H_k is not in C so by definition it must contain one of the G_n as a minor. by transitivity, this contradicts the fact that the G_n form an anti-chain.

□

Chapter 4

Logics over Graphs

The most important classical time and space complexity classes, such as PTIME, NP, or PSPACE, have clean definitions in terms of resource-bounded Turing machines. It is well-known (though still surprising) that most natural decision problems are complete for one of these classes; the consequence is a clear and simple complexity theoretic classification of these problems. However, if more refined complexity issues such as approximability, limited nondeterminism, or parameterizations are taken into account, the landscape of complexity classes becomes much more unwieldy. This means that the natural problems tend to fall into a large number of apparently different classes. Furthermore, these classes usually do not have straightforward machine characterizations, but can only be identified through their complete problems.

Logic can serve as a tool to get a more systematic understanding of such classes. The basic results of descriptive complexity theory [39] show that all of the standard classical complexity classes have natural logical characterizations. For example, Fagin's Theorem characterizes NP as the class of all problems that can be defined in the fragment Σ^1_1 of second-order logic. One advantage that such logical characterizations have over machine characterizations is that they allow for more fine tuning. For instance, one may ask which problems can be defined by a Σ^1_1 -formula whose first-order part only contains universal quantifiers. While for decision problems in NP such restrictions do not lead to any remarkable new classes, there are interesting classes of NP- optimization problems obtained by restricting syntactic definitions this way. The best-known of these classes is Papadimitriou and Yannakakis' MAXSNP.

This approach seems to open the door to an endless variety of syntactically defined complexity classes, but fortunately it turns out that a fairly limited number of syntactic forms suffices to define those classes that have natural complete problems. Remarkably, these syntactic forms tend to be similar even in different application domains.

In this chapter, we provide the necessary prerequisites from mathematical logic and review some basic facts about the complexity of propositional, first-order, and second-order logic.

4.1 Propositional Logic

Formulas of propositional logic are built up from a countable infinite set of propositional variables by taking conjunctions, disjunctions, and negations. The negation of a formula α is denoted by $\neg\alpha$. Besides the normal binary conjunctions and disjunctions, it will be useful to explicitly include conjunctions and disjunctions over arbitrary finite sequences of formulas in our language (instead of just treating them as abbreviations). The normal binary conjunctions of two formulas α, β are called *small conjunctions* and are denoted by $(\alpha \wedge \beta)$. Similarly, binary disjunctions are called *small disjunctions* and are denoted by $\vee$.

Conjunctions over finite sequences $(_i)i \in I$ of formulas are called *big conjunctions* and are denoted by $\bigwedge_{i\ I\ i}$. Here I may be an arbitrary finite nonempty index set. Disjunctions over finite sequences of formulas are called *big disjunction*s and are denoted by $\bigvee$. A formula is small if it neither contains big conjunctions nor big disjunctions. Of course, every formula is equivalent to a small formula, but the precise syntactic form of formulas is important. For example, the formulas

$$\bigwedge_{i\ [5]} {}_i \text{ and } {}_1 \wedge {}_2 \wedge {}_3 \wedge {}_4 \wedge {}_5$$

Propositional variables are usually denoted by the uppercase letters X, Y, Z, and propositional formulas by the Greek letters $\alpha, \beta, \gamma, \delta, \lambda$($\lambda$is specifically used for literals). A literal is a variable or a negated variable.

The class of all propositional formulas is denoted by PROP. For $t \geq 0, d \geq 1$, we inductively define the following classes $\Gamma_{t,d}$ and $\Delta_{t,d}$ of formulas:

$$\Gamma_{0,d} = \{\lambda_1 \wedge \ldots \wedge \lambda_c | c \in [d], \lambda_1, \ldots, \lambda_c : \textit{literals}\} \Delta_{0,d} = \{\lambda_1 \vee \ldots \vee \lambda_c | c \in [d], \lambda_1, \ldots, \lambda_c : \textit{literals}\}$$

And:

$$\Gamma_{t+1,d} = \{\bigwedge \delta_i | \ I \textit{ finite nonempty index set, and } \delta_i \in \Delta_{t,d} \forall i \in I\} \ \Delta_{t+1,d} = \{\bigvee \gamma_i | \ I \textit{ finite non}$$

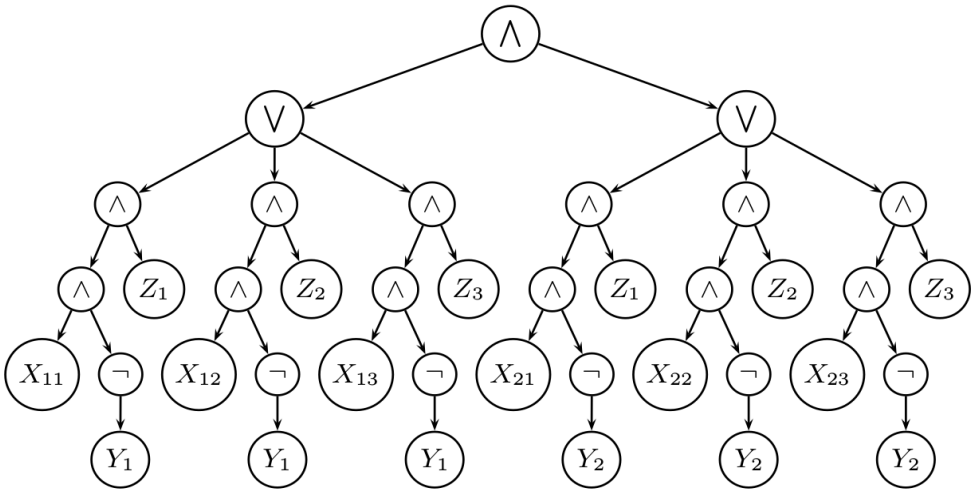
$_{2,1}$ is the class of all formulas in *conjunctive normal form*, which we usually denote by CNF. For $d \geq 1$, $_1$,d is the class of all formulas in d-conjunctive normal form, which we often denote by d-CNF. Similarly, $_2, 1$ is the class of all formulas in *disjunctive normal form* (DNF), and $_{1,d}$ the class of all formulas in d-disjunctive normal form (d-DNF). If $\alpha = \bigwedge_{i \in I} \bigvee_{j \in J_i} \lambda_{ij}$ is a formula in CNF, then the disjunctions $\bigvee_{j\ J_i\ ij}$ are the *clauses*

A formula α is in negation normal form if negation symbols occur only in front of variables. A formula α is positive if it contains no negation symbols, and it is negative if it is in negation normal form and there is a negation symbol in front of every variable. Each formula of propositional logic has a **parse tree**, which may formally be defined as a "derivation tree" in the grammar underlying the formula formation rules. For example, the parse tree of the $\Gamma_{2,3}$ -formula

$$\bigwedge_{i\in 2[]} \bigvee_{j\in[3]} ((X_{ij} \wedge Y_i) \wedge Z_j)$$

is displayed in Fig. 4.1

Figure 4.1: Parse tree of the example formula



4.1.1 Satisfiability Problems

For each class A of propositional formulas or (Boolean) circuits, we let Sat(A) denote the satisfiability problem for formulas or circuits in A. For example, Sat(3-CNF), that is, Sat$(1, 3)$, is the familiar 3-satisfiability problem, and Sat(CIRC) is the satisfiability problem for circuits, which we denoted by Circuit-Sat in the previous chapter. p-Sat(A) is the parameterization by the number of variables of the input formula, a problem which we considered in the introductory chapter for the class of all propositional formulas . It is well known that Sat(A) is NP-hard for every class $A \supseteq \Gamma 1, 3$ and that Sat$(\Delta_{2,2})$ is in polynomial time. p-Sat(A) is fixed-parameter tractable for every class A of formulas whose membership problem is fixed-parameter tractable. So for now, the parameterized problems p-Sat(A) are not particularly interesting.

However, for now the following version of the satisfiability problem is much more important; it is the decision problem associated with the optimization problem that tries to maximize (or minimize) the number of variables set to true in a satisfying assignment. The weight of an assignment V is the number of variables set to true. A formula α is k-satisfiable, for some nonnegative integer k if there is a satisfying assignment V : var(α) $\rightarrow$ { true, false} for α of weight k. We often identify an assignment V : var(α) $\rightarrow \{true,\ false\}$ with the set $\{X|V(X) = \ true\}$. For any class A of propositional formulas, the weighted 3 satisfiability problem for A is defined as follows:

p-WSAT(A)
Input: $\alpha \in$ A and k $\in$ N.
Output: Decision whether α is k-satisfiable

We consider the parameterized weighted satisfiability problem for A:

p-WSAT(A)
Input: $\alpha \in$ A and k $\in$ N.
Parameter: an integer k
Output: Decision whether α is k-satisfiable

It is known that WSat(2-CNF) and WSat(CIRC) and hence WSat(A) for all polynomial time decidable classes A of formulas or circuits containing 2-CNF are NP-complete; thus all these problems are equivalent under polynomial time reductions. This equivalence does not seem to carry over to the parameterized problems and fpt-reductions.

Theorem 4.1.1. *For $d \geq 1$, the problem $p - WSat(\Gamma_{1,d})$ for formulas in A that are positive, that is, that contain no negation symbols, is fixed-parameter tractable.*

From which with trivial work found in [41] we can derive.

Theorem 4.1.2. *For every $d \geq 1$, the problem $p - WSat(\Delta^{+}_{2,d})$ is fixed-parameter tractable.*

4.2 First-Order Logic

4.2.1 Relational Structures

A (relational) vocabulary τ is a set of relation symbols. Each relation symbol R has an arity (noted arity(R)) ≥ 1. A structure A of vocabulary τ , or τ -structure (or simply structure), consists of a set A called the universe and

an interpretation $R^A \subseteq A^{arity(R)}$ of each relation symbol $R \in \tau$. We synonymously write $\bar{a}\ R^A$ or $R^A bara$ to denote that the tuple $bara \in A^{arity(R)}$ belongs to the relation R^A .

> We only consider nonempty finite vocabularies and finite structures, that is, structures with a finite universe. The arity of τ is the maximum of the arities of the symbols in τ .

Example 4.2.1. (Graphs). Let τ_{Graph} be the vocabulary that consists of the binary relation symbol E. A directed graph may be represented by a τ_{Graph} structure $G = (G, E^G)$. An undirected graph, or just graph, may be represented by a

$$\tau_{Graph}$$

-structure G in which the edge relation E^G) is symmetric. We always assume graphs and directed graphs to be loop-free, that is, we assume the edge relation to be irreflexive.

Unless we want to emphasize in some situations that we view a graph G as an $\{E\}$-structure, we continue to denote the vertex set of a graph G by V and the edge set by E (instead of G and E(G)). We usually denote undirected edges in set notation (as in $\{v, w\}$).

Example 4.2.2. (Circuits). Let τ_{Circ} be the vocabulary consisting of the binary relation symbol E and unary relation symbols OUT , AND, OR, NEG, IN , TRUE , FALSE . A (Boolean) circuit may be represented by a τ_{Circ}-structure C, where:

- $(C, E^C))$ is a directed acyclic graph.
- OUT^C contains exactly one node, and this node has out-degree 0 (the output node)

The sets AND^C , OR^C , NEG^C form a partition of the set of all nodes of in-degree at least 1 (the and-nodes, or-nodes, and negation nodes, respectively). Nodes in NEG^C have in-degree 1. The sets IN^C , $TRUE^C$,$FALSE^C$ form a partition of the set of all nodes of in-degree 0 (the input nodes and the nodes computing the Boolean constants true and false, respectively).

4.2.2 First Order Syntax and Semantics

We fix a countably *infinite set of (individual) variables*. Henceforth, we use the letters $x, y, \ldots$ with or without indices for variables. Let τ be a vocabulary. **Atomic formulas** of vocabulary τ are of the form $x = y$ or $Rx1...xr$, where R is r-ary and $x1, ..., xr, x, y$ are variables.

First-order **formulas** of vocabulary τ are built from the atomic formulas using:

- the Boolean connectives $\neg, \wedge, \vee$
- existential and universal quantifiers $\exists, \forall$

.

The connectives $\rightarrow$ and $\leftrightarrow$ are not part of the language, but we use them as abbreviations:

$$\phi \rightarrow \phi \;\; for \;\; \neg\phi \vee \psi \;\; and \;\; \phi \leftrightarrow \psi \;\; for \;\; (\phi \rightarrow \phi) \wedge (\phi \rightarrow \phi)$$

.

By free(φ) we denote the set of *free variables* of φ, that is, the set of variables x with an occurrence in φ that is not in the scope of a quantifier binding x. A **sentence** is a formula without free variables. We write $\varphi(x_1, \ldots, x_k)$ to indicate that φ is a first-order formula with free(φ) $\subseteq \{x_1, \ldots, x_k\}$. We also use the notation $\varphi(x_1, \ldots, x_k)$ to conveniently indicate substitutions. For example, if φ(x) is a formula, then φ(y) denotes the formula obtained from φ(x) by replacing all free occurrences of x by y, renaming bound variables if necessary.

To define the semantics, for each first-order formula $\varphi(x_1, \ldots, x_k)$ of vocabulary τ and each τ-structure A we define a relation $\varphi(A) \subseteq A$ k inductively as follows:

- if $\varphi(x_1, \ldots, x_k) = Rx_{i1} \ldots x_{ir}$ with $i1, \ldots ir \in [k]$ then

  $$\varphi(A) := \{(a_1, ..., a_k) \in A^k | (a_{i1}, \ldots, a_{ir}) \in R^A\}$$

  Equalities are treated similarly

- If

  $\varphi(x_1, \ldots, x_k) = \psi(x_{i_1}, \ldots, x_{il}) \wedge \chi(x_{j1}, \ldots, x_{jm}) \;\; with \;\; i1, \ldots, il, \;\; j1, \ldots, jm \in [k]$, then

  $$\varphi(A) := (a_1, ..., a_k) \in A^k | (a_{i1}, \ldots, a_{il}) \in \psi(A), \;\; and \;\; (a_{j1}, \ldots, a_{jm}) \in (A).$$

  The other connectives are treated similarly.

- If

  $\varphi(x_1, \ldots, x_k) = \exists x_{k+1} \psi(x_{i_1}, \ldots, x_{il}) \;\; with \;\; i1, \ldots, il \in [k+1]$, then

  $$\varphi(x_1, \ldots, x_k) \in A^k \;\; there\, exists\, an \;\; a_{k+1} \in A \;\; such\, that \;\; (a_{i1}, \ldots, a_{il}) \in \psi(A)$$

  Universal quantifiers are treated similarly.

The definition also applies for k = 0; in this case,$\varphi(A)$ is either the empty set or the set consisting of the empty tuple. If $\varphi(x_1, \ldots, x_k)$ is a formula and A a structure of a vocabulary τ that does not contain all relation symbols occurring in $\varphi(x_1, \ldots, x_k)$, then we let $\varphi(A) := \varnothing$ We usually write $A \vDash \varphi(x_1, \ldots, x_k)$ instead of $(a_1, ..., a_k) \in \varphi(A)$. If φ is a sentence, we simply write $A \vDash \varphi$ instead of $\varphi(A) \neq \varnothing$ and say that A satisfies φ or A is a model of φ . Note that for a sentence φ the condition φ (A)$\neq \varnothing$ just means that φ(A) contains the empty tuple.

Example 4.2.3. Recall that $\tau_{Graph} = \{E\}$ and that we represent directed graphs and graphs as τ_{Graph} -structures $G = (G, E^G)$

Let$k \geq 1$ and consider the following formula:

$$vc'(x_1, \ldots, x_k) := \forall y \forall z(Exy \rightarrow \bigvee_{i \in [k]} (x_i = y \vee x_i = z)).$$

Then for every graph G and every tuple $(a_1, \ldots, a_k) \in G^k \Leftrightarrow \{a_1, \ldots a_k\}$, is a vertex cover of G. A bit sloppily, we will say that the above formula defines the set of all vertex covers of at most k elements of a graph." Let

$$vc_k := \exists x_1 \ldots \exists x_k(\bigwedge_{1 \leq i \leq j \leq k} x_i \neq x_j \wedge vc'_k(x_1, \ldots, x_k))$$

Then a graph G satisfies the sentence vc k if and only if G has a vertex cover of k elements.

4.3 Monadic Second Order Logic of Graphs (MSO)

One of the most important milestones of the arrea of algorithmic meta-theorems that will be revised later was Courcelle's theorem appearing in [19, 18] for the first time. It will later be a very important part of the paradigms of this thesis.

The syntax of the second-order monadic logic of graphs includes the logical connectives $\wedge, \vee, \neg$, variables for vertices, edges, sets of vertices, and sets of edges, the quantifiers $\forall, \exists$ that can be applied to these variables, and the five binary relations:

1. $u \in U$, where u is a vertex variable and U is a vertex set variable.
2. $d \in D$, where d is an edge variable and D is an edge-set variable.
3. $inc(d, u)$, where d is an edge variable, u is a vertex variable, and the interpretation is that the edge d is incident on the vertex u.
4. $adj(u, v)$, where u and v are vertex variables and the interpretation is that u and v are adjacent vertices.

5. Equality for vertices, edges, sets of vertices and sets of edges.

We will use lowercase letters for vertices or edge variables and uppercase letters for variables representing sets of edges or sets of vertices.

Example 4.3.1. (Hamiltonicity) A graph G is Hamiltonian if and only if it has a spanning cycle. the edges of G can be partitioned into two sets red and blue such that:

- Each vertex has exactly two incident red edges, and
- The subgraph induced by the red edges is a connected spanning subgraph of G.

In developing the MS_2 formula that expresses the property of Hamiltonicity, we will represent the set of red edges by the variable R and the set of blue edges by the variable B. The exact formula is given in example 8.1.1

Actually, this language is sometimes referred to in the literature as the ***extended monadic second-order*** language, and the logic the extended monadic second-order logic or MS_2 logic (e.g. Arnborg [2], Seese [24], and Arnborg, Lagergren, and Seese [4]). This is because we are looking at a two-sorted structure with predicates for edges and vertices plus an incidence relation. Another natural language has only vertex symbols and one must use binary relations for edges. Following Courcelle,we call this one-sorted logic the MS_1 logic. Naturally, the monadic second-order theories are different. MS_2 also allows for multiple edges whereas MS does not. Clearly there are properties definable in MSO_2 that are not expressible in MSO_1

Further on in chapter 7 we will see another version of MSO the Matroid MSO. The analytic definitions of that are not given here since they are a bit out of the scope of the current section. One can find more details in appendix .2.

Chapter 5

The trade-off function

In this chapter the trade-off relation is defined and studied with respect to the fields that have been explained already.

5.1 Elementary Description

This far we have parameterized the model checking problem with respect to the length of the property to be checked and the k as a parameter regarding the structural properties of the input instance.

Without further ado here follows a rough shape of the desired result.

Theorem 5.1.1. *Assume a logic class A of expressive power x that parameterized by k_0 is FPT. For every increment on the expressive power of A there exist a structural parameter k, that classifies the Model Checking Problem of all properties expressible in A as FPT on instances of bounded k .*

The above theorem has a very trivial proof similar to theorem 2.2.1 and as always by accepting that parameterizing by a number that can bind the size or the input is on the table.

Proof. There are two ways to derive a proof:

- Without the use of parameter k_0 we can immediately define k = g(n) where n is the size of the input model for the Model-Checking of A'. k classifies A' as FPT.
- The Model Checking Problem of A will be of some complexity $O^*(F(k_0))$ where the notation O^* is ignoring any polynomial terms of measure n. The increment of the expressive power is an expansion that can be applied to all the properties expressible in A. So there is an FPT algorithm that requires $O^*(F(k_0))$ and can me called an h(k) number of times for the model checking of A' as long as h(k) remains irrelevant of n. We can pick for h(k) a function for which

$$h(k_n) \geq n \; is \; true \; \forall n$$

By doing so we ensure that the running time of the Model-Checking problem of A' will be bound by $O^*(h(k)^{(k_0)})$ which is still FPT.

□

In order to produce some better result we must study the above relation with respect both to the *generality* of the parameters and the relationships between them.

So the first question of interest would be : Can we extract information towards the relationship of the parameters by looking at the relationship between expressibility classes?

Definition 5.1.1 (***Stronger***)**.** We will say that a parameter x is stronger than a parameter y if given a fixed k the percentage of instances that have x bounded by k is smaller than the percentage of instances that have y bounded by k.

In simple words x is stronger than y if it is more likely given a random input to end up with an instance of bounded y than a bounded x.

> From now on when mentioning a parameter of a Logic Class we assume that this parameter is the the weakest we can find for this class. Meaning that it is the one corresponding to the largest amount of instances without covering the hole universe.

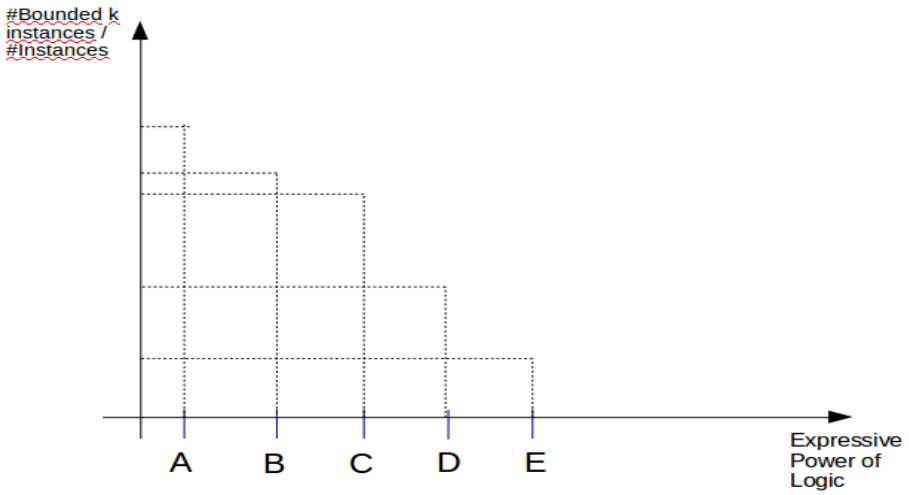
Theorem 5.1.2. *Given two logic classes A and B and if A has a higher expressive power than B then the parameter that classifies A as FPT is stronger than the parameter that classifies B as FPT.*

a trivial first look into the above theorems proof could be:

Proof. If A is of higher expressive power than B then all properties definable in B are also definable in A. Without loss of generality we can say that parameter x classifies A as FPT and y classifies B.
$\forall$ properties $p \in B$, the MCP for (p,x) $\in$ FPT since $p \in B \rightarrow p \in A \rightarrow$, and $\forall p \in AMCP(p, x) \in$ FPT. Therefore if p is expressible in B , we can simply express it using A and use the corresponding parameter for an FPT algorithm. So if A is of higher expressive power than B then all properties in B can be solved in FPT time using the parameter of A. Our definition of *stronger* requires to assume a smaller percentage of instances to have bounded the stronger parameter.

Suppose now that the parameter y of B is **stronger** than x of A. That means that for each property of B the amount of instances of the MCP(p,y) that we can solve efficiently with an FPT algorithm with parameter y is

Figure 5.1: Percentage of bounded k instances over strict Logic inclusions



smaller than the amount we would solve if we expressed p in A and used the FPT algorithm that solves the MCP(p,x) in A. $\perp$

□

Now of course although the above is a start it doesn't really give any information rather than just supporting the argument that there might be a larger scale correlation besides the trivial one. I have not presented yet any information towards how can someone either utilize existing knowledge on this field or at least use some process to produce results.

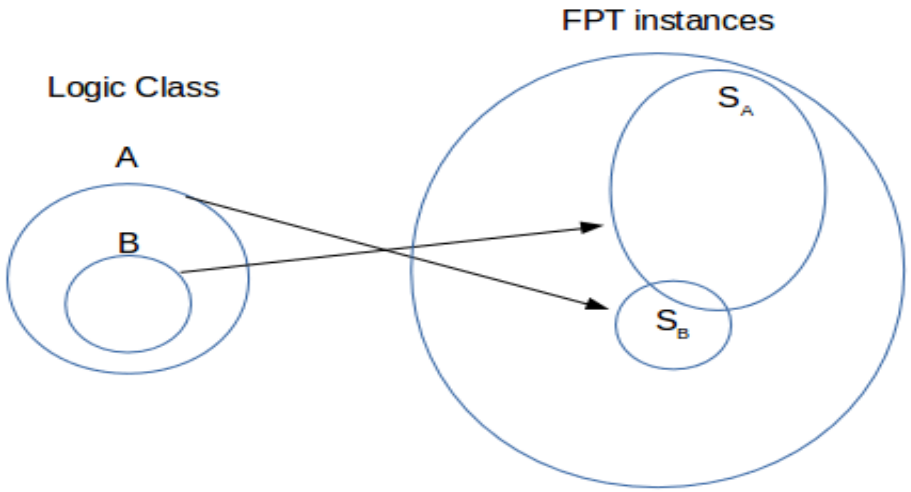
The above only means that while using strict bounds on language needed to express a property, when a property needs more expression power you need to sacrifice an amount on instances to maintain the MCP in FPT.

The new parameter though (of the more expressive complexity class) even if it requires a stronger parameter does not necessarily associate the parameter with the same instances. Of course the empty graph as a input will always be part of all graphs bounded by a random parameter k and therefore there is reason to believe that in most cases there is a part of the universe of instances that given two different parameters k_1, k_2 will have them both bounded. But in the general case we are just interested on the *cardinality* of the set that has each parameter bounded.

> For a logic class A and the parameter k that classifies A as FPT the set of instances that have the parameter bounded by k on an input of size n will be called S_A

One here might guess that depending on the parameter we can actu-

Figure 5.2: The relation between the parameters of logic classes A and B when A is of higher expressive power than B



aly determine or approximate the cardinality of S_A . This is not only very interesting but also quite useful. There are some results regarding FPT checkable logics that can be interpreted in this framework and start relativizing the results.For instance one could start arguing on the relationship of the parameters that classify as FPT $MSO_1 \& MSO_2$. Another thing one must keep in mind is that in order to properly relativize the parameters we must know that each one of them is tightly bound to the Logic class . An example would be

> If A is a logic class then for $p \in A$ and an instance I, if I has bounded k then p is checkable in FPT time. And all the set or instances that can be checked in FPT time for a property in A all have the parameter k bounded.

If no such result exists for a logic class then the above cannot be applied in a straightforward way. Thankfully there are cases where such results exist and in the absence of such we can think of workarounds. In some cases correlating structural parameters of graphs is a step towards understanding the nature of the properties that led to them. In other cases the relationship between the parameters is the only clue towards the relationship of the Logics although that is a topic not studied extensively here. The parameter classification is a very large project and i have started an attempt on it in chapter 6. After some basic results are presented, there is space for interpreting affected existing works.

Definition 5.1.2. From now on when referring to this function the following apply:

- The trade-off function will be noted as F
- in agreement with the above note for a logic class A the parameter that classifies A as FPT will be noted k_A
- the above parameter will be considered to be the weaker (= least **strong**) known one.
- The straightforward relation will be considered to be the one refering to a parameter that classifies a Logic class as FPT (i.e. the MCP of A is FPT parameterized by k_A)
- The converse relation will be considered to be the one associating a family of graphs and a checkable logic with a bounded parameter (i.e. if family F has a checkable logic A then it has bounded k_A

Some of the results that i will be quoting later on might not be the optimal known ones. However they might be the most appropriate to cite in this framework. This doesn't mean that later ones cannot be also interpreted and utilized. Of course to ensure optimality there must also exist some short of converse theorem. In the cases there isn't there are some extra steps that need to take place in order to return to a point where we can produce useful results.

PROBLEM: There might exist a case where a converse theorem is not known for a Logic class. For reference lets say that for class A all properties can be checked n FPT time with parameter k_A. Now we can discriminate two scenarios:

1. A converse does not exist
2. A converse exists but for a parameter k'_B

In both cases and for completeness purposes we can temporarily accept the converse parameter to be the one of the right higher Complete for our intent Logic class. Also there might be a case where only some short of converse is known. Each one of these cases needs a very small explanation as to how it is going to be treated in the future.

A small representation of all above cases is represented in the following table

In the above table logic A is considered less expressive than B , B than C and so on. Lets begin with the second row of the table.

What we know is that the MCP for Logic class B is FPT for k_B. What we need is at least some parameter for witch the Converse holds

Theorem 5.1.3. *For Logic B the Converse holds for parameter k_A*

Logic	Related parameter	Converse
A	k_A	k_A
B	k_B	-
C	k_C	k'_C
D	-	k_D
E	k_E	k_E

Table 5.1: Possible Discontinuities

Proof. B is more expressive than A $\rightarrow \forall p \in A, p \in B$. From the converse theorem of A: If a family of graphs F has a checkable A then it is of bounded k_A.Suppose F has a checkable B. $A \subseteq B \rightarrow$ F has a checkable A $\rightarrow$ F has a bounded k_A □

For the fourth row of table 5.1 we know that for a family of graphs F if D has a checkable Logic D then F is of bounded k_D

What we want is a parameter k that for witch the relation in question holds. I.e. The MCP of D is FPT parameterized by k.

Theorem 5.1.4. *The MCP of D is FPT parameterized by k_E*

Proof. The MCP of E is FPT parameterized by k_E . D is less expressive than E $\rightarrow \forall p \in D, p \in E$. Therefore the MCP of D is FPT parameterized by k_E □

Regarding the third row the situation gets more complicated in case that the exact relation between k_C & k'_C is not known. Here it is not enough to know witch parameter is **stronger** (is some cases we do not know even that) . We need to be able to decide for each instance I the bound on both parameters if such a value exists. Again this can only be done by analysis on specific parameters and therefore this is a part of chapter 6.

Definition 5.1.3. In the above context we can distinguish three scenarios.

1. $\forall I$ instances of bounded k_C, I is of bounded k'_C
2. $\forall I$ instances of bounded k'_C, I is of bounded k_C
3. The exact relation between k_C & k'_C is not known.

for the above we can define as $K_C = \mathbf{stronger}(k_C,\ k'_C,\ k_C + k'_C)$

the definition of **stronger** here is enough to cover all scenarios. If for instance $S_C \subseteq S'_C$ then $K_C = k'_C$. The rest of the cases can be tested similarly.

Through this trick we can summarize table 5.1 as:

Logic	Related parameter	Converse
A	k_A	k_A
B	k_B	k_A
C	K_C	K_C
D	k_E	k_D
E	k_E	k_E

Table 5.2: Corrected Discontinuities

We will see further on that the above table is not actually very far from the reality. Of course this table only summarizes relations between logics for which we know strict inclusion properties.

5.2 Refinements

This far i have only given proof on the existence of a somewhat inversely function between parameters and Logics. Lets assume though for the sake of argument that this function could be further determined.

What kind of input and output would it have?

Could it be some deterministic description?

What kind of mathematical properties would it have?

Each of the above is accompanied with understanding an aspect of the nature of the problem .

So lets start at the beginning. Suppose that a strict relation exists. How can we measure such a thing? Reminder here that we are trying to describe the way expression power in logics interacts with parameters of input instances.

Imagine an F function that as an ipput there is a variable describing in a way how expressive a Logic is and as an output we have a quantity that gives some short of information toward what kind of parameter would correspond to that language.

Its quite obvious early on that F would not be of the short

$$F(x) = x^2 + 4$$

or any other continuous function over the real numbers for example.

There are quite a lot of reasons why such a thing wouldn't work and the first one i am going to focus on is the fact that in any case the relation i am looking for will be in any way continuous. Even if we end up with a number as a input the values this number can be assigned will not be continuous since they have to correspond to expression power. Lets focus then to the kinds of logics that exist and what kind of increments are applied form one Expression Power to the other. The way we can prove if a property is expresible in a Logic has been covered in Chapter 4

1. propositional logic
2. FO logic
3. MSO_1
4. MSO_2
5. SO

In each one of these the difference is some operator , a part of the vocabulary, the use of Predicates or limitations on them.

One could of course try to associate some number with each one of these.

It is widely known and easily proven that all of the above have countable infinitely many properties defined in each one. This was proven in chapter 4 on theorem !!!!!!!

So any measure regarding the cardinality of the set of properties expressed in each one is irrelevant.

There are some studies on how can someone measure the expressive power of a language but none relevant enough. Some interesting measure would be to take into account th length of the expression of a specific properties in each Logic. But this becomes less useful when the optimal length is reached because even Logics of higher complexity require the same number of symbols.

For now then lets keep the idea of an increment in expressive power happening through its grammar or it vocabulary. Thankfully for the purpose of this thesis only strict inclusions are studied and therefore the idea of increment has a one-way interpretation. In order though for someone to

repeat such a process for logics of not strict inclusion a more detailed study will be needed on this area.

So as proved already the relation exists but it is yet to be determined how tight it is. We could also try to define it in the opposite direction. Results in this case would look something like this:

> If a family of graphs is checkable for all properties in Logic A
> then the family of graphs has a bounded parameter k.

Now for a finite family of graphs one can argue that all parameters are bounded. But thankfully i have here a proof that this does not interfeere with my work

Theorem 5.2.1. *On finite families of graphs a finite set of Logics is always FPT- checkable.*

The trick here is not to actually check all properties in a Logic rather than label the models that satisfy the property.

Proof. Select a random enumeration of the graphs in the family F. For a all these graphs we can select an upper bound for each possible parameter that would be of use in th MC of a Logic. Out parameters would be $k_L1, k_L2, ..., k_L$ for our N logics respectively.

So we define $k = max(k_L1) + ... + max(k_LN)$ where the individual parameters are parsing through all of our models.

k is an upper bound for all individual parameters in all of the graphs of F and therefore for all Logics in our set the MCP is FPT parameterized by k. □

so we can focus on infinite families of graphs where the bounding of a specific parameter would come from structural properties and by looking at the definition of the class.

Could we after producing a bounded parameter result derive on the set of properties that are checkable on those graphs?

Of course we could! Some results already exist but in a very local scale. In fact based on the work already presented one can derive such results after giving a closer look to the existing knowledge over parameters. We will study such paradigms later on.

The above results are not necessarily the most recent in this area but for the purpose of parameters of graphs they are more appropriate.

So now if someone finds a infinite family of graphs with bounded treewidth he can immediately know that there must exist a property expressible in SO Logic but not in MSO_2 that is not checkable in FPT time for this family.

The above actually is not trivially a necessity but it can be proven. As always when mentioning the parameter of a Logic class we refer to a relationship of the form shown in defiminition (CHAPTER 2.4 somewhere)

Theorem 5.2.2. *Assume logic classes A and B with B having a higher expressive power than A. If a family of graphs F has parameter k_A of Logic Class A bounded but k_B of Logic Class B unbounded then there exist a property p expressible in B but not in A that is not checkable in FPT time on F with parameter k_B.*

Proof. Suppose no such property exists i.e. all properties of B are FPT checkable for F with parameter k_B. Therefore Logic B is FPT-checkable for F. But as in the converse of Courcelle's theorem given by Seese which is or the form i am assuming then if a family of graphs has an FPT checkable A Logic then it must have bounded the corresponding parameter. $\perp$

□

As we are slowly getting somewhere let me summarize progress made this far.

1. While increasing expression power one must sacrifice an amount of instances.

2. Structural Characteristics of graphs can be used to determine what kind of properties are checkable form them.

3. With a little bit of further study we can correlate the times an algorithm runs fast with the logic required to define a property !!!!!!!

4. There might be a way to relativize the increment in expression power with the decreasing of number of instances.

Finally i will focus on one of the listed questions from further up. Lets just suppose that we have agreed on a measure that expresses how massive is the expressive power of logic A. From now this measure will be denoted as χ.

> What can we say about $\digamma(\chi)$? Technically χ is a mapping from all possible Logics to the real numbers with respect to the following properties:
>
> - $\forall$ Logic A $\exists$ $\chi(A) \in \Re$
> - iff A is more expressive than B then $\chi(A) \geq \chi(B)$

Remember that our output is a measure concerning the number of instances that correspond to a bounded value of k_A

Theorem 5.2.3. $\digamma(\chi)$ *is decreasing.*

Proof. We need to prove that

$$\forall x, y \ : x \geq y \rightarrow F(x) \leq F(y)$$

.

$x \geq y$ means that logic A_x is more expressive than $A_y \Rightarrow \forall p \in A_y, p \in A_x$
Suppose now that F is not decreasing $\Rightarrow$

$$\exists x, y \ x \geq y \ : F(x) \geq F(y)$$

That would mean that there are properties p expressible in A_y that can be checked in FPT time in less instances (S_y) when p is expressed in A_y than when expressed in A_x. That would mean that k_y is **stronger** than k_x which is a contradiction according to definition 5.1.2 □

5.3 Parameterized Framework

Although the work done this far is quite enlightening it lacks a certain amount of specific definitions and results. To continue further without worry for results being not definite enough there is some work to be done.

As you see above all reductions and proofs take place in a more mathematical framework. In order to be positively sure that not only the correlation F exists but all mentioned properties are true i have to translate them in the parameterized framework.

To begin...

The first result of section 5.1 is theorem 5.5.1. regarding the certainty of an existent parameter that classifies the MCP of a Logic as FPT.

According to the phrasing of 5.1.1 we have to describe the transformation of a weft 0 circuit for logic A to a weft 0 circuit for the logic A'.

Of course without knowing the increment it is very difficult to design a specific circuit the following can be done:

The increment will be of some form of extra symbols , Predicates or rules. Without any massive assumptions i will use a primitive form of enumeration of these changes.

Proof. Note C the family of circuits that recognizes property $p \in A$. The Logic A' will be the set of all properties expressible in A repeated an enumerable infinity number of times , each time with another aspect of the new rules being applied. Therefore each property p' ∈ A' will be a combination of a property p ∈ A and *expansion* from A' (even if the property in A is an empty string) .

Only thing remaining is to describe a parameterized reduction between the circuit recognizing p to a circuit recognizing p' where p is the part of p' in A.

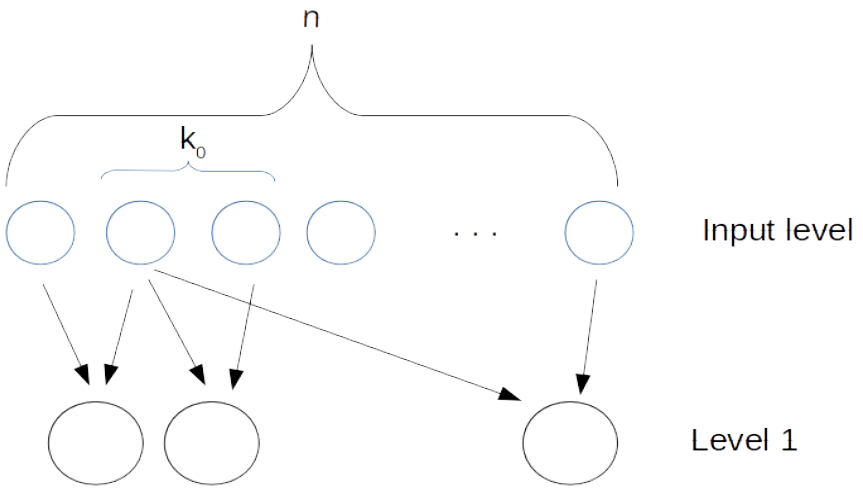


Figure 5.3: A circuit recognizing a property $p \in A$ for $k_0 = 2$

Each one of the changes in A will be represented on a gate of C'. Since C is of weft 0 each of the gates in C will have an in-degree at most $k_0 \Rightarrow$ in C each level contains at most a

$$g(k) * poly(n) \tag{5.1}$$

gates.

> * the above number is a result of each level being composed of k combinations of the previous one. This process only happens for finite polynomial over n steps and therefore each gate of a level cannot have more than 5.1 inputs from gates of the previous one.

A gate in C' will always be a rule being applied on a property p (which has its own weft 0 circuit) or on gates of C'.
We can recursively solve the case where it is applied on gates of C' by explaining what happens in gates of C. Similarly with theorem 2.2.1 we assign A' the parameter $k'_A = g(k) * poly(n)$.

Now no gate in C' is of unbounded in-degree $\Rightarrow C$ is of weft 0. □

The immediate next thing we need to interpret is theorem 5.1.2. But this is not something that could be deciepted through circuits due to the fact that circuits do not take into account instances that do no have the parameter bounded.

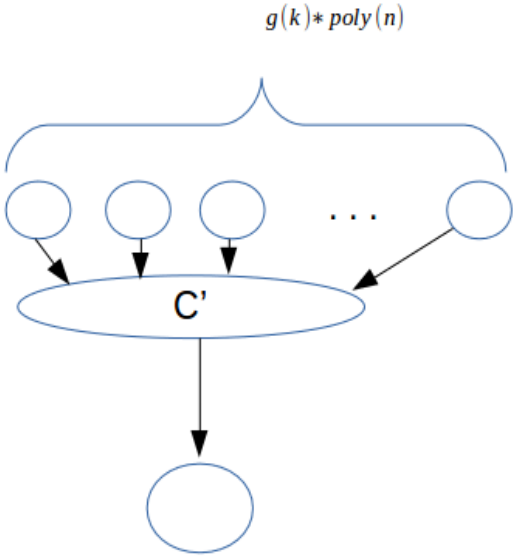


Figure 5.4: The gate of the circuit recognizing property $p' \in A'$

However the results of table 5.2 can all be interpreted through similar procedures.

Chapter 6

Parameter Analysis

In this chapter the graph metrics we defined in section 2.5 will be studied autonomously at first and later in contrast with each other. Here i will try to start the needed work in order for the previous chapter's results to be interpreted in specific structures.

6.1 Treewidth

As follows from the definition what is the most important thing to determine treewidth (noted form now on as tw) is the tree decomposition. This will be also the base of the following calculations.Since tree decompositions are heavily counting on *labels* our graphs will also be labeled, undirected ones. Our purpose is to determine at least approximately how massively bounding tw by k will affect the number of instances is size n. Si

For starters we will use the following which are considered trivial.

- The complete graph of size n has $\binom{n}{2} = n * (n-1)/2$ edges
- In total there exist $2^{n*(n-1)/2}$ posible subsets of these edges and each one corresponds to a different labeled graph.

Definition 6.1.1. For handiness purposes i will use form now on the term T_k to describe the number of graphs of size n and treewidth equal to k .

Note here that even if T_k is deterministically described it will not mean that we have a way to give an upper bound on n depending only on k. This study is focusing on number of instances and not specific ones.

If a graph is of bounded tw k then there exists a tree decomposition where the maximum number of nodes in labels is $k + 1$.

in order to produce a first result we will need the following:

Definition 6.1.2. A **smooth** tree decomposition is one for which each bag has k+1 nodes and all the neighboring bags have k common nodes .

Theorem 6.1.1. *For all tree decomposition of tw k there exist a smooth one of tw k.*

Proof. Given a tree decomposition of size k a *smooth* one can be constructed by applieng the following in each edge(p.q) of the tree decomposition.

1. if $X_p \in X_q$ then contract pq and let the bag be X_q . Siimilarly if $X_q \in X_p$
2. if $X_p \nsubseteq X_q$ but $|X_q| < k+1$ take any node from X_p and add it to X_q
3. if $|X_p| = |X_q| + k + 1$ and $|X_p - X_q| > 1$ then interpolate new bags between them.

□

So! using the above we say that when having a graph of size we are now looking for the ones that have a smooth tree decomposition of tw k. This means that we need to determine how many different tree decompositions of tw k exist over a set of n nodes.

We can also use the following theorem by Bodlaender[**?**] :

Theorem 6.1.2. *If T is a* smooth *decomposition of G then T contains exactly $|V(G) - k|$ nodes.*

The proof of this will be gives through the following procedure.

Each bag here is a set of $k+1$ nodes since T is smooth. And each node can pick from a pool of n nodes

So as a first result we know that the graphs of tw k are at most

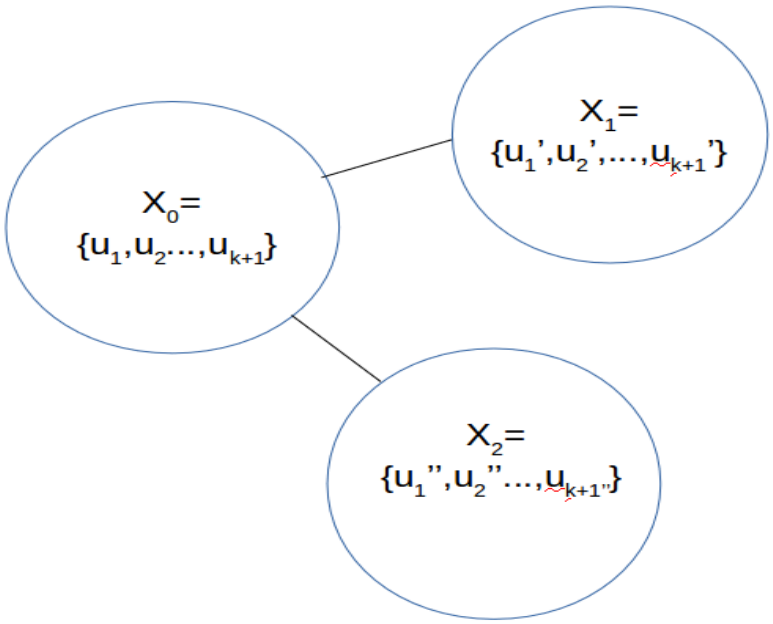
$$\binom{n}{k+1}^{|V(G)-k|} \tag{6.1}$$

Now of course when considering bounded by k tw we are not focusing on the tree decompositions of tw *exactly* k. But b using the above even a graph without edges will be mapped to a tree decomposition of tw k we are safe this far.

Now we need to make sure that on the number of (1) we are taking into account valid tree decompositions. That means we need to subtract from (1) a number of all the trees that are not actually tree decompositions of their labels.

We have $|V(G) - k|$ bags and we want all of them to have k common label nodes with their neighbors. So Lets construct one X_0 at random by inserting k nodes on the label. In equation (1) we where construction the other by inserting at random k_1 on the next one etcetera. Now for each one of its adjacent ones we have to choose $\binom{k+1}{k}$ that will be duplicated and can only replace one of the X_0 node with one of the $|V(G) - k - 1|$ remaining.

Figure 6.1: Constructing the new nodes



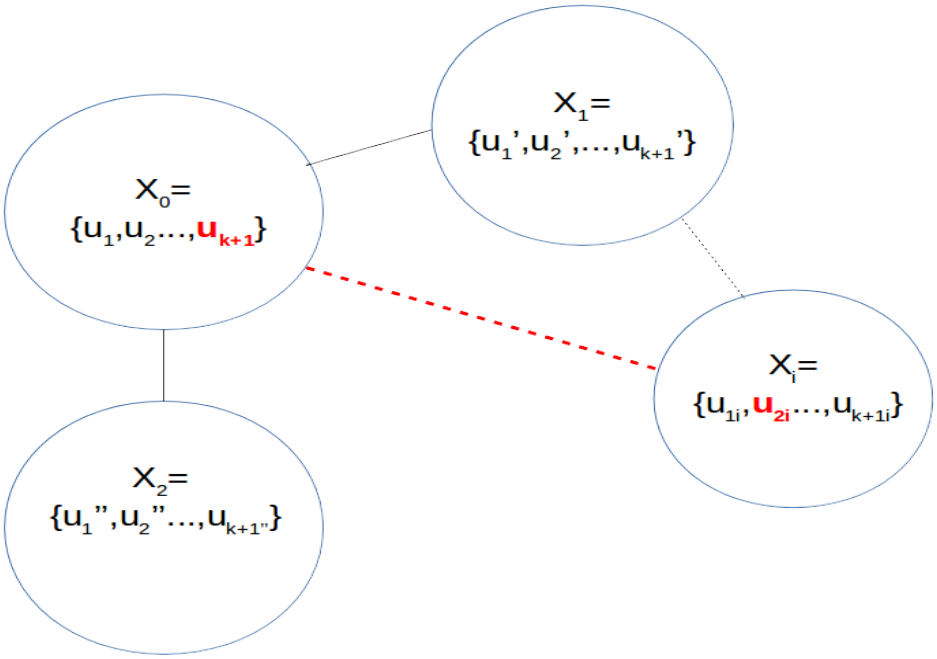
So far the construction of each one of the new nodes we are always picking $\binom{k+1}{k}$ from the existing neighbor and adding one more label that must not be already in use.

In the above figure for each new bag k labels must remain same. When choosing the k labels that would be reused we are safe since we are only expanding on the existing tree and therefore the new labels since they existed on a valid bag will not have to be the reason for any new edges besides the one with the parent node.

When choosing the new label that will be added to the bag we must be very careful. If the new label is in use somewhere in the tree adding it on the new node would cause a circle 6.2.

Suppose we are construction node i. If we are construction node i there are $i-1$ nodes already in the tree decomposition and we have to choose one of them as the parent of our current node This however will not contribute to the total number since we are not actualy interested in what order each bag node will be added to the tree decomposition. We can pick at random the k labels that will be reused from the previous node and we have to also pick the new label from the ones that have not been used. Note here that Since we are adding the $i-th$ node in the tree we have used so far $i-1$

Figure 6.2: Reusing labels would cause a circle



extra labels. So right now in our tree there are used

$$k+i, i=1,\ldots,|V(G)|-k-1$$

which means there are left $|V(G)|-k-i$ to choose from.

The product for all i would give us the number of all possible *smooth* tree decompositions of tw k $= T'_k$.

Now since all tree decompositions of tw k have a *smooth* one the number here will be an upper bound for T_k

$$T'_k = \binom{V(G)}{k+1} * \prod_{i=1}^{|V(G)|-k-1} i * \binom{k+1}{k} * (|V(G)|-k-i) =$$

$$\binom{V(G)}{k+1} * \binom{k+1}{k}^{|V(G)|-k-1} * \prod_{i=1}^{|V(G)|-k-1} i * (|V(G)| - k - i) =$$

$$[\binom{|V(G)|}{k+1} * (k+1)^{(|V(G)|-k-1)} * (|V(G)| - k - 1)^2! \tag{6.2}$$

As you will see the above seems to be very accurate since we can test it on easy cases. For instance:

On cliques were k=n-1 we have :

$$(6.2) = \binom{|V(G)|}{|V(G)|} * |V(G)|^0 * (0)! =$$

$$1 * 1 * 1 = 1$$

Which is exactly the number of graphs of size $|V(G)|$ with tw $|V(G)| - 1$ ($K_{|V(G)|}$) So right now we have the following:

$$T_k \leq T'_k = \binom{V(G)}{k+1} * (k+1)^{(|V(G)|-k-1)} * (|V(G)| - k - 1)^2!$$

The above then can be translated into an upper bound for graphs of ***bounded*** k by calculating the sum over all k.

$$\sum_{j=1}^{k} \binom{V(G)}{j+1} * (j+1)^{(|V(G)|-j-1)} * (|V(G)| - k - 1)^2! \tag{6.3}$$

Our first approximation about the number of size n graphs with bounded by k tw was :

$$\binom{n}{k+1}^{|V(G)-k|} \quad 6.1$$

Result 6.3 is obviously much better witch can be shown through trivial calculations.

To summarize then about treewidth we have that form all possible graphs of size n only 6.2 are of tw bounded by k hence the quantity:

$$\frac{\sum_{j=1}^{k} \binom{V(G)}{j+1} * (j+1)^{(|V(G)|-j-1)} * (|V(G)|-j-1)^2!}{2^{\binom{|V(G)|}{2}}} \tag{6.4}$$

As shown in the following diagram for some usual cases of real world input and in order to be able to relativize with other graph metrics later on.

6.2 Cliquewidth

In the search for cliquewidth one will notice from the definition that there in no such thing as a clique decomposition or something similar as to the one existing with treewidth. Cliquewidth originates from a constructive process focused on generating the original graph. This means that we cannot use the same teqnique or a similar one to find an upper bound to the quantity of instances of cliquewidth bounded by k .

The same trivial equations apply for the whole of graphs of $V(G) = n$ nodes.

As shown in section 2.5 cliques are of cliquewidth 2 as are very few other graphs.

Now the reader must keep in mind intuitively that the more we deviate from a clique of size n the larger the cliquewidth of the graph gets.

Now as per the definition there are four kinds of opperations one can use to produce a graph. We are comparing for a graph of $V(G)$ nodes and therefore we can derive the following:

- There will be exactly $V(G)$ in total node creations (easily since we end up with this much nodes in the graph)
- There will be exactly $V(G)-1$ disjoint union operations (a very short proof of this is given in the appendix)
- There will be at most $n-1$ recoloring operations (worst case = every recolor operation causes the change of color of exactly one node).
- There will be at most $(n-1)*(k-1)$ connection operations (This requires a short proof)

Now of the create operations the first two must necessarily happen first since none of the rest can can take place when less than 2 nodes exist in the graph.

What i am going to do is i am going to try to produce an upper bound for the total number of possible graphs of cliquewidth k by counting how many sequences of the above opperations can take place.

Now as i said we have cliquewidth k and therefore can utilize k colors in the creation of the graph . However i do not want to discriminate between lets say a pink and a blue clique of size n. Therefore since i am demanding $V(G) - 1$ recoloring operations in means that in the end i will be left with a graph of a unique color. In the flowing relations this will be translated to the mathematical property that when choosing a color for a new vertex we can pick from $k - 1$ colors. This is not of course the case. What actually happens is that each time we multiply and event by the number of possible colors we are then dividing by the number of isomorphic regarding color graphs. So:

Definition 6.2.1. From now on:

- The number of graphs of cliquewidth at most k will be noted as C_k
- The percentage of graphs of cliquewidth at most k will be noted as p_k

To begin giving some upper bounds :

There will be n vertex creations in the construction. When in a new number of vetrices we have to take into account the graphs concerning this many edges. There are

$$\binom{|V(G)|}{i}, i = 1 \dots n$$

ways to pick i vetrices from the final $|V(G)|$ that will exist in the graph.

For each one of them now we can perform a number of the other operations given in the definition. Since i am counting possible graphs per number of nodes involved disjoint union are not counted as will be resulting in a change of the number of vetrices studied. For the rest though we can say:

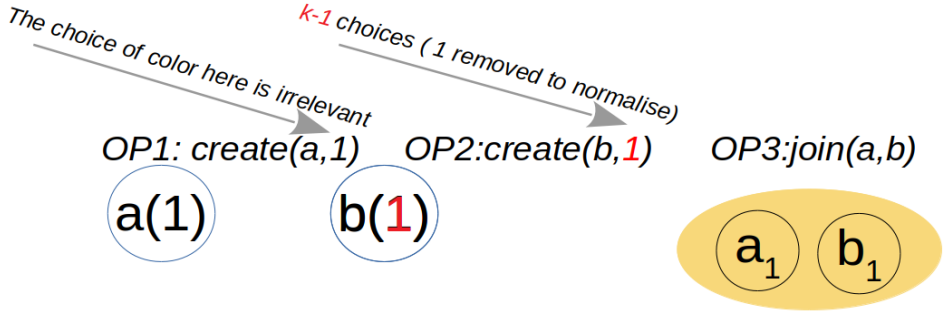
1. Each create operation correspond to an initial choice of color for the vertex created. the number of colors is k but as explained above this results in $k - 1$ possible choices
2. Each connect operation has at most $\binom{k}{2}$ possible inputs.
3. each recolor operation has at most $\binom{k}{2}$ possible inputs.

Now we are close to a first result.

Before a final solution is given i will explain a final point. The above operations have been left free to have as possible input any number of the colors and such. However the reader will probably notice that in many cases

there are not yet k colors in the graph yet. A closer look though will reveal that by not determining the sequence in which the disjoint union operations took place we can at least assume for the purposes of an upper bound that all above combinations are allowed and the way they will be arranged is given by their input.

Figure 6.3: First steps of the sequence describing graphs of cliquewidth k



For a number of nodes n_0 There are

$$c'_k = \binom{|V(G)|}{n_0}^{(k-1)} * (n_0 - 1)^{\binom{k}{2}} * [(n_0 - 1) * (k-1)]^{\binom{k}{2}}$$

$$= \binom{|V(G)|}{n_0}^{(k-1)} * (n_0 - 1)^{2*\binom{k}{2}} * \binom{k}{2}^{\binom{k}{2}} \tag{6.5}$$

The numbers $(n_0 - 1)^{2*\binom{k}{2}} * (k-1)^{\binom{k}{2}}$ correspond to the possible choices on can make when using the recolor or the connect operations . Summing for all n_0 up to k we will have a first result:

$$c_k \leq \sum_{i=0}^{n} \binom{|V(G)|}{i}^{(k-1)} * (|V(G| - 1)^{2*\binom{k}{2}} * (k-1)^{\binom{k}{2}} \tag{6.6}$$

The above result seems very reasonable since for $k = 2$ it count all cliques with size up to n plus some extra graphs of small cliquewidth per vertex number

$$k = 2 \Rightarrow c_2 \leq \sum_{i=0}^{|V(G)|} \binom{|V(G)|}{i} * (|V(G)| - 1)^2$$

Notice that in the above numbers while computing the sum for all i, i replaced i with n in term corresponding to choices over recoloring and

connection operations. This is because in the total number for c_k even smaller cliques and graphs are considered as part of a graph of n vetrices in total depending on the disjoint union operations. This means that a recolor or connect operation can indeed have as an input any of these vetrices.

Note here that the uper bound for small $|V(G)|$ is very loose because of the generalizations made in order to construct a unified result. However after not much increment in n (which is after all the case we are interested in the bounds become way more realistic) Much to our luck result 6.6 can now be further analyzed as follows:

$$6.6 = (|V(G)| - 1)^{2*\binom{k}{2}} * (k-1)^{\binom{k}{2}} * \sum_{i=0}^{|V(G)|} \binom{|V(G)|}{i}^{(k-1)}$$

And in the more familiar case were $k = 2$

$$6.6 \ ^{(k=2)} = \ (|V(G)| - 1)^2 * \sum_{i=0}^{|V(G)|} \binom{|V(G)|}{i} =$$

$$(n-1)^2 * 2^n$$

Now as a final result we can describe the number p_k defined in the beginning of this section:

$$p_k \leq \frac{(|V(G)| - 1)^{2*\binom{k}{2}} * (k-1)^{\binom{k}{2}} * \sum_{i=0}^{|V(G)|} \binom{|V(G)|}{i}^{(k-1)}}{2^{\binom{|V(G)|}{2}}} \tag{6.7}$$

6.3 Branchwidth

We have seen this far how one can derive upper bounds on the number of graphs that have some specific parameter bounded. In this attemt i tried to utilize some part of the definition to construct an somewhat loose numeration of possible cases. In the case of Branchwidth unfortunately such techniques will not do. As seen in the definition the branch decomposition all of the decompositions have the same structure (a tree with |E(G)| leaves). The only thing that changes over this structure is which edge of the original graph will be matched to each leave. Then the branchwidth is calculated by checking on each one of the branch decompositions for a specific quantity.

Facing this problem i will try to approach this matter from a different angle.

Suppose that we are in possession of a graph G of branchwidth k. This means that on all of its branch decompositions all of its possible separators are of size at most k. In is branch decompositions that would mean that

one each on every one of them for every inner edge of the decomposition at most k vertex labels will exist in both sides of the cut. Now in order to be able to produce specific numbers we define:

Definition 6.3.1. For a graph G=(V,E)

- The number of graphs with size n of a branchwidth k will be denoted B_k
- The percentage over all possible graphs will be denoted as p_k

Now some easy first useful facts are:

Theorem 6.3.1. *If T is the branch decomposition of graph G=(V,E) then the branch decomposition has exactly E(G) leaves and exactly* $2 * |E(G)| - 3$ *edges*

Proof. We will first construct a binary tree from the decomposition:

1. Choose arbitrarily a vertex u
2. *Hang* the hole tree from vertex u
3. Delete vertex u from T

Now we have a tree T' that is binary (for each inner node of the decomposition one node has assumed the role of the father for the other 2) with $|E(G)| - 1$ leaves.

For T' we know tat the number of inner nodes is $(|E(G) - 1) - 1 = |E(G)| - 2 \Rightarrow T'$ currently has

$$2 * (|E(G)|) - 3 \; nodes. \Rightarrow \; T' \; has 2 * (|E(G)|) - 4$$

edges (this is an elementary result)

As a final step we re-add vertex u to T' and the number of edges changes only by one giving a total of $2 * (|E(G)|) - 3$ edges. □

So now we know that given a Branch decomposition there are $2*(|E(G)|)-3$ possible places we can cut the tree.

Since our initial graph G is of Branchwidth k no matter where we cut we will end up with two subgraphs that share $\leq$ labels.

I will to inversely construct from these facts a way to derive possibilities of origination.

So for a graph that we have the information that its branchwidth is k we know that there is optimal branch decomposition (e.i the one resulting in a max e-separation of size k). Lets try to count then all the graphs of n vetrices that would result in a such.

Lets for starters see what happens if there is only one edge of the decomposition T that reaches the number of k common label nodes for each side.

By cutting there we end up with two trees $T_1 \& T_2$ for that contain in $n_1 \& n_2$ labels respectively. For the labels of T_2 we know that $k+1$ must be shared with the labels of T_1.

So in such a case we have

$$\binom{labels(|V(T_1)|)}{k+1}$$

possible different graphs that such a could have originated from.

To intuitively understand the above i am mostly trying to the graphs that have exactly one e-separation of branchwidth $k+1$ and counting all the different vetrices that could be a part of it. Since i am looking for an upper bound the largest quantity that $labels(|V(T_1)|)$ could have is $|V(G)|-k-1$. If it contained more labels then it couldn't possibly share $k+1$ with the other side T_2 So: for a graph containing exactly one e-saparation of branchwidth k_1 we have the upper bound:

$$\binom{|V(G)|-k-1}{k+1} \tag{6.8}$$

How many such e-separations can we have? Well as proved earlier we can cut the decomposition in $2*(|E(G)|)-3$ many places. But! quite a lot of them would not be eligible. We know that even branchwidth 1 separations did not originate form cuts made to edges that had leaves as one endpoint. Since the labels that correspond to such a cut are 2 the separation number would be 1 in all cases. Since branchwidth is computed as the maximum number of all such cuts these cases should not be counted. The leaves are of course $|E(G)|$ and therefore we are left with $|E(G)|-3$ meaningful cuts.

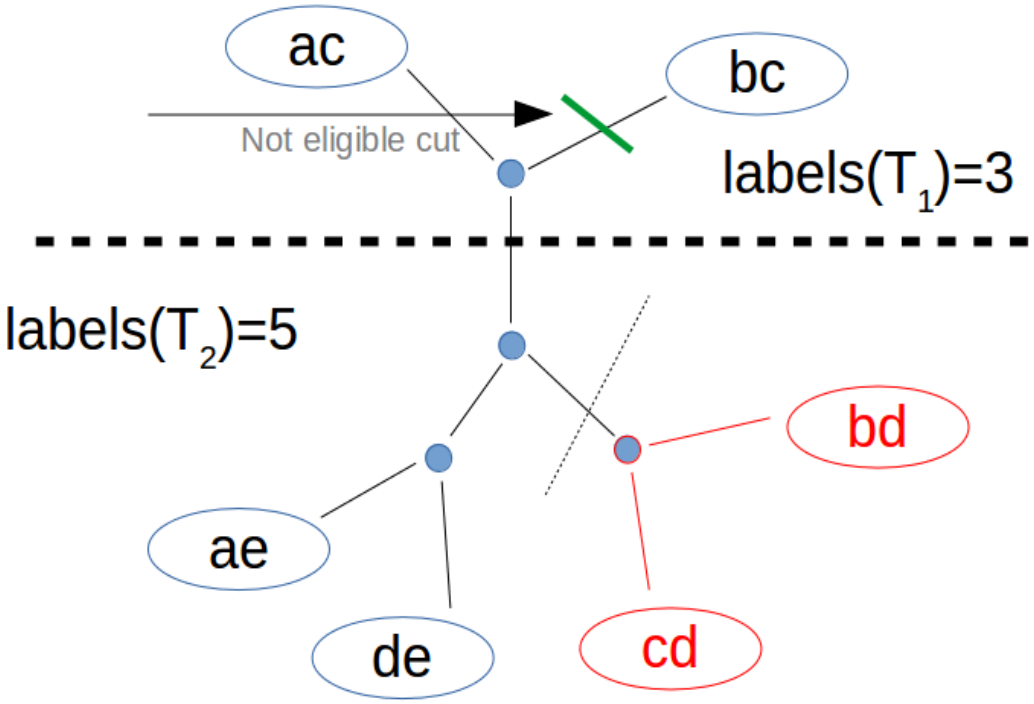
And of course trivialy E(G) is bound by $\binom{|V(G)|}{2}$. So when choosing an eligible graph that could result in a branch decomposition of branchwidth k we are also making a choice on which of the inner nodes of T the large e-separation took place.

We could have each one of those positions to actually be a large e-separation and for each one we have 6.8 choices for the possible shared labels.

$$(\binom{|V(G)|}{2}-3)*\binom{|V(G)|-k-1}{k+1} \tag{6.9}$$

Now there is one extra bound we can derive. If a Graph G has branchwidth k and n vetrices then we can derive an upper bound for E(G). This is very important since our current result is growing very fast exactly becase of all the possible edges in G. So i am looking for the largest number of edges

Figure 6.4: There are $|E(G)| - 3$ meaningful cuts



(because i want an upper bound) that if only one more edge is added the graph cannot have a branchwidth k .

It is very easy to prove as a first step that all graphs that have a minimum degree of $k+2$ cannot have a branchwidth of k. Therefore i am looking for the number of edges in the most complete graph of minimum degree $k+1$. Now it very easy to say that this graph since is it the more complete would only have one vertex of $k+1$ degree. All the other vetrices would have maximal allowed degree.

So the lonely edge u has k+1 neighbors. Those are the only vetrises that are allowed to connect with all the others in the graph.

The $|V(G)| - k - 2$ remaining can form a clique innerly and will be connected with the neighborhood of u but not u. So in total we have

$$\alpha = \frac{(|V(G)| - k - 2) * (|V(G)| - k - 3)}{2} + (k+1) * (n-1) - \frac{(k+1) * k}{2}$$

The last term is due to the edges between the vetrices of N(u) being counted twice in the previous term.

Now we can count the common labels in both size of the cuts for every possible combination of cuts and for every possible number of cuts. We can

now summarize all that up to :

$$B_k \leq \binom{|V(G)|-k-1}{k+1} * \sum_{i=0}^{\alpha-3} \binom{\alpha-3}{i}$$

and

$$p_k \leq \frac{\binom{|V(G)|-k-1}{k+1} * \sum_{i=0}^{\alpha-3} \binom{\alpha-3}{i}}{2^{\binom{|V(G)|}{2}}}$$

$$= \frac{\binom{|V(G)|-k-1}{k+1} * 2^{\alpha-3}}{2^{\binom{|V(G)|}{2}}} \tag{6.10}$$

Te above is a very accurate result since it can be verified for small values of n and k.

6.4 Pathwidth

Pathwidth is another metric on graphs tightly associated with a decomposition. In this case its a Path decomposition as presented in 2.5 . Following from the definition we can produce easily a trivial result.

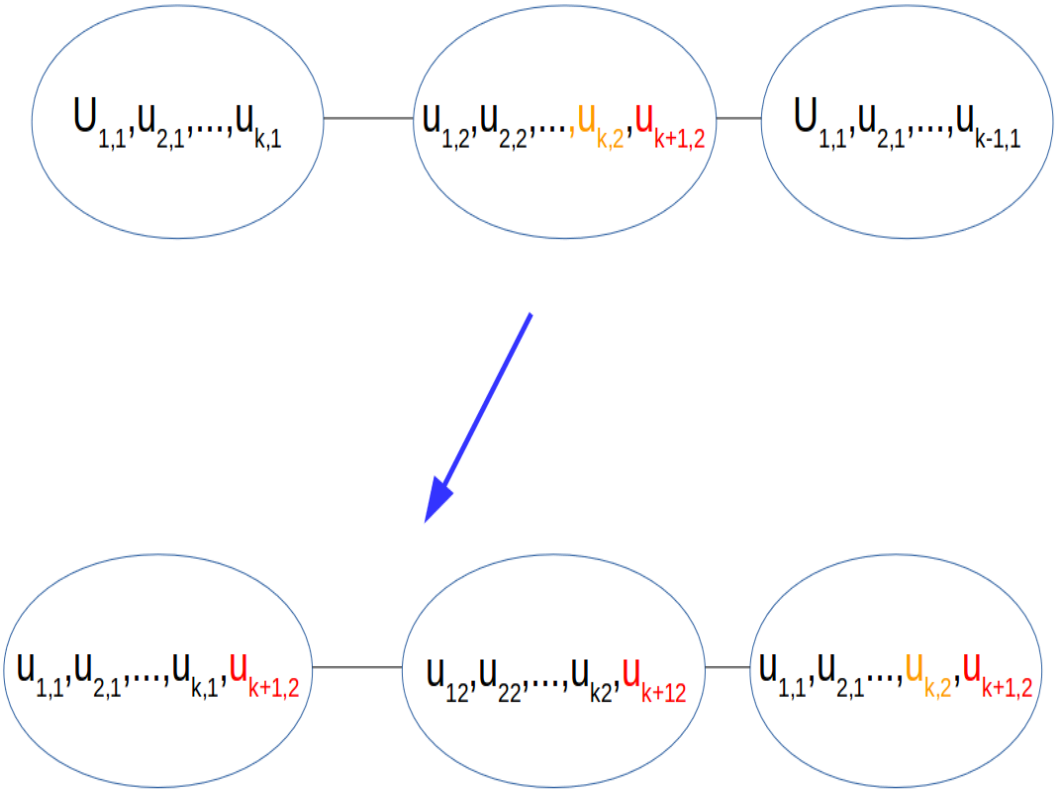
If the graph is of pathwidth k then the largest of its nodes will contain $k+1$ labels. To make things easier on a first attempt we can assume that *all* nodes contain this much labels. We can even do that deterministically using the following steps:

1. Decide a start and an end for the path decomposition and enumerate the nodes accordingly
2. starting from node 0 follow the decomposition until a node with $k+1$ labels is reached.
3. pick a label and add it to all neighbor (as far as the next $k+1$ node or the end of the path) nodes that contain less than $k+1$ nodes.
4. repeat until all nodes in the path contain $k+1$ labels

The above procedure is efficient and will produce a valid path decomposition. A specific label will only appear in a connected subpath of the decomposition. In figure 6.6 we see how on can reuse labels of nodes. Here we do not have the property of *smooth* tree decompositions as we had in treewidth but we can still produce some results.

Definition 6.4.1. The number of graphs G with $|V(G)|$ nodes and k pathwidth will be denoted as P_k

Figure 6.5: Filling up nodes with adjacent labels



The above seems very similar to the treewidth case since we again could construct bags of size $k+1$

Before we start applying tighter bounds on the number of decompositions we can have a first estimation of P_k Without knowing yet how many nodes are in the path decomposition that an upper bound of the number of possible path decompositions. There can be

$$\binom{|V(G)|}{k+1} \tag{6.11}$$

many nodes on the path decomposition.

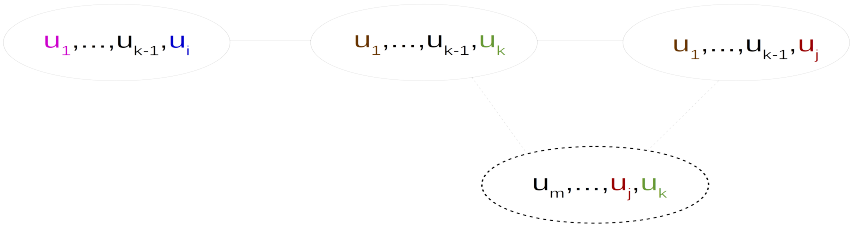
In no case of course all of the above are acceptable path decompositions.By choosing $k+1$ vetrices of the original graph (with an arbitrary enumeration of its vetrices) we will positively end up with a situation where:

- There exists a bag B_1 which contains labels u_i, u_j, u_k.

- There exist a bag B_2, which contain labels u_i, u_j but not u_k
- There exist a bag B_3 which contain labels u_j, u_k but not u_i

This can be easily shown if one considers that for the number of nodes in the path decomposition we requested all possible unique combinations of $k+1$ labels. So for a short proof:

Figure 6.6: We cannot have all combinations of k+1 labels



Proof. Say that all bags that contain label u_1 have been connected through a path graph. We can pick two different labels from two non consecutive bags of this path graph. Such labels will always exist since on each new node of the graph at least one new label will appear. Focusing on those two labels no say (u_k, u'_k) there will be a case in the later on bags to be added to the path graph were they will appear in the same set of labels (i.e. a new bag) since all sets that contain labels u_1 have been used already the above holds □

In the above case all of these bags must be connected but there is no way that this is done through a path graph.

So even though the above is not tight we do know that no more than

$$\binom{|V(G)|}{k+1}^{\binom{|V(G)|}{k+1}}$$

exist.

We can however produce better results. The first thing we will use is the following:

Theorem 6.4.1. *For a path decomposition graph P over a graph G of $|V(G)|$ nodes there will be at most $|V(G)| - k$ bags in T.*

The proof of this follows from the similar result about treewidth in section 6.1 . Only difference here is that although there is no proof that the bags will have k common labels we can accept this numbers as an upper

bound since for the construction of a new bag at least one label changes and the old one can never be used again.

Now we can summarize a similar result only for a slightly smaller percentage of graphs:

$$T'_k = \binom{V(G)}{k+1} * \prod_{i=1}^{|V(G)|-k-1} \binom{k+1}{k} * (|V(G)| - k - i) = \tag{6.12}$$

Notice how the only difference with the similar treewidth result is that in the treewidth case there was for each new bag i a multiplicative i to express the number of choices for a *parent* bag of the current one. In our case no such term exists since the resulting graph must be a path one.

We can head forward now and calculate the final result which is

$$\sum_{j=1}^{k} \binom{V(G)}{j+1} * (j+1)^{(|V(G)|-j-1)} * (|V(G)| - k - 1)!$$

This clearly not only results in less graphs of bounded pathwidth than bounded treewidth k but also maintains the good properties of 6.3 over low k.

To calculate the final percentage we have the similar case of

$$p_k \leq \frac{\sum_{j=1}^{k} \binom{V(G)}{j+1} * (j+1)^{(|V(G)|-j-1)} * (|V(G)| - j - 1)!}{2^{\binom{|V(G)|}{2}}} \tag{6.13}$$

Chapter 7

Parameterized Algorithmic Meta-Theorems

In this chapter all the work done above will be brought together in order to describe independent results as part of this larger framework. There are some Logic classes that have been studied and associated with a parameter already. For these logics with or without strict inclusion relationships between them i will be explaining how the trade-of function described in chapter 5 works .

7.1 Logics and families of Graphs

Note here that the existence of such theorems is the reason for the initiation of this hole approach. Existing results regard the following classes:

7.1.1 FO

In this section, we investigate the complexity of the problems Eval(FO) and the MC(FO)which are respectively:

Definition 7.1.1. Let Φ be a class of formulas.

- The evaluation problem for Φ is the following problem:

  EVAL(Φ)
  Input: A structure A and a formula$\phi \in \Phi$.
  Output: (A)

- And the Model Checking would be the above but restricted to the decision version:

  MC(Φ)
  Input: A structure A and a formula$\phi \in \Phi$.
  Output: Decide whether (A) holds.

A crucial parameter is the width of a first-oder formula ϕ, which we define to be the maximum number of free variables of a subformula of ϕ. The width is trivially bounded by the total number of variables appearing in ϕ, and, of course, by the length of ϕ.

Theorem 7.1.1. *Eval(FO) and MC(FO) can be solved in time* $O(|\phi| * |A|^w \cdot w)$, *where w denotes the width of the input formula .*

Proof. The recursive definition of ϕ(A) immediately gives rise to a recursive algorithm. Observe that for a formula $\phi(x_1, \ldots, x_k)$, computing ϕ(A) from the immediate subformulas of ϕ requires time $O(w*|A|^w)$. For example, suppose that $\phi(x_1, \ldots, x_k) := \psi(x_{i1}, \ldots, x_{ir}) \wedge \chi(x_{j1}, \ldots, xjs)$, *where* $\{i_1, \ldots, i_r\} \cup \{j_1, \ldots, j_s\} = [k]$. Suppose that $\{i_1, \ldots, i_r\}$ *cap*$\{j_1, ..., j_s\} = l_1, ..., l_t$. We sort the tuples in the relations $\psi(A)$ and $\chi(A)$ lexicographically by the components $l_1, \ldots, l_t$(based on an arbitrary order of the underlying universe A). Then we 'join' the two sorted lists to obtain $\phi(A)$. If we use bucket sort, the sorting requires time $O(t * |A|^{max\{r,s\}}k)$. Joining the two lists requires time $O(w * |A|^k)$. Since the number of subformulas of a formula ϕ is bounded by $|\phi|$, this algorithm achieves the claimed time bound. □

Corollary. *Let* $k \geq 1$, *and let* FO^k *denote the fragment of FO consisting of all formulas with at most k variables. Then Eval(*FO^k*) and MC(*FO^k*) can be solved in polynomial time.*

To be absolutely precise here, we have to add $O(|enc(A)$

$|totherunningtime.Thisisbecausethewholeinputhastobereadtoextracttherelevantpartsandbuil$

Actually, it can be proved that MC(FO^2) is complete for PTIME under logarithmic space reductions. Occasionally, we are interested in the restrictions of the problem to a fixed formula ϕ.

Corollary. *For every first-order formula , the eval(*ϕ*) and MC(*ϕ*) problems can be solved in polynomial time:*

This result can be strengthened. It is not hard to see that the problem MC(ϕ) belongs to the circuit complexity class uniform-AC_0 .

Let us turn to the complexity of the model-checking problem.

Theorem 7.1.2. *The following hold:*

- *For every* $t \geq 1$, *the problem MC(*Σ_t *) is complete for the t-th level* Σ_P^t *of the polynomial hierarchy.*
- *MC(FO) is complete for PSPACE*

The proof for the above is not in the spectrum of this analysis. However we can see how the above are not very useful in the case that one is building an algorithm that check a FO logic property. To that end we know:

Theorem 7.1.3. *Fix $l > 0$. Then the model-checking problem for FO on structures of bounded by k degree is fixed-parameter linear.*

The proof of the above can be found in [38]. An effective algorithm is presented where all structures up until a bounded by l integer are evaluated over the property in FPT time.

> Can one prove a similar result for FO queries on arbitrary structures?

The answer is most likely no, assuming some separation results in complexity theory . In fact, these results show that even fixed-parameter tractability is very unlikely for arbitrary structures. Nevertheless, fixed-parameter tractability can be shown for some interesting classes of structures.

Theorem 7.1.4. *[22] If C is a minor-closed class of graphs which does not include all the graphs, then model-checking for FO on C is fixed-parameter tractable.*

The proof of the above is derived from the following facts. H is an excluded minor of a class of graphs C if no G C has H as a minor. If such an H exists, then C is called a class of graphs with an excluded minor.

- If C is a minor-closed class of graphs, membership in C can be verified in Ptime [9]

- If C is a Ptime-decidable class of graphs with an excluded minor, then checking Boolean FO queries on C is fixed-parameter tractable ([42]).

Corollary. *Model-checking for FO on the class of planar graphs is fixed-parameter tractable.*

Proof. Planarity is a property with the finite set of obstructions being = $\{K_5, K_{3,3}\}$ so 7.1.4 holds. □

So finally we are in possession of a parameter (vertex degree) that defines the instances that the Model-Checking Problem of FO is tractable over.

The corresponding percentage for $|V(G)| = n$ and k the bound of the vertex degree is: The most full graph of degree bound by k is a k-canonical graph of n vetrices. Therefore all the possible graphs of degree bound by k are:

$$2^{\frac{n*k}{2}} \tag{7.1}$$

7.1.2 MSO_2 - Courcelle's Theorem

Courcelle's Theorem is a logic-based meta-theorem for establishing that various graph-theoretic properties are decidable in linear FPT time, when the parameter is input graph treewidth. Similar results were obtained independently by Borie, Parker and Tovey [5].

Courcelle's Theorem has the form:

> If the property of interest is expressible in MS 2 logic, then, parameterizing by the treewidth of the input, it can be determined in linear FPT time whether the graph has the property.

More formally:

Theorem 7.1.5. *If F is a family of graphs described by a sentence in second-order monadic logic, then F has finite index in the large universe of t-boundaried graphs.*

In the language of logic, we consider structures that satisfy the relevant formulae. Here the structures are graphs and we say that $G \vDash \phi$ for a formula *phi* if the interpretation of ϕ in G is true.

TW φ-Model Checking for ϕ
Input: A graph G = (V , E), and Property φ.
Parameter: tw(G) = t + $|\varphi|$
Output: "Yes" iff G $\models \varphi$, "No" otherwise.

The main theorem means that this problem is linear time FPT. Later, we will look at a theorem of Seese which is something of a converse.

Proof. We outline the idea of the proof.

1. Given a graph G, compute, in linear time, its tree decomposition, consisting of a tree T and a set B_t for each node t of T This can be done thanks to Bodlaender [31] .Since the treewidth is fixed, say k, each B_t is of size at most k + 1, and thus all the graphs generated by B_t 's can be explicitly enumerated.

2. This allows us to express MSO quantification over the original graph G in terms of MSO quantification over T. Thus, we are now in the setting where MSO sentences have to be evaluated over trees.

3. The above can be done in linear FPT time: Suppose we have a sentence Φ and a structure A (string or tree). We convert Φ into a deterministic automaton. This can be done thanks to the theorems of Buchi [1] *A language is definable in MSO iff it is regular.* and : *A set of trees is definable in MSO iff it is regular.* due to Thatcher and Wright [40]

4. Evaluate the formula over the constructed automaton.

Following the above steps one can evaluate in linear FPT time any MSO_2 formula . □

Anothem more detail proof of this that describes an aglorithm for doing the above can be found in the book of Downey and Fellows [35]

Detlef Seese proved a converse to Courcelle's Theorem. Note that this is extremely important as it fullfils a situation as the one described in the first row of table 5.1

Theorem 7.1.6. *(Seese [24]) Suppose that F is any family of graphs with a decidable monadic second-order (MSO_2) logic. Then, there is a number n such that for all$G \in F$, the treewidth of G is less than n.*

Notice the very interesting dichotomy from linear time decidability: the monadic second-order logic of bounded treewidth graphs to the undecidability of the monadic second-order logic for families of graphs failing to have a bound on the treewidth.

7.1.3 MSO_1

The material on Courcelle's Theorem and Seese's Theorem both looked at the MS_2 logic based on the two-sorted language with predicate symbols for edges, vertices, and incidence. As we mentioned earlier, there is a long history concerning the "basic" monadic second-order logic where we have no predicate for edges but need binary relations for these objects. Now, the expressive power changes.

Naturally, the methods we have used for Courcelle's MS_2 Theorem still work. Seese [24] was able to prove the following theorem

Theorem 7.1.7. *If a class of planar graphs has a decidable MSO_1 theory, then that class has uniformly bounded treewidth.*

This is somewhat of an analogous of theorem 5.1.3.

The decisive lemma is the following.

Lemma 7.1.8. *Let K be any class of graphs such that for every planar graph H there is a planar G K with H minor G. Then, the monadic second order (MSO_1) theory of K is undecidable.*

The principle difficulty in the proof is then to prove that MSO_1 interpretability occurs. This is quite intricate and heavily relies on planarity to be able to interpret the class of grids of size n into the class as minors. We refer the reader to [24] for details. Subsequently, Courcelle and others have extended Theorem 7.1.7 to much wider classes of graphs. In particular, Courcelle and Oum have Seese's MS_2 theorem extended to MS_1 via the notion of cliquewidth, a width metric with a similar parse language to treewidth, as we saw in chapter 6. That is, Courcelle and Oum prove the following.

Theorem 7.1.9. *(Courcelle and Oum [20]) If a set of directed or undirected graphs has a decidable MSO_1 theory (even with the addition of the "even cardinality" predicate), then it has bounded cliquewidth.*

7.1.4 Matroid MSO

The analytical definition of Matroid MSO is given in the appendix .2. It is quite similar with the definition of simple MSO but it allows of elements of countably infinite cardinality in a Structure.

The idea that a graph is tree-like if it can be decomposed entirely across small separations can be extended to algebraic structures, and, in particular, matroids using algebraic independence instead of topological separation as the central decomposition criteria. This programme was initiated by Hliněný and Whittle [13, 14], and is part of a long-term program to generalize the Graph Minors Project of Robertson and Seymour (the originators) to an analogous matroid structure theory (and associated FPT algorithmic methods). In the setting of matroid theory, matroid branchwidth is easier and more natural to define than the matroid analog of treewidth .In the definition of branchwidth for graphs, the key idea in the representation of the separation properties is to make a "data-structure" for the graph, where the edges of the graph (the essential elements of topological connectivity), are in one-to-one correspondence with the leaves of a ternary tree.In the setting of matroids, we need an analog of the notion of topological separation, and this is provided by the rank of a set of vectors, a measure of algebraic linear independence of the set of vectors. The connectivity or width function is

$$\lambda(A) = r(A) + r(E\backslash A) - r(E) + 1$$

, where r is the rank function of the matroid. A separation can be defined in the same way as for graphs, and this results in a partition of the set E of matroid elements into two subsets A and $B = E\backslash A$. The branchwidth of a graph and the branchwidth of the corresponding graphic matroid may differ. For instance, the three-edge path graph and the three-edge star have different branchwidths, 2 and 1, respectively, but they both induce the same graphic matroid with branchwidth 1. Mazoit and Thomassé showed also that for graphs that are not forests, the branchwidth of the graph is equal to the branchwidth of its associated graphic matroid.

Robertson and Seymour conjecture that the matroids representable over any finite field are well-quasi-ordered by matroid minors, analogously to the Robertson– Seymour theorem for graphs. So far, this conjecture has been proven only for the matroids of bounded branchwidth. It is possible to prove a version of Courcelle's Theorem for matroids of bounded branchwidth. The syntax consists of variables for matroid elements and predicates $e \in F$ where F is a variable for sets of elements, and indep(F) which is true if and only

if F is an independent set. The above description is clearly more expressive than that of MSO.

Theorem 7.1.10. *(Hliněný [15]) Let F be a finite field and φ a sentence of MMS, matroid monadic second order logic as described above. Suppose that the n element matroid M is given a vector representation over F together with a branch decomposition of width k. Then there is a linear FPT algorithm (in F, , k) deciding whether $M \vDash$.*

7.2 Parameter Correlations

For the parameters studied in Chapter 6 and utilized in the section there are some results that instead of describing how *strong* they are as parameter describe a set of bounds between them. These bounds are more suitable for expressing the impact of changes in one parameter to the others. Such relations contain very useful info that can be utilized in the parametric algorithm design but they are less important in the comparison of expressive power.

We can say for the following:

Pathwidth Since path-decompositions are a special case of tree decompositions, the pathwidth of any graph is greater than or equal to its treewidth. We can deduce easily therefore that for a graph G:

$$tw(G) \leq pw(G)$$

There is not yet an upper bound for pathwidth in measurement of treewidth.

Cliquewidth The graphs of treewidth w have clique-width at most $3 * 2^{w-1}$. The exponential dependence in this bound is necessary: there exist graphs whose clique-width is exponentially larger than their treewidth.[17] In the other direction, graphs of bounded clique-width can have unbounded treewidth; for instance, n-vertex complete graphs have clique-width 2 but treewidth n − 1. However, graphs of clique-width k that have no complete bipartite graph K_t,t as a subgraph have treewidth at most $3 * k * (t-1)-1$. Therefore, for every family of sparse graphs, having bounded treewidth is equivalent to having bounded clique-width.

Branchwidth Having bounded branchwidth imposes strong structure on a graph. As we will see branchwidth imposes a 1/3 -approximation over treewidth.

Theorem 7.2.1. *(Robertson and Seymour) Suppose that* $bw(G) > 1$. *Then*

$$bw(G) \leq tw(G) + 1 \leq [3/2 * bw(G)]$$

. the proof is thanks to Hliněný, Oum, Seese, and Gottlob [16]

In conclusion we can say that according to the above the most strict parameters are treewidth and branchwidth. The relationship between them is not definitive though. The following diagram describes our known relations:

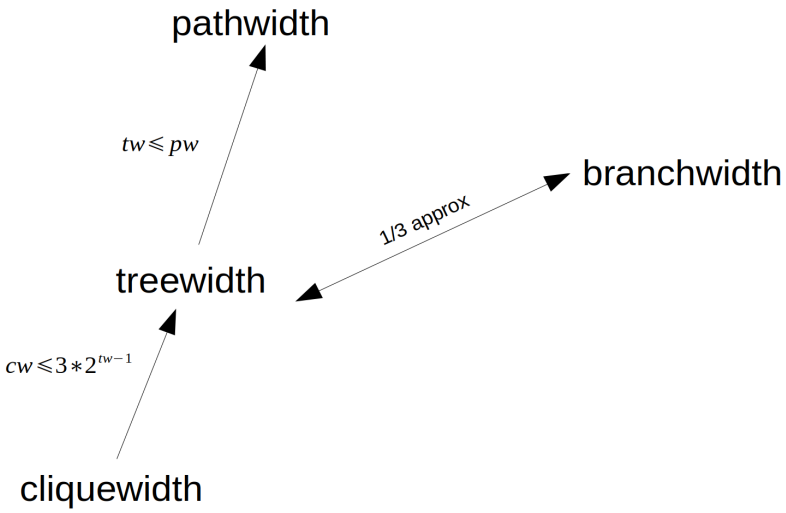


Figure 7.1: The parameters studied in chapter 6

An exactly inverse situation holds over how *strong* each parameter is. As seen in chapter 6 each one has been tied with an upper bound . This situation is reproducing the fact that the more strong a parameter is and the more aspects of graphs structural properties it bounds the less instances are expected to correspond to a specific value of it.

7.3 Combining the Results

A summary on results in this sector are presented on the following table. Each row corresponds to a Logic among with the paired parameter that classifies is as FPT. Each row consecutively corresponds to two theorems. In some cases both are known while in others only one is. The expressive power grows as we continue to lower rows. Each one of those increments therefore according to theorem 5.1.2 should lead to a stronger parameter

i.e a parameter that when for an input of size n coresponds to a smaller percentage of instances of bounded k . The above claim holds for the bounds that are given i chapter 6 for the parameters appearing here.

FPT Checkable Logic	Parameter	Proof	Converse
FO	degree	Seese [22]	-
MSO_1	cliquewidth	-	Courcelle & Oum [20]
MSO2	treewidth	Courcelle [18]	Seese [24]
MMSO	branchwidth	Hliněný [15]	-

Table 7.1: A summary of the existing works over this framework.

The percentages of each parameter for input size n and the parameter bounded by k are given in summary bellow in table NEW. The it is obvious to notice from those the impact of the inversely proportional between expression power and parameter *power* described in section 5.1.

Chapter 8

Conclusion

8.1 Explaining the results

In full scope now one can use table **??** to utilize meta-knowledge of a problem to help him choose the correct parameter for solving the problem efficiently.

For instance while checking a property p one could be absolutely sure that there exists a parametrized algorithm that runs in FPT time and recognizes property p. Now the programmer has an extra tool that he can take advantage of while designing algorithms. Now his resources can be spend in finding the optimal solution or a fast one instead of searching blindly for an existing one. An example of such use could be:

Example 8.1.1. For the property of Hamiltonicity: In developing the MS_2 formula that expresses the property of Hamiltonicity, we will represent the set of red edges by the variable R and the set of blue edges by the variable B.

$$\exists R \exists B \forall u \forall v \{part(R,B) \wedge deg(u,R) = 2 \quad \wedge span(u,v,R) \wedge \forall x \forall y \exists W(con(u,v,W,R)\}$$

where span, deg and part are described in the appendix .1.

The above logic is MSO_2 and therefore Hamiltonicity is FPT parameterized by treewidth. However the above expression is minimal(there is no MSO_1 equivalent) and therefore Hamiltonicity is at least W[1]-hard parameterized by cliquewidth - even if no such reductions or algorithms exist yet.

In addition the results can be used in reverse. An unknown relationship between logics will have an impact on the relevant running time of each ones Model-checking. Therefore some short of comparison between them can arise just by looking at the running times of algorithms that recognize the properties of each one. This might seem trivial but in the world of formal

verifications where there is need for a minimal usage of expansions over a specific logic such results can be utilized massively. For instance by checking what kind of processes over inputs run in a machine one could derive what kind of parameter bounds all of them. Consecutively they could request a machine that is designed to be compatible with the respectable logic.

8.2 Further Research

Besides the obvius expantion of such research that is to analyse more Logic Classes in a way as done in this thesis there are many other ways this framework could be expanded.

8.2.1 Fine grained approach

For starters the existing(and further) work in parameters could be explored in a fine grained manner. We could try checking changes in the complexity of a problem while remaining FPT. What would happen for instance to a problem when parameterized by many parameters each one stronger than the last while even the weakest is enough for an FPT time. Through such search we could learn even more about the nature of parameters and the limits of computation over bounded universes.

This of course requires a solid framework of parameterized fine grained reductions that would be able to capture discrimination of complexity inside the FPT class among with a reduction that would not allow to the changing of parameters to "jump" between said discriminations. This is a third way of regarding the model checking problem, this time by requesting a FPT running time and toggling the parameterization between known efficient ones.

8.2.2 Complexity

The reason that this whole work took place over clases defined through logic is no other that the fact that they are the most proximate to the tools needed to relate with a computational model. Language-theoretic approaches to complexity are one of the most popular and effective areas of computer science. That said this is no reason to study the parameterised time requirements of classes of problems defined by ***other*** means.

Those could be classes appearing in the approximations framework or the probabilistic one. There is no one forbiding for isntace all efficient approximable problems to also be FPT parameterized through lets say degree(this is just an example , it is not true). It is way harder of course to categorize the structural properties of approximable problems but an interesting point could be made in defense of such a thought (interesting results in this direction are those of C. Bazgan [12] , L. Cai [26] and L. Cai, J. Chen [21])

Furtherer the results of chapter 6 could be utilized to produce probabilistic schemes for the recognizing the membership in a family of graphs. Such a thought is exremely young and one can find some interesting works from M. Muller [43].

8.2.3 Logic Metrics

A very interesting work could be done in continuation of chapter 5 regarding the way of measuring the expression power of a logic. For the above work to make valid points it is important to be able to compare such power. Of course in the strict inclusion case this is not of the same importance but as seen already from existing work and even more noticeable when considering extensions that is not always the case . Serious theoretical and applied work could take place in search for a proper way to express how "massive" is a class of properties.

Appendix

.1 Property Descriptions

The descriptions of conn, span, deg and part are given here :

- part(R, B) : $\forall e(e \in R \vee e \in B) \wedge \neg(e \in R \wedge e \in B)$,
- deg(u, R) = 2 : $\exists e1, e2 \neg(e1 = e2) \wedge inc(e1, u) \wedge inc(e2, u) \wedge e1\ R \wedge e2\ R \wedge \neg(\exists e1, e2, e3, \neg(e1 = e3) \wedge \neg(e2 = e3) \wedge inc(ei, u) \wedge ei \in R\ for\ i \in 1, 2, 3)$,
- span(u, v, R) : $\exists V, W part(V, W) \wedge u \in V \wedge v \in W \rightarrow \exists(e, x, y inc(e, x) \wedge inc(e, y) \wedge x \in V \wedge y \in W \wedge e \in R$,
- conn(x, y, W, R) :$\exists V1, V2 \subseteq V[V1 \cup V2 = V \wedge V1 \cap V2 = \varnothing \wedge x \in V1 \wedge y \in V2] \rightarrow \exists r \in R \exists p \exists qp \in V1 \wedge q \in V2 \wedge inc(r, p) \wedge inc(r, q)$

.2 Matroid MSO

Definition of matroid MSO logic: First, we present basic definition concerning monadic second-order logic.

Definition .2.1. We assume two countably infinite set of variables: element variables and set variables. Element variables are denoted by lower-case letters, set variables are denoted by upper-case letters. **Matroid monadic second order** formulas are defined inductively as follows:

- If x and y are element variables, then $x = y$ is a formula.
- If x is an element variable and X is a set variable, then $x \in X$ and $x \in cl(X)$ are formulas.
- If ϕ is a formula, then $\neg\phi$ is a formula.
- If ϕ and ψ are formulas, then $\phi \wedge \psi$ is a formula.
- If ϕ is a formula and x is an element variable, then $\exists x\phi$ is a formula.
- If ϕ is a formula and X is an element variable, then X is a formula.

Bibliography

[1] *Buchi.* Weak second-order arithmetic and finite automata. Z. Math. Log. Grundl. Math. 6, 66–92 (1960)

[2] *Arnborg, Stefan* (1985), "Efficient algorithms for combinatorial problems on graphs with bounded decomposability – A survey", BIT, 25 (1): 2–23

[3] Menger,1927

[4] *S. Arnborg, J. Lagergren, D. Seese*, Easy problems for tree-decomposable graphs. J. Algorithms 12(2), 308–340 (1991)

[5] *R. Borie, G. Parker, C. Tovey*, Automatic generation of linear-time algorithms from predi- cate calculus descriptions of problems on recursively constructed graph families. Algorith- mica 7, 555–581 (1992)

[6] *Robertson, Neil; Seymour, Paul D*, (1984), "Graph minors III: Planar tree-width", Journal of Combinatorial Theory, Series B, 36 (1): 49–64,

[7] *Neil Robertson and P.D. Seymour*. Graph minors. XX. wagner's conjecture. Journal of Combinatorial Theory, Series B, 92(2):325–357, 2004.

[8] *Robertson, Neil; Seymour, Paul D* , "Graph minors. X. Obstructions to tree-decomposition", Journal of Combinatorial Theory, 52 (2): 153–190,

[9] *N. Robertson, P. Seymour*, Graph minors. XIII. The disjoint paths problem. J. Comb. Theory, Ser. B 63(1), 65–110 (1995)

[10] *M. Rabin*, Decidability of second-order theories, and automata on infinite trees. Trans. Am.Math. Soc. 141, 1–35 (1969)

[11] *Seymour, Paul D. Thomas, Robin* (1994), "Call routing and the ratcatcher", Combinatorica, 14 (2): 217–241,

[12] C. Bazgan, Schémas d'approximation et complexité paramétrée, Rapport de stage de DEA d'Informatique á Orsay, Université Paris-Sud, 1995

[13] P. Hliněný, G. Whittle, Matroid tree-width. Eur. J. Comb. 27, 1117–1128 (2006)

[14] *P. Hliněný, G. Whittle*, Addendum to "Matroid treewidth". Eur. J. Comb. 30(4), 1036–1044 (2009)

[15] *P. Hliněný*, Branch-width, parse trees, and monadic second-order logic for matroids. J. Comb. Theory, Ser. B 96(3), 325–351 (2006)

[16] *P. Hliněný, S.-I. Oum, D. Seese, G. Gottlob*, Width parameters beyond treewidth and their applications. Comput. J. 51(3), 326–362 (2008)

[17] *Corneil, Derek G.; Rotics, Udi* (2005), "On the relationship between clique-width and treewidth", SIAM Journal on Computing, 34 (4): 825–847

[18] *B. Courcelle*, The monadic second-order logic of graphs. I. Recognizable sets of finite graphs. Inf. Comput. 85(1), 12–75 (1990)

[19] *B. Courcelle*, On context-free sets of graphs and their monadic second-order theory, in Graph-Grammars and Their Application to Computer Science 3rd International Workshop, Warrenton, Virginia, USA, December 2–6, 1986, ed. by H. Ehrig, M. Nagl, G. Rozenberg, A. Rosenfeld. LNCS, vol. 291 (Springer, Berlin, 1987), pp. 133–146

[20] *B. Courcelle, S. Oum*, Vertex-minors, monadic second-order logic, and a conjecture of Seese. J. Comb. Theory, Ser. B 97(1), 91–126 (2007)

[21] *L. Cai, J. Chen*, On fixed-parameter tractability and approximability of NP optimization problems. J. Comput. Syst. Sci. 54(3), 465–474 (1997)

[22] *D. Seese*, Linear time computable problems and first-order descriptions. Math. Structures Comput. Sci., 6(6):505–526, 1996.

[23] *D. Seese*, Entscheidbarkeits- und Interpretierbarkeitsfragen monadischer Theorien zweiter Stufe gewisser Klassen von Graphen, PhD thesis, Humboldt-Universitat, Berlin, 1976

[24] *D. Seese*, The structure of models of decidable monadic theories of graphs. Ann. Pure Appl. Log. 53(2), 169–195 (1991)

[25] *Arnborg, S. Corneil, D. Proskurowski*, A. (1987), "Complexity of finding embeddings in a k-tree", SIAM Journal on Matrix Analysis and Applications, 8 (2): 277–284,

[26] *L. Cai*, Fixed parameter tractability and approximation problems, Project report, June 199

[27] *H. Bodlaender, T. Kloks*, Efficient and constructive algorithms for the pathwidth and treewidth of graphs. J. Algorithms 21(2), 358–402 (1996)

[28] *Courcelle Bruno, Engelfriet Joost, Rozenberg, Grzegorz* (1993), "Handle-rewriting hypergraph grammars", Journal of Computer and System Sciences, 46 (2): 218–270,

[29] Bodlaender, Hans L.; Thilikos, Dimitrios M. (1997), "Constructive linear time algorithms for branchwidth", Proc. 24th International Colloquium on Automata, Languages and Programming (ICALP '97), Lecture Notes in Computer Science, 1256, Springer-Verlag, pp. 627–637

[30] *Bodlaender, Hans L.; Gilbert, John R.; Hafsteinsson, Hjálmtýr; Kloks, Ton* (1992), "Approximating treewidth, pathwidth, and minimum elimination tree height", Graph-Theoretic Concepts in Computer Science, Lecture Notes in Computer Science, 570, pp. 1–12

[31] A linear-time algorithm for finding tree-decompositions of small treewidth. SIAM J. Comput. 25(6), 1305–1317 (1996)

[32] *Rod Downey, Michael Fellows*, Fixed-parameter tractability and completeness. III. Some structural aspects of the W-hierarchy, in: Complexity Theory, Cambridge Univ. Press,Cambridge, 1993, pp. 191–225

[33] *Rod G. Downey, Michael R. Fellows*, Fixed-parameter tractability and completeness. I. Basic results, SIAM J.Comput. 24 (4) (1995) 873–921.

[34] *Rod G. Downey, Michael R. Fellows*, Fixed-parameter tractability and completeness II: On completeness for W[1], Theoret. Comput. Sci. 141 (1–2) (1995) 109–131

[35] *Rodney G. Downey r Michael R. Fellows* Fundamentals of Parameterized Complexity , Texts in Computer Science , p 386-341 Springer 2013

[36] *M. Garey, D. Johnson*, Computers and Intractability. A Guide to the Theory of NP- Completeness (Freeman, San Francisco, 1979)

[37] *M. Vardi*, The complexity of relational query languages (extended abstract), in Proceedings of 14th ACM Symposium on Theory of Computing (STOC '82), San Francisco, California, USA, May 5–May 7, 1982, ed. by H. Lewis, B. Simons, W. Burkhard, L. Landweber (ACM, New York, 1982), pp. 137–146.

[38] *Leonid Libkin* Elements in Finite model theory . Texts in theoretical Computes Science Springer 2004

[39] *N. Immerman*. Descriptive Complexity. Springer-Verlag, 1999.

[40] J. Thatcher and J. Wright. Generalized finite automata theory with an ap- plication to a decision problem of second-order logic. Mathematical Systems Theory, 2 (1968)

[41] *J. Flum · M. Grohe* Parameterized Complexity Theory , Texts in computer Science . Springer pp 69-72

[42] *J. Flum and M. Grohe.* Fixed-parameter tractability, definability, and model- checking. SIAM Journal on Computing 31 (2001), 113–145.

[43] *M. Muller.* Parameterized Randomization . PhD thesis, Albert-Ludwigs-Universit¨at Freiburg im Breisgau, 2008.

[44] *Stathis Zachos* Computability and Complexity Notes. Athens 2017