UNIVERSITÄT ZU LÜBECK
INSTITUT FÜR
THEORETISCHE INFORMATIK

On the Space and Circuit Complexity of Certain Parameterized Problems

Über die Platz- und Schaltkreiskomplexität bestimmter parameterisierter Probleme

Masterarbeit

im Rahmen des Studiengangs
Informatik
der Universität zu Lübeck

vorgelegt von
Max Bannach

ausgegeben und betreut von
Prof. Dr. Till Tantau

mit Unterstützung von
Dipl.-Inf. Christoph Stockhusen

Lübeck, den 14. November 2014

Hiermit erkläre ich an Eides statt, dass ich die vorliegende Masterarbeit selbstständig und ausschließlich unter Verwendung der angegebenen Quellen angefertigt habe.

Ort, Datum Unterschrift

Abstract

Parameterized complexity theory provides important tools to generate and analyze reasonable fast algorithms for hard problems. Current research mainly focuses on the question whether parameterized problems are fixed-parameter tractable or not, i. e., if there is a fast algorithm solving them. Thus, the only considered resource is time, while parameterized space and circuit complexity is often omitted, although these resources are useful in classical complexity theory as well. A recent work from Elberfeld, Stockhusen, and Tantau provides a framework of parameterized space and circuit classes to cover this area [26]. Although the framework provides parameterized space and circuit classes, as well as tools to analyze them, there is still a lack of complete problems for most of these classes. The objective of this thesis is to resolve this issue by analyzing widely known problems with respect to the framework and, thus, populate the different classes. This will lead to new upper bounds for many natural problems, together with a couple of new lower bounds for some of them. Moreover, we will also be able to collide the upper and lower bounds of some of these problems and, hence, finally resolve the complexity of them.

Zusammenfassung

Die parametrisierte Komplexitätstheorie liefert wichtige Werkzeuge zum finden und analysieren schneller Algorithmen für harte Probleme. Im Fokus aktueller Forschung liegt dabei hauptsächlich die Frage, ob ein parameterisiertes Problem fixed-parameter tractable ist und somit ein schneller Algorithmus zum Lösen jenes Problems existiert. Daher ist die Zeit meistens die einzige Ressource, die untersucht wird, und das, obwohl sich Komplexitätsklassen, die auf Platz oder Schaltkreisen basieren, als sehr nützlich in der klassischen Komplexitätstheorie erwiesen haben. Eine aktuelle Arbeit von Elberfeld, Stockhusen und Tantau behandelt diese Problematik und stellt ein Framework mit parametrisierten Platz- und Schaltkreisklassen zur Verfügung [26]. Obwohl dieses Framework neben den Klassen auch Werkzeuge enthält, um diese zu untersuchen, so fehlen doch für viele Klassen vollständige Probleme. Das Ziel dieser Arbeit ist es daher diese Klassen mit Leben zu füllen; dazu untersuchen wir bekannte Probleme in Bezug auf dieses Framework und präsentieren entsprechende Ergebnisse. So werden wir für viele natürliche Probleme neue obere Schranken, sowie einige neue untere Schranken erhalten. In einigen Fällen werden wir diese Schranken des Weiteren zusammenführen und die entsprechenden Probleme somit endgültig bezüglich ihrer Komplexität klassifizieren.

CONTENTS

1 Introduction 1
- 1.1 A Brief Introduction to Parameterized Complexity 1
- 1.2 Contributions of this Work 7
- 1.3 Related Work 8
- 1.4 Structure of this Work 10

2 Automata Intersection Problem 11
- 2.1 Parameterized by the Number of Automata 12
- 2.2 Parameterized by the Length of the Word 18
- 2.3 Parameterized by the Number of States 21
- 2.4 Parameterized by the Languages 24
- 2.5 Complexity Map 29

3 Vertex Ordering Problems 31
- 3.1 Bandwidth 31
- 3.2 Cutwidth 34
- 3.3 Imbalance 39
- 3.4 Complexity Map 41

4 Colored Reachability Problems 43
- 4.1 Color Selection Reachability Problems 44
- 4.2 Color Deletion Reachability Problems 47
- 4.3 Vision Based Reachability Problems 52
- 4.4 Combined Vision Problems 59
- 4.5 Vision Problems in Trees 69
- 4.6 Versions on Directed Graphs 72
- 4.7 Complexity Maps 74

5 Graph Separation Problems 77
5.1 Cutting Vertices 78
5.2 Vertex- and Edge-Bisection 81
5.3 Separation Problems with Wardens 82
5.4 Complexity Map 85

6 Conclusion and Outlook 87

Register of Problems 89

References 90

1 INTRODUCTION

Complexity theory is the part of theoretical computer science that deals with the question of how much resources are needed to solve an instance of a particular problem. Measuring the amount of needed resources with respect to the size of the instance allows a classification of the problem. However, for many practical problems not only the instance size is known, but also many different details about the instance, too. Parameterized complexity theory is the part of complexity theory that takes these additional information into account in order to classify problems more precisely.

Besides a few exceptions, the only considered resource in the field of parameterized complexity is time. This fact stands in contrast to the objective of a more precise classification of problems, since other resources like space and circuit-depth play an important roll in classical complexity theory, too. It therefore seems reasonable to study parameterized space and circuit classes in order to obtain new insights into the complexity of problems of theoretical and practical interest. This work is based on a recent work from Elberfeld, Stockhusen, and Tantau whom provided a framework of parameterized space and circuit classes. Most of the classes of this framework are still unexplored and miss natural complete problems. Moreover, most natural problems were never studied with respect to parameterized space and circuit classes. The objective of this work is to change the situation by analyzing different natural problems with respect to the mentioned framework. This will give new insights into the complexity of many of these problems and will help us to better understand the complexity of them. Furthermore, this approach will allow us to resolve the complexity of some problems, for which a classification with respect to parameterized time complexity did fail so fare.

1.1 A Brief Introduction to Parameterized Complexity

Before we discuss the objective and results of this work in detail, we will give a brief introduction to parameterized complexity to define the area we are operating in. The following story with Jacqueline will serve as a little guide throughout this introduction: Jacqueline was asked to organize a scientific workshop and already has received a long list of scientists whom she should invite. Unfortunately, she knows that some of these scientists do not got along with each other and since the workshop should be as peaceful as possible, she has to ensure that no two scientists are invited whom do

not get along with each other. Of course, she still wants to invite as many people as possible. To ensure that the workshop will become a real success, Jacqueline got another restriction: There are a couple of scientists that closely work together in groups. These groups are known to be highly productive on workshops and, thus, she has to ensure to invite these groups. She wants to do this independently from their relation to other researchers, since the members of such groups will most likely stay in their group and, hence, are always peacefully. It may intuitively seems "easy" or "hard" for Jacqueline to solve these problems, but she is more interested in a precise, objective measurement of this "difficulty".

So our task starts here, since providing such measurements is the main objective of complexity theory. The usual way to measure how "difficult" it is to solve a problem, is to formalize the scenario as a decision problem and measure the amount of resources a classical computational model, like the Turing machine or circuits, needs to actually solve an instance of the problem. Natural resources for Turing machines are time and space; in case of circuits, this corresponds to the depth of the circuit and the total amount of gates it is using, respectively. Both problems from above can be formalized as a graph problem: For the first problem, each scientist is represented by a vertex and between two vertices exists an edge if, and only if, the two researchers do not get along with each other. We now have to find a set of vertices such that the deletion of these vertices removes all edges from the graph, i. e., we are looking for a *minimum vertex-cover*. For the second problem, each scientist is again represented by a vertex, this time we place an edge between two vertices if, and only if, the corresponding researchers collaborate with each other. We now search for a set of vertices such that each vertex of the set is connected to the rest of the set, i. e., we have to solve the problem of finding a *maximum clique*. Throughout this work, we will indicate formal problems in small caps, e. g., VERTEX-COVER or CLIQUE, and always will define them in the following manner:

► Problem 1 (VERTEX-COVER)
Instance: A graph $G = (V, E)$ together with an integer k.
Question: Is there a set $C \subseteq V$ with $|C| \leq k$ such that each edge of G is incident to at least one vertex of C? ◄

► Problem 2 (CLIQUE)
Instance: A graph $G = (V, E)$ together with an integer k.
Question: Is there a set $C \subseteq V$ with $|C| \geq k$ such that the vertices of C are pairwise adjacent? ◄

If we are a bit more accurately, a decision problem is defined as language over an alphabet Σ, like the Latin alphabet or the Binary alphabet $\Sigma_{\text{bin}} = \{0, 1\}$. The computational task is then to decide whether a given word $w \in \Sigma^\star$ is a member of this language or not. For example, in case of VERTEX-COVER the corresponding language is defined as:

$$\textsc{vertex-cover} = \{\, w \mid\ w \in \{0,1\}^\star \text{ with } w \text{ encodeing a graph } G = (V, E) \text{ and an integer } k \text{ such that there is a subset } C \subseteq V \text{ with } |C| \leq k \text{ that contains at least one endpoint of each edge.} \,\}.$$

In most cases, the representation as a problem is sufficient and we do not have to explicitly consider the underlying languages. Throughout this work, this will almost always be the case.

Once we have defined our scenario as a formal decision problem, we can analyze how much resources a computational model needs in order to solve it. We can classify a problem by grouping all problems that are solvable within the same resource bound in a *complexity class*. Famous examples are P and L, the classes of all problems solvable in polynomial time or with a logarithmic amount of space, respectively. The most important question in the most scenarios is, whether a problem is *tractable* or not. Simply spoken, it is the question of whether a reasonably fast algorithm exists that decides the problem or not. Unfortunately for us, the problems of the workshop organization problem from above, i. e., VERTEX-COVER and CLIQUE, are known to be *intractable* and, thus, under standard assumptions there is no fast algorithm solving them.

Jacqueline was a little bit disappointed by our results and already started to think about how she can explain this to her boss. Nevertheless, after a couple of coffees she had an idea. Last time, she only explained the general problem, but she omitted some details. However, Jacqueline knows that it is a workshop about theoretical computer science and as far as she remembers, theoretical computer scientists are quite calm, so that she does not expect to uninvite more than 20 researchers. Moreover, she also knows that groups of theoretical computer scientists are almost always pretty small, only about 5 to 10 members. Now Jacqueline asks herself if these problems are still so hard, if we consider these additional details.

This question is reasonable, since standard complexity theory measures the 'difficulty" of solving a problem only with respect to the instance size. But in the new scenario, we know a lot more about the instance. We call such a restriction a *parameter*, which describes certain properties of the instance. Formally, a *parameterization* of a language $L \subseteq \Sigma^\star$ is a function $\kappa : \Sigma^\star \to \mathbb{N}$ that assigns a natural number to each instance of the problem. The parameterized version of the problem then is the tuple (L, κ). In our case, the parameter is either the number of people we have to uninvite or the maximum size of a group of scientists, respectively. More precisely, we consider instances $w = (G, k)$ and have $\kappa_1(w) = k$. Throughout this work, we indicate the parameterized version of a problem with the prefix p-, e. g., p-VERTEX-COVER and p-CLIQUE. In the definition of parameterized problems, we add a line with the corresponding parameter. The definition of p-VERTEX-COVER is as example shown on the next site:

► Problem 3 (p-VERTEX-COVER)
Instance: A graph $G = (V, E)$ together with an integer k.
Parameter: k
Question: Is there a set $C \subseteq V$ with $|C| \leq k$ such that each edge of G is incident to at least one vertex of C? ◄

Many natural problems have a whole list of possible parameterizations. If we address the problem with a specific parameterization, we add the corresponding parameter to the index of p: p_k-VERTEX-COVER would be VERTEX-COVER parameterized with κ_1. If we consider another parameter, for example the degree of the graph δ, i. e., $\kappa_2(w) = \delta(G)$, we would denote the corresponding problem by p_δ-VERTEX-COVER. If we use multiple parameters in the index of p, we implicitly use a corresponding parameterization function, e. g., $p_{k,\delta}$-VERTEX-COVER is the problem parameterized with $\kappa_3(w) = k + \delta(G)$.

In parameterized complexity theory, we measure how "difficult" it is to solve a problem as a function of resource cost with respect to both, the instance size and the parameter. This allows us to analyze problems in much more detail and gives us a deeper understanding of the complexity of these problems. Corresponding to the notation of classical complexity theory, an important part is to decide whether a problem is *fixed-parameter tractable* or not. Which, again simply spoken, means that a reasonable fast algorithm exists for the problem and a certain parameter. Formally, the set of fixed-parameter tractable problems is defined by the parameterized complexity class FPT, which is the set of parameterized problems solvable by a Turing machine in time $f(\kappa(w)) \cdot \text{poly}(|w|)$ for a computable function f. Fortunately, p_k-VERTEX-COVER is known to be fixed-parameter tractable and, thus, efficiently solvable for Jacqulines instances. On the other hand, this is not the case for p_k-CLIQUE, since this problem is known to be fixed-parameter intractable.

Nevertheless, Jacqueline attended the theoretical computer science workshop and it was a great success through our help. However, Jacqueline wonders why the most people at the workshop only consider parameterized time classes and check off a problem's complexity as resolved, once it is shown to be fixed-parameter tractable or intractable. She ask herself if this stands in contrast to the claim of parameterized complexity to understand the complexity of problems more accurately. After all, she knows that the study of other resources like space or circuit-depth has generated important results in classical complexity theory as well.

She is right: At this point – one problem classified as fixed-parameter tractable, one as fixed-parameter intractable – parameterized complexity theory stops most of the time and, thus, the only considered resource is time. But in order to obtain a deeper understanding of the complexity of these problems, it seems logical to study parameterized space and circuit classes as well. A recent work by Elberfeld, Stockhusen, and Tantau introduced a framework of parameterized space and circuit classes [26]. The basic parameterized space classes of this framework are defined as parameterized counterparts to L and NL, in a similar fashion as FPT is defined with respect to P. These classes are called paraL and paraNL and are defined as the sets of parameterized problems solvable

by a deterministic or nondeterministic Turing machine using $f(\kappa(w)) + \mathcal{O}(\log|w|)$ space for a computable function f, respectively. Besides these para-classes, the framework also considers so called X-classes, namely XL and XNL. The idea behind these X-classes is to consider *sliced parameterization*, that is, considering only one *slice* of the problem. A slice of a parameterized problem is the set of all instances of the problem for one fixed parameter, i. e., X-classes hold problems that are efficiently solvable if we consider the parameter as a constant. Thus, the corresponding classes XL and XNL are defined as the sets of problems solvable by a deterministic or nondeterministic Turing machine using $f(\kappa(w)) \cdot \mathcal{O}(\log|w|)$ space for a computable function f, respectively.

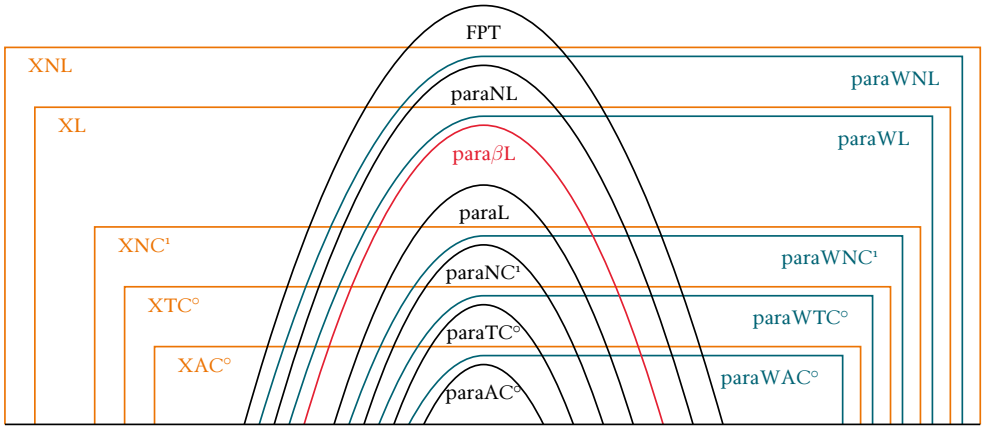
Elberfeld, Stockhusen, and Tantau went one step further and also analyzed parameterized circuit classes as well. Important circuit classes are paraNC^i, paraAC^i, and paraTC^i, which are the sets of problems solvable by NC-, AC-, or TC-circuits of size $f(\kappa(w)) \cdot \text{poly}(|w|)$ and depth $f(\kappa(w)) + \mathcal{O}\big(\log^i|w|\big)$, or constant depth in case of $i = 0$. The corresponding X-classes are XNC^i, XAC^i, and XTC^i, which are the sets of problems solvable by NC-, AC-, and TC-circuits of size $\text{poly}(|w|)^{f(\kappa(w))}$ and depth $f(\kappa(w)) \cdot \mathcal{O}\big(\log^i|w|\big)$.

They also introduced concepts like simultaneous time and space bounds to the world of parameterized complexity, which does not make sense in classical complexity theory, but becomes interesting if we consider parameterized problems. An important class of this kind is XNL_{FPT}, the set of problems solvable by a XNL-machine in time $f(\kappa(w)) \cdot \text{poly}(|w|)$. Another concept they introduced is *bounded nondeterminism*, which describes deterministic computational models that get a few nondeterministic bits, which we call *choice bits*, as additional input.

The semantic of these choice bits is, with respect to nondeterminism, quite natural: A corresponding computational model accepts if, and only if, there is at least one possible selection of choice bits such that the underlying computation accepts. The most important classes of this kind are the paraW$\mathcal{C}$-classes, which are the problems solvable by a $\mathcal{C}$-machine or circuit that gets additional $f(\kappa(w)) \cdot \mathcal{O}(\log|w|)$ choice bits as input. More precisely: An instance w of a parameterized problem (Q, κ) is accepted by a paraW$\mathcal{C}$-computational model if there is a word $b \in \Sigma^\star$ with a size of $|b| \leq f(\kappa(w)) \cdot \mathcal{O}(\log|w|)$ such that the tuple (w, b) is accepted by a $\mathcal{C}$-machine. These bits can be seen as a few pointers to the input, for example to mark a couple of vertices in a graph problem. The resulting classes are paraWL, paraWNL, paraWNC^i, paraWAC^i, and paraWTC^i.

A second kind of bounded nondeterminism is used in para$\beta\mathcal{C}$-classes, which are the sets of problems solvable by a $\mathcal{C}$-machine or circuit that gets $f(\kappa(w)) \cdot \mathcal{O}(\log|w|)$ read-once choice bits. These read-once choice bits can be seen as a few nondeterministic decisions. The only class that really fits to this definition and that is of interest is paraβL.

For more details about the different classes and their connection consult the paper by Elberfeld, Stockhusen, and Tantau [26]. The figure on the next site summarizes the classes that are important for this work in different colors: paraW-classes, paraβ-classes, and X-classes.



We have now defined all necessary classes. Furthermore, we also have discussed the concepts of *tractability* and *intractability*. Nevertheless, we have not discussed *how* we can classify a problem with respect to these properties. Here we distinguish two cases: The first thing we want to do, is to prove that a problem is tractable, or more generally that is just a member of a particular class. For a language $L \subseteq \Sigma^\star$ and a complexity class $\mathcal{C}$, a proof for $L \in \mathcal{C}$ is quite natural: We just have to provide an algorithm that is computable within the resource bounds of $\mathcal{C}$ and which decides for every $w \in \Sigma^\star$ whether w is a element of L or not. Such results can be seen as *upper bounds* on the problem's complexity. The second objective, the proof that $L \notin \mathcal{C}$, turns out to be much harder in most cases. In order to describe this property, we need the terms of *reduction* and *$\mathcal{C}$-hardness*.

Reductions. A reduction from a problem $L_1 \subseteq \Sigma_1^\star$ to a problem $L_2 \subseteq \Sigma_2^\star$ is, simply said, a way to describe how we can decide if a word w is an element of L_1, by deciding if another word w' is an element of L_2. In other words, a reduction defines an algorithm that solves L_1 and that uses a L_2-solver as subroutine. The complexity theoretic interpretation of such a reduction is, that it can not be more "difficult" to solve L_1 than to solve L_2. If we are able to solve L_2 within some resource bounds, then we are also able to solve L_1 within the same resource bounds with the algorithm defined by the reduction (if the computation of the reduction itself does not need significant more resources). In parameterized complexity theory, the most used reduction is the FPT-reduction, which is defined as follows:

► Definition 4
Let (Q_1, κ_1) and (Q_2, κ_2) be two parameterized problems over the alphabets Σ_1 and Σ_2, respectively. Let furthermore $f : \mathbb{N} \to \mathbb{N}$ and $g : \mathbb{N} \to \mathbb{N}$ be two computable functions. A mapping $R : \Sigma_1^\star \to \Sigma_2^\star$ is called a FPT-reduction from (Q_1, κ_1) to (Q_2, κ_2) if it fulfills the following three properties for all $w \in \Sigma_1^\star$:

- $w \in Q_1$ if, and only if, $R(w) \in Q_2$;
- $R(w)$ is computable by a Turing machine in time $f(\kappa_1(w)) \cdot \text{poly}(|w|)$;
- $k_2(R(w)) \leq g(\kappa_1(w))$. ◄

Since we consider parameterized space and circuit classes that are partly "weaker" than FPT, we also need "weaker" reductions. To obtain such reductions, we use the

definition from above and change the resource bound in which we have to compute $R(w)$. If $R(w)$ is computable by a Turing machine using $f(\kappa_1(w))+\mathcal{O}(\log|w|)$ space, then we call the corresponding reduction a paraL-reduction. If $R(w)$ is computable by a DLOGTIME-uniform family of constant depth AC-circuit with a maximum size of $f(\kappa_1(w))\cdot \text{poly}(|w|)$, then we call the reduction a paraAC°-reduction. The latter is also known as *parameterized first-order reduction* (*pfo-reduction*). We say furthermore that two languages L_1 and L_2 are *equivalent* with respect to a certain reduction, if L_1 reduces to L_2 and L_2 reduces to L_1 with respect to the corresponding reduction.

Hardness and Completeness. Since we are now able to compare the "difficulty" of two problems, a reasonable question is which problems are the "hardest ones" within a complexity class $\mathcal{C}$.

► Definition 5
A language L is called to be *hard* for a complexity class $\mathcal{C}$ (to be *$\mathcal{C}$-hard*) with respect to a certain reduction, if each language $L' \in \mathcal{C}$ reduces to L via such a reduction. ◄

Thus, if $L \subseteq \Sigma^\star$ is $\mathcal{C}$-hard, then we can solve each problem in $\mathcal{C}$ using L. Therefore we obtain the important result:

► Fact 6
Let $\mathcal{C}_1$ and $\mathcal{C}_2$ be two complexity classes with $\mathcal{C}_1 \subseteq \mathcal{C}_2$. For every problem L that is hard for $\mathcal{C}_2$ with respect to a reasonable weak reduction, we have $L \in \mathcal{C}_1$ if, and only if, $\mathcal{C}_1 = \mathcal{C}_2$. ◄

Under the assumption that the complexity hierarchy is not collapsing, we therefore can interpret hardness-results as *lower bounds* for the complexity of a problem. If the lower and upper bounds collide, we define a problem as *complete* for the corresponding complexity class:

► Definition 7
Let $L \subseteq \Sigma^\star$ be a language and $\mathcal{C}$ be a complexity class. We say L is *complete* for $\mathcal{C}$ (or *$\mathcal{C}$-complete*) if, and only if, $L \in \mathcal{C}$ and L is $\mathcal{C}$-hard. ◄

1.2 Contributions of this Work

The crux with the framework from Elberfeld, Stockhusen, and Tantau is that it is still fairly unexplored and, thus, there are only a few problems classified into the different classes and intersections of classes. The objective of this work is to tackle this issue by analyzing certain problems that are well known in the parameterized complexity community with respect to parameterized space and circuit complexity. Roughly summarized, we will analyze problems of the following "types":

Automata intersection problems. We study the well known automata intersection problem, in which the objective is to determine if a set of automata accepts a common word. We will prove that different parameterizations of this problem obtain different new upper bounds in the framework of parameterized space and circuit classes. This approach will also resolve an open problem from Wareham, which was reported in the compendium by Cesati [13]. Moreover, we will collide the new upper bound with

a new lower bound for the standard parameterization of the problem (the number of automata) and, hence, provide a natural complete problem for parameterized space classes.

Vertex ordering problems. Vertex ordering problems are problems of the following kind: Given a graph, can we order the vertices such that the resulting embedding of the graph has some specific property? Such problems are quite useful in different applications like graph drawing or bioinformatics. Unfortunately, they are also intractable and, moreover, most parameterized approaches with respect to the resource time failed. Thus, we will analyze these problems with respect to parameterized space classes and provide some new upper bounds for them. Unfortunately, new lower bounds or completeness results are still missing.

Colored reachability problems. We will study problems that have a very natural structure and that fit well into the setup of parameterized complexity theory. In such colored reachability problems the instance is a graph with colored edges, the objective is to determine if there is a path between two given vertices that only uses edges of a given amount of different colors. We will use such problems to provide new complete problems with a handy structure, which can be used for further reductions. However, in order to do so, we will provide complete problems for many different classes and will find problems in the intersection of these classes as well. Furthermore, we will show that many natural variants of these problems require their own class. This will again indicate that the intersections of complexity classes play an important role in parameterized complexity.

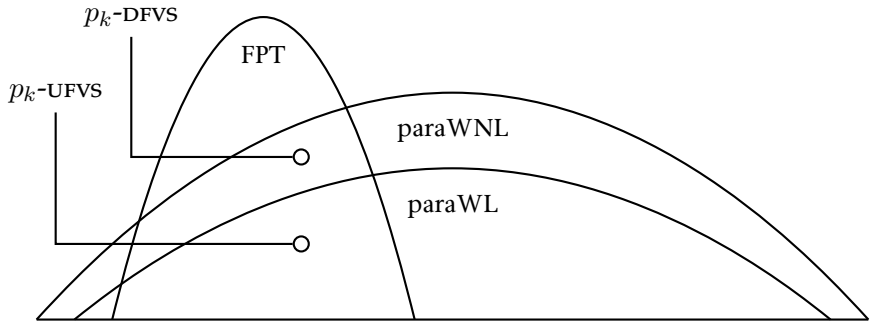
Graph separation problems. The last kind of problems we consider are graph separation problems. Here the task is to cut a given graph into pieces such that these pieces have some specific property. These problems have applications in image and video processing. We will mainly focus on the problem of cutting ℓ vertices away from a graph. This problem is for different parameterizations known to be W[1]-hard. But considered with respect to parameterized space and circuit complexity, we will show that different versions of the problem lie in different complexity classes. Thus, we obtain a more detailed understanding of this problem.

1.3 Related Work

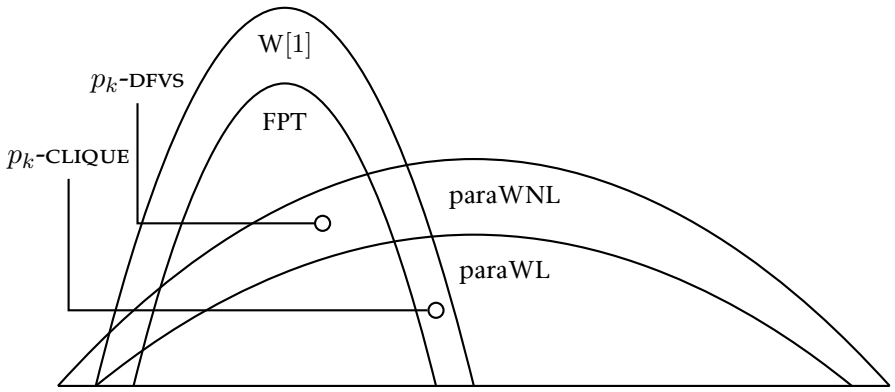
First research in the field of parameterized space classes, especially about parameterized logspace, was made by Cai, Chen, Downey, and Fellows [12]. They introduced the classes paraL and paraNL under different names and showed that several problems from FPT lie in these classes. Later Chen, Flum, and Grohe [14] and Flum and Grohe [34] provided some (quite technical) complete problems for natural parameterized space classes. Finally, Guillemot showed that some natural problems are complete for (quite technical) parameterized space classes [38]. Besides these works, a detailed study of parameterized space classes did, as far as the author knows, not happen until Elberfeld, Stockhusen, and Tantau [26] presented a framework covering a couple of parameterized space and circuit classes. This framework contains natural parameterized space classes as well as natural complete problems for them. However, they did

only study a few problems with respect to this framework and a classification of many problems is still missing.

A first application of the framework that Elberfeld, Stockhusen, and Tantau provided, is the problem of finding a feedback vertex set, i. e., the question whether it is possible to delete k vertices from a given graph in order to make it cycle free. While it was already known that the problem lies in FPT for both, directed and undirected graphs, they showed that the undirected version lies in paraWL while the directed version lies in paraWNL [26]. This fact already shows that the consideration of space provides a more detailed view on the complexity of the problem. These results are especially interesting, because they also connect to classical complexity theory, since the directed version only falls to paraWL if L = NL holds [26]. The following figure illustrates the mentioned parameterized time and space classes as well as both versions of the problem.



The importance of the framework becomes even clearer, if we consider the previously introduced problem of finding cliques. The parameterized clique problem is known to be W[1]-complete, a class which lies above FPT in the parameterized complexity hierarchy. Thus, one would expect a weak reduction, like a paraL-reduction, from the directed feedback vertex set problem to the problem of finding a clique. But parameterized space complexity tells us, that the parameterized clique problem lies in paraWL and this implies that the mentioned reduction only exists if L = NL holds.



Results like these suggest that our understanding of classical complexity theory does

not perfectly mirror into parameterized complexity theory, or at least, that the consideration of parameterized time classes is not enough.

The results also show that parameterized problems can lie in the intersection of different parameterized complexity classes, and we will see during this work, that this is quite often the case. This indicates that such intersections play an important role in the understanding of parameterized complexity. Maybe these intersections are even more important than classical completeness-results, since if a problem lies in the intersection of two complexity classes, this often implies that is unlikely that the problem is complete for one of these classes (unless some classes do collapse).

1.4 Structure of this Work

Each chapter of this work discusses one particular problem or a set of similar problems. Thus, the chapters are self-contained and have not to be read in order. At the end of each chapters, the reader will find a complexity map – a little diagram showing the relationship between the complexity classes considered in the chapter. Such complexity maps were already shown earlier in this introduction. A complexity class is visualized as a parabola in these diagrams. If a parabola is completely inside another one, this means that the corresponding complexity class is a subset (or equal) to the other complexity class. If two parabolas intersect each other, then the corresponding complexity classes intersect each other as well. Problems are visualized as little dots. An empty dot inside a parabola means that the corresponding problem lies in the complexity class represented by the parabola; a filled dot means that the problem is complete for the class. If the dot is inside the intersection of parabolas, this naturally means that the problem lies in the intersection of the corresponding complexity classes.

This work handles problems in the order we already used in the "Contribution of this Work" section: the first problem we consider is the automata intersection problem in Chapter 2. Afterwards, we move forwards to vertex ordering problems in Chapter 3. In the following chapter, Chapter 4, we will consider colored reachability problems, which will somehow represent the core of the work. Here we will study many different complexity classes "around" paraL and paraWL, and provide complete problems for most of them. Finally, we close the thesis with graph separation problems in Chapter 5.

2 AUTOMATA INTERSECTION PROBLEM

Automata theory is at the hearth of computer science. While the computational power of a single deterministic finite automaton is rather limited, the cooperation of many automata becomes quite powerful. Such a cooperation can be modeled as an automata intersection problem, in which we are given a set of automata and have to decide whether there is a common word accepted by all automata or not. Unfortunately, this intersection problem is known to be PSPACE-complete [49]. On the other hand, such problems are well suited for a parameterized approach, since they deliver many natural parameters. For example, the size of the alphabet can often be used as a parameter, since the alphabet is often known to be the binary or the latin alphabet. Also the size of the common word can be used as a parameter in many applications; for example, if the automata should work with a block code, the block length is a natural parameter for this particular problem. The reader can find an introduction to automata theory with respect to the intersection problem in a paper by Saks and Statman [67], an introduction to Turing machines can be found in the textbook from Reischuk [64]. Let us start this chapter with a formal definition of the problem:

► Problem 8 (DFA-INTERSECTION)
Instance: A set of k deterministic finite automata $A_1, \ldots, A_k$, each defined as the tuple $A_i = (\Sigma, Q_i, \delta_i, q_0^i, F_i)$ with $|\Sigma| = s$ and $|Q_i| = q$. Furthermore, a number m is given with $m \in \mathbb{N} \cup \{\infty\}$.
Question: Is there a word $w \in \Sigma^\star$ with $|w| \leq m$ that is accepted by all k automata?
◄

In an equivalent manner we define FDFA-INTERSECTION and NFA-INTERSECTION, the versions in which the automata are acyclic, i. e., the accepted languages are finite, or the automata are not restricted to be deterministic, respectively. Todd Wareham spend a lot of effort to analyze the parameterized version of the automata intersection problem with respect to parameterized time classes [72]. His results are summarized in the table on the right. We will analyze the parameterized automata intersection problem with respect to parameterized space and circuit classes in this

Parameter	Complexity
k	W[t]-hard
m	W[2]-hard
k, m	W[1]-hard
q, k	open
q, m	W[2]-hard
k, m, q	FPT

chapter. This will allow us to resolve the open problem and get a deeper inside into the structure of the problem. Our results are shown in the table on the right side. Thus, we give first completeness results for the parameterized version of the problem and are able to classify different versions of the problem along the complexity hierarchy. By proving that the automata intersection problem parameterized by q, k lies in paraAC°, we also solve an open problem by Wareham, since this result implies membership in FPT as well. We will furthermore adapt the results for the case of nondeterminstic finite automata and other special cases of the problem.

Parameter	Complexity
k	XNL-complete
m	paraWL ∩ XAC°
k, m	paraWL ∩ XAC°
q, k	paraAC°
q, m	paraWNC¹ ∩ XAC°
k, m, q	paraAC°

2.1 Parameterized by the Number of Automata

We first analyze p_k-DFA-INTERSECTION, the version just parameterized by the number k of automata. In classical parameterized complexity theory, this version is known to be W[t]-hard for every $t \in \mathbb{N}$ [13]. The following theorem provides, as far as the author knows, the first completeness result for a version of the parameterized automata intersection problem.

► Theorem 9
The problem p_k-DFA-INTERSECTION is complete for XNL under paraAC°-reduction.

Proof. We first prove membership. A nondeterministic Turing machine can solve the problem by guessing the word w and simulating all the automata in parallel. Since the machine has only a logarithmic amount of space, it will guess one symbol after another and simulate the automata step by step. To guess the symbols, the machine will guess and store $\lceil \log_2 |\Sigma| \rceil$ bits, which represent the symbol. To simulate the automata, the machine needs additional $k \cdot \lceil \log_2 q \rceil$ bits of memory to store the current states of the automata. Thus, the overall amount of needed storage is bounded by $f(k) \cdot \mathcal{O}(\log_2(|\Sigma| + q))$ and therefore the algorithm is computable by a XNL Turing machine.

For hardness, we will reduce from p-BOUNDED-NTM-HALTING, which is known to be XNL-complete [26].

► Problem 10 (p-BOUNDED-NTM-HALTING)
Instance: A nondeterministic Turing machine M and an integer k.
Parameter: k
Question: Does M halt on the empty string using at most k memory cells? ◄

For the reduction let $M = (\Sigma, \Gamma, \Delta, q_S, F)$ and $k \in \mathbb{N}$ be given. We first transform M to an equivalent machine M' with the following properties: The tape alphabet is $\Gamma' = \Gamma \cup \{\xi\}$ for $\xi \notin \Gamma$; the behavior of M' is the same as of M, but M' has only one accepting configuration. In this configuration, the head of the Turing machine has to point to the first cell of the tape, which is the cell to which the head pointed at the

beginning of the computation. Furthermore, each cell used during the computation has to contain ξ in this configuration. We now face the following task: Describe k automata, such that the accepted languages of these automata share a word if, and only if, M' halts on the empty string using at most k memory cells. The nondeterminism of M' will be "simulated" through the question whether or not there is a word that is accepted by all automata. The instructions for the machine will be encoded in a new alphabet used by the automata and the behavior of the machine then will be simulated by the automata itself. First, we will define the alphabet Π on which the automata work. This alphabet consists of the following symbols, representing the natural meaning of Turing machine instructions: $(\sigma_1, \sigma_2, \diamond)$ with $\sigma_1, \sigma_2 \in \Gamma'$ and $\diamond \in \{\triangleright, \triangleleft, \bowtie\}$. Such an instruction defines the following behavior: If the current cell contains σ_1, then replace it with σ_2 and move the head corresponding to $\diamond$ (move right, move left, or do not move).

Let us now describe the automata. Each automaton has $|\Gamma'| \cdot k$ states, which can be arranged in a matrix with $|\Gamma'|$ rows and k columns. We will interpret the states as follows: If the automaton A_i is in a state in column c, the head of the Turing machine points to cell c (note that this does not depend on i, so all automata are always in the same column). The rows correspond to the alphabet Γ', such that if the automaton A_i is in row r, then the ith cell of the memory tape contains the rth symbol of Γ' (for any fixed order of Γ'). We now define the transition relation δ_i for the automata $A_1, \ldots, A_k$. Let q_c^r be the state in the cth column and the rth row. For $c \neq i$, $c \in \{1, \ldots, k-1\}$ we then define $\delta_i(q_c^r, (\sigma_1, \sigma_2, \triangleright)) = q_{c+1}^r$, which means that the automaton just notices that the head of the Turing machine moves to the right, but since the head is not on the ith cell, the content of the tape cell (and therefore the row) does not change. In the same manner we define $\delta_i(q_c^r, (\sigma_1, \sigma_2, \triangleleft)) = q_{c-1}^r$ and $\delta_i(q_c^r, (\sigma_1, \sigma_2, \bowtie)) = q_c^r$. In case of $c = i$, the transitions will only be defined, if the symbol that should be read by the Turing machine corresponds to the symbol the automaton has stored in its state (its row). Since such a transition will possibly change the symbol of the ith cell on the tape, it will also change the row of the automaton, i. e., we define $\delta(q_c^{r_1}, (\sigma_{r_1}, \sigma_{r_2}, \triangleright)) = q_{c+1}^{r_2}$ (assuming σ_{r_1} and σ_{r_2} to be the r_1th and r_2th symbol in Γ', respectively).

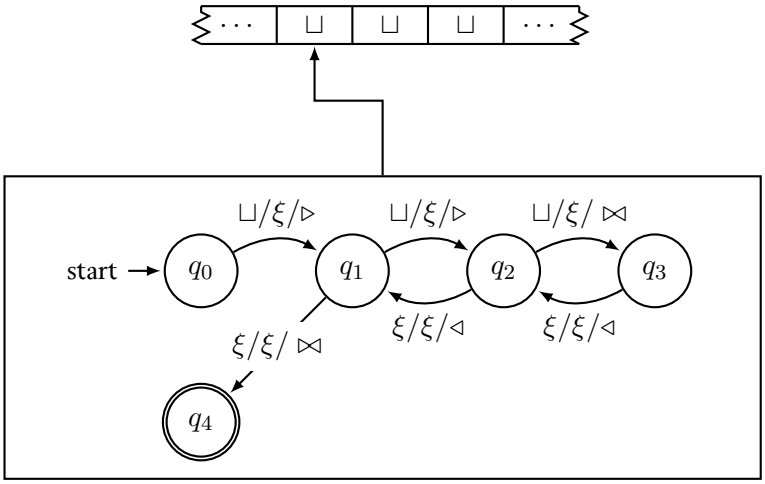
The initial state of all automata is q_0^0, which corresponds to an empty tape and the head is pointing to cell 0, which we will refer to has the first cell. Moreover, the only accepting state is $q_0^{|\Gamma'|}$, which corresponds to a ξ on the tape cell of the automaton and the head of the machine is pointing to the first cell (remember that M' has just one accepting configuration in which the head points to the first cell and in which all used cells contain ξ). Since these k automata only describe the tape content during the computation, we need an extra automaton A_0 that is keeping track of the current state of the Turing machine and makes sure that a given transition is possible at all. This automaton is based on the transition function of the Turing machine: It has one state for every state of the Turing machine and the same initial and accepting states. Furthermore, this automaton has the same state transitions as the Turing machine (where the instructions of the Turing machine are described as symbols over Π). Notice that we can assume that this automaton is deterministic, since the nonde-

terminism of a Turing machine can be fully encoded in the nondeterministic guess of writing a specific symbol to the tape, the successor state of the underlying automaton can be defined deterministically (with respect to symbols of Π).

For correctness consider the case that M indeed halts on the empty string using at most k memory cells. In this case an instruction sequence w exists such that the head of M is never moved before the first or after the kth cell. This sequence is a word accepted by all automata. The automaton corresponding to the transition function of the Turing machine will accept, since the sequence induces a valid computation. Furthermore, since the computation is valid, the Turing machine will never use too many memory cells or do an invalid transition on any cell and, hence, the automata corresponding to the tape cells will accept as well. On the other hand, an instruction sequence that does not induces a valid computation of the Turing machine will lead to at least one rejecting automaton. Either the Turing machine (and therefore A_0) is ordered to do a transition from a state it is not in (and A_0 would reject); or the Turing machine is ordered to read a symbol on the tape that is not there. This would be noticed by one of the automata $A_1, \ldots, A_k$, which then rejects. □

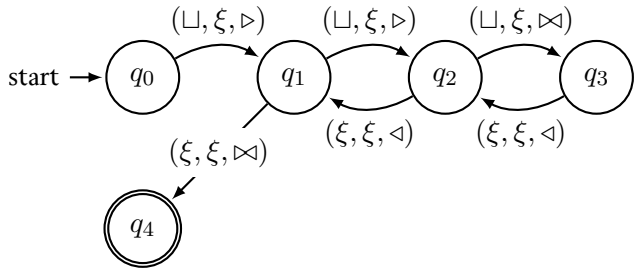
► Example 11

Consider the following Turing machine, which uses $k = 3$ cells to accept the empty string. The machine is already converted to a version that accepts by writing ξ on the tape and moving the head back to the first used cell.

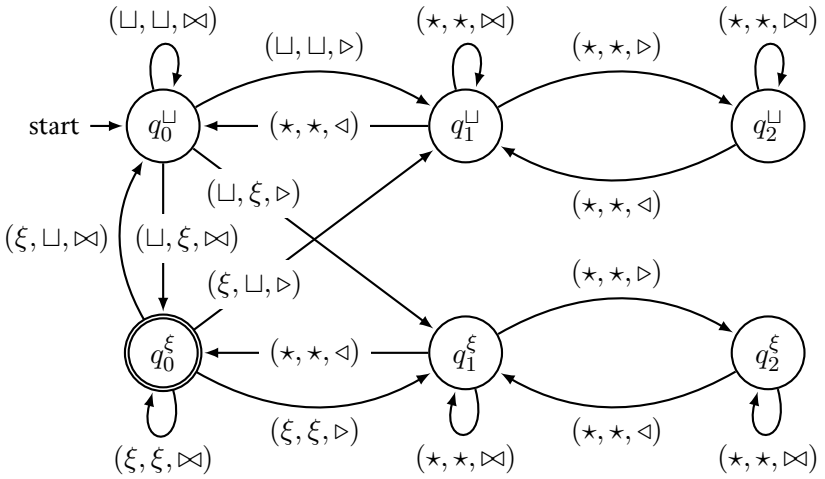


In order to check if the machine accepts the empty string, we can also test if the following four automata all accept a common word over the constructed alphabet $\Pi = \{ (\sigma_1, \sigma_2, \diamond) \mid \sigma_1, \sigma_2 \in \{⊔, \xi\}, \diamond \in \{▷, ◁, ⋈\} \}$. The $\star$ in a transition is just for better readability and is a wildcard for any symbol at the corresponding transition.

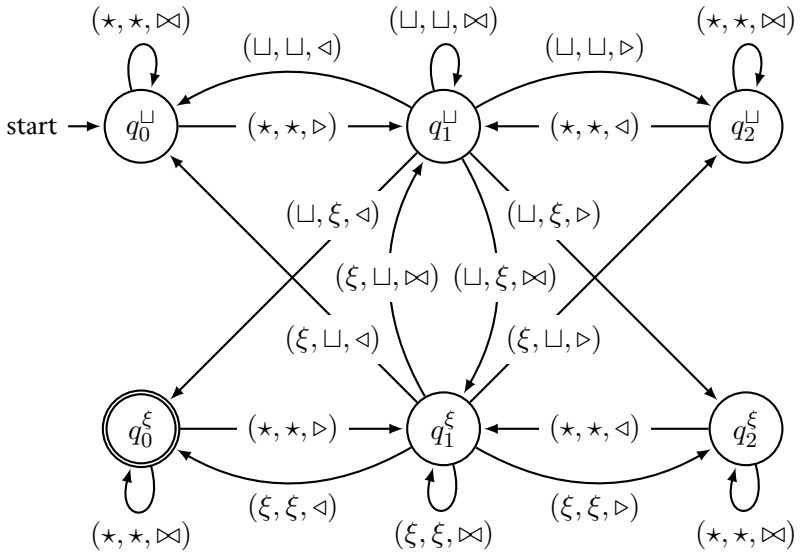
A_0

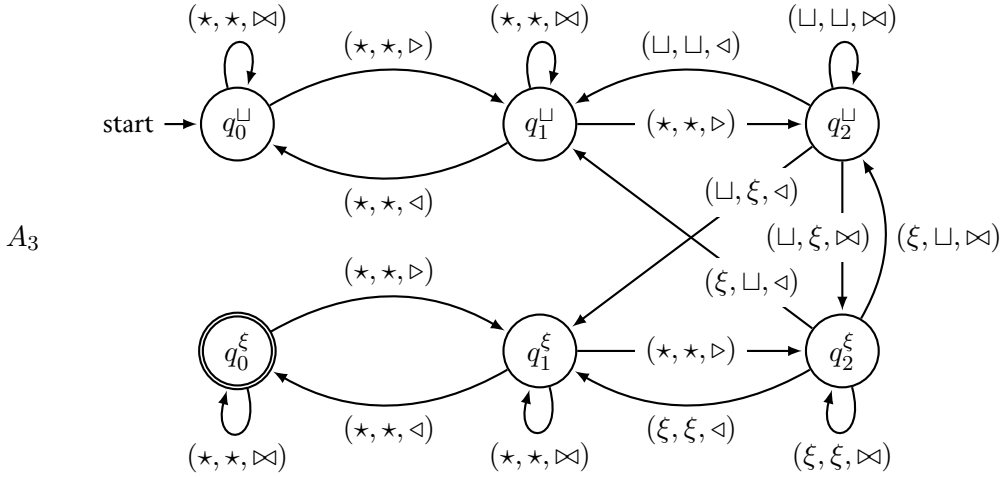

start
q_0
q_1
q_2
q_3
q_4
$(\sqcup, \xi, \triangleright)$
$(\sqcup, \xi, \triangleright)$
$(\sqcup, \xi, \bowtie)$
$(\xi, \xi, \triangleleft)$
$(\xi, \xi, \triangleleft)$
$(\xi, \xi, \bowtie)$

A_1


start
$q_0^{\sqcup}$
$q_1^{\sqcup}$
$q_2^{\sqcup}$
q_0^{ξ}
q_1^{ξ}
q_2^{ξ}
$(\sqcup, \sqcup, \bowtie)$
$(\sqcup, \sqcup, \triangleright)$
$(\star, \star, \bowtie)$
$(\star, \star, \triangleright)$
$(\star, \star, \bowtie)$
$(\star, \star, \triangleleft)$
$(\sqcup, \xi, \triangleright)$
$(\star, \star, \triangleleft)$
$(\xi, \sqcup, \bowtie)$
$(\sqcup, \xi, \bowtie)$
$(\xi, \sqcup, \triangleright)$
$(\star, \star, \triangleright)$
$(\star, \star, \triangleleft)$
$(\xi, \xi, \triangleright)$
$(\star, \star, \triangleleft)$
$(\xi, \xi, \bowtie)$
$(\star, \star, \bowtie)$
$(\star, \star, \bowtie)$

A_2


start
$q_0^{\sqcup}$
$q_1^{\sqcup}$
$q_2^{\sqcup}$
q_0^{ξ}
q_1^{ξ}
q_2^{ξ}
$(\star, \star, \bowtie)$
$(\sqcup, \sqcup, \triangleleft)$
$(\sqcup, \sqcup, \bowtie)$
$(\sqcup, \sqcup, \triangleright)$
$(\star, \star, \bowtie)$
$(\star, \star, \triangleright)$
$(\star, \star, \triangleleft)$
$(\sqcup, \xi, \triangleleft)$
$(\sqcup, \xi, \triangleright)$
$(\xi, \sqcup, \bowtie)$
$(\sqcup, \xi, \bowtie)$
$(\xi, \sqcup, \triangleleft)$
$(\xi, \sqcup, \triangleright)$
$(\star, \star, \triangleright)$
$(\star, \star, \triangleleft)$
$(\xi, \xi, \triangleleft)$
$(\xi, \xi, \triangleright)$
$(\star, \star, \bowtie)$
$(\xi, \xi, \bowtie)$
$(\star, \star, \bowtie)$



A word accepted by all automata is:

$$w = (\sqcup,\xi,\triangleright)(\sqcup,\xi,\triangleright)(\sqcup,\xi,\bowtie)(\xi,\xi,\triangleleft)(\xi,\xi,\triangleleft)(\xi,\xi,\bowtie). \qquad \blacktriangleleft$$

The same argument holds for p_k-NFA-INTERSECTION, since a nondeterministic Turing machine can simulate nondeterministic finite automata with the same time and space bound as a deterministic Turing machine can simulate deterministic finte automata:

▶ Corollary 12
p_k-NFA-INTERSECTION is complete for XNL under paraAC°-reduction. ◀

Often it is not necessary to consider languages with infinite size but only the ones restricted to be finite. For example the set of allowed instructions in a programming language is typically finite. Also block codes are typically finite and there recognition could be done by a finite automata. These examples motivate us to study the problem p_k-FDFA-INTERSECTION, the same problem as above defined with automata which only accept finite languages.

▶ Theorem 13
The problem p_k-FDFA-INTERSECTION is complete for XNL_{FPT} via paraAC°-reduction.

Proof. First of all we can make the following observation: Since the languages are finite, the transition graph of each automaton is acyclic, since a cycle would directly imply that the accepted language has infinite size through the following fact:

Let us consider an automaton with a cyclic transition graph that accepts a word of the form $w = xyz$. Without loss of generality, we can assume that y is the part of the word that is accepted by a cycle in the transition graph. Therefore, the automaton will also accept the word $w' = xyyz$ and, by induction, it will accept infinitely many words.

If the transition graphs of the automata are acyclic, then the length of an accepted word is bounded by q. The machine does therefore need only $\mathcal{O}(q)$ simulation steps per automaton. Hence, we can use the algorithm from the last theorem with the

modification that the Turing machine only guesses q symbols. If the machine has guessed a word of size q and not all automata have accepted, then the machine can reject. It follows that p_k-FDFA-INTERSECTION lies in XNL_{FPT}.

For a proof of hardness, we reduce from a more restricted version of the problem p-BOUNDED-NTM-HALTING.

▻ Problem 14 (p-NTSC)
Instance: Given a nondeterministic Turing machine M and two unary encoded integers t and s.
Parameter: s
Question: Does M halt on the empty string in t steps using at most s cells? ◅

The problem p_s-NTSC is known to be complete for XNL_{FPT} [26] and can be reduced to p_k-FDFA-INTERSECTION by the same reduction used in the last theorem. The only difference this time is that the Turing machine has an explicit bound on its computation time (since t is unary encoded given with the input). This behavior can be translated into automata which accept finite languages, since we do not have to consider any possible computation, but only the ones that use at most t computational steps. Therefore, the word that should be accepted by the automata has at most length t and, thus, the language of the automata is finite. □

2.2 Parameterized by the Length of the Word

In the last section, we have parameterized the automata intersection problem with k, the number of automata. Another reasonable parametrization is the parameter m, the length of the word. This implies that we do not have to check words up to a length of q^k, but only up to a length of m. This version will drop down further in the complexity hierarchy and we will have to deal with circuit classes. We will prove in this section that the problem p_m-DFA-INTERSECTION lies in the intersection of paraWL and XAC°.

In order to do so, we first present a lemma that describes how a family of circuits can simulate an automaton. It is well known that the relation REG $\subseteq$ NC¹ holds [2], but in this scenario the automata are fixed. We describe the more general case in which the automaton is part of the input, i. e., as a decision problem we consider:

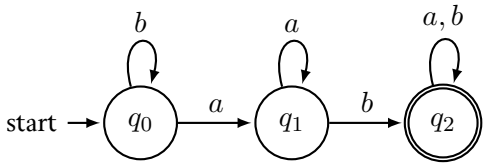
► Problem 15 (DFA-SIMULATION)
Instance: An automaton $A = (\Sigma, Q, \delta, q_0, F)$ with $|Q| = q$ and $|\Sigma| = s$. Furthermore, a word $w \in \Sigma^\star$ with $|w| = m$ is given.
Question: Does A accept w? ◄

The following lemma describes an uniform family of circuits that can solve the problem. We renounce to classify the problem into a certain class, since we will use the lemma later to study parameterized problems with different parameterizations, i. e., selections of q, m, and s. Notice that the constructed family of circuits is a NC¹-family for fixed automata, i. e., for fixed q, and corresponds in this way to the known REG $\subseteq$ NC¹ result.

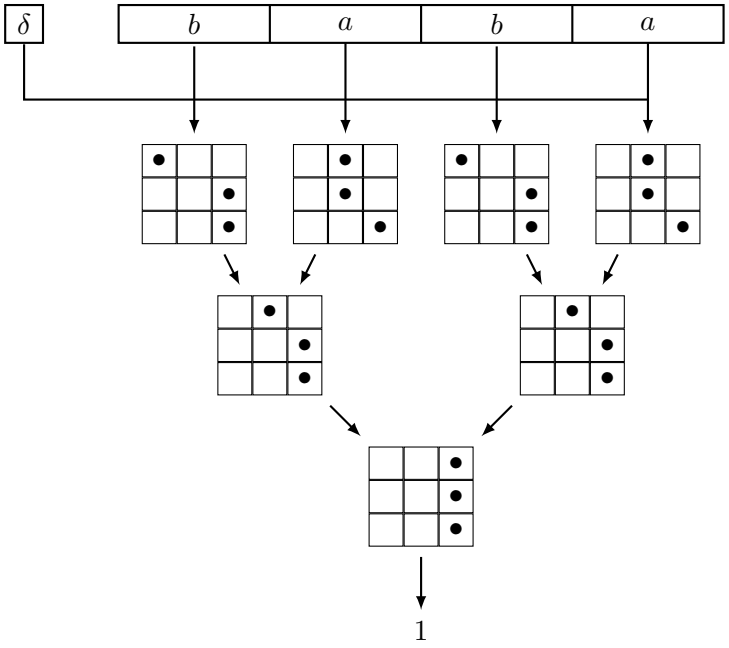
► Lemma 16
The simulation of a deterministic or nondeterministic automaton A with q states on a word w of length m can be done by an AC-circuit of depth $\mathcal{O}(\log m)$ and size $\mathcal{O}\big(q^2 \cdot \log m + |\Sigma|\big)$, i. e., the DFA-SIMULATION-problem can be solved by a corresponding family of circuits.

Proof. Let the input be $w = \sigma_1\sigma_2 \dots \sigma_m$. The circuit first computes for each symbol σ_α in w a $q \times q$ binary matrix M^0_α with $M^0_\alpha[i][j] = 1$ if, and only if, there is a transition from q_i to q_j using σ_α. These matrices can be computed in constant depth using the transition matrix of the automaton. Afterwards the circuit computes in a divide and conquer manner for each two neighboring matrices M^β_α, $M^\beta_{\alpha+1}$ a new matrix $M^{\beta+1}_\alpha$ by multiplying both of them. Such a new matrix describes the reachability of the automaton if it handles both symbols σ_α and $\sigma_{\alpha+1}$ together. Since the matrices are binary, these multiplications can be done by an AC°-circuit: The entry at index (i, j) is the pairwise "and" of the kth symbol in the ith row of the first matrix and the kth symbol of the jth column of the second matrix; the results of all these operations are merged by a big or-gate, i. e., $M^{\beta+1}_\alpha[i][j] = \bigvee_{k=1}^{q}(M^\beta_\alpha[i][k] \wedge M^\beta_{\alpha+1}[k][j])$. The circuit iterates this procedure for $\beta \in \{0, \dots, \lceil \log m \rceil\}$ and $\alpha \in \{0, \dots, m\}$ until only one matrix is left, which then describes the reachability of the automaton reading the whole word. Since in each iteration the number of used matrices is halved, the number of iterations is $\log m$. □

► Example 17
Consider the following automaton over $\Sigma = \{a, b\}$.



To simulate this automaton on the word $w = baba$, a circuit would make the following computation. The binary matrix is showed symbolically, a • indicates a 1, an empty cell a 0.



◄

► Theorem 18
The problem p_m-DFA-INTERSECTION lies in paraWL ∩ XAC°.

Proof. We get membership in paraWL by the following parameterized logspace Turing machine M: On the choice tape the word w is presented to the machine. This is possible since m is the parameter and the size of w is bounded by $m \cdot \lceil \log n \rceil$. Now M can simulate the automata one by one and just has to keep track of the current automaton, its current state, and the current position in w. All of these values can be stored within logarithmic space.

Now we focus on membership in XAC°. Since m is the parameter, the AC-circuit of depth $\mathcal{O}(\log m)$ from Lemma 16 is an XAC°-circuit. Therefore we can provide

a family of XAC°-circuits that can solve the problem by considering every possible input word in parallel (of which there are s^m) and simulate each automaton on each word. We can check if a word is accepted by all automata via a big and-gate. With another big or-gate, we test if this is the case for any of the possible words. Clearly all these operations are computable using XAC°-circuits. □

For p_m-NFA-INTERSECTION, the family of XAC°-circuits remains the same, however, we have to replace paraWL by paraWNL, since we need nondeterminism to guess the transitions of the nondeterministic automata.

► Corollary 19
The problem p_m-NFA-INTERSECTION lies in paraWNL ∩ XAC°. ◄

Since p_m-DFA-INTERSECTION is a member of both, paraWL and XAC°, we should not invest to much hope for a completeness result, since this would imply AC° = L due to the following results.

► Lemma 20
Let $\mathcal{C}_1, \mathcal{C}_2$ be two complexity classes with $\mathcal{C}_1 \supseteq \mathcal{C}_2$; furthermore, let L be a problem that lies in para$W\mathcal{C}_1 \cap X\mathcal{C}_2$. If L is complete for para$W\mathcal{C}_1$ via a reduction that is weaker than para$W\mathcal{C}_1$ and than $X\mathcal{C}_2$, then we have $\mathcal{C}_1 = \mathcal{C}_2$.

Proof. Recall that all problems in $\mathcal{C}_1$ also lie in para$W\mathcal{C}_1$ using a trivial parameterization. By assumption L lies in the intersection of para$W\mathcal{C}_1$ and $X\mathcal{C}_2$ and L is complete for para$W\mathcal{C}_1$ via a sufficiently weak reduction. Thus, we can reduce each problem in $\mathcal{C}_1$ to L and solve this within $X\mathcal{C}_2$. Because of the trivial parameterization the underlying unparameterized problem can be solved within $\mathcal{C}_2$ and, hence, we get $\mathcal{C}_1 = \mathcal{C}_2$. □

► Conjecture 21
There are problems in XAC° that are not in paraWL, i. e., XAC° $\not\subseteq$ paraWL. ◄

We make this conjecture because for a problem instance w with parameterization κ, the size of an XAC°-circuit is bounded by $f(\kappa(w)) + |w|^{f(\kappa(w))}$ for a computable function f. A paraWL-machine on the other hand has only $f(\kappa(w)) + \mathcal{O}(\log |w|)$ space and therefore, the machine can not even iterate over all gates of the circuit; thus it remains unclear how such a machine could be able to simulate an XAC°-circuit.

We have studied the p-DFA-INTERSECTION problem parameterized by the number of automata k and the word length m. One could ask for a version parameterized by both, i. e., $p_{k,m}$-DFA-INTERSECTION. This version is known to be W[1]-complete with respect to parameterized time classes [13], however since m seems to be the stronger parameter, we can conclude the same results as above. Hence, the version is probably not complete for paraWL or XAC° since then these classes would be a subset of W[1]. However, our result classifies the problem more detailed within the set of W[1]-problems.

► Corollary 22
The two automata problems $p_{k,m}$-DFA-INTERSECTION and $p_{k,m}$-NFA-INTERSECTION lie in paraWL ∩ XAC° and paraWNL ∩ XAC°, respectively. ◄

2.3 Parameterized by the Number of States

In addition to either k or m, another interesting parameter is q, the number of states each automaton has. The resulting versions of the problem – p_q-DFA-INTERSECTION and $p_{q,m}$-DFA-INTERSECTION – are both known to be W[2]-hard [13]. While we did not improve the bound for the first version, new results in space complexity can be obtained for the second one.

► Theorem 23
Both problems, $p_{q,m}$-DFA-INTERSECTION and $p_{q,m}$-NFA-INTERSECTION, lie in the intersection of the complexity classes paraWNC^1 and XAC$^\circ$.

Proof. Lemma 16 provides us with an AC-circuit of depth $\mathcal{O}(\log m)$ to simulate an automaton. Recall how this circuit is designed: It consists of $\mathcal{O}(\log m)$ layers and on each layer a couple of $q \times q$ binary matrices are multiplied. Using AC-gates this results in a circuit of depth $\mathcal{O}(\log m)$, however, if we only allow NC-circuits, then the depth of the circuit becomes $\mathcal{O}(\log q \cdot \log m)$. Since both, q and m, are parameters, this circuit is a paraNC^1-circuit, because such a circuit has depth $f(\kappa(w)) + \mathcal{O}(\log |w|)$ on an input w. Notice that this is not a paraAC$^\circ$-circuit, since these are restricted to have constant depth. In some sense, we have to use the higher classes only to allow a depth with respect to the parameter, the logarithmic depth on the input size is thereby not really used.

To solve the problems, a paraWNC^1-circuit gets $m \cdot \lceil \log n \rceil$ choice bits as input. These choice bits corresponds to the word w. Now, the circuit just has to check if all the automata accept this word. This can be done by simulating all automata in parallel, each with the paraNC^1-circuit described above. Afterwards, a paraNC^1-circuit can check whether all the automata have accepted or not.

For membership in XAC$^\circ$, notice that the circuit from Lemma 16 is in this case also a XAC$^\circ$ circuit and that the paraNC^1-circuit – which checks if all automata accept – can easily be replaced by a paraAC$^\circ$-circuit as well. Furthermore, instead of using choice bits for w, a circuit could hard-wire all s^m possible words and check them in parallel with the algorithm from above. An XAC$^\circ$-circuit can do this, since for a fixed m this number is polynomial and, thus, the circuit is huge enough to test all words in parallel. Afterwards, one can test with a big or-gate if any of these words were accepted by all automata. For fixed q and m, this circuit has constant depth and polynomial size, hence it is an XAC$^\circ$-circuit. □

Instead of parameterizing the number of states together with the length of the word, one could parameterize the number of states together with the number k of given automata. Interestingly, this parameterization implies a parameterized bound on more parts of the input. In order to prove this, we need another common term of parameterized complexity, which we have not used yet: *Kernelization.* A kernelization of a problem is another way to describe a parameterized computation; simply spoken is a *kernel* of a problem instance a representation of this instance with a size depending only on the parameter.

► Definition 24
Let (Q, κ) be a parameterized problem.We call a mapping $K : \Sigma^\star \to \Sigma^\star$ a *kernelization* of (Q, κ) if it fulfills the following properties for all $w \in \Sigma^\star$:

- ▷ $w \in Q$ if, and only if, $K(w) \in Q$;
- ▷ $|K(w)| \leq f(\kappa(w))$ for a computable function $f : \Sigma^\star \to \mathbb{N}$.

We call $K(w)$ a *kernel* of w. ◄

Since the size of the kernel is bounded by the parameter, a kernelization implies a parameterized algorithm: First compute the kernel and find a solution for the kernel via "brute force" afterwards. It is well known that this technique implies a FPT, paraL, paraTC°, or paraAC° algorithm if the kernel K can be computed by a Turing machine using polynomial time or logarithmic space, or by a family of TC° or AC° circuits, respectively [26].

► Lemma 25
The two problems $p_{k,q}$-DFA-INTERSECTION and $p_{k,q,m,s}$-DFA-INTERSECTION are complexity theoretic equivalent with respect to an AC°-kernelization.

Proof. We will first describe, why m is bounded by the parameter, afterwards we provide a AC°-kernelization to an instance, where s is bounded by the parameter as well.

In these problems, we are searching for a word accepted by k automata, each with q states. The system of all k automata has at most q^k configurations and, hence, there is a word w of length q^k that forces the system to take each reachable configuration at least once. If a word exists that is accepted by all automata, then there is a configuration of the system in which all automata accept. This accepting configuration can be reached by a prefix $\tilde{w}$ of w in at most q^k steps and, therefore, m is bounded by the parameter.

Let us now describe the AC°-kernelization to an instance where the size s of the alphabet Σ is bounded by the parameter as well. We will call two symbols $\sigma_1, \sigma_2 \in \Sigma$ a *pattern*, if we have $\delta(q_i, \sigma_1) = q_j$ if, and only if, $\delta(q_i, \sigma_2) = q_j$ for any two states q_i, q_j over all automata, i. e., σ_1 and σ_2 always yield to the same transitions. If two symbols constitute a pattern, we can remove one of them together with all its transitions from all automata. This leads to the question: How big can Σ be, before a pattern is forced to exist?

An automaton for $|\Sigma| = 1$ can be viewed as a directed graph and there are $2^{(q^2)}$ of such graphs. For $|\Sigma| > 1$ the automaton can be seen as a graph with labeled multi-edges that arises by overlaying some of the graphs from the case of $|\Sigma| = 1$. This overlay can be described as follows: Take a $|\Sigma| = 1$ graph, label it with the corresponding symbol and put all the transitions to the new graph; then take the next graph and repeat the procedure. The resulting multigraph is the graph representing the automaton for $|\Sigma| > 1$. Hence, for $|\Sigma| > 2^{(q^2)}$ a pattern has to exist in an automaton. Since we have k automata, a pattern has to exist if we have $|\Sigma| > 2^{(q^2)^k}$. That means we can always reduce the size of Σ to at most $2^{(q^2)^k}$ and, therefore, to be bounded by the parameter.

It remains to show that this kernelization can be done via AC°-circuits. The circuit first checks if $|\Sigma| \leq 2^{(q^2)^k}$ holds and, if so, just does nothing. If, on the other hand,

we have $\Sigma > 2^{(q^2)^k}$, then the circuit will check if we have $2^{(q^2)^k} < \log n$. If not, then we get

$$2^{2^{(q^2)^k}} \geq n,$$

i. e., the instance size is bounded by functions of the parameter and, hence, the instance is already a kernel. If we have $2^{(q^2)^k} < \log n$, then the circuit can check in parallel for each pair of symbols if they build a pattern. To do so, the circuit checks for all pairs of states in constant depth, if the two symbols appear together. If two symbols σ_1 and σ_2 constitute a pattern, the circuit will only remove the one with the higher index.

The instance size was not reduced by the circuit until now, actually only a few symbols are marked as remaining and others are not (the ones with higher indices in the patterns). The circuit now has to compactify the instance, i. e., compute the index of the remaining symbols in the new alphabet. This index of a symbol σ can be computed by counting the number of remaining symbols before σ. This counting can be realized by threshold gates: One with threshold 1, one with threshold 2, and so on. These gates are wired to all the symbols with a smaller index and, thus, count how many of them are remaining. The first gate for which the threshold is not reached leads to the index of σ. Since at most $2^{(q^2)^k}$ symbols will remain and, furthermore, since $2^{(q^2)^k} < \log n$, these threshold gates can be simulated with AC°-circuits [60]. Overall we obtain an instance of $p_{k,q}$-DFA-INTERSECTION in which m and s are bounded by the parameter and, thus, an instance that is equivalent to $p_{k,q,m,s}$-DFA-INTERSECTION. □

With Lemma 25 we can provide the following theorem.

► Theorem 26
The two problems $p_{q,k}$-DFA-INTERSECTION and $p_{q,k}$-NFA-INTERSECTION lie in paraAC°.

Proof. From Lemma 25 we know that a parameterization with q and k implies that m and s are bounded by the parameter as well. Therefore a paraAC°-circuit can hard-wire all $\mathcal{O}(s^m)$ possible words and check in parallel, if one of these is accepted by all automata. Let us consider one fixed word and describe how the circuit can check in constant depth if it is accepted by an automaton; since q is a parameter, the circuit can make this test for all automata in parallel. In order to do so, the circuit hard-wires all q^m possible state sequences for the considered word, this is possible since q and m are parameters. Such a sequence $q_{i_1} q_{i_2}, \ldots, q_{i_q}$ describes that the automaton is in state q_{i_1} after reading the first symbol, in state q_{i_2} after reading the second symbol, and so on. We will call a sequence a *good* sequence, if there is a valid computation for the automaton on the word, traversing the states in the order of the sequence and ending in an accepting state. The automaton accepts the word if, and only if, there is a good sequence. Checking if a sequence is good can be done in constant depth AC: The circuit just has to check if there is a transition between adjacent states together with the corresponding symbol in w. Afterwards the circuit checks if the last state is an accepting one, which is just a simple look up on the input. □

2.4 Parameterized by the Languages

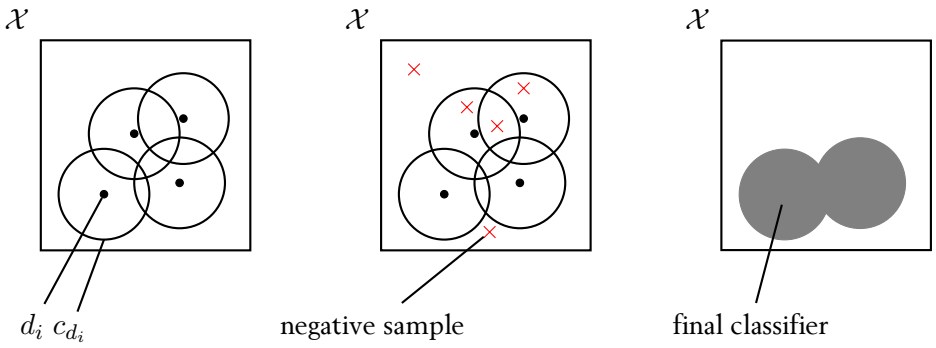
Until now, we have analyzed the general version of the p-DFA-INTERSECTION problem with different parameterizations. However, the problem is in practice often not so general. The languages, for example, have often specific properties in specific applications. In this section, we will analyze such versions for one application. A *negative selection* algorithm is a learning procedure that is trained only by negative examples. The algorithm is designed after the model of the immune system in nature. In the immune system, the organism uses *T cells* to protect itself from *antigens*. A T cell is a special killer cell, which eliminates antigens that have a certain similarity to the surface of the T cell. This similarity of the surfaces lets the T cell connect to the antigen. The organism has to ensure that the T cells, indeed, eliminate only exogenous antigenes and not endogenous ones. To do so, the organism trains the T cells as it matures. This training is done in two steps: The organism starts by producing a huge amount of randomly generated T cells; afterwards, it presents its endogenous antigenes to the T cells and destroys each T cell that bounds to one of them. Thus, after this process only T cells remain that do not attack endogenous antigenes, but rather recognize exogenous antigens. This approach has some nice properties, for example, each organism of a species has its own, unique set of T cells. This makes it hard for exogenous antigens to adapt to all the T cells of a whole species.

In computer science, we can model this process as a learning scenario. Our learning domain is $\mathcal{X} = \Sigma^n$ for some fixed alphabet Σ. Thus, the antigens are just strings of length n over the given alphabet Σ. The considered concept class then is the set $\mathcal{C}_\delta = \{ c_d \subseteq \Sigma^n \mid d \in \Sigma^n \text{ and for each } w \in c_d \text{ we have } \delta(w, d) \leq r \}$ where δ is some predefined similarity measurement and r is some constant. One concept c_d represents all strings that are covered by a *detector* d, which is also just a string.

We can think of this setup as follows: d is a T cell and the encoding string represents the surface of the cell. Then δ is the similarity function of the surface, which describes if another cell would bind to d or not. The concept c_d is the set of all strings w that are similar enough to d, i. e., all antigenes that would be eliminated by the T cell.

Now a negative selection algorithm works as described for nature above. First draw a set $D \subseteq \mathcal{C}_\delta$ of detectors (T cells). Then present a sample set $S \subseteq \Sigma^n$ of negative examples (endogenous antigenes) to the algorithm and remove all the detectors $d \in D$ that cover at least one sample $s \in S$ from D. The resulting set $D' \subseteq D$ is the final classifier. Note, that in difference to a classical learning algorithm – in which we try to find the correct target concept $c \in \mathcal{C}$ – in a negative selection algorithm we identify the concepts $c' \in \mathcal{C}$ that are for sure wrong. The whole process is illustrated in the image on the next page.

A possible application for negative selection algorithms is *spam detection*. Here the learning domain is the set of emails, and in the learning phase a few emails, which are no spam, are presented to the algorithm. Afterwards, the algorithm detects emails that are spam. For more details about applications or the whole field around negative selection algorithms, we refer to the dissertation of Textor [68].



In this work, we are only interested in the application of the p-DFA-INTERSECTION problem for negative selection algorithms. Clearly, a concept $c \in \mathcal{C}$ can be represented by an automaton A_c that accepts all strings of the concept. To construct D' and, later, to classify with D', one has to decide if a word $w \in \Sigma^n$ is accepted by any of the automata represented by D'. Thus, one has to compute the union of these automata, which is through De Morgan the same as computing the intersection of them. Moreover, in applications like spam detection or the simulation of the immune system, it is often the case that the number of automata, the string length, or r is noticeably smaller than the rest of the input. Thus, a parameterized point of view lends itself.

Let us remark at this point, that the languages in our following problem definition do not necessarily have to be represented as automata, indeed, we define the automata just as the corresponding word $d \in \Sigma^m$ which defines the concept. Nevertheless, many practical implementations of negative selection algorithms are usually based on automata [68, 54]. Since this will furthermore fit better to our previous notations, we will continue to speak about automata and define the problem in a corresponding manner:

► Problem 27 (δDFA-INTERSECTION)
Instance: A set of k automata $A_1, \ldots, A_k$, each represented as a string $d_i \in \Sigma^m$ for an alphabet Σ with $|\Sigma| = s$. Each automaton A_i accepts all words w with $\delta(w, d_i) \leq r$.
Question: Is there a word $w \in \Sigma^m$ that is accepted by all automata, i. e., has $\delta(d_i, w) \leq r$ for each A_i? ◄

The last thing that has to be defined is δ. Two frequently used functions are the *hamming distance* and the *contiguous distance* [68, 54, 27]. The hamming distance $\delta_{\mathrm{H}}(w_1, w_2)$ is defined as the number of string positions in which w_1 and w_2 differ, i. e., we define $\delta_{\mathrm{H}}(w_1, w_2) = |\{\, i \mid w_1[i] \neq w_2[i] \,\}|$. The contiguous distance $\delta_{\mathrm{C}}(w_1, w_2)$ measures the similarity of w_1 and w_2 in local regions. Let x be the maximum number of connected string positions in which w_1 and w_2 are equal. We then define the contiguous functions as $\delta_{\mathrm{C}}(w_1, w_2) = 2r - x$, i. e., we will have $\delta_{\mathrm{C}}(w_1, w_2) \leq r$ if there is an index i with $w_1[i] = w_2[i], w_1[i+1] = w_2[i+1], \ldots, w_1[i+r] = w_2[i+r]$.

A first natural parameterization is k, the number of automata. This parameter is motivated by the fact that there are often a lot more strings than detectors. For example in

the immune system, there are plenty of endogenous antigens, but only a few T Cells that have to be tested. In spam detection, we most likely have a lot of mails that should not be marked as spam, and only a few criteria that mark a mail as spam.

▶ Theorem 28
The problem p_k-δ_{C}DFA-INTERSECTION lies in $\mathrm{XNL}_{\mathrm{FPT}}$.

Proof. An $\mathrm{XNL}_{\mathrm{FPT}}$ Turing machine can solve the problem as follows: For each automaton A_i, the Turing machine initializes a counter. Then the machine iterates over the string length and guesses nondeterministically one symbol after another. For each automaton, the machine checks if the guessed symbol equals the symbol of the representation string at the corresponding position; if this is the case, the machine increments the counter for this automaton. If the symbol does not match, the machine resets the corresponding counter to zero. If one of the counters reaches r, the machine will freeze this counter, i. e., will stop to increase or reset this counter. When the machine reaches the last symbol, it simply checks if each counter is r and accepts, if this is the case. The machine needs only $\mathcal{O}(k \cdot \log n)$ storage bits and since k is a parameter, this computation can be done by a $\mathrm{XNL}_{\mathrm{FPT}}$ machine. □

We can modify the algorithm slightly to obtain a similar algorithm for the problem p_k-δ_{H}DFA-INTERSECTION. To do so, we count only the mismatches and do not reset the counters at any point.

▶ Corollary 29
The problem p_k-δ_{H}DFA-INTERSECTION lies in $\mathrm{XNL}_{\mathrm{FPT}}$. ◀

The next parameter that we will consider is m, the length of the words that represent the automata. There are some natural arguments for this parameter as well: In case of the immune system, antigenes have a similar size, but the amount of antigenes can differ quite a lot. In spam detection, the size of different emails compared with the total amount of emails is usually very small. Thus, it makes sense to assume a small m, while the instance size can be huge.

▶ Theorem 30
The problems p_m-δ_{H}DFA-INTERSECTION and p_m-δ_{C}DFA-INTERSECTION both lie in the class paraWAC°.

Proof. Let us first consider p_m-δ_{H}DFA-INTERSECTION. A paraWAC° circuit can solve the problem as follows: On its choice gates, the description of the word w that has hamming distance r to all the strings that corresponds to the automata is presented. The circuit then can test for each string d_i, which represents the automaton A_i, if $d_i[j] \neq w[j]$ holds for all indices $j \in \{1, \ldots, m\}$. Afterwards, the circuit can count with a threshold gate in how many positions the strings differ. If at most r positions differ, then w would be accepted by A_i. The final result of the circuit will be computed by a big and-gate, which checks if all the A_i would accept w. The resulting circuit is a paraWAC°-circuit, since it has constant depth and all the gates, except the threshold gate, are available within AC°. The threshold gate can be simulated by

an AC°-circuit as well, since if n is the instance size and we have $r < \log n$, then the counting is possible with hashing [60]. If $r \geq \log n$, then we have $2^m \geq n$, since $m \geq r$. Hence, the whole instance has only parameterized size and, thus, the counting can be "brute forced."

The same argument also holds for p_m-δ_{C}DFA-INTERSECTION as well. The difference is that we in this case count the matches and not the mismatches. Furthermore, we do not count over the whole string, but only in blocks of size r. But since the number of such blocks is polynomial in the instance size, a paraWAC° circuit can do this in parallel. □

When we talk about strings in biology or in emails, another parameter that comes to mind is s, the size of the alphabet Σ. This is because in biology we generally have 4 DNA-bases and about 20 Amino acids, which both build the corresponding alphabets. In emails the alphabet can often be restricted to the Latin alphabet. Thus, in both cases the alphabet is fixed and much smaller than the input size. If s is the parameter, the algorithms used in the last theorem do not need the choice gates anymore. The reason is, that only s^m different strings exist and, if s and m are both parameters, a circuit can test all these strings in parallel.

► Corollary 31
Both, $p_{m,s}$-δ_{H}DFA-INTERSECTION and $p_{m,s}$-δ_{C}DFA-INTERSECTION, lie in paraAC°. ◄

A new parameter, which could be used in case of p-δDFA-INTERSECTION, is r, the threshold for the distance. This often makes sense, since the similarity between antigenes could be strongly restricted, and in the case of spam detection, marked emails should not be to different from spam emails, in order to prevent the detector to mark important messages as spam. However, r as a parameter for its own does not make to much sense, but becomes interesting in combination with k.

► Theorem 32
The problems $p_{k,r}$-δ_{H}DFA-INTERSECTION and $p_{k,m}$-δ_{C}DFA-INTERSECTION both lie in paraNL.

Proof. Let us first consider $p_{k,r}$-δ_{C}DFA-INTERSECTION. A paraNL Turing machine starts solving the problem by initializing k counters. Afterwards, the machine guesses nondeterministically one symbol after another and checks for each automaton, if the representing string has this symbol at the corresponding index. If this is the case, the machine increases the counter for this automaton; if it is not the case, the machine resets the corresponding counter. If a counter reaches r at any point, the machine will freeze the corresponding counter, i. e., the machine will not increase or reset this counter anymore. If the machine has guessed all symbols, it simply checks if all counters have counted up to r and accepts if this is the case. For the k counters, the machine needs at most $\mathcal{O}(k \cdot \log r)$ bits, but since k and r are both parameters, this can be stored in the $f(\kappa(x)) + \mathcal{O}(\log |x|)$ space of the Turing machine.

The same algorithm also works for $p_{k,r}$-δ_{H}DFA-INTERSECTION; the only difference here is that we count the mismatches and not the matches. Moreover, in this case the Turing machine will not reset the counters at any point. □

In combination with k, a stronger parameter than r would be m. Stronger means in this case, that the parameter m also implies that r is a parameter as well. For the problems $p_{k,m}$-δ_{H}DFA-INTERSECTION and $p_{k,m}$-δ_{C}DFA-INTERSECTION, we can use the same algorithms as in the last theorem. But since m is a parameter, we do not need unlimited nondeterminism and, thus, the problems fall to paraβL. Together with Theorem 30, we then can conclude the following corollary:

▻ Corollary 33
The two problems $p_{k,m}$-δ_{H}DFA-INTERSECTION and $p_{k,m}$-δ_{C}DFA-INTERSECTION both lie in paraβL $\cap$ paraWAC$^\circ$. ◅

2.5 Complexity Map

3 VERTEX ORDERING PROBLEMS

In this chapter we will consider a family of graph layout problems that are called vertex ordering problems and which have received a lot of attention in the parameterized complexity theory recently. In these problems the objective is to find an ordering π of the vertices of a given graph that minimizes some functions, in other words to find a layout with specific properties. The diversity of vertex ordering problems is huge, with BANDWIDTH, CUTWIDTH and IMBALANCE – to name just a few – we have a selection of problems useful in very different applications like graph drawing and protein engineering. All three problems are NP-complete and remain NP-hard even in restricted versions. The BANDWIDTH-problem remains NP-hard when restricted to trees [33], CUTWIDTH remains NP-hard for graphs with a maximum degree of 3 [40], and IMBALANCE remains NP-hard for planar graphs of maximum degree 6 and for 5-regular graphs [55]. Furthermore, these problems are some of the very few natural graph problems for which, parameterized by the treewidth, no FPT-algorithm is known. Considering the natural parameter, the parameterized complexity of these problems remained open for a long time and current research focuses on more structural parameterizations like the size of a minimum vertex-cover or the maximum leaf number of the input graph [33, 31, 20]. The latest results considering the natural parameter and parameterized time classes are quite technical and needed a lot of effort [55, 69]. Nevertheless, we will analyze these problems with the natural parameter and with respect to parameterized space classes as well as with respect to nondeterministic parameterized space classes that are restricted by a time bound. This will allow us to consider the problems from a new perspective and lead to results that split up the complexity of these problems more accurately. The results stand diagonal to the current results that consider parameterized time classes.

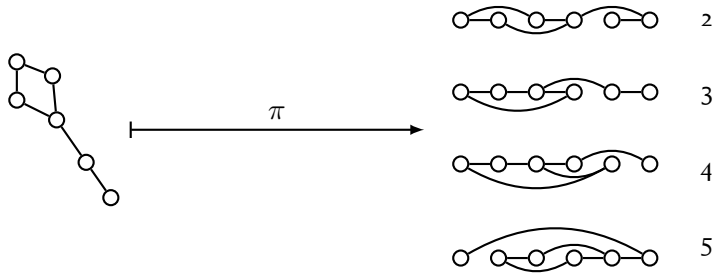
3.1 Bandwidth

We start this chapter with the BANDWIDTH problem, which has a couple of applications in the area of symmetric sparse and band matrices through its similarity to these structures [15, 52]. The objective in the p-BANDWIDTH problem is to place the vertices of a given graph in a line, such that each edge "jumps" over as few vertices as possible. For a given ordering, the *bandwidth* of this ordering is the maximum number of vertices skipped by an edge. The bandwidth $\mathrm{bw}(G)$ of a graph G is the minimum

bandwidth over all possible orderings of the vertices of G. As a decision problem, we define p-BANDWIDTH as follows:

► Problem 34 (p-BANDWIDTH)
Instance: A graph $G = (V, E)$ and a natural number k.
Parameter: k
Question: Is there an ordering π of the vertices of the graph, such that we have $|\pi(u) - \pi(v)| \leq k$ for every edge (u, v)? ◄

► Example 35
Consider the following graph with 6 vertices and four of its possible layouts. Each layout has a different bandwidth, which is shown next to the illustration.



◄

Parameterized by the natural parameter k, the problem p_k-BANDWIDTH is in classical terms known to be W[t]-hard for all t [13]. While this gives strong evidence that there is no FPT-algorithm for the problem, it does not say much about parameterized space classes, since some of them lie diagonal to the parameterized time classes. We will prove that p_k-BANDWIDTH is solvable within XNL_{FPT}, thus we consider time efficient, nondeterministic Turing machines, which are allowed to use $f(k) \cdot \log n$ space.

► Theorem 36
The problem p_k-BANDWIDTH lies in XNL_{FPT}.

Proof. We get membership with the following nondeterministic logspace Turing machine that can accept a slice of the problem. Our first observation is that the order of connected components is not important for π, i. e., we can compute a permutation of the vertices of each connected component and concatenate the results. Since the computation of connected components is possible within a logarithmic space bound, our Turing machine can compute this concatenation and we can safely assume that we only have one connected component; let n be the number of vertices of this connected component. Furthermore, any vertex can have at most $2k$ incident edges or the instance has no solution at all, thus we will assume in the following algorithm that the graph has maximum degree $2k$.

The Turing machine will start by guessing the first $2k$ vertices and their order. It will store all the $2k$ vertices and all edges incident to them (which are at most $4k^2$).

We will call these vertices the current *window* of the layout, observed by the Turing machine. The algorithm will make sure, that the layout before the window is valid and will extend the layout vertex by vertex behind the window.

In each step of the algorithm, the Turing machine checks if the current window is *valid*, for which it performs the following operations:

- ▷ check if no vertex appears twice in the window and reject otherwise;
- ▷ for each edge stored, check if both ends are part of the window and, if so, remove the edge from the storage;
- ▷ check if no edge is stored for the first vertex of the window and reject otherwise.

If the window is valid, the layout up to this point is valid as well and all edges from the first vertex are already considered. Now the machine will move the window by one vertex: It removes the first vertex from the storage and guesses a new one at the end of the window, also it adds all edges that are incident to the new vertex to the storage. Now the Turing machine checks if the new window is valid and, if so, goes ahead by moving the window further. It will repeat this steps until it has placed a total of n vertices. At the end, the last check the machine will perform, is if the last window is valid and that no edge is left in the storage. If this is the case the machine will accept.

For correctness first notice that the machine always produces a layout of n vertices in which the bandwidth is less or equal to k, since it keeps track of the edges. The only thing we have to consider carefully is that the machine will notice a wrong guess, which in this case means that it will not guess the same vertex twice. This is ensured by the fact that the machine can move the window only if all edges of the first vertex are considered (i. e., all the corresponding vertices are placed). If the machine would guess the same vertex v twice, then there is a vertex w not guessed by the Turing machine (since it places exactly n vertices). But since we consider a connected component, w has an adjacent vertex $\tilde{w}$ placed by the Turing machine; furthermore, $\tilde{w}$ could leave the window only if w was placed: A contradiction. □

3.2 Cutwidth

In the CUTWIDTH problem, we are given a graph G and are asked to find an ordering of the vertices such that the maximum number of edges that is intersected by a line inserted between two consecutive vertices is as small as possible. The maximum number of such edges at any point of the layout is called the *cutwidth* of the layout. The cutwidth $\mathrm{cw}(G)$ of the graph G is the minimum cutwidth over all possible orderings of the vertices of G. As a decision problem, we define more formally:

► Problem 37 (p-CUTWIDTH)
Instance: A graph $G = (V, E)$ and a natural number k.
Parameter: k
Question: Is there an ordering π of V, such that for every $i \in \{1, \ldots, |V| - 1\}$ and the following family of functions:

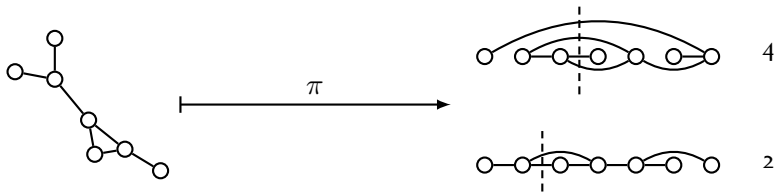
$$f_i(x, y) = \begin{cases} 1 & \text{if } x \leq i \text{ and } y > i, \\ 0 & \text{else;} \end{cases}$$

we have

$$\sum_{(u,v) \in E} f_i\big(\pi(u), \pi(v)\big) \leq k?$$

◄

► Example 38
Consider the following graph on 7 vertices and two of its layouts, one with cutwidth 4 and one with cutwidth 2.



◄

Fellows et al. summarized the wide field of applications of the CUTWIDTH problem in [33], including circuit design [1, 57], protein engineering [4], network reliability [47], graph drawing algorithms [59], information retrieval [8] and even a TSP-algorithm [43]. A fixed-parameter linear-time algorithm for CUTWIDTH was recently presented by Thilikos, Serna, and Bodlaender [69]. However, in this section we will analyze the complexity of p_k-CUTWIDTH with respect to parameterized space complexity: First we present an $\mathrm{XNL}_{\mathrm{FPT}}$-algorithm similar to the one for p_k-BANDWIDTH, afterwards we will use the immersion-characterization of CUTWIDTH to prove XL membership as well. It may seem weird at the first look that we prove membership in $\mathrm{XNL}_{\mathrm{FPT}}$ and XL, since one could imagine that $\mathrm{XNL}_{\mathrm{FPT}}$ is a class above XL, but this is not necessarily the case. As far as we known, $\mathrm{XNL}_{\mathrm{FPT}}$ and XL lie diagonal to each other. The reason

is that although an XNL_{FPT} machine is allowed to use nondeterminism, it is also restricted by a time bound. We can think of XNL_{FPT} as time efficient, nondeterministic XL and since we consider these three resources (time, space, and nondeterminism), the relationship between these classes is unknown.

► Theorem 39
The problem p_k-CUTWIDTH can be decided within XNL_{FPT}.

Proof. We will prove membership with an algorithm similar to the one used in Theorem 36. First notice that it is, like in the case of p_k-BANDWIDTH, sufficient to consider only connected components with a maximum degree of $2k$.

The Turing machine will, like in Theorem 36, start by guessing a window of $2k$ vertices. It will store the vertices of the window and all incident edges. Now it will check if this window is valid, which is slightly different as in the last algorithm:

- ▷ check if no vertex appears twice in the window and reject otherwise;
- ▷ for every edge stored, check if both ends are part of the window and, if so, remove the edge from the storage;
- ▷ check if the cuts inside the window are valid, i. e., count the edges and consider also the still stored edges (which move from the start vertex to a vertex behind the window and thus can be used to compute the cuts).

If the window is valid and the graph has a cutwidth of at most k, then there are at most k edges that lead from the window to a not yet placed vertex, since otherwise their would be a cut greater than k behind the last vertex of the window. Thus, there is a vertex inside the window for which there is no stored edge left. In order to move the window, the Turing machine will remove such a vertex and guess a new one at the end of the window.

The machine will repeat this procedure until it has placed exactly n vertices and it will accept if, and only if, at each step the window is valid and there is no stored edge left at the end. The proof of correctness is similar to the one in Theorem 36. □

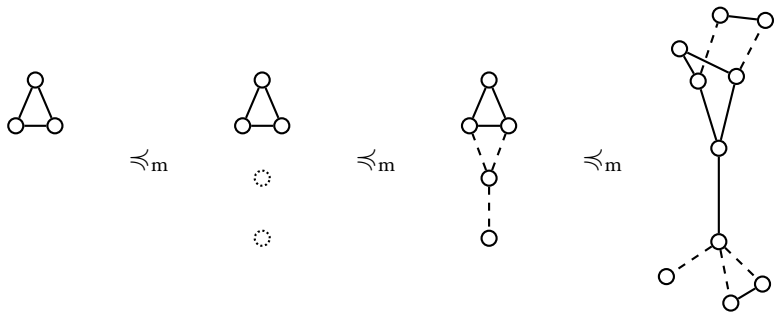
In the p_k-BANDWIDTH-problem, XNL_{FPT}-membership was everything what we were able to achieve. For p_k-CUTWIDTH some more tools are available, which will allow us to consider the class XL as well. Thus, we will drop the nondeterminism and the time bound at the same time. The result is based on a framework of meta-theorems and is *non-constructive*, hence we do not know the algorithm, but we can prove that there is one. In the following, we will first provide a short introduction to the framework of used meta-theorems, afterwards we will conclude the existence of an XL algorithm for p_k-CUTWIDTH.

Minors and Immersions. A special way to look at graph theory is to describe a *partial order* over a family $\mathcal{F}$ of graphs. Of great importance are the *minor relation* $\preccurlyeq_{\text{m}}$ and the *immersion relation* $\preccurlyeq_{\text{i}}$ between two graphs. We will only give a short overview, for more details consult the survey by Lovász [56] or the textbook by Courcelle and Engelfriet [18].

► Definition 40
A graph H is called a *minor* of a graph G, denoted by $H \preccurlyeq_{\mathrm{m}} G$, if an isomorphic graph of H can be obtained from G by applying a finite sequence of the following three operations:

- ▷ removing a vertex;
- ▷ removing an edge;
- ▷ contracting an edge, i. e., removing the edge and merging the two incident vertices. ◄

► Example 41
The leftmost graph is a minor of the rightmost one. We can see this by applying from right to left the following operations: Contract the dashed edges, delete the dashed edges, delete the dotted vertices.



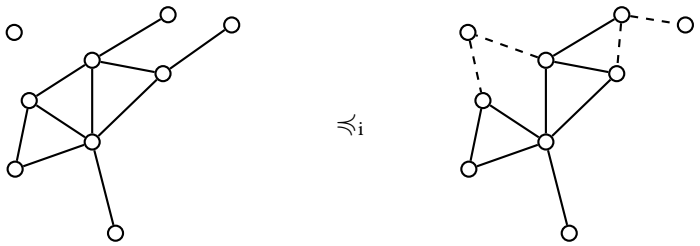
◄

In a similar way the *immersion relation* is defined:

► Definition 42
A graph H is called *immersed* in G, denoted by $H \preccurlyeq_{\mathrm{i}} G$, if an isomorphic graph of H can be obtained from G by applying a finite sequence of the following three operations:

- ▷ removing a vertex;
- ▷ removing an edge;
- ▷ lifting two adjacent edges, i. e., replacing two edges $\{a, b\}$, $\{b, c\}$ by $\{a, c\}$. ◄

► Example 43
The left graph is a subgraph of the graph obtained from the right one by lifting the dashed edges. Thus, the left graph is immersed into the right graph.



◄

The strength of these relations lies in the following theorems, conjectured by Wagner [71] and proved by Robertson and Seymour in a series of over twenty papers [66], which show that these relations build *well-quasi-orderings* for families of graphs. A detailed overview over the concept of well-quasi-ordering can be found in a survey by Kruskal [50]. However, for an understanding of this work a look at the following definition is sufficient.

► Definition 44
A relation $\preccurlyeq$ is a *quasi-ordering* of a set X, if it is reflexive and transitive. The relation is a *well-quasi-ordering* of X if it is a quasi-ordering and if each strictly decreasing sequence is finite and if each set of pairwise incompatible elements is finite as well. ◄

► Theorem 45 (Graph Minor Theorem)
Each family $\mathcal{F}$ of graphs is well-quasi-ordered by the minor relation. □

In other words: Each family $\mathcal{F}$ of graphs contains a finite set of *minimum* graphs with respect to the minor relation. Recently Robertson and Seymour proved that the graph minor theorem still holds if the minor relation is replaced by the immersion relation [66].

► Theorem 46 (Graph Immersion Theorem)
Each family $\mathcal{F}$ of graphs is well-quasi-ordered by the immersion relation. □

To see the enormous impact of these theorems, we need another definition first.

► Definition 47
Let G, H be two graphs and $\mathcal{F}$ be a family of graphs with $G \in \mathcal{F}$. We call $\mathcal{F}$ *minor-closed* (*immersion-closed*) if $H \preccurlyeq_{\mathrm{m}} G$ or $H \preccurlyeq_{\mathrm{i}} G$ implies $H \in \mathcal{F}$. ◄

We are now able to connect the last definition with the previous theorems to obtain a tool that allows us to describe a family of graphs by a finite set of minors or immersions, which we will call the *minor-characterization* or *immersion-characterization* of the family. Let $\mathcal{G}$ be the set of all graphs and $\mathcal{F}$ a minor-closed (immersion-closed) family. From Theorem 45 (46) we know that the family $\mathcal{G} \setminus \mathcal{F}$ has a finite set of minimum minors (immersions), called the *obstruction set* of $\mathcal{F}$. We will denote the obstruction set of a family $\mathcal{F}$ with respect to the minor or immersion relation with $\mathrm{os}_m(\mathcal{F})$ or $\mathrm{os}_i(\mathcal{F})$, respectively.

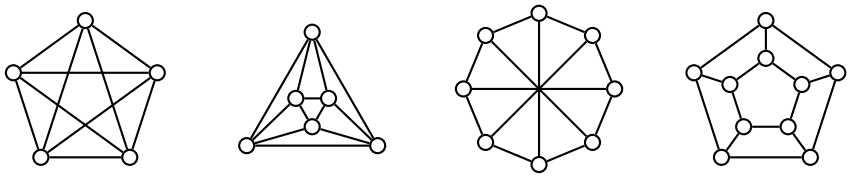
► Corollary 48
For a graph G and a family of graphs $\mathcal{F}$ we have $G \in \mathcal{F}$ if, and only if, there is no graph $H \in \mathrm{os}_m(\mathcal{F})$ or $H \in \mathrm{os}_i(\mathcal{F})$ with $H \preccurlyeq_{\mathrm{m}} G$ or $H \preccurlyeq_{\mathrm{i}} G$, respectively. ◄

Hence, we can describe a minor-closed or immersion-closed family of graphs by its finite obstruction set, i. e., every minor-closed (immersion-closed) family of graphs is well-defined by a finite set of *forbidden minors (immersions)*.

► Example 49
From the definition of the minor relation we immediately get, that the family of *cycle free graphs* is defined by the obstruction set including only the graph  . Wagners Theorem, a restricted version of Theorem 45, tells us that the family of *planar graphs*

is defined by the obstruction set containing the complete graph on 5 vertices and the complete bipartite graph on 4 vertices [71]:

Another example is the family of *series-parallel graphs* with the obstruction set that only contains the complete graph on 4 vertices ⊠ [22]. A little more complex is the family of graphs with *treewidth at most 3*. Such graphs are represented by the following obstruction set [7]:



◀

Furthermore, Robertson and Seymour gave an $\mathcal{O}\big(|V(G)|^3\big)$ algorithm, which decides for a fixed graph H whether $H \preccurlyeq_{\mathrm{m}} G$ holds or not [65]. This leads us to the following corollary:

▶ Corollary 50
Every graph property describable through a minor-closed graph family is decidable in polynomial time. ◀

Unfortunately, these results are *non-constructive*, i. e., we do not know the obstruction set for all families of graphs and, even worse, Fellows and Langston proved that these obstruction sets can not be computed [30]. However, from an algorithmic point of view this may be disappointing, but from the aspect of complexity theory we are satisfied by the pure existence of such an algorithm. And we can do even more: These relations can be described with second order logic, which will allow us to use further meta theorems. For more details about the minor relation and its logical representation consult the textbook of Courcelle and Engelfriet [18]. We refer to the original paper by Robertson and Seymour for more details about the characterization of the immersion relation [66] and to the textbook by Flum and Grohe for an introduction to MSO-logic [35].

Such MSO-characterisations become useful in combination with the following meta theorems and their logspace variants.

Theorems of Bodlaender and Courcelle. The Theorem of Bodlaender provides us with a linear time algorithm to compute the tree decomposition of a graph with bounded treewidth [6]. Thus, the Theorem of Courcelle yields to a linear time algorithm to evaluate monadic second order formulas over graphs with bounded treewidth [17]. Elberfeld, Jacoby, and Tantau showed that in both theorems "linear time" can be replaced by "logarithmic space" [23]. We therefore get the following corollary:

► Corollary 51
Each graph property that is describable by a minor-closed or immersion-closed graph family can be tested in linear time or in logarithmic space on graphs with bounded treewidth. ◄

We are now ready to connect Corollary 51 with the p_k-CUTWIDTH problem. To do so, we need a result from Fellows and Langston, who proved an *immersion-characterization* of CUTWIDTH [32]. Furthermore, Thilikos, Serna, and Bodlaender showed that for each graph G we have $\mathrm{cw}(G) \geq \mathrm{pw}(G) \geq \mathrm{tw}(G)$ [69, 5]. This implies that a "yes"-instance of p_k-CUTWIDTH has bounded treewidth. With these facts taken together with Corollary 51 from above, we can conclude the existence of a (non-uniform) XL-algorithm for p_k-CUTWIDTH.

► Corollary 52
There is a $\mathrm{XL}_{\mathrm{NU}}$-algorithm for p_k-CUTWIDTH. ◄

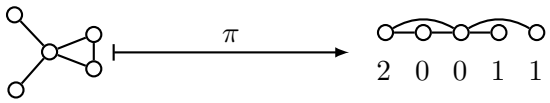
► Corollary 53
The problem p_k-CUTWIDTH lies in $\mathrm{XNL}_{\mathrm{FPT}} \cap \mathrm{XL}_{\mathrm{NU}}$. ◄

3.3 Imbalance

At the end of this chapter, we will consider the very natural problem IMBALANCE, which was introduced by Biedl et al. [3]: Given an undirected graph, our task is to determine a permutation of the vertices such that each vertex is as "balanced" as possible, i. e., has equally many neighbors one the left and right side. For a graph $G = (V, E)$ and an ordering π of V, we denote the "left" and "right neighborhood" of $v \in V$ by $L_\pi(v)$ and $R_\pi(v)$, respectively. Now, we can define the *balance* of a vertex v as $\mathrm{balance}(v) = |L_\pi(v) - R_\pi(v)|$; moreover, the *imbalance* of a graph is defined as $\mathrm{ib}(G) = \sum_{v \in V} \mathrm{balance}(v)$. Finally, the p-IMBALANCE problem can be formulated as follows:

► Problem 54 (p-IMBALANCE)
Instance: A graph $G = (V, E)$ and a natural number k.
Parameter: k
Question: Is there an ordering π of V, such that we have $\mathrm{ib}(G) \leq k$? ◄

► Example 55
Below, a graph and its layout of imbalance 4 are illustrated. The numbers below the vertices represent the balance of each vertex with respect to π.



◄

The p-IMBALANCE problem has a lot of applications in many graph drawing algorithms [44, 45, 61, 73, 74] and has a strong connection to the CLEANING problem [37].

First complexity analysis for IMBALANCE were made by Karà et al. [46]; Lokshtanov, Misra, and Saurabh later presented a FPT-algorithm [55]. Like in the case of BANDWIDTH and CUTWIDTH, we will present an XNL_{FPT} algorithm for p_k-IMBALANCE. For the XNL_{FPT} algorithm we need a result from Lokshtanov et al. [55], which connects the cutwidth and imbalance of a graph:

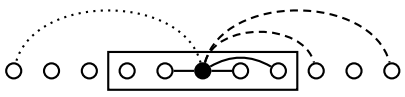
$$\text{tw}(G) \leq \text{pw}(G) \leq \text{cw}(G) \leq \text{ib}(G)/2.$$

A direct consequence is that bounded imbalance implies bounded maximum degree, i. e., a graph with imbalance k can only have a degree of at most k. This is because a graph with cutwidth at most k has a maximum degree of at most $2k$.

▻ Theorem 56
The problem p_k-IMBALANCE lies in XNL_{FPT}.

Proof. With the notes from above, we can reject if the graph has degree greater than k and, thus, we can assume that the graph has degree at most k. Furthermore, with the same argumentation as in Theorem 36 and Theorem 39, we can assume that the graph is connected as well.

On input of a graph G, an XNL_{FPT} Turing machine can guess the first k vertices of the layout and store these vertices together with the corresponding edges (which are at most k^2). As in the Theorems 36 and 39, we will call these vertices the *window*. There are always three kind of edges for the window observed by the Turing machine: Edges inside the window (——), edges with one end in the window that are stored (---), edges with one end in the window that are not stored (·····). With respect to one vertex, the direction (left or right) of edges inside the window is clear, for the other two kinds we define it as follows: Is the edge still stored, then it points behind the window and thus is right of the vertex inside the window; if the edge is not stored, the other end lies before the window and thus, is left of the vertex. The setup is illustrated in the image at the right site; the Turing machine has all information needed to compute balance(●) = 2.



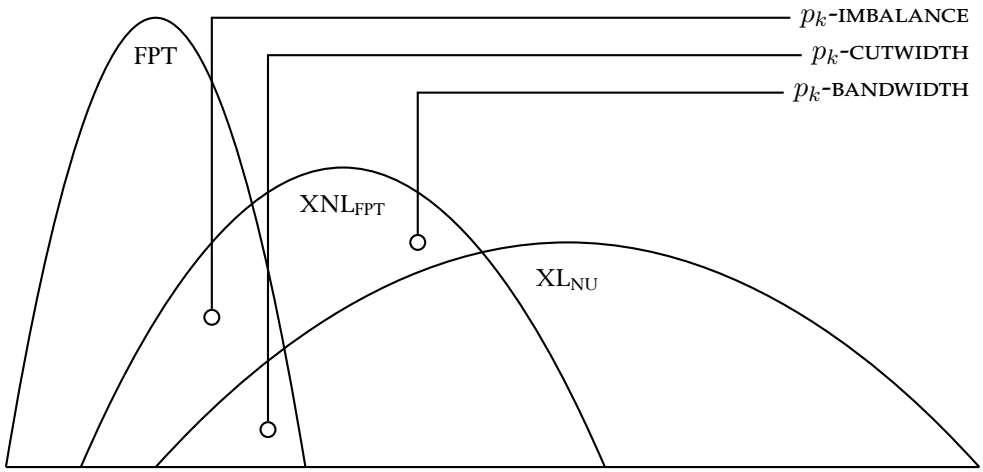
The Turing machine now computes a sequence of *validation* steps. Within each validation step, the Turing machine can compute the balance of each vertex inside the window with the stored information and the graph representation. It will add this balance to a global counter and reject if this counter becomes greater than k at any point. At the end of a validation step, the Turing machine does three things: First it removes all edges with both endpoints inside the window from the storage, then it removes vertices for which we have no stored edges left (these are not needed for any further computations), and it will add one vertex and its corresponding edges nondeterministically to the window. Another validation step is following afterwards.

If the Turing machine has loaded exactly n vertices to the window, has no stored edges left and if the global counter is smaller or equal to k, then the machine will accept; otherwise it will reject. Since the machine can remove vertices from the window only

if all the edges of this vertex are covered, i. e., the corresponding vertices are placed in the window as well, we can be sure that each vertex will be placed exactly once (otherwise there would be an edge left in the storage at the end).

Let us now consider why this is possible within logarithmic space. This is due to the following fact: There are at most k vertices that add more edges to the storage as they remove. This is because such a vertex always automatically increases the imbalance of the graph by at least 1. Thus, the machine will never have to store more than $k^2 + k$ edges and therefore not more than $k^2 + k$ vertices. □

3.4 Complexity Map

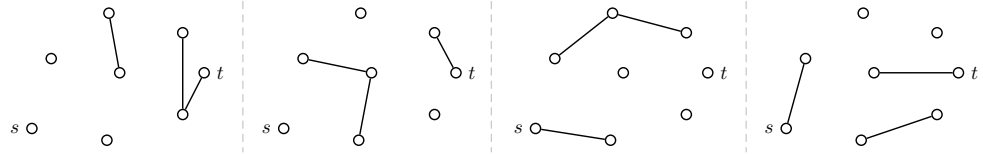


○ Membership

4 COLORED REACHABILITY PROBLEMS

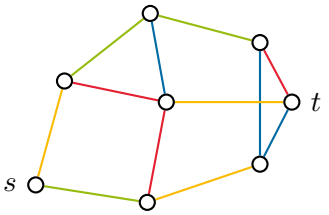
The present chapter will take us on a long journey ranging from secret agent missions to modern energy problems. On this journey we will face known problems and complexity classes, and encounter new complexity classes and concepts. The journey is motivated by a simple observation: There are some natural problems that naturally generalize a graph problem to a parallel version of this problem on multiple graphs. An example for such a problem is the reachability problem in big cities.

Let us say a student wants to travel from his home s to the university t. In the city in which the student lives exists a couple of public traffic transportations, namely two different bus lines by different companies and a tram. Besides that, the city has a couple of bicycle pathways. Consider the vertices of the following graphs as important waypoints in the city, the four described traffic nets (from left to right: Bus 1, Bus 2, Tram, and Biscyle) may look like the ones in the following figure:



Unfortunately, the student has not enough money to buy two bus tickets, a tram ticket, and a new bicycle. His little student salary is only sufficient for two of these items. In order to resolve his problem, the student has to solve the graph reachability problem in multiple graphs at the same time. In other words, he has to ask himself: "Is it possible to overlay two of the above visualized graphs, such that there is an s-t-path in the result?" In this case the student is lucky and there is indeed a solution, can you guess which one?

A way to formalize such problems is to consider a single graph with colored edges. For the example from above we could use the following color scheme: Bus 1, Bus 2, Tram, and Bicycle. The problem then becomes to decide if it is possible to select two colors such that there is an s-t-path in the right graph, using only edges colored by these two colors. This version of the problem is also known as p-COLORED-REACHABILITY and was already studied by Elberfeld, Stockhusen, and



Tantau [26]. Besides its reference to some natural problems, a nice property of this problem is its close relation to the computanional model of a parameterized logspace Turing machine with bounded nondeterminism. We will generalize this problem to a class of problems that we call *color selection problems*, in which the setup is like above, but one must not necessarily solve a reachability problem. A logical consequence of considering color selection problems is to consider *color deletion problems* as well, which we will also do in this chapter. One may think that these two classes of problems are the same, since instead of selecting k colors out of n, one always could simply delete $n - k$ colors and vice versa, but this is not possible in a trivial way if k is the parameter. Thus, the two problem classes are parameterized dual to each other and, hence, both are interesting.

4.1 Color Selection Reachability Problems

We will start this chapter with different *color selection problems*, in which the objective is s-t-reachability. Besides the case of colored edges, which we have mentioned in the introduction, we will later in this section also consider versions with colored vertices. However, we start with a formal definition of the version with colored edges:

► Problem 57 (p-COLORED-EDGE-SELECTION-φ)
Instance: A graph $G = (V, E)$ and a multi-coloring of the edges $c : E \to 2^C$, where 2^C is the power set of the set of possible colors of which there are at most $|V|$ different. Also a natural number k is given.
Parameter: k
Question: Is there a selection of k colors such that the graph obtained from G by removing all edges that have none of the k colors has the graph property φ? ◄

Two interesting variants of the introduced problem are the natural color selection problems p-COLORED$_1$-EDGE-SELECTION-φ and p-COLORED$_{\mathcal{F}}$-EDGE-SELECTION-φ. In the first problem each edge is just colored with only one color as in the example from the introduction; in the second problem we are given k sets of colors instead of n colors and then have to select exactly one color from each set. The setup of the second version can be thought of as a set of color pools – for example a pool of different shades of red and one with different shades of blue and so on – then the task is to select one shade of each color in order to obtain a graph with property φ. Formally we define these versions on the next page.

► Problem 58 (p-COLORED$_1$-EDGE-SELECTION-φ)

Instance: An undirected graph $G = (V, E)$ with an edge coloring $c : E \to C$, where C is a set of at most $|V|$ colors. Also a natural number k is given.

Parameter: k

Question: Is there a selection of k colors such that the subgraph of G that contains only edges of these colors has the graph property φ? ◄

► Problem 59 (p-COLORED$_{\mathcal{F}}$-EDGE-SELECTION-φ)

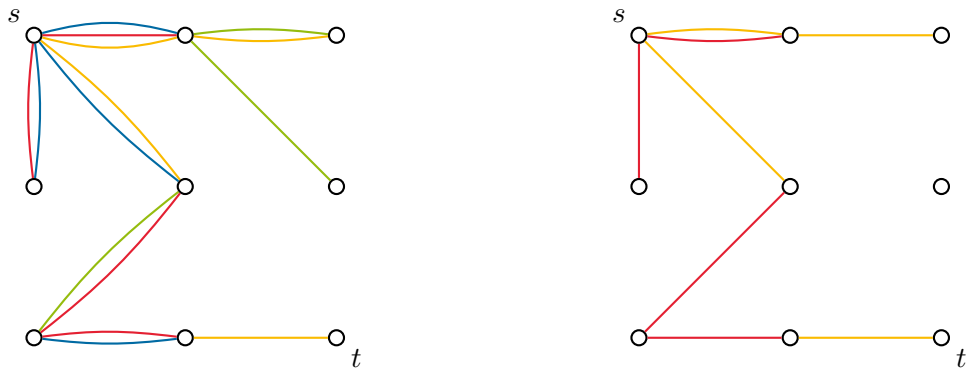
Instance: An undirected graph $G = (V, E)$ with a multi-coloring $c : E \to 2^C$ on the edges (2^C is the power set of the set C of colors). Furthermore, a family $\mathcal{F} = \{C_1, \ldots, C_k\}$ of pairwise disjoint color sets with the property $\bigcup_{i=1}^{k} C_i = C$ is given.

Parameter: k

Question: Is it possible to select exactly one color of each of the k color sets such that the subgraph of G that contains only edges of these colors has the graph property φ? ◄

► Example 60

As example we have a look at the problem considered by Elberfeld, Stockhusen, and Tantau: p_k-COLORED-EDGE-SELECTION-UREACH [25]. In this problem we are given an undirected graph with multi-colored edges and two vertices s and t. Such a graph is shown on the left side in the image below.



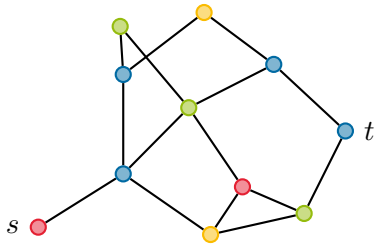
In this graph it is not possible to select one color in order to obtain a path from s to t. However, it is possible to select two colors to do so, namely: Yellow and red. The solution is shown in the right graph. ◄

Elberfeld, Stockhusen, and Tantau showed, that the very natural problems obtained by setting φ to the standard problems REACH, DAG-REACH, CYCLE or to the logspace complete problems UREACH, TREE, UCYCLE are complete for paraWNL or paraWL, respectively [26]. They generalized this concept, such that the resulting versions are complete for any paraW-class, if φ is complete for the underlying class under certain conditions. Moreover, they also showed that in case of REACH and UREACH, these results also hold for p-COLORED$_1$-EDGE-SELECTION-φ and p-COLORED$_{\mathcal{F}}$-EDGE-SELECTION-φ.

If we talk about color selection problems with colored edges, a natural further step is to also consider variants with colored vertices. In this scenario one could think of a company buying railway stations instead of the railway tracks between the stations. A natural example is the problem p-COLORED$_1$-VERTEX-SELECTION-UREACH, which will be very useful in the present chapter.

► Problem 61 (p-COLORED$_1$-VERTEX-SELECTION-UREACH)
Instance: An undirected graph $G = (V, E)$ together with a coloring on the vertices $c : V \to \{1, \ldots, |V|\}$, two marked vertices s, t with $s, t \in V$, and a natural number k.
Parameter: k
Question: Is there a selection of k colors such that in G exists an s-t-path that only uses vertices of these colors? ◄

► Example 62
In the p_k-COLORED$_1$-VERTEX-SELECTION-UREACH-problem, we are given a single colored graph, like the one on the right side. The question then is, if we can select some colors such that we can walk from s to t by just using vertices of the chosen colors. In this graph, it is possible to traverse from s to t by using only ●, ●, and ● vertices. As an alternative, one could also use ● vertices instead of the ● ones. However, it is not possible to use fewer colors, for instance ● and ● vertices are always required, but not enough by itself. ◄



Similar to the edge-selection version, the variant of the problem with colored vertices is complete for paraWL, as we will see in the following theorem.

► Theorem 63
The problem p_k-COLORED$_1$-VERTEX-SELECTION-UREACH is complete for paraWL under paraAC°-reduction.

Proof. For membership consider the following Turing machine, which works in two phases. The first phase is preprocessing, here the Turing machine gets the description of the k colors in form of $\mathcal{O}(k \cdot \log |x|)$ nondeterministic bits on its choice tape. The machine deletes all vertices that are not colored with one of these colors. In the second phase the Turing machine tests s-t-reachability on the remaining graph with the logspace algorithm from Reingold [63].

For completeness we will reduce from p_k-COLORED$_1$-EDGE-SELECTION-UREACH, the version of p_k-COLORED-EDGE-SELECTION-UREACH in which each edge is colored with exactly one color. This version of the problem is still paraWL-complete [26]. For a given graph, we replace each edge ○—○ by a new vertex ● colored by the color of the edge, and two new edges ○—●—○. Moreover, we color all the original vertices with a new color and set the new parameter to $k + 1$. Clearly, the new color has to be selected since otherwise the path could not even include s and t. Each selection of the remaining k colors in the new graph corresponds to the same color selection in

the old graph. Thus, a path in the new graph, visiting only vertices of these colors, corresponds to a path in the old graph, visiting only edges of these colors. □

A natural question from the studies above is what happens if we replace "selection" with "deletion"? For logspace properties, problems of this kind seem to be both, very natural and good candidates to be paraWL complete. Thus, we will investigate such problems in the following section.

4.2 Color Deletion Reachability Problems

There are different possible ways to define a *color deletion problem*. All of them, similar to the *color selection problems*, are based on undirected graphs with a multi-coloring on the edges. The objective in these problems is to choose k colors in order to obtain a graph with some specific properties. The question we face is, whether we delete all edges having one of the k colors or we delete all edges having all the k colors (and maybe more). While the first version may look a little bit more natural, the second variant has a closer relation to the *color selection problems* which we have already considered. We will analyze candidates for both versions in this section. However, we will start with an investigation of the variant in which we only delete edges that are colored by all k colors.

► Problem 64 (p-COLORED-EDGE-DELETION-UREACH)
Instance: An undirected graph $G = (V, E)$ and a multi-coloring $c : E \to 2^C$ of the edges (2^C is the power set of the set C of colors). Also two marked vertices s, t with $s, t \in V$ and a natural number k.
Parameter: k
Question: Is there a set $S \subseteq C$ with $|S| = k$ such that in the graph obtained from G by removing all edges e with $S \subseteq c(e)$ there is a path from s to t? ◄

Note again that we, in contrast to the classical p-COLORED-EDGE-SELECTION-φ problems, delete only edges that are colored with all k colors and not the ones colored with at least one of the colors.

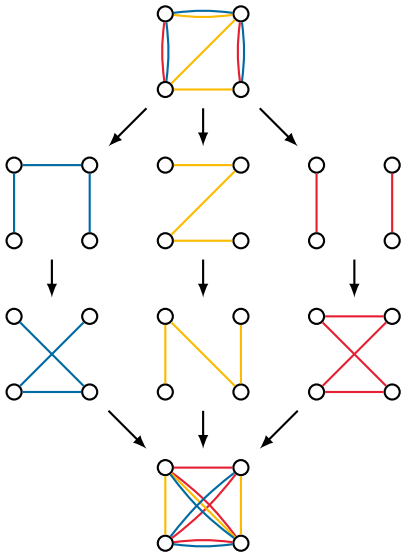
► Example 65
Consider the following path graph s  t. Any selection of two colors from {blue, red, yellow} will remove one of the edges of the path and, thus, will remove the possibility to walk from s to t. However, if we modify the graph to be s  t, then every selection of two colors will still lead to a graph with a path between s and t. ◄

► Theorem 66
The problem p_k-COLORED-EDGE-DELETION-UREACH is paraWL-complete with respect to paraAC°-reduction.

Proof. We obtain membership with a Turing machine that works similar to the one used in the proof of Theorem 63. In the same way as mentioned in this Theorem, the

machine gets the description of k colors on its choice tape. Afterwards, the machine deletes all edges that are colored with at least these k guessed colors. Finally, the Turing machine solves the problem by computing s-t-reachability with Reingold's algorithm [63].

To prove hardness, we reduce from p_k-COLORED-EDGE-SELECTION-UREACH, a problem known to be complete for paraWL [26]. Let therefore $G = (V, E)$, c, k be an instance of p_k-COLORED-EDGE-SELECTION-UREACH. Let G' be the *color complementary graph* of G. The color complementary graph for a multi-colored graph is the graph that we obtain if we split the original graph into multiple graphs (one for each color), build the complementary graphs of these graphs, and merge them back afterwards. See the figure on the right side for an illustration. The obtained graph is the new instance for p_k-COLORED-EDGE-DELETION-UREACH. Let us now consider the case that it is possible to select k colors $c_1, \dots, c_k$ such that in G exists an s-t-path on which each edge has one of these colors. Since each edge of this path has one of the k colors in G, each of the edges misses at least one of these colors in G'. Hence, deleting edges with at least all colors $c_1, \dots, c_k$ in G' results in the same s-t-path as selecting the corresponding edges in G. Let us now consider the case that it is not possible to select k colors in G in order to obtain a path from s to t. Then for every selection of k colors and for every path between s and t exists at least one edge e that does not have one of the k colors. Therefore, e has all k colors in G' and, hence, will be deleted for this selection of k colors. Hence, it is not possible to delete k colors in order to obtain s-t-reachability in G'. □



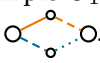
In the p_k-COLORED-EDGE-DELETION-UREACH problem, each edge can be colored by several colors and for the deletion an edge has to be colored by all the selected colors. This may seem rather unnatural and a version where each edge has exactly one color could be more interesting. In such a version the selection of some colors implies a concrete selection of forbidden edges. We will denote this version with p_k-COLORED$_1$-EDGE-DELETION-UREACH.

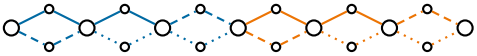
► Problem 67 (p-COLORED$_1$-EDGE-DELETION-UREACH)
Instance: An undirected graph $G = (V, E)$ and a coloring of the edges $c : E \to C$ as well as an integer $k > 0$ and two vertices $s, t \in V$.
Parameter: k
Question: Is there a selection of k colors such that the graph obtained from G by removing all edges having one of these colors has an s-t-path? ◄

This version has not such a close relation to p_k-COLORED-EDGE-SELECTION-UREACH, however, it has a close relation to p_k-COLORED$_{\mathcal{F}}$-EDGE-SELECTION-UREACH.

▶ Theorem 68
The problem p_k-COLORED$_1$-EDGE-DELETION-UREACH is complete for paraWL with respect to paraAC°-reduction.

Proof. Membership works similar as in the proofs of Theorem 63 and Theorem 66. A corresponding Turing machine gets the description of k colors as $\mathcal{O}(k \cdot \log |x|)$ nondeterministic bits on its choice tape. In a first phase the machine deletes all edges that are colored with one of the k guessed colors. Afterwards the machine computes s-t-reachability with the algorithm from Reingold in a second phase [63].

We reduce from p_k-COLORED$_\mathcal{F}$-EDGE-SELECTION-UREACH in order to prove hardness. Let therefore $G = (V, E)$, $\mathcal{F} = \{C_1, \ldots, C_k\}$ be an instance of this problem. We replace each edge $\{u, v\} \in E$ by a couple of paths, more precisely by one path for each color c_i the edge has. Such a path has one edge for each color in the color pool C_i of c_i, except for c_i itself. Thus, the path corresponding to c_i has $|C_i| - 1$ edges and these edges are colored with the colors of $C_i \setminus \{c_i\}$ (each color is used exactly once for a unique edge). If we have for example $C_1 = \{—, \text{--}, \cdots\}$, $C_2 = \{—, \text{--}, \cdots\}$, and the edge  , we would construct  .

A deletion of a color in the new graph corresponds to a selection of the same color in the original graph. Let an edge be colored by c_i in the original graph and, thus, be usable for the s-t-path if c_i is selected. This edge is replaced by a path in the new graph that does not include an edge colored by c_i and, hence, will be usable if c_i is deleted. We now have to enforce that a solution deletes exactly one color of each of the original color pools. In order to do so, we add a new start vertex s' with a path from s' to s. This path has two edges between two neighboring vertices colored with two different colors of the same color pool. In this fashion all combinations of colors of the same color pool are encoded to the path and, thus, the deletion of two colors of the same color pool will destroy the path. Since we are furthermore forced to delete k colors, it follows that a solution will delete exactly one color of each of the original color pools. For the color pools C_1 and C_2 from above the path would be: s'  s.

This reduction can be performed by a paraAC°-circuit since the replacement of a single edge introduces at most k paths of length at most $|C_i| \leq n$ and, thus, at most $k \cdot n$ vertices. Furthermore, for the path from s' to s there are at most $|C_i|^2$ color combinations for each pool and, hence, the length of this path is bounded by $k \cdot n^2$. □

We can adapt these results to versions defined on colored vertices. In these versions membership works in the same way, the only difference is that the corresponding Turing machine does not delete certain edges, but certain vertices. For hardness, we can reduce the versions defined on colored edges to the versions defined on colored vertices using the tools of the proof of Theorem 63. Thus, we get the following corollaries for the problems that are defined in consistent manner to the previous problems.

► Corollary 69
The problem p_k-COLORED-VERTEX-DELETION-UREACH is complete for paraWL with respect to paraAC°-reduction. ◄

► Corollary 70
The problem p_k-COLORED$_1$-VERTEX-DELETION-UREACH is complete for paraWL with respect to paraAC°-reduction. ◄

Finally, let us consider a deletion problem without colors. Such a version is interesting, since it is quite natural to ask if one could reduce the size of a graph while preserving reachability. Consider for example a rail network and the question if it is possible to reduce the number of rail connections while still covering the whole region.

► Problem 71 (p-VERTEX-DELETION-UREACH)
Instance: An undirected graph $G = (V, E)$, two marked vertices $s, t \in V$, and a natural number k.
Parameter: k
Question: Is it possible to delete k vertices from G such that there is still an s-t-path? ◄

Interestingly, in contrast to the problems previously studied, the complexity of this problem drops dramatically in the complexity hierarchy. To prove this, we first need a helpful lemma, which allows the computation of an *iterative depth first search* in certain graphs using a paraL-machine.

► Lemma 72
Given a parameterized graph problem (Q, κ) and an instance $G = (V, E)$ of the problem. If we have $\delta(G) \leq f(\kappa(G))$ and $|\{\, v \mid v \in V \text{ and } \delta(v) > 2 \,\}| \leq f(\kappa(G))$ for a computable function $f : \mathbb{N} \to \mathbb{N}$, then a Turing machine can compute an *iterative depth first search* starting on any vertex $s \in V$ using $f(\kappa(G)) + \mathcal{O}(\log |G|)$ space.

Proof. We will first prove that the Turing machine can compute a *bounded depth first search* for fixed depth d starting by s using its $f(k)$ storage. In order to do so, the machine indexes the high degree vertices with respect to the natural ordering of the vertices in the input. Furthermore, the machine stores for each of these vertices if it was already visited in the current path of the depth first search and, if so, which neighbor was used as next vertex in the path. Since the number of neighbors is bounded by the parameter, this is possible within $f(k)$ space by not storing the description of the corresponding vertex, but only the number the neighbor has, e. g., the 5th neighbor. In this fashion the machine can traverse the complete graph and measure the length of the currently considered path. Whenever the length of the considered path exceeds d, the machine will return in the recursion.

Starting with $d = 1$, the machine can increase this counter and iteratively start a bounded depth first search on vertex s. The machine can repeat this procedure until we have $d = n$, or until a predefined target vertex $t \in V$ was visited. The machine will not need more than $f(\kappa(G)) + \mathcal{O}(\log |G|)$ space for the complete procedure. □

Notice the implied results of a iterative depth first search: We can not only test s-t-reachability, we can furthermore explicitly compute the path between s and t, as well as the distance between s and t.

► Theorem 73
The problem p_k-VERTEX-DELETION-UREACH is complete for the complexity class paraL with respect to paraAC°-reduction.

Proof. To prove membership, we start with a few observations. First notice that if the shortest path between s and t is, well, "short", then we can easily delete some vertices that are not on this path. In particular, if the shortest path between s and t has a length of at most $n - k$, then it is possible to delete k vertices without destroying s-t-connectivity. Thus, a Turing machine solving the problem has to decide if the shortest path between s and t has a length of at most $n - k$.

A further observation is that a graph with shortest path of length more then $n - k$ is sparse. We can think of such a graph as a long chain of vertices of degree 2, and up to $k - 1$ other vertices. Each of these $k - 1$ vertices can be connected to at most 3 vertices in the chain, since otherwise they would allow a "shortcut" and the shortest path would have length of at most $n - k$. Thus, there are at most $4k - 4$ vertices with degree greater than 2.

A Turing machine solving the problem now works as follows: First the machine computes the connected component that contains s. The machine rejects if t is not in this connected component, otherwise the machine deletes all vertices that are not part of the connected component. Let from now on n be the number of vertices in the connected component and k be the number of vertices which still have to be deleted in the connected component.

The machine counts the number of vertices with degree higher than 2 and accepts if there are more than $4k - 4$ such vertices. If the machine has not decided till now, the situation is as follows: There is a path from s to t and the graph is very sparse, in particular at most $4k - 4$ vertices have a high degree. Now the machine computes the maximum degree δ of the graph. A further observation is that for a vertex v at most itself and two neighbors can be part of the shortest s-t-path. Thus, if we have $\delta > k + 2$ the machine can accept, since we can for sure delete k neighbors of any vertex v with $\delta(v) > k + 2$ without destroying s-t-connectivity.

If the machine has still not decided, the situation now is that the number of vertices with degree greater than 2 and the maximum degree itself are both bounded by the parameter. Hence, the Turing machine can compute an iterative depth first search starting by s using its $f(k)$ storage with Lemma 72. Using this iterative depth first search, the machine can compute the distance between s and t and accept if, and only if, the distance is not greater then $n - k$.

Since the slice for $k = 0$ is the standard reachability problem in undirected graphs and, thus, L-complete, we obtain that p_k-VERTEX-DELETION-UREACH is complete for paraL. □

This result is especially interesting since p_k-VERTEX-DELETION-UREACH can be seen as the parameterized dual problem of the parameterized distance problem. In the parameterized distance problem the question is, if the the distance between s and t is less or equal to k. In p_k-VERTEX-DELETION-UREACH the question is, as argued in the proof of Theorem 73, if the shortest path has a length of at most $n - k$. The crucial fact is, that the standard parameterized distance problem is complete for paraβL [25], while we have proven that p_k-VERTEX-DELETION-UREACH is solvable within paraL.

4.3 Vision Based Reachability Problems

```
Sender:     Secret Institute of Computer Science
Recipient:  Agent 042
Status:     Top Secret
Message:    All current missions are revoked with immediate
            effect. Priority subject Beta Three was spot-
            ted in building G in room t. A helicopter takes
            you to the roof, from which you will reach room
            s. From here you have to find your own way to
            room t, but be careful there are multiple war-
            dens guarding different rooms. The wardens use
            a new camera system that allows a single warden
            to guard a couple of rooms. In room s is a port
            to the server of the building, here you can hack
            the system of a warden, which will allow you to
            travel through guarded rooms. But beware, if
            you hack more than k cameras, the general alarm
            goes off. Choose carefully which k wardens you
            will neutralize, the mission fails if you are
            not able to travel unseen from s to t.
            Good luck agent!

            This message destroys itself in 5 seconds.
```

Now, agent, how do you handle this mission? The present section will investigate this question and will further explain what this illustrated scenario has to do with the *color selection* or *color deletion* problems that we have analyzed in the previous section. We will introduce a new family of problems to cover this scenario: *Graph vision problems*. In these problems we are given an undirected graph (the building) with a couple of special vertices (the wardens). The wardens observe their neighborhoods, which could be the set of adjacent vertices or the set of vertices with a given distance to the warden. The objective is to find a path between two given vertices s and t that uses no guarded vertex. The last piece of the problem is the possibility to delete k wardens, in order to obtain such a path (the hack).

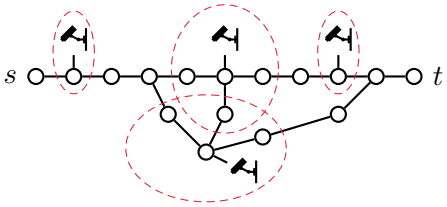
We investigate the resulting problems since they fit very naturally in the complexity class paraWL. The nondeterminism of machines of this class allows us to guess the

wardens that should be deleted; the result is a reachability problem that is solvable by the underlying logspace machine. The first graph vision problem that we consider is p_k-WARDEN-UREACH, which we formally define as follows:

► Problem 74 (WARDEN-UREACH)
Instance: An undirected graph $G = (V, E)$, a set $W \subseteq V$ of *wardens*, a *vision range function* $\nu : W \to \{1, \ldots, r\}$, two vertices s, t with $s, t \in V$ and two natural numbers k, r.
Question: Is it possible to delete k wardens such that there is an s-t-path on which for all vertices v we have $d(v, w) > \nu(w)$ for all remaining wardens w. Here, d is the distance function in graphs. ◄

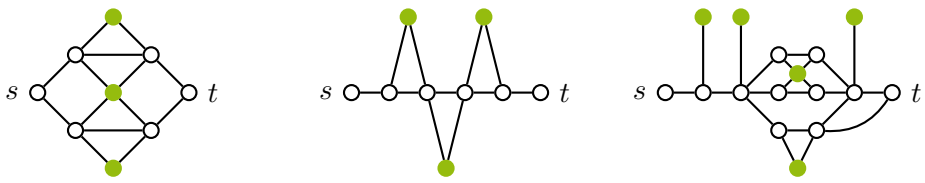
Since this problem has a couple of aspects – s-t-reachability, distance problems to the wardens, and the search through the solution space for the k wardens – we will consider different versions of the problem to analyze it more detailed. Let therefore p-WARDEN-UREACH be the version in which r and k are part of the input, further let p-WARDEN$^{r=1}$-UREACH and p-WARDEN$^{k=0}$-UREACH be the versions in which each warden has vision range 1 or in which we are not allowed to delete wardens, respectively. We will study the problem parameterized by k, r, and $k + r$.

► Example 75
Our spies were able to obtain the following plan from the building. The roof s and the target room t are drawn in. Wardens are visualized as and their vision range is illustrated by a red circle.



One of our spies also found out that the general alarm in this building will go off if more than 2 camera systems get hacked. Fortunately, this is enough! Which camera systems will you hack? ◄

► Example 76
We will generally visualize wardens as ●. The following image shows some more instances of the problem, all with $\nu(●) = 1$:



In the leftmost graph, it is possible to travel unseen from s to t, by deleting just two wardens (the one in the center and either the top or bottom one). In the graph in

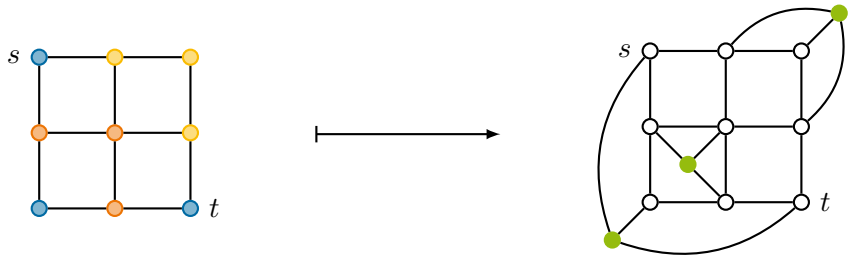
the center of the picture all three wardens have to be deleted in order to open a path between s and t. The rightmost instance can also be solved by deleting three of the four wardens, finding them is left as an exercise to the reader. ◀

We start slowly by investigating p_k-WARDEN$^{r=1}$-UREACH. Remember that this is the version in which all wardens have vision range 1, i. e., all wardens observe only their direct neighborhood and, thus, we have no distance problems. The resulting version is closely related to the color selection problems from the previous section, as the proof of the following theorem is showing.

▶ Theorem 77
The problem p_k-WARDEN$^{r=1}$-UREACH is complete for paraWL via paraAC°-reduction.

Proof. Membership follows by the following Turing machine that gets the description of the k wardens on its choice tape. The machine first deletes the k guessed wardens. Afterwards the machine deletes each vertex that is connected to one of the remaining wardens. Finally, the machine deletes the remaining wardens and computes s-t-reachability on the resulting graph with Reingold's algorithm [63].

To show hardness, we reduce from p_k-COLORED$_1$-VERTEX-SELECTION-UREACH. The central idea is to introduce one warden ● for each color and connect such a warden to all vertices that have the corresponding color. The procedure is visualized in the following image.



Now, every vertex is blocked by a certain warden and selecting a color translates into deleting the corresponding warden, since each vertex is connected to exactly one warden and all vertices of the same color are connected to the same warden. Thus, deleting k wardens is equivalent to selecting k colors and since we are not allowed to walk over wardens or delete other vertices, this fulfills the proof. □

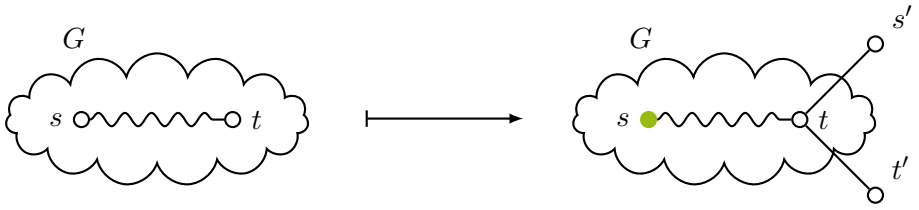
In the p_k-WARDEN$^{r=1}$-UREACH-problem, the vision range of each warden is 1, i. e., wardens only observe their direct neighborhood. A more general version assigns a vision range $\nu(w)$ to each warden w, and the warden then observes all vertices that have a distance of at most $\nu(w)$ to the warden. This version becomes harder to solve, since the underlying problem itself already becomes harder. While the underlying problem in p_k-WARDEN$^{r=1}$-REACH is basically reachability, it is a kind of a distance problem in the version with a vision range function. Since the distance aspect makes the problem harder itself, we will first analyze a version in which we are not allowed to delete any

wardens. We will first consider the problem unparameterized in Theorem 78 to see that the underlying problem already becomes harder. Then we consider the parameterized version in which we are not allowed to delete wardens in Corollary 79, finally we put both aspects together and analyze $p_{k,r}$-WARDEN-UREACH in Theorem 80.

▶ Theorem 78
The problem WARDEN$^{k=0}$-UREACH is complete for NL.

Proof. We will first prove membership by providing a corresponding Turing machine that solves the problem. The machine uses the standard nondeterministic *guess and forget* algorithm for reachability, i. e., it guesses $|V|$ times a successor vertex and does always only store the vertex currently considered. Every time the machine guesses a vertex v, it tests if the distance from v to all wardens w is larger than $\nu(w)$, i. e., if v is usable. The Turing machine rejects if this is not the case. These tests are possible within NL since the undirected distance problem lies in NL [42] and since NL is closed under complement [41]. The machine accepts if it reaches t during this procedure.

Since the undirected distance problem is even NL-complete [42], we get completeness of WARDEN$^{k=0}$-UREACH by the following reduction. Let $G = (V, E)$ with $s, t \in V$ and $d \in \mathbb{N}$ be an instance of the undirected distance problem. We have to translate the question of whether there is a path between s and t of length at most d to the question whether it is possible to walk between two vertices such that each vertex on this walk is unguarded. In order to do so, we add two new vertices s' and t' together with the edges $\{s', t\}$ and $\{t, t'\}$ to the graph. Furthermore, we let s be a warden with $\nu(s) = d$. The construction is illustrated in the following image.



In the new graph exists an unguarded path from s' to t' if, and only if, the distance between s and t is *greater* than d, since otherwise s would observe t and make it unusable for the path. Hence, we have reduced to the complement of the problem, but since NL is closed under complement [41], this fulfills the proof. □

This result directly implies paraNL-membership for the parameterized version of the problem, where the parameter is the vision range r. Moreover, this version also lies in XL, since such a machine can determine for each warden which vertices are covered by the corresponding warden. To determine if a vertex v is covered by a warden w, the XL-machine can compute every path starting by w of length r and check, if v is part of one of these paths. This is possible, since their are at most $|V|^r$ such paths, which can be enumerated within $f(r) \cdot \mathcal{O}(\log |V|)$ space. Afterwards the machine removes

all covered vertices and computes s-t-reachability with Reingold's algorithm [63]. Together, we obtain the following corollary:

► Corollary 79
The problem p_r-WARDEN$^{k=0}$-UREACH lies in paraNL ∩ XL. ◄

This intersection already tells a lot about the complexity of the problem and a completeness result for one of the two classes is unlikely. If the problem is complete for paraNL, then we have paraNL ⊆ XL and, thus, L = NL, since the reachability problem in directed graphs lies in paraNL for the default parameter. Completeness for XL would imply XL ⊆ paraNL, which would not imply such a strong result. But it is unclear how a paraNL-machine should simulate the huge amount of storage that a XL machine has. We will come back to this intersection later in this chapter and discuss it in more detail.

For the most general version, $p_{k,r}$-WARDEN-UREACH, the complexity remains mostly open, our best result is membership in paraWNL ∩ XL.

► Theorem 80
The problem $p_{k,r}$-WARDEN-UREACH lies in paraWNL ∩ XL.

Proof. Let us first prove membership in paraWNL. A corresponding Turing machine solves the problem in two phases: First preprocessing and second reachability. To do so, the machine gets the description of k wardens as $f(k)\cdot\mathcal{O}(\log|V|)$ nondeterministic bits on its choice tape and deletes the corresponding k wardens. Afterwards, the machine computes the distance between each vertex and each warden. Every vertex that is in the range of a warden will be deleted as well. This phase is possible within paraWNL, since the distance problem is solvable within NL [42]. Finally, in the second phase the Turing machine computes s-t-reachability in the resulting graph with a simple *guess and forget* algorithm, which successively guesses $|V|$ vertices that build a path from s to t. The machine accepts if it reaches t and rejects if no such path exists.

Let us now consider membership in XL. The Turing machine follows the same basic concept as the paraNL-machine from above: First find the k wardens, then remove guarded vertices, and finally compute s-t-reachability. The XL-machine has no nondeterminism to guess the k wardens, but it can iterate over all $\binom{n}{k} \leq n^k$ possibilities of deleting k wardens within its $f(k)\cdot\mathcal{O}(\log|V|)$ space. For any such selection, the machine computes which vertices are observed by one of the other wardens, removes them, and tests s-t-reachability in the resulting graph. The reachability-test can be performed by the XL-machine with the logspace algorithm from Reingold [63], the computation of the distance problems can be performed by simulating the corresponding paraβL-machine [26]. □

Better results for the problem remain open, since it is neither clear how a paraNL-machine could determine which wardens should be deleted, nor is clear how a paraWL-machine should solve the huge amount of arising distance problems.

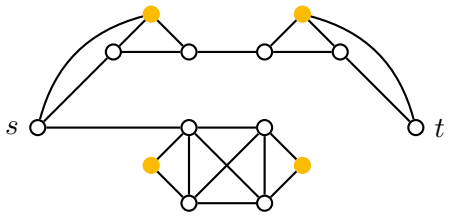
Until now, the wardens were our "enemies" protecting some vertices on our path between s and t. Instead, one could think about applications where the wardens

where our "friends" who make the path between the vertices safe for us. To prevent confusion with the naming, we will consider another scenario to describe this kind of problems: Energy management in modern cites. Consider a city in which every crossing is illuminated by a lantern and several lanterns are connected to the same energy hub. If someone shut down such a hub, all the connected lanterns will turn off. The city wants to save energy and, thus, wants to power off some of the energy hubs. Nevertheless, the mayor wants to ensure that there is still a completely illuminated path between key locations in the city. We can model this setting by the following graph problem.

► Problem 81 (LANTERN-UREACH)
Instance: An undirected graph $G = (V, E)$, a set $L \subseteq V$ of lanterns, a range function $\nu : L \to \{1, \dots, r\}$, some marked vertices $s, t \in V$ and two integers k, r.
Question: Is there a selection of lanterns such that if these lanterns are removed from the graph, there is still an s-t-path in which each vertex is connected to a lantern ℓ by a path of length at most $\nu(\ell)$, but is no lantern itself? ◄

As for the warden problem, we let p-LANTERN-UREACH be the version in which k and r are part of the input, further are p-LANTERN$^{r=1}$-UREACH and p-LANTERN$^{k=0}$-UREACH the versions where the range of each lantern is 1 or where we are not allowed to remove lanterns, respectively.

► Example 82
The graph on the right side consists of normal vertices ○ and lanterns ●. It is possible to delete two lanterns in order to preserve an illuminated path from s to t. This is possible by deleting the lanterns at the bottom. However, in this graph it is not possible to delete more lanterns, because one would be forced to delete one of the lanterns at the top of the graph. These two vertices are both required to illuminate the path between s and t. ◄



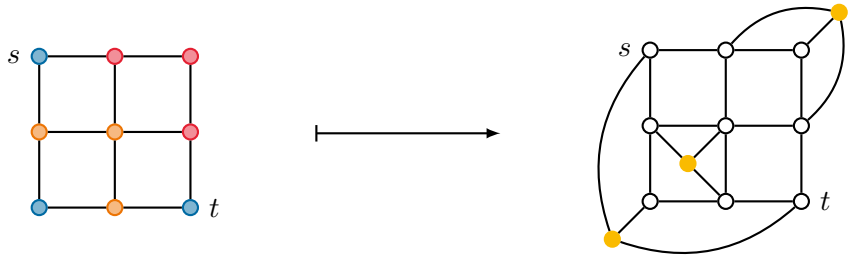
We have already seen that the p_k-WARDEN$^{r=1}$-UREACH-problem is closely related to the color reachability problem p_k-COLORED$_1$-VERTEX-SELECTION-UREACH. In the following we will see that this translates to lantern problems as well, i. e., the problem p_k-LANTERN$^{r=1}$-UREACH is closely related to p_k-COLORED$_1$-VERTEX-DELETION-UREACH.

► Theorem 83
The problem p_k-LANTERN$^{r=1}$-UREACH is complete for paraWL via paraAC°-reduction.

Proof. Membership follows as for p_k-WARDEN$^{r=1}$-UREACH by a Turing machine that works in two phases. The machine gets the description of k lanterns on its choice tape. In the first phase the Turing machine deletes the corresponding lanterns and

all vertices that are not adjacent to another lantern. In the second phase, the Turing machine tests s-t-reachability with Reingold's algorithm [63].

Hardness follows by a reduction from p_k-COLORED$_1$-VERTEX-DELETION-UREACH. Similar to the proof of Theorem 77, we introduced one lantern for each color. Then each lantern is connected to each vertex of this color, as illustrated in the following image:



Thus, if we delete one lantern, all the adjacent vertices are not usable for a path from s to t, since each vertex in the original graph has only one color and hence, is only adjacent to one lantern. Therefore, deleting a lantern is equivalent to deleting a color and since we have to delete k lanterns, this is equivalent to delete k colors. □

Notice similarity of Theorem 77 and Theorem 83, but notice moreover the difference in detail. In Theorem 77 we reduced from a *color selection* problem, in Theorem 83 from a *color deletion* problem. Also the problems p_k-WARDEN$^{r=1}$-UREACH and p_k-LANTERN$^{r=1}$-UREACH may seem quite similar, it remains unclear how a direct reduction between the problems could look like, since such a reduction is not allowed to set the parameter to something like $n - k$. We remark at this point, that the two problems seem to be parameterized dual to each other.

The result of Theorem 78, where we have proved that WARDEN$^{k=0}$-UREACH is complete for NL, directly transfers to LANTERN$^{k=0}$-UREACH. The only difference is, that we do not have to use the property that NL is closed under complement. These observations lead to the following corollary:

► Corollary 84
The problem LANTERN$^{k=0}$-UREACH is NL-complete. ◄

Furthermore, with the same idea we used for Corollary 79, we also get the following corollary for p_r-LANTERN$^{k=0}$-UREACH:

► Corollary 85
The problem p_r-LANTERN$^{k=0}$-UREACH lies in paraNL ∩ XL. ◄

And finally, the results of Theorem 80 also apply and we obtain:

► Corollary 86
The problem $p_{k,r}$-LANTERN-UREACH lies in paraWNL ∩ XL. ◄

4.4 Combined Vision Problems

So far, we have considered vision-based reachability problems in which our task was to find a path from a vertex s to another one t while avoiding wardens or while staying in the range of lanterns. One could imagine these versions as one in which the marked vertices close paths (wardens) and one in which they open paths (lanterns); equally the marked vertices are in some sense "good" or "bad" for the objective. A natural generalization of this scenario is a problem in which we have both, wardens and lanterns. Such a version is especially interesting since we were not able to prove completeness results for versions in which we have only wardens or lanterns and, hence, a combined version could deliver more insights. We have seen in Corollary 79 that p_r-WARDEN$^{k=0}$-UREACH lies in the intersection of paraNL and XL, and in an analog manner we have seen in Theorem 80 that the problem $p_{k,r}$-WARDEN-UREACH lies in the intersection of paraWNL and XL. Furthermore, we obtained the same results for these versions defined with lanterns instead of wardens. We will show in the present section that these problems are, in some sense, complete for these intersections. To achieve this, we will introduce new complexity classes that cover the intersections of XL with paraNL and paraWNL, respectively. But let us start by first introducing the combined version with wardens and lanterns.

► Problem 87 (WARDEN-LANTERN-UREACH)

Instance: An undirected graph $G = (V, E)$ with a set of wardens $W \subseteq V$ and a set of lanterns $L \subseteq V$ with $W \cap L = \emptyset$. Furthermore, a vision range function $\nu : (W \cup L) \to \{1, \ldots, r\}$ is defined and two vertices s, t with $s, t \in V$ and two natural numbers k, r are given.

Question: Is it possible to delete k wardens or lanterns such that there is a path P from s to t in $G' = (V \setminus (W \cup L), E)$ in which we have for each $v \in P$ a lantern $\ell \in L$ with $d(v, \ell) \leq \nu(\ell)$, but no warden $w \in W$ with $d(v, w) \leq \nu(w)$? Here, d is the distance function in graphs. ◄

As before, we denote with p_r-WARDEN$^{k=0}$-LANTERN$^{k=0}$-UREACH the version in which r is the parameter and in which k is constant zero and not part of the input, i. e., the version without the possibility of deleting wardens and lanterns. As announced, we need to define new complexity classes in order to analyze the complexity of the problems from above. For the first new class, we will use a paraL Turing machine together with a paraβL oracle. Let us therefore briefly recall the concept of oracle Turing machines:

► Definition 88

An *oracle Turing machine* is a standard Turing machine that is defined for an oracle language $X \subseteq \Sigma^\star$, has an additional write-only question tape, and has three additional special states: ⓠ, ⓨ, ⓝ.

The computation of an oracle Turing machine on input of a word $w \in \Sigma^\star$ works as follows:

- ▷ the machine does the normal computation of the underlying Turing machine;
- ▷ during the computation, the machine can write a word x to the question tape;

- ▷ let x be the content of the question tape, if the Turing machine enters state ?, it will be transferred to ⊤ or ⊥ in one time step, depending on whether $x \in X$ holds or not, respectively;
- ▷ whenever the Turing machine visits ?, the content of the question tape will be erased after $x \in X$ was evaluated;
- ▷ whenever the machine visits ?, the content of all other tapes as well as all head positions of the Turing machine are not modified. ◁

Let M be a Turing machine and $X \subseteq \Sigma^\star$ be a language, we denote with M^X the oracle Turing machine that is based on M and has X as oracle. We generalize this notation for a complexity class $\mathcal{C}$, i. e., we denote with $\mathcal{C}^X$ a $\mathcal{C}$-Turing machine that is extended by an X oracle. Moreover, if X is complete for a complexity class $\mathcal{C}'$ with respect to a reasonable weak reduction, then we can also write $\mathcal{C}^{\mathcal{C}'}$, i. e., we can use any language of $\mathcal{C}'$ as oracle language.

We are now ready to define paraL$^{\text{para}\beta\text{L}}$, i. e., we set $\mathcal{C} = \text{paraL}$ and $\mathcal{C}' = \text{para}\beta\text{L}$. However, if we consider parameterized oracle Turing machines we have to consider some details about the oracle tape:

► Definition 89
The complexity class paraL$^{\text{para}\beta\text{L}}$ is the set of all languages that can be recognized by a paraL Turing machine using a paraβL oracle.

More precisely, a paraL$^{\text{para}\beta\text{L}}$ Turing machine that decides a parameterized problem (Q_1, κ_1) using the oracle language $(Q_2, \kappa_2) \in \text{para}\beta\text{L}$ has the following properties for two computable functions $f : \mathbb{N} \to \mathbb{N}$ and $g : \mathbb{N} \to \mathbb{N}$:

- ▷ it uses at most $f(\kappa_1(w)) + \mathcal{O}(\log |w|)$ space;
- ▷ for each $x \in \Sigma^\star$ that is written to the question tape during the computation, we have $\kappa_2(x) \leq g(\kappa_1(w))$. ◁

Since the distance problem parameterized by the distance is complete for paraβL, we can conclude:

$$\text{paraL}^{\text{para}\beta\text{L}} = \text{paraL}^{p_d\text{-DISTANCE}}.$$

We will first analyze some basic properties of this complexity class. The idea behind paraβL is to provide a machine with just a little nondeterminism. More precisely, the number of nondeterministic steps that the machine can do is bounded by the parameter. However, a machine with a paraβL oracle has a lot more nondeterminism, since the oracle can be called multiple times. But these nondeterministic calls are in some sense independent: The machine cannot use more than a parameterized amount of nondeterministic steps at once, i. e., it only can compute a bunch of small, independent nondeterministic computations, and the rest of the program is deterministic.

► Lemma 90
The class paraL$^{\text{para}\beta\text{L}}$ is a subset of paraNL ∩ XL.

Proof. We will first show that a paraL$^{\text{para}\beta\text{L}}$ machine M can be simulated by an XL machine M_{XL}. The XL machine can simulate the underlying paraL machine M_{paraL}.

Whenever M_{paraL} makes a call to the oracle, M_{XL} can simulate the corresponding paraβL machine $M_{\text{para}\beta\text{L}}$ and provide M_{paraL} with the result. This is possible since the parameter of the oracle-call is bounded by the parameter of the input instance and, thus, M_{XL} can iterate over all possible contents of the choice tape of $M_{\text{para}\beta\text{L}}$ within its $f(\kappa(x)) \cdot \mathcal{O}(\log|x|)$ space.

To prove that a paraNL machine M_{paraNL} exists that simulates M, we prove that M_{paraNL} can even simulate a paraL$^{\text{paraNL}}$ machine. Again, M_{paraNL} can easily simulate the underlying paraL Turing machine M_{paraL}. Whenever M_{paraL} makes a call to the oracle, M_{paraNL} has to simulate the corresponding paraNL machine. This would be obvious for a single call, but since the oracle is called multiple times, and since M_{paraNL} has to provide the result of the oracle computation to M_{paraL}, it is not.

Let us therefore precisely describe how M_{paraNL} can simulate a paraL$^{\text{paraNL}}$ machine, therefore we describe how M_{paraNL} can simulate a paraL^{X} machine for a fixed oracle language $X \subseteq \Sigma^\star$ with $X \in$ paraNL. Since $X \in$ paraNL and since NL, and thus also paraNL, is closed under complement [41], there are two paraNL Turing machines M_X and $M_{\bar{X}}$ that accept X and the complement of X, respectively. The M_{paraNL} machine can easily simulate the paraL^{X} machine until it reaches a ?-state. When the oracle is called, the M_{paraNL} machine simulates M_X and traverses to ⊤ if M_X accepts the content x of the question tape. This can safely be done, since M_X can only accept if $x \in X$ holds. However, if M_X does not accept, or more precisely does not reach an accepting configuration, then we can not conclude that $x \notin X$ holds, since the machine could just have made the wrong guesses. Thus, if M_X does not accept, the M_{paraNL} machine simulates $M_{\bar{X}}$ on input x. This machine can only accept if $x \notin X$ holds and, hence, if $M_{\bar{X}}$ accepts x then M_{paraNL} can safely traverse to ⊥. If $M_{\bar{X}}$ does not accept, i. e., both machines have rejected, then the nondeterministic guesses of M_{paraNL} were for sure wrong and, thus, M_{paraNL} rejects.

The last problem we face is that M_{paraNL} has no oracle-tape and that the word x that is passed to M_X or $M_{\bar{X}}$ could have polynomial size, i. e., M_{paraNL} can not write x to its work tape and then start the simulation of M_X or $M_{\bar{X}}$. Instead, we have to use a *virtual tape*. Therefore, we modify M_X and $M_{\bar{X}}$ such that these machines "ask" the simulating machine for the ith bit of x, i. e., for instance M_{paraNL} simulates M_X and provides it only with the first bit of x. After some simulation steps, M_X needs the second bit and asks M_{paraNL} for it. This procedure can be repeated until M_X has finished its computation. Since the parameter of x is furthermore bounded by the parameter of the whole instance, we can conclude that the space of M_{paraNL} is sufficient to simulate M_X and $M_{\bar{X}}$. Hence, M_{paraNL} is able to simulate a paraL$^{\text{paraNL}}$ machine and, thus, is also able to simulate M. Together we can conclude:

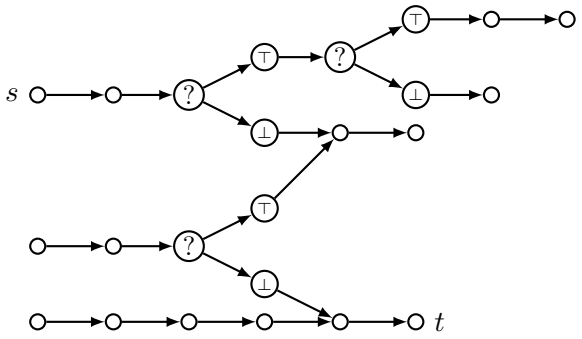
$$\text{paraL}^{\text{para}\beta\text{L}} \subseteq \text{paraNL} \cap \text{XL}. \qquad \square$$

In order to fill our new complexity class paraL$^{\text{para}\beta\text{L}}$ with life, we introduce a first complete problem: p_r-WARDEN$^{k=0}$-LANTERN$^{k=0}$-UREACH. The following theorem will show that the problem fits in a natural way to this class and, thus, gives evidence that paraL$^{\text{para}\beta\text{L}}$ is the right way to look at problems of this kind.

▶ Theorem 91
The problem p_r-WARDEN$^{k=0}$-LANTERN$^{k=0}$-UREACH is complete for paraL$^{\text{para}\beta\text{L}}$ with respect to paraL-reduction.

Proof. We will first prove membership by providing a corresponding paraL$^{\text{para}\beta\text{L}}$ Turing machine that decides the problem. The machine works in two phases: It will first compute which vertices are *usable* and delete the other ones, afterwards the machine will compute s-t-reachability on the resulting graph. We call a vertex *usable* if it is in range of at least one lantern, but is not in range of any warden. The machine can decide for each vertex if it is usable or not by computing the distance from this vertex to each warden and each lantern. This computation can be done using the paraβL oracle, since the size of the vision range is parameterized and since the parameterized distance problem lies in paraβL. Afterwards the machine deletes all vertices that are not usable and computes s-t-reachability on the resulting graph using Reingold's algorithm [63].

To prove hardness we will bootstrap to the problem, i. e., we will reduce any problem in paraL$^{\text{para}\beta\text{L}}$ to p_r-WARDEN$^{k=0}$-LANTERN$^{k=0}$-UREACH by encoding the simulation of a paraL$^{\text{para}\beta\text{L}}$ Turing machine that can decide the problem as instance of the problem p_r-WARDEN$^{k=0}$-LANTERN$^{k=0}$-UREACH. Since the underlying paraL Turing machine is deterministic, the configuration graph $G_{M,x}$ of a paraL$^{\text{para}\beta\text{L}}$ Turing machine M and an input word x is a directed graph of outdegree 1 or 2, where outdegree 2 vertices correspond to configurations in which the oracle is consulted. The two out-edges of these vertices correspond to the possible results of the oracle call, i. e., "yes" and "no". Furthermore, we can modify the Turing machine in a way that it has a unique start configuration s and an unique accepting configuration t. This can be obtained, for example, by letting the machine delete the whole tape and move the head to the first visited tape cell after the machine would have accepted, and let it accept afterwards. The following figure illustrates such a configuration graph.

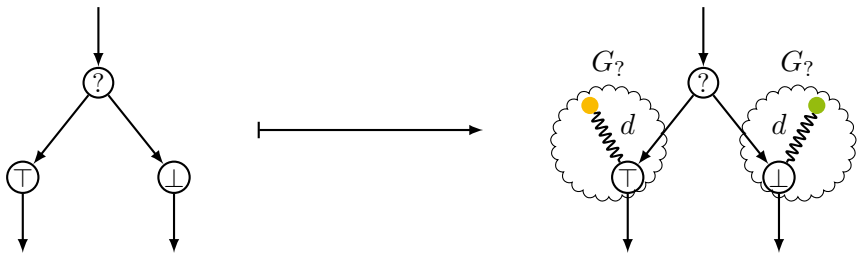


Like in the classical bootstrapping from L to UREACH, our intention is that there exists a path from s to t in this graph if, and only if, the corresponding machine would accept its input. However, we are left with two problems: First of all the graph is directed, and since the graph has vertices with degree greater than 1, we cannot simply forget about the direction. For example in the graph above exists no directed path from s to t; however, if we forget the edge directions there is such a path. The second

problem is, that the vertices with outdegree 2 do not correspond to nondeterminstic configurations, i. e., configurations in which the machine could use both transitions. These vertices correspond to oracle calls and the machine must use the transition that corresponds to the answer of the oracle. Thus, these vertices have outdegree 1 in some sense, but to determine which edge has to be used, one must determine the result of the oracle. Therefore, our task now is to block outgoing edges from vertices with outdegree 2 in correspondence to the result of the oracle. If we achieve this, the graph has *quasi outdegree one*, which means that each vertex has at most one outgoing edge that is usable. Hence, if we delete the outgoing edges of t, we could forget about the edge directions afterwards.

The tools that we have in order to achieve these goals are wardens and lanterns, which we have not used until now. Recall that each consultation of the oracle is the solution of a paraβL problem and since p_d-UDISTANCE is complete for paraβL, an oracle call is not more than the solution of a parameterized distance problem.

Let ⓠ be a question configuration of the Turing machine in which it consults the oracle. The paraL Turing machine that computes the reduction can traverse backwards from ⓠ in order to compute the content x of the question tape. This is possible since the path between two ⓠ configurations is deterministic and since the parameter of x is furthermore bounded by the parameter of the whole instance. Let $G_? = (V, E)$ with $u, v \in V$ and $d \in \mathbb{N}$ be the instance of p_d-UDISTANCE that corresponds to the oracle call in ⓠ, i. e., the instance encoded in x. Informally spoken, the Turing machine will transfer from ⓠ to ⓣ if the distance between u and v in $G_?$ is smaller or equal than d, other wise the machine will transfer to ⓛ. We add $G_?$ two times to the configuration graph and connect v once with ⓣ and the v from the other graph to ⓛ. Now we let u in the first graph be a lantern with range d and in the second graph a warden with range d. The following figure illustrates this process.



If the oracle call is a "yes"-instance, i. e., the distance between v and u in $G_?$ is smaller than d, then ⓣ is covered by a lantern and, hence, usable. On the the other hand, in this scenario is ⓛ covered by a warden and therefore not usable. The other way around: Is the oracle call a "no"-instance, i. e., the distance between v and u in $G_?$ is greater than d, then ⓣ is not covered by a lantern and, thus, not usable. Furthermore, ⓛ is not covered by a warden and therefore usable if it is covered by a lantern as well. To achieve this, we add a global lantern to the graph, which has range 1 and is connected to each vertex of the original configuration graph, without the ⓣ vertices.

The situation now is as follows: Each vertex of the configuration graph which is not Ⓣ is connected to a lantern. Moreover, for each configuration in which the oracle is consulted, the following Ⓣ vertex is connected to a lantern if, and only if, the corresponding oracle call is positive. In this case, the Ⓛ vertex for this configuration is covered by a warden. In the case of a negative result, the vertex Ⓣ is not covered by a lantern, but therefore is Ⓛ. Thus, the graph, which has only the vertices that are covered by a lantern and are not covered by a warden, has an outdegree of at most one. Hence, if we remove all outgoing edges from t and forget the directions of the edges, we obtain an undirected graph in which an usable path between s and t exists if, and only if, there was a directed one in the original graph.

All together, this implies that the constructed undirected graph has an s-t-path in which each vertex is covered by at least one lantern and by no warden if, and only if, the Turing machine would accept its input. This is because if such a path exists, there is a sequence of computation configurations that the Turing machine does, matching to the results of the oracle calls and leading to the accepting configuration. □

With the last theorem, we found a way to describe the intersection of paraNL and XL by a new oracle class. This motivates us to search for an analog description for the intersection of paraWNL and XL. To do so, we introduce analog to paraWL$^{\text{para}\beta\text{L}}$ the class paraWL$^{\text{para}\beta\text{L}}$. In this oracle class the underlying Turing machine is able to use nondeterminism and, thus, we have to be careful about how this nondeterminism can influence the content of the question tape:

► Definition 92
The complexity class paraWL$^{\text{para}\beta\text{L}}$ is the set of all languages that can be recognized by a paraWL Turing machine using a paraβL oracle.

More precisely, a paraWL$^{\text{para}\beta\text{L}}$ Turing machine that decides a parameterized problem (Q_1, κ_1) using the oracle language $(Q_2, \kappa_2) \in$ paraβL has the following properties for two computable functions $f : \mathbb{N} \to \mathbb{N}$ and $g : \mathbb{N} \to \mathbb{N}$:

- ▷ it uses at most $f(\kappa_1(w)) + \mathcal{O}(\log |w|)$ space;
- ▷ it gets $f(\kappa_1(w)) \cdot \mathcal{O}(\log |w|)$ choice bits as additional input;
- ▷ for each $x \in \Sigma^\star$ that is written to the question tape during the computation, we have $\kappa_2(x) \leq g(\kappa_1(w))$;
- ▷ if the machine starts to write content to the question tape, it is not allowed to use its choice bits until it reaches Ⓠ, i. e., the content of the question tape has to be deterministic. ◄

This class may seem a little awkward at the first look, since with paraβL $\subseteq$ paraWL, the oracle is weaker than the underlying Turing machine itself. But the concept of bounded nondeterminism makes this class interesting anyway. This is because a paraWL Turing machine is indeed able to simulate a paraβL machine, but it is not able to simulate such a machine multiple times. Informally spoken, the bounded nondeterminism of a paraWL machine is barely enough to simulate the paraβL machine, but for a second simulation the amount of nondeterministic bits is just not enough. Unfortunately, for a problem like $p_{k,r}$-WARDEN-LANTERN-UREACH, the machine needs to

solve certain problems very often, and for the computation of the solution for each of these problems, the machine needs a bounded amount of nondeterminism.

The way we can think about such a machine is as follows: Also a paraWL$^{\text{para}\beta\text{L}}$ Turing machine as an overall unbounded amount of nondeterminism available, it can only use a bounded amount of this nondeterminism for local computations. Thus, it is for example not clear how such a machine should simulate a paraNL machine or solve the directed graph reachability problem. It is simply not clear how the machine could (in a trivial way) implement the guess and forget algorithm and check the correctness of the computation globally. On the other hand, it is also not clear how a paraNL machine should simulate a paraWL$^{\text{para}\beta\text{L}}$ machine, since it is not even clear how it could simulate a paraWL machine. Thus, paraWL$^{\text{para}\beta\text{L}}$ seems to fit nicely into the complexity hierarchy and can be placed in it by the following lemma:

▻ Lemma 93
The class paraWL$^{\text{para}\beta\text{L}}$ is a subset of paraWNL $\cap$ XL.

Proof. Since paraWL is a subset of XL, paraWL$^{\text{para}\beta\text{L}} \subseteq$ XL follows analogously to the proof in Lemma 90. Furthermore, paraWL$^{\text{para}\beta\text{L}} \subseteq$ paraWNL follows analogously to the proof of paraL$^{\text{para}\beta\text{L}} \subseteq$ paraNL in Lemma 90. The only difference is that the underlying Turing machine this time is a paraWL machine, which can be simulated by a paraWNL machine. □

We will close this section by presenting a natural complete problem for the new complexity class paraWL$^{\text{para}\beta\text{L}}$. In a similar way to paraL$^{\text{para}\beta\text{L}}$, we will introduce the problem $p_{k,r}$-WARDEN-LANTERN-UREACH as a complete problem for paraWL$^{\text{para}\beta\text{L}}$.

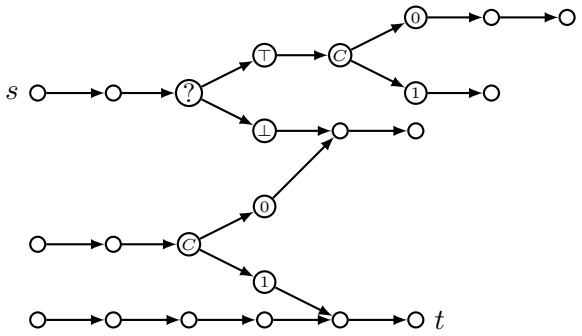
▻ Theorem 94
The problem $p_{k,r}$-WARDEN-LANTERN-UREACH is complete for paraWL$^{\text{para}\beta\text{L}}$ with respect to paraL-reduction.

Proof. Membership follows analogously to the proof of Theorem 91, i. e., the machine computes for each vertex if it is usable or not, deletes all vertices that are not usable, and computes s-t-reachability afterwards. The only difference this time is, that the machine obtains the description of k wardens or lanterns on its choice tape and deletes the corresponding wardens and lanterns before it starts with the actual computation.

For completeness, we will bootstrap all problems within paraWL$^{\text{para}\beta\text{L}}$ to an instance of $p_{k,r}$-WARDEN-LANTERN-UREACH. Thus, we will describe a way to construct a graph $G_{M,x} = (V \cup W \cup L, E)$ with two vertices $s, t \in V$ for each paraWL$^{\text{para}\beta\text{L}}$ Turing machine M and each word $x \in \{0,1\}^\star$ such that in $G_{M,x}$ exists a path from s to t on which each vertex is covered by lantern $\ell \in L$ and is not covered by a warden $w \in W$ if, and only if, M accepts x. To do so, we will first modify M such that M uses a binary work tape and choice tape alphabet, has a unique start configuration s and a unique acceptance configuration t. This is possible, for example, by letting M clear the work tape and move the head to the first tape cell after it would have accepted. Now let $G_{M,x}$ be the configuration graph of M on input x. Thus, if in $G_{M,x}$ exists a path

from s to t, the machine would accept x, since then there exists a sequence of valid configuration steps that lead to the acceptance configuration and which corresponds to x. However, we are left with a couple of problems here. The first obvious problem is that $G_{M,x}$ is directed; and even worse, there are vertices in $G_{M,x}$ with outdegree greater than 1 and, hence, we can not simply forget about the edge directions. The next problem is the certificate: M does no simple nondeterministic steps, which would produce configurations with outdegree 2 (since the guess could be a zero or a one bit). Instead, the nondeterministic guesses of M are dependent, i. e., if a couple of configurations define the successor configuration by the value of the first bit of the choice tape, then of course this bit must be the same for all these configurations. The oracle calls represent our last problem: A configuration that consults the oracle defines its successor configuration by the outcome of this oracle call. However, the configuration itself has outdegree 2, but only one of these outgoing edges can be used, and it must be the one corresponding the answer of the oracle.

The following image illustrates a configuration graph, where ? represent configurations in which the machine calls the oracle, ⊤ and ⊥ represent the possible result configurations of these calls. Furthermore, © represents a configuration in which the machine looks at the choice tape, 0 and 1 then represents the configurations in which a 0 or 1 stands on the choice tape at the requested position.



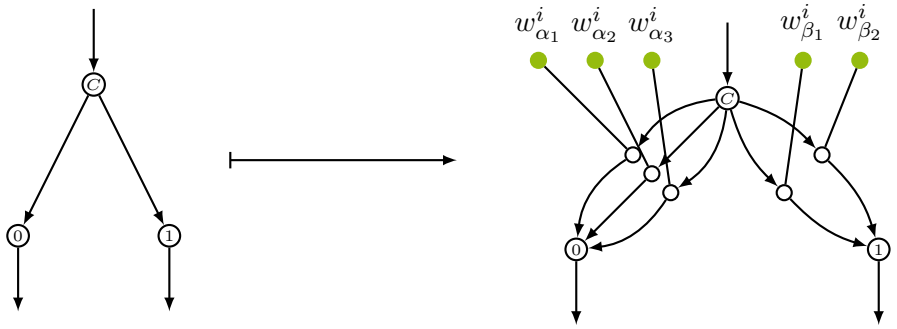
We will first handle the problem of the choice tape, for which we will use wardens and the possibility to delete some of them. Remember that the size of the choice tape of the machine is bounded by $\tilde{f}(\kappa(x)) \cdot \mathcal{O}(\log|x|)$ for a computable function $\tilde{f} : \mathbb{N} \to \mathbb{N}$. Notice that we can write

$$\tilde{f}(\kappa(x)) \cdot \mathcal{O}(\log|x|) = \tilde{f}(\kappa(x)) \cdot c \cdot \lceil \log|x| \rceil = f(\kappa(x)) \cdot \lceil \log|x| \rceil$$

for $|x| > 0$, a constant $c \in \mathbb{N}$, and a computable function $f : \mathbb{N} \to \mathbb{N}$.

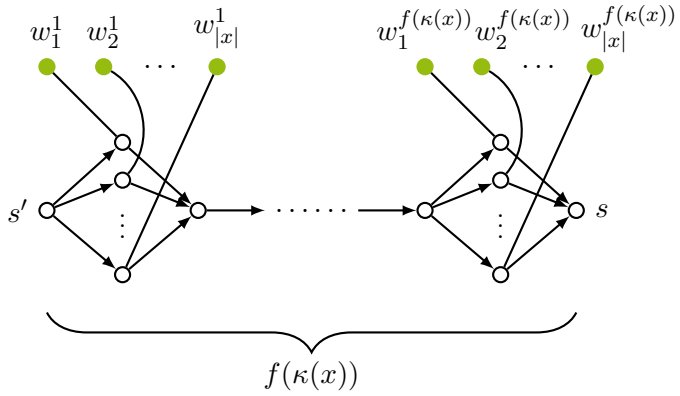
We will split the choice tape in blocks $B_1, B_2, \ldots, B_{f(\kappa(x))}$ of size $\lceil \log|x| \rceil$. Hence, each block B_i can represent $|x|$ different strings $s^i_1, \ldots, s^i_{|x|}$. We will introduce a warden w^i_j for each of these strings, so $f(\kappa(x)) \cdot |x|$ wardens in total. Now let © be the vertex corresponding to a configuration in which the machine asks for a bit on the choice tape. Let this bit be in block B_i and let $s^i_{\alpha_1}, \ldots, s^i_{\alpha_p}$ be the strings of this block in which this bit is 0 and let $s^i_{\beta_1}, \ldots, s^i_{\beta_q}$ be the ones in which the bit is 1. Now we

replace the edge from Ⓒ to ⓪ by p disjoint path of length two. The center vertex of path j then is connected to warden $w^i_{\alpha_j}$. Thus, all these paths are blocked by wardens and the only way to get from Ⓒ to ⓪ is by deleting one of the wardens $w^i_{\alpha_1}, \ldots, w^i_{\alpha_p}$. We do the same for the edge from Ⓒ to ① and the wardens $w^i_{\beta_1}, \ldots, w^i_{\beta_q}$. The process is illustrated in the following figure.



The deletion of a warden $w^i_1, \ldots, w^i_{|x|}$ corresponds to the guess of $\lceil \log |x| \rceil$ bits on the choice tape and has the same effect in the whole graph, i. e., for each configuration using a bit out of the block B_i.

In order to correspond to a valid computation, we have to ensure that exactly one warden of each block will be deleted. Otherwise it would, for example, be possible to open the path to both, the ⓪ and ① configuration. In order to do so, we add a new start vertex s' to the graph. Afterwards, we add a path of $f(\kappa(x))$ layers between s' and s. Each layer represents one of the blocks $B_1, \ldots, B_{f(\kappa(x))}$ and has $|x|$ vertices. Each of these vertices is connected to one of the wardens corresponding to the block and, hence, to travel from one layer to another, one of the wardens corresponding to the block must be deleted. Since we are only allowed to delete $f(\kappa(x))$ wardens overall, we are forced to delete exactly one warden of each block, since otherwise there would be no path between s' and s and therefore no path between s' and t. This gadget is illustrated in the figure below.



The situation now is as follows: In order to have a chance to get an usable path from s' to t, we must delete $f(\kappa(x))$ wardens, one of each block B_i. Thus, each vertex that does not represent a configuration in which the oracle is called now has *quasi outdegree one*. Where quasi outdegree means, that at most 1 edge leads to a vertex which is not covered by a warden. The only vertices which still have real outdegree 2 are the ones which call the oracle. For these, we use the gadget that we have already used in Theorem 91. Notice that it can not happen that we delete wardens in this gadget, since we have to use all $f(\kappa(x))$ deletions to open a path between s' and s. We close the construction by adding a global lantern that covers all vertices except the negative results of oracles (since these have a local lantern in the gadget from Theorem 91). Finally, we remove all outgoing edges from t and can then forget about the edge directions, since each vertex has a quasi outdegree of 1.

In the resulting undirected graph exists a path from s' to t that only uses vertices covered by lanterns and not covered by a warden if, and only if, the machine M accepts the word x. This is because if such a path exists, then a certificate exists for the choice tape – defined by the selection of the $f(\kappa(x))$ wardens – such that there is a sequence of configurations that correspond to x, namely the path, which leads to the accepting configuration. □

4.5 Vision Problems in Trees

We have discovered different versions of warden and lantern reachability problems in the last section. In the present section, we will restrict the problems to trees, or more specificly: The part of the input graph without the wardens (lanterns) has to be a tree.

► Problem 95 (p-WARDEN-TREACH)
Instance: An undirected graph $G = (V, E)$ with a set of pairwise non adjacent wardens $W \subseteq V$ and the property, that $T = (V \setminus W, E)$ is a tree. A vision range function $\nu : W \to \{1, \dots r\}$ is defined on the wardens and two vertices s, t as well as a natural number k is given.
Parameter: k, r
Question: Is it possible to delete k of the wardens such that there is path from s to t in T on which no vertex v has a distance to a warden w of at most $\nu(w)$? ◄

To analyze the complexity of this problem, we will use the fact that the problem of finding an s-t-path in a directed forest with outdegree 0 or 1 is L-complete [16]. We will refer to this problem as DIRECTED-FOREST-REACH and denote the problem parameterized by a trivial parameter as p_0-DIRECTED-FOREST-REACH. Clearly, the parameterized version is complete for paraL. With similar techniques as the ones used in [16] to reduce from DIRECTED-FOREST-REACH to other L problems, we can prove paraL-hardness results with DIRECTED-FOREST-REACH.

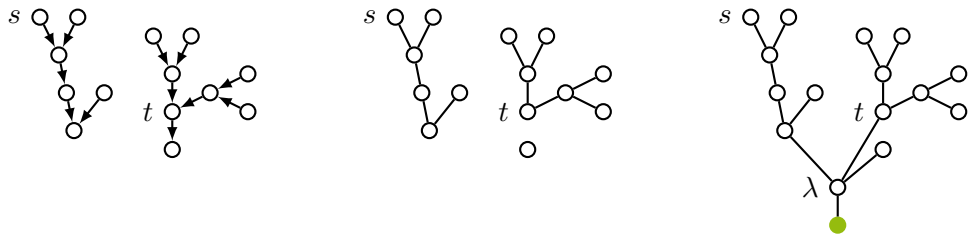
► Theorem 96
The problem $p_{k,r}$-WARDEN-TREACH is complete for paraL under paraAC°-reduction.

Proof. We will first handle the membership in paraL. A corresponding Turing machine can remove the vision range function ν in a preprocessing step as follows: For each warden w, the machine computes the distance to all vertices on the tree and, if the distance is smaller than $\nu(w)$, add a new edge from the warden to these vertices. Thus, each warden has then an edge to each vertex it would cover with the vision range function and, hence, we can drop this function. This preprocessing step is possible within paraL, since computing the distance on a tree is possible with a logarithmic amount of space. After the preprocessing, the Turing machine computes the unique path $s \to v_1 \to \dots \to v_\ell \to t$ in T. For each vertex v_i on the path, the machine checks for each warden w if v_i is covered by w. To do so, it simply has to check if there is an edge between v and w. If v_i is covered by w, then the machine checks for each v_j with $j < i$ if v_j is also covered by w. If none of the v_j is covered by w, then the Turing machine increases a counter by one. The counter now stores the number of wardens that cover the path (and no warden is counted twice). Thus, the machine accepts if the value of the counter is less or equal to k and rejects otherwise.

For hardness, we reduce from p_0-DIRECTED-FOREST-REACH. Let therefore $G = (V, E)$ be a directed graph with an outdegree of at most 1 and $s, t \in V$ be the two marked vertices. We will construct an undirected tree T out of G with the property, that there

is an uncovered path between s and t if, and only if, there was a directed s-t-path in G. In order to do so, we remove the outgoing edge from t, if such exist, and forget the direction of each edge afterwards. We will now show that in the resulting graph G' an undirected s-t-path exists if, and only if, there was a directed s-t-path before. The first direction is clear, a directed path becomes an undirected path. Thus, consider the case that there is no directed path from s to t and, for a contradiction, assume that there is an undirected one in the new graph. Then there is a series of vertices $s \leftrightarrow v_1 \leftrightarrow \cdots \leftrightarrow v_\ell \leftrightarrow t$ in G, here $\leftrightarrow$ should indicate that there is an edge between the two vertices, but the direction of this edge is uncertain. Since this series of vertices becomes a path in G', the edge between v_ℓ and t has to be $v_\ell \to t$ (otherwise we would have deleted the edge). But since each vertex has outdegree of at most one, the edge between $v_{\ell-1}$ and v_ℓ has to be $v_{\ell-1} \to v_\ell$ as well. Thus, for each $1 < i < \ell$ we have $v_{i-1} \to v_i \to v_{i+1}$ and, hence, we have a directed path from s to t, a contradiction.

Now the s-t-reachability in G' remains the same as in G, but G' is just a forest and not a tree. To overcome this issue, we build T by adding a new vertex λ to G'. Furthermore, we add an edge from λ to t and to each vertex with outdegree 0 – i. e., each leaf – in G. The result is a tree, since each connected component in G has exactly one vertex with outdegree 0. Because T now is a tree, s-t-reachability becomes trivial. To obtain the original reachability properties, we add a warden w with $\nu(w) = 1$ and the edge $\{\lambda, w\}$ to T. The result is a tree in which an uncovered path from s to t exists if there was a directed s-t-path in G. If there was no such path in G, then the corresponding path in T is covered by a warden. The whole reduction is visualized in the following figure, from left to right the graphs G, G' and T are illustrated. The warden is, as before, drawn as ●.



To complete the proof, we set $k = 0$ and $r = 1$. Thus, in the resulting instance one is not allowed to remove any warden and hence, a path covered by the warden is not usable for s-t-reachability. □

Note, that our proof implicitly has shown that at least one slice of the problem $p_{k,r}$-WARDEN-TREACH is complete for L, which also directly implies that the problem is paraL-complete [35]. Nevertheless, it also shows that this holds for all the slices and that $p_{k,r}$-WARDEN-TREACH indeed is complete for L.

This result lets the reachability problems with wardens – for which we have proved different results in the last section – drop massively down in the complexity hierarchy if the graph is restricted to be a tree. We will close this section by proving that this drop does not happen for a similar problem defined for lanterns.

► Problem 97 (p-LANTERN-TREACH)
Instance: An undirected graph $G = (V, E)$ and a set of lanterns $L \subseteq V$ with the property that $T = (V \setminus L, E)$ is a tree. Furthermore, a vision range function $\nu : L \to \{1, \ldots, r\}$ is defined on the lanterns and two marked vertices s, t as well as a natural number k are given.
Parameter: k, r
Question: Is it possible to delete k lanterns such that there exists a path from s to t in T on which for each vertex v exists a remaining lantern ℓ with $d(v, \ell) \leq \nu(\ell)$? ◄

We will will first prove that $p_{k,r}$-LANTERN-TREACH is complete for W[1], which implies that the problem lies probably not in paraNL or in a class below. This will also imply, that it is unlikely that the problem is complete for paraWL (for which we will prove membership later), since that would further imply a collapse of the hierarchy. It is also unlikely that the problem falls to a class below paraWL, since its not clear how the path from s to t or the distance to the lanterns should be computed in lower classes. To prove these results, we will show complexity theoretical equality of $p_{k,r}$-LANTERN-TREACH and a version of the famous hitting set problem, which we will call p_k-HUGE-HITTING-SET.

► Problem 98 (p-HUGE-HITTING-SET)
Instance: A hypergraph $H = (V, E)$ and a natural number k.
Parameter: k
Question: Is there a subset $V' \subseteq V$ of $n - k$ vertices such that each edge of H contains at least one vertex of V'? ◄

In contrast to the normal hitting set problem, this version searches for a hitting set of size $n - k$ and not of size k; however, k is still the parameter. Gutin, Jones and Yeo showed that p_k-HUGE-HITTING-SET is W[1]-complete [39].

► Lemma 99
The problems $p_{k,r}$-LANTERN-TREACH and p_k-HUGE-HITTING-SET are equivalent with respect to parameterized complexity under paraL-reduction.

Proof. We first show that we can reduce an instance $G = (V \cup L, E)$ of the problem $p_{k,r}$-LANTERN-TREACH to an instance $H = (V', E')$ of p_k-HUGE-HITTING-SET. Let therefore $s \to v_1 \to \cdots \to v_{|P|} \to t$ be the unique s-t-path P in $T = (V, E)$, which can be computed by a logspace Turing machine. We define for each vertex $v \in P$ the set of lanterns who cover v as $N_L(v) \subseteq L$. This set can be computed by a logspace Turing machine, since the distance problem on trees is solvable within logarithmic space. Now we define the hypergraph for p_k-HUGE-HITTING-SET as $H = (L, E')$ with $N_L(v) \in E'$ for each $v \in P$. Thus, we have for each vertex on the unique s-t-path one edge in the p_k-HUGE-HITTING-SET-instance. This edge contains all lanterns that can cover the corresponding vertex and, thus, a hitting set – i. e., a set of vertices such that in each edge of H is at least one of these vertices – is a selection of lanterns such that each vertex on the path is covered by at least one of these lanterns. If we now

search for a hitting set of size at most $n - k$, clearly there remain at least k lanterns that can be deleted in the original graph.

Let us now reduce an instance $H = (V, E)$ of p_k-HUGE-HITTING-SET to an instance $G = (V' \cup L, E')$ of $p_{k,r}$-LANTERN-TREACH. In order to do so, we will let $L = V$ and add $|E|$ vertices to V' which build with E' a path on which the first vertex is labeled as s and the last one as t. Each vertex on this path corresponds to an edge $e \in E$ of the hypergraph and we will add an edge to E' from this vertex to each lantern that is part of e in the original graph. If we are now able to delete k lanterns such that each vertex on the path from s to t is still covered by a lantern, then there exists a corresponding hitting set of size $n - k$ in the hypergraph, i. e., all the lanterns that we have not deleted. □

► Lemma 100
The problem $p_{k,r}$-LANTERN^r_k-TREACH lies in paraWL.

Proof. A paraWL Turing machine gets the description of k lanterns on its choice tape in form of $f(k) \cdot \mathcal{O}(\log |x|)$ nondeterministic bits. The Turing machine first deletes the k corresponding lanterns. Then the machine computes the distance of each vertex to each lantern, which is possible within logarithmic space, since the underlying graph is a tree. Finally, the machine deletes all vertices that are not in range of any lantern, together with all remaining lanterns. The Turing machine then computes s-t-reachability on the resulting forest. □

4.6 Versions on Directed Graphs

We will close this chapter by revisiting the discussed problems on directed graphs. First results for problems of this kind where again made by Elberfeld, Stockhusen, and Tantau [26], who provided paraWNL-completness results for the three natural problems p_k-COLORED-EDGE-SELECTION-REACH, p_k-COLORED$_1$-EDGE-SELECTION-REACH as well as p_k-COLORED$_\mathcal{F}$-EDGE-SELECTION-REACH. We can apply the tools of Section 4.1 and 4.2 in order to obtain the following corollary:

► Theorem 101
All of the following problems are complete for paraWNL under paraAC°-reduction:

- ▷ p_k-COLORED-EDGE-DELETION-REACH;
- ▷ p_k-COLORED$_1$-EDGE-DELETION-REACH;
- ▷ p_k-COLORED$_1$-VERTEX-SELECTION-REACH;
- ▷ p_k-COLORED$_1$-VERTEX-DELETION-REACH.

Theorem 101 furthermore implies that also the problems p_k-WARDEN$^{r=1}$-REACH and p_k-LANTERN$^{r=1}$-REACH are complete for paraWNL.

► Corollary 102
Both, p_k-WARDEN$^{r=1}$-REACH and p_k-LANTERN$^{r=1}$-REACH, are complete for paraWNL under paraAC°-reductions. ◄

For the other problems, the results follow not so easily. We will first consider the warden and lantern problems with a parameterized vision range, in which we are not allowed to delete wardens or lanterns. In the directed case, it is not clear how these problems could lie in XL. This guess gets evidence though the following theorem:

▻ Theorem 103
The problem p_r-WARDEN$^{k=0}$-REACH is complete for paraNL under paraAC°-reduction.

Proof. Membership follows from the previous used nondeterministic Turing machine, which first computes the distance of each warden to each vertex and deletes all vertices that are covered by a warden. Afterwards, the machine guesses an s-t-path of length at most $|V|$ one vertex by another. The machine accepts if it reaches t in this procedure.

For completeness, we can use the standard bootstrap argument from NL to REACH. The only difference is, that we add a parameter sized set of isolated wardens to the graph. Thus, it is always possible to delete k wardens and the parameter size stays the same. □

This result also holds for p_r-LANTERN$^{k=0}$-REACH:

▻ Corollary 104
The problem p_r-LANTERN$^{k=0}$-REACH is complete for paraNL via paraAC°-reduction.
◅

In a previous section we studied the problem $p_{k,r}$-WARDEN-UREACH and were, unfortunately, not able to prove completeness for any class. We can resolve this issue for directed graphs and prove paraWNL-completeness.

▻ Theorem 105
The problem $p_{k,r}$-WARDEN-REACH is paraWNL-complete under paraAC°-reduction.

Proof. The problem clearly lies in paraWNL: A corresponding Turing machine gets the description of k wardens on its choice tape. The machine deletes these k wardens and computes the distance of the remaining wardens to each vertex. Afterwards the machine deletes all vertices that are in range of any warden and computes s-t-reachability on the resulting graph using a standard guess and forget algorithm.

Hardness follows from the fact that the problem is a general case of the problem p_k-WARDEN$^{r=1}$-REACH, for which we have already proven that it is paraWNL-complete. □

▻ Corollary 106
The problem $p_{k,r}$-LANTERN-REACH is paraWNL-complete under paraAC°-reduction.
◅

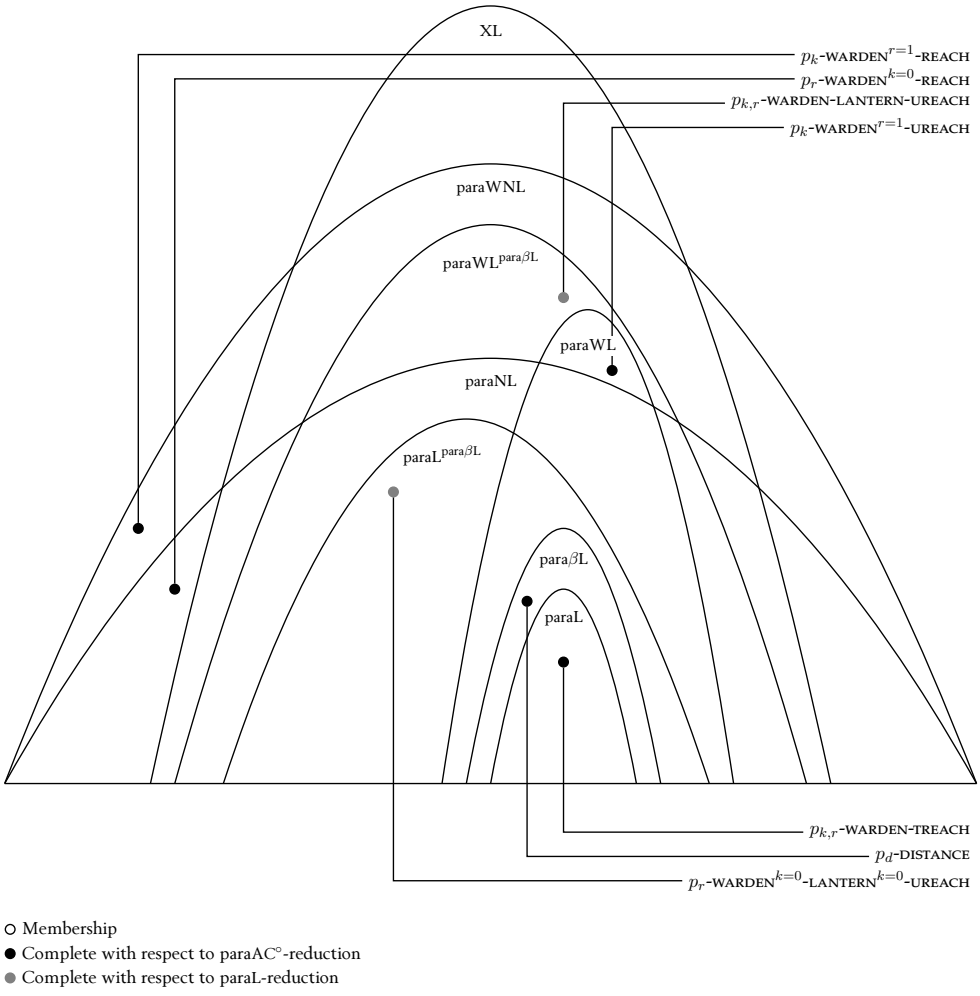
The last directed graph problem that we will consider in this section is the problem p_k-VERTEX-DELETION-REACH, i. e., the problem of determining if it is possible to delete k vertices of a graph in order to preserve a path from s to t.

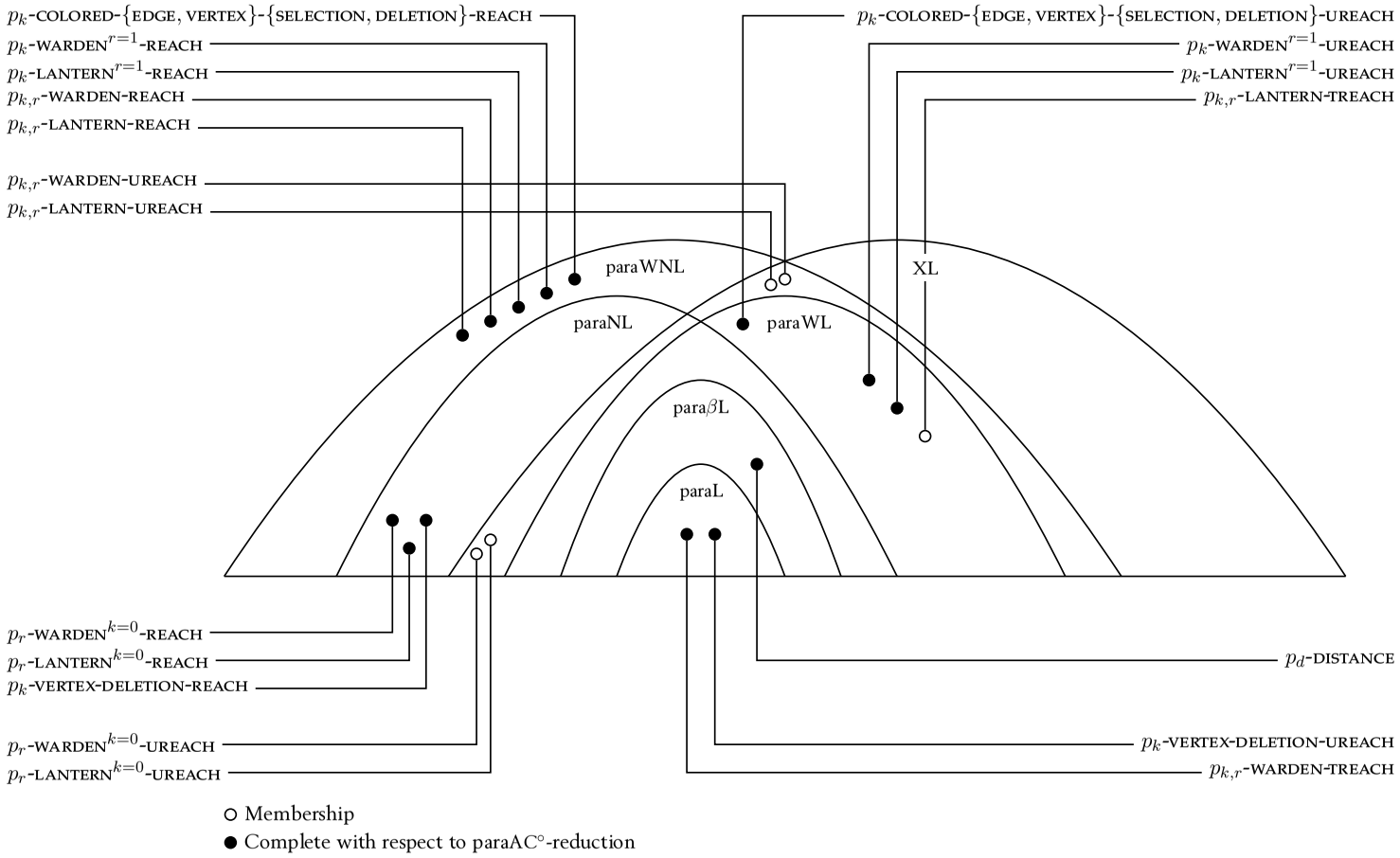
▶ Theorem 107
The problem p_k-VERTEX-DELETION-REACH is complete for paraNL with respect to paraAC°-reduction.

Proof. For membership, we use the following paraNL-algorithm: Start at s and guess nondeterministically an s-t-path one vertex after another. Increment a counter for each used vertex. If after $|V| - k$ steps the vertex t was not reached, then reject; accept otherwise. Correctness follows from the fact that the instance has a solution if, and only if, the shortest path between s and t has a length of at most $|V| - k$. Since if this is the case, there are at least k vertices not on the path, and these can be deleted. On the other hand, if the shortest path has more than $|V| - k$ vertices, then removing k of them will destroy the path.

For completeness, we use the same bootstrapping argument as we have used in Theorem 103. □

4.7 Complexity Maps



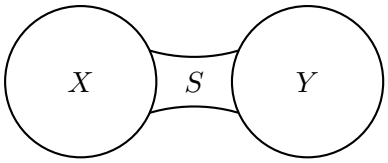

p_k-COLORED-{EDGE, VERTEX}-{SELECTION, DELETION}-REACH
p_k-WARDEN$^{r=1}$-REACH
p_k-LANTERN$^{r=1}$-REACH
$p_{k,r}$-WARDEN-REACH
$p_{k,r}$-LANTERN-REACH
$p_{k,r}$-WARDEN-UREACH
$p_{k,r}$-LANTERN-UREACH
p_k-COLORED-{EDGE, VERTEX}-{SELECTION, DELETION}-UREACH
p_k-WARDEN$^{r=1}$-UREACH
p_k-LANTERN$^{r=1}$-UREACH
$p_{k,r}$-LANTERN-TREACH
paraWNL
paraNL
paraWL
XL
paraβL
paraL
p_r-WARDEN$^{k=0}$-REACH
p_r-LANTERN$^{k=0}$-REACH
p_k-VERTEX-DELETION-REACH
p_r-WARDEN$^{k=0}$-UREACH
p_r-LANTERN$^{k=0}$-UREACH
p_d-DISTANCE
p_k-VERTEX-DELETION-UREACH
$p_{k,r}$-WARDEN-TREACH
○ Membership
● Complete with respect to paraAC°-reduction

5 GRAPH SEPARATION PROBLEMS

In this chapter, our focus lies on graph separation problems. These are problems in which we are given a graph and are asked to determine whether it is possible to delete a given amount of vertices or edges in order to separate the graph in a certain way. Marx has studied different separation problems in [58]. A general version, which he has studied and which we will analyze with respect to space and circuit complexity as well, is the p-CUTTING-ℓ-VERTICES-problem. In this problem we are given a graph and two integers k, ℓ. Our task is to decide, if it is possible to delete k vertices in order to separate a set of ℓ vertices from the rest of the graph. If we set ℓ to be half the size of the remaining graph, i. e., that the deletion of k vertices splits the graph in two equally sized subgraphs, then we obtain the p-VERTEX-BISECTION-problem. This version was recently studied by Bevern, Feldmann, Sorge, and Suchý, who showed that the problem parameterized by k is W[1]-hard for general graphs [70]. They further provided an FPT-algorithm for the case that one can guarantee that the graph has a constant number of connected components after the deletion of the k vertices. They motivated this version by the consideration that graphs in practice tend to break into very few components. Furthermore, they argued with the fact that the separated sets of the optimal bisection are connected with high probability on random regular graphs [10]. If we restrict the problem further to a version in which we are asked to delete k edges instead of vertices, we obtain the p-EDGE-BISECTION-problem (in the literature often referenced as BISECTION or MINIMUM-BISECTION). The EDGE-BISECTION problem is one of the classical NP-complete problems [36]. Bevern et al. [70] summarized the set of applications of EDGE-BISECTION, which ranges from divide-and-conquer algorithms [53] and route planning [21], up to image and video texture synthesis in computer vision algorithms [51]. Also the range of applications is huge, Cygan, Logshtanov, Philipczuk, and Philipczuk mentioned that EDGE-BISECTION is one of the last classical problems, whose status with respect to parameterized complexity was open for a long time. They summarized the huge effort, which was provided by the computer science community, to handle this separation problems: EDGE-BISECTION was massively studied with respect to approximation algorithms [29, 28, 48, 62], under different heuristics [9, 11], and with the aspect of average case complexity [10]. It took until 2014, till Cygan et al. where finally able to provide an FPT-algorithm for p_k-EDGE-BISECTION, which is based on a new graph decomposition technique [19].

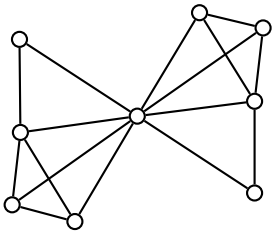
5.1 Cutting Vertices

We start our journey through the world of graph separation problems with the most general version: The p-CUTTING-ℓ-VERTICES-problem. As mentioned earlier in the introduction of this chapter, we are asked to cut ℓ vertices away by deleting k vertices. With other words, we are searching for a separator S of size at most k, which splits the graph into two parts X and Y such that X has exactly size ℓ. This process is illustrated in the image at the right site. Formally, the problem can be defined as follows:

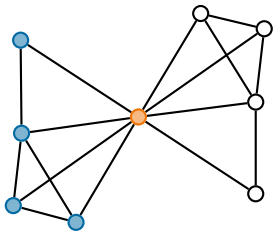


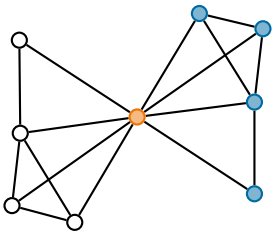
► Problem 108 (CUTTING-ℓ-VERTICES)
Instance: A graph $G = (V, E)$ and two integers k, ℓ.
Question: Is it possible to partition V into three disjoint sets X, S, and Y with $|X| = \ell$ and $|S| \leq k$ such that each vertex in X has no neighbor in Y? ◄

► Example 109
We will start with a small example to illustrate the p-CUTTING-ℓ-VERTICES problem. Let us consider the following graph:

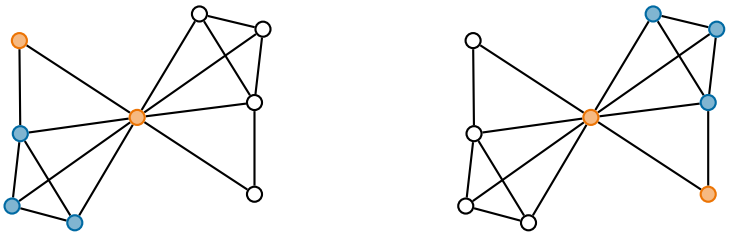


In the following, we will mark X with blue and S with orange. If we set $k = 1$, we can separate $\ell = 4$ vertices, as shown in the following images:





For $k = 2$, one could cut the following sets of size $\ell = 3$:



Other combinations for k and ℓ are possible, for example one also could cut $\ell = 2$ vertices by deleting $k = 3$ vertices. ◀

Natural parameters of this problem are k and ℓ, or both at the same time. Marx has shown that all three versions – p_k-CUTTING-ℓ-VERTICES, p_ℓ-CUTTING-ℓ-VERTICES, and $p_{k,\ell}$-CUTTING-ℓ-VERTICES – are W[1]-hard [58]. We will see in this section that these problems fall into different complexity classes, if we analyze them with respect to parameterized space and circuit complexity.

▶ Theorem 110
The problem p_k-CUTTING-ℓ-VERTICES lies in paraWL.

Proof. We consider the following paraWL Turing machine in order to prove membership. The machine gets the description of k vertices as $f(\kappa(x)) \cdot \mathcal{O}(\log |x|)$ nondeterministic bits on its choice gates. The machine starts by deleting the corresponding k vertices. Afterwards, the machine computes all connected components of the remaining graph. In order to do so, it will consider the natural order of the vertices, which is defined by the input. Each connected component then is represented by the vertex with the smallest index (with respect to this order) that is part of the component. In this way, a logspace Turing machine is able to identify all connected components and their size. Now the question is, whether it is possible to split the set of connected components into two parts such that one part has exactly size ℓ. This is a SUBSET-SUM instance and since the numbers in this instance are represented by connected components, the instance is encoded in unary. Thus, it is possible to solve the SUBSET-SUM-problem with a TC°-circuit [24]. Together with the fact that a TC°-circuit can be simulated by a logspace machine, we get membership for paraWL of p_k-CUTTING-ℓ-VERTICES. □

Unfortunately, it remains open if p_k-CUTTING-ℓ-VERTICES is also paraWL-hard. However, we will prove paraWL-hardness for a similar problem based on wardens, which we will define in Section 5.3. But for now, let us further analyze p-CUTTING-ℓ-VERTICES. The second parameterization that Marx has studied was the parameterization by ℓ, the number of vertices that we want to cut away. Informally said the problem now is to find a huge set of vertices that separate a tiny (parameter sized) part of the vertices from the rest of the graph. Although Marx has proved that this version is W[1]-hard as well [58], intuitively it seems a little bit "easier". And indeed, with respect to parameterized space and circuit complexity, this problem drops in the complexity hierarchy, as we will see in the following theorem.

▶ Theorem 111
The problem p_ℓ-CUTTING-ℓ-VERTICES lies in paraWTC°.

Proof. A paraWTC°-circuit can solve p_ℓ-CUTTING-ℓ-VERTICES as follows: The description of the ℓ vertices of X is presented on the circuit's choice gates. Using threshold gates, the circuit can determine, if the number of vertices $v \notin X$ that are connected to vertices in X is smaller than k. If this is the case, there clearly exists a separator S of size at most k, since one could just delete all vertices connected to X. If the circuit on the other hand counts more than k neighbors, then it is not possible to separate the guessed ℓ vertices by deleting only k vertices. □

To prove paraWTC°-hardness turns out to be quite difficult, since (as far as the author knows) there are no known paraWTC°-complete problems and also no known tools that would provide complete problems.

At the end of this section, we will analyze a version of p-CUTTING-ℓ-VERTICES in which both, k and ℓ, are parameters. Thus, given a graph, we are searching for a tiny (parameter sized) set of vertices that separates another tiny (parameter sized) set of vertices from the rest of the graph. This version still remains W[1]-hard, although it seems to be much more parameterized and, hence, "easier" [58]. The following theorem shows that the problem, indeed, becomes "easier", since it is solvable by weaker circuits.

▶ Theorem 112
The problem $p_{k,\ell}$-CUTTING-ℓ-VERTICES lies in paraWAC°.

Proof. A paraWAC°-circuit for this problem has a fairly simple structure: The description of the sets X and S is presented on the choice gates. This is possible, since $|X| = \ell$ and $|S| \leq k$, and since both, k and ℓ, are parameter. The circuit just checks for each edge $(u, v) \in E$ if u is in X if, and only if, v is in X or S. Afterwards, a single big and-gate can check if this is the case for all edges. □

We should not invest too much hope in a completeness result for this version, since $p_{k,\ell}$-CUTTING-ℓ-VERTICES is not only W[1]-hard, but also W[1]-complete, as stated by the following lemma. Thus, completeness for paraWAC° would imply the relationship paraWAC° $\subseteq$ W[1]. Until now, we have omitted any definition of the W-hierarchy, since this work has not the aim to handle this hierarchy in any form. We refer the interested reader to the textbook from Flum and Grohe for an introduction to the W-hierarchy [35]. This textbook also provides a detailed introduction to the corresponding logic. However, for the following lemma, the following very rough description of W[1] is sufficient. A parameterized problem (Q, κ) lies in W[1] if it can be FPT-reduced to the question, whether or not a logical structure $\mathcal{A}$ is a model for a first-order formula $\varphi(X)$ with one free relation variable X such that the satisfying assignment of X has cardinality k. Furthermore, $\varphi(X)$ has to have the form $\varphi(X) = \forall x_1 \ldots \forall x_\ell \, \psi$ for a quantifier-free formula ψ, and k has to be bounded by the parameter.

► Lemma 113
The problem $p_{k,\ell}$-CUTTING-ℓ-VERTICES lies in W[1].

Proof. The following formula φ uses only $\forall$-quantifiers and has, thus, no changes between quantifier types.

$$\varphi(X, S) = \forall u, v \,.\, E(u, v) \wedge X(u) \rightarrow X(v) \vee S(v)$$

The formula φ describes the problem p-CUTTING-ℓ-VERTICES, since it checks for two sets of vertices X and S, if there is no edge from X to a part of the graph that is neither X nor S. Thus, a graph G together with two integers k and ℓ build a word of p-CUTTING-ℓ-VERTICES if there are sets X and S with $|X| = \ell$ and $|S| = k$, such that G is a model for $\varphi(X, S)$. Moreover, we can replace the two unary relation variables X and S by a single binary relation B and adapt φ correspondingly:

$$B = \{\, (x, 0) \mid x \in X \,\} \cup \{\, (s, 1) \mid s \in S \,\}.$$

Since both, k and ℓ, are parameterized, this implies $p_{k,\ell}$-CUTTING-ℓ-VERTICES $\in$ W[1]. □

5.2 Vertex- and Edge-Bisection

Although we already have analyzed the most general graph separation problem with p-CUTTING-ℓ-VERTICES, we will briefly summarize the results for the well known problems p-VERTEX-BISECTION and p-EDGE-BISECTION in the present section. As described in the introduction of this chapter, the problem p-VERTEX-BISECTION is the special case of p-CUTTING-ℓ-VERTICES where we set $\ell = (|V| - k)/2$.

► Problem 114 (p-VERTEX-BISECTION)
Instance: A graph $G = (V, E)$ and an integer k.
Parameter: k
Question: Is it possible to split V into three disjoint sets X, S, Y such that there is no edge between vertices in X and Y, and that we have $|S| \leq k$ as well as $\big||X| - |Y|\big| \leq 1$? ◄

It is known that p_k-VERTEX-BISECTION is W[1]-hard in the general case, and in FPT if it is guaranteed that the graph $G' = (V \setminus S, E)$ has a constant number of connected components [70]. However, since both versions are special cases of p_k-CUTTING-ℓ-VERTICES, we can conclude that both versions lie in paraWL.

► Corollary 115
The problem p_k-VERTEX-BISECTION lies in paraWL. Therefore, the version in which it is guaranteed that after the deletion of the k vertices only a constant number of connected components exists, lies in paraWL as well. ◄

These results also apply for the more restricted problem p-EDGE-BISECTION, in which we are asked to delete edges instead of vertices in order to construct a bisection of the graph.

► Problem 116 (p-EDGE-BISECTION)
Instance: A graph $G = (V, E)$ and an integer k.
Parameter: k
Question: Is there a partition of V into two disjoint sets X and Y such that we have $\big||X| - |Y|\big| \leq 1$ and such that there are at most k edges between vertices in X and Y? ◄

As mentioned before, this problem is fixed parameter tractable [19]. It is also not surprising that it falls into the same parameterized space complexity class as the problem p-VERTEX-BISECTION.

► Theorem 117
The problem p_k-EDGE-BISECTION lies in paraWL.

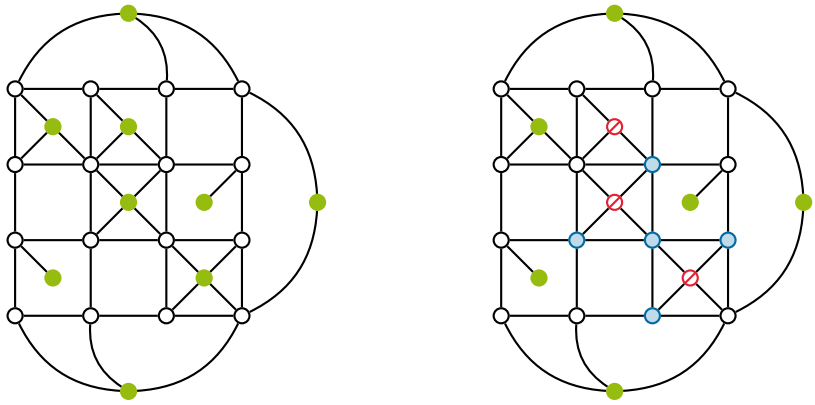
Proof. A paraWL Turing machine can solve the problem as follows: It gets the description on k edges presented on its choice tape and deletes the corresponding edges in a first step. The machine afterwards computes all connected components in the resulting graph. Through the natural order of the vertices in the input, the machine can identify each connected component by the vertex with the smallest index within this connected component. Then the Turing machine computes the size of each of these connected components and is left with the question, if it is possible to divide the set of connected components into two equally sized parts. This task is an instance of unary encoded SUBSET-SUM and, thus, can be computed by a TC^0-circuit [24], which can be simulated by a logspace Turing machine. □

5.3 Separation Problems with Wardens

So far, we have analyzed classical separation problems, in which the objective is to, well, separate a set of vertices from a graph by deleting some vertices or edges. This setup can be generalized if we search for a set of vertices with more properties. Consider for example the following scenario: The owner of a huge piece of ground wants to construct a building of size ℓ on her land. However, her land is covered by a couple of obstacles that block the construction in some parts of the area. A first question would be, if there is a connected field of size ℓ that is not blocked? If this is the case, the owner can simply construct her building there. If it is not the case, a logical next question is how many obstacles have to be removed, in order to obtain a free connected field of size ℓ? To model this problem as a graph problem, remember Chapter 4, where we have introduced warden problems. These are problems in which we are given a graph with two kinds of vertices: Normal ones and wardens. In the mentioned chapter, we have studied reachability problems on such graphs and have seen that they fit very well in the setup of parameterized space complexity with bounded nondeterminism. In the present section, we will analyze a warden problem that models the scenario from above. More specifically, we are given a graph $G = (V, E)$ together with a set of wardens $W \subseteq V$. The resulting question is whether it is possible to delete k wardens such that there are at least ℓ connected vertices $X \subseteq V$ that are not covered by a warden.

► Problem 118 (p-WARDEN-REGION-UNCOVER)
Instance: A graph $G = (V \cup W, E)$ with a set of wardens W, together with two integers k, ℓ.
Parameter: k
Question: Is it possible to delete k wardens $w \in W$ such that there is a set of ℓ connected vertices $X \subseteq V$ that are all not adjacent to any warden? ◄

► Example 119
For an example, consider the grid graph illustrated in the following image, which is covered by a couple of wardens. As usual, we denote the wardens by ●.



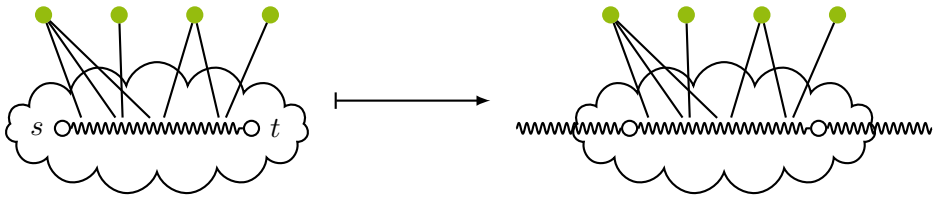
In the graph (left image), each vertex is covered by at least one warden and, thus, there is no connected uncovered area at all. Let us say we are allowed to delete $k = 3$ wardens. Then we could uncover a connected area of size $\ell = 5$ as illustrated in the right image. The wardens that will be deleted are marked as ⊘, and the vertices of the uncovered connected area are marked as ○. ◄

► Theorem 120
The problem p_k-WARDEN-REGION-UNCOVER is complete for paraWL with respect to paraAC°-reductions.

Proof. We will first prove membership for paraWL by constructing a corresponding Turing machine. Such a machine gets the description of k wardens presented on its choice tape. The machine first deletes the corresponding k wardens, then it deletes all vertices that are adjacent to a remaining warden, and finally it deletes all residual wardens. This operation cuts the graph in a couple of connected components, each completely unguarded by the wardens. Thus, the machine now has to check if one of these connected components has a size of at least ℓ. Since both, identifying connected components and measuring the size of connected components, is possible within logspace, it follows that p_k-WARDEN-REGION-UNCOVER lies in paraWL.

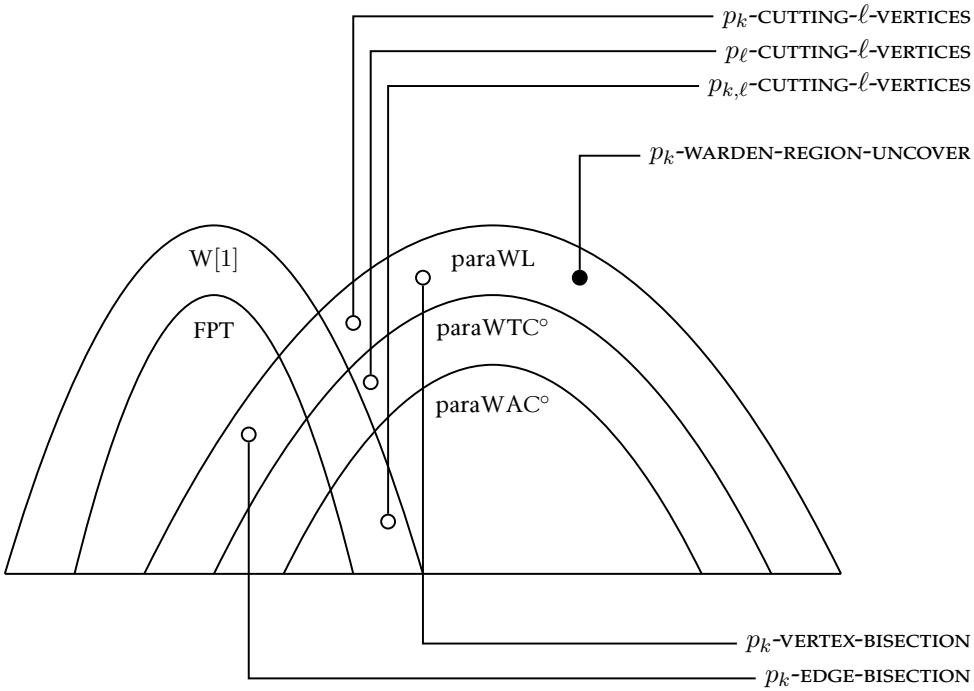
For paraWL-hardness, we reduce from p_k-WARDEN$^{r=1}$-UREACH. A problem for which we have already proved paraWL-completness in Theorem 77. Remember, in this problem we are given a graph $G = (V, E)$ with a set of wardens $W \subseteq V$ and two vertices

s, t with $s, t \in V$. Our objective is to determine, whether or not it is possible to delete k wardens such that there is a path from s to t on which no vertex is a warden nor connected to a warden. The basic idea of the reduction is to add huge connected components to s and t such that an uncovered connected component of a certain size only can be obtained, if there is an uncovered path between s and t. We construct a new instance $I = (\ell, k, G' = (V', E'))$ for p_k-WARDEN-REGION-UNCOVER in the following way: Let $|V \setminus W| = n$, we add a path of n vertices to s and another path of n vertices to t. All these $2n$ vertices will not be connected to any warden. We now set $\ell = 2n$ and let k be the same as in the instance of p_k-WARDEN$^{r=1}$-UREACH. The construction is illustrated in the following figure.



For correctness of the construction imagine that it is possible to delete k wardens such that there is an uncovered path P from s to t. Then by deleting the same wardens in G', we uncover a region of a size of at least $2n + |P|$, which is greater than ℓ. On the other hand, if it is possible to delete k wardens in G' such that there is an uncovered region with a size of at least ℓ, then either s or t must be part of this region. If both would not be part of the region, then the region could have a size of at most $n - 2$. Without loss of generality, we can assume that s is part of the region. Since the size of the uncovered region is greater than ℓ, there have to be n other vertices (beside s and the path we added to s) in the uncovered region. There are two possible ways that could happen, either there is a path from s to t and, hence, the n vertices we added to t are part of the region as well; or all n vertices of the original graph are part of the region, which also implies a path from s to t. Together it follows, that it is possible to delete k wardens in G in order to obtain an uncovered path from s to t if, and only if, it is also possible to delete k wardens in G' in order to obtain an uncovered connected region of size ℓ. Thus, we get paraWL-completness for p_k-WARDEN-REGION-UNCOVER. □

5.4 Complexity Map



○ Membership
● Complete with respect to paraAC°-reduction

6 CONCLUSION AND OUTLOOK

The main objective of this work was to fill the different parameterized space and circuit classes from the framework by Elberfeld, Stockhusen, and Tantau with life [26]. We achieved this with a couple of different problems for which we where able to prove completeness for most of these classes. Furthermore, we have provided problems that fall into the intersections of different complexity classes, which gives evidence that all aspects of these classes are necessary. We started the work with the automata intersection problem in Chapter 2. For different parameterizations of the problem, most of them are hard for a layer of the W-hierarchy, we proved membership in different paraW-classes. We also were able to prove that the version parameterized by the number of automata and the number of states of these automata lies in paraAC°, which implies that it lies in FPT. This resolves an open problem from Wareham, reported in the compendium by Cesati [13]. In Chapter 3 we then have analyzed different vertex ordering problems and where able to prove, that these problems can be solved within time efficient XNL.

The core of the work was Chapter 4, where we have analyzed different colored reachability problems. Such colored reachability problems where already considered by Elberfeld, Stockhusen, and Tantau and were proved to be paraWL-complete. However, many natural parameterized graph problems that lie in paraWL are more of the kind: "Is it possible to delete k vertices such that . . .?" For example in the parameterized feedback vertex set problem we have: "Is it possible to delete k vertices such that the remaining graph is cycle free?"; in vertex-bisection problems we have: "Is it possible to delete k vertices such that the connected components of the remaining graph can be partitioned into two equal sized sets?" These problems motivated us to define handy versions of the colored reachability problems, in which the task is to delete some vertices. The results where the warden and lantern reachability problems, which are both good candidates for further reductions. We also have analyzed intersections of complexity classes in this chapter. Elberfeld, Stockhusen, and Tantau already filled the intersection of paraNL and paraWL with paraβL. However, the intersections of XL and paraNL or paraWNL where still undiscovered. In this thesis we filled these intersections with $\text{paraL}^{\text{para}\beta\text{L}}$ and $\text{paraWL}^{\text{para}\beta\text{L}}$ and provided complete problems for both classes. These problems where based on warden and lantern reachability problems.

The last chapter was Chapter 5, where we have considered graph separation prob-

lems. An eye catcher in this chapter was the problem of cutting ℓ vertices away from a graph. This problem is known to be W[1]-hard for different parameterizations, and we showed that these different versions fall into different complexity classes in parameterized space and circuit complexity, i. e., paraWL, paraWTC°, and paraWAC°. This shows the amount of detailed information about a problem that we can obtain by considering parameterized space and circuit classes. It also shows the strength of parameterized circuit classes if local graph properties are considered.

While the framework by Elberfeld, Stockhusen, and Tantau is quite detailed and while we have provided a couple of problems for the different classes, there are still a lot of unanswered questions. Obvious is, that there are a lot more "certain" problems that are famous in the parameterized complexity scene and which are still not considered with respect to parameterized space and circuit complexity. Here, one can hope for a deeper understanding of many different problems and maybe even for some new and fast algorithms. Another point are parameterized circuit classes. While parameterized space classes already obtained a lot of attention by Elberfeld, Stockhusen, and Tantau and where also heavily analyzed in Chapter 4, this is not the case for parameterized circuit classes. For them, many open questions remain, as there are still no complete problems for many of these classes. Also, in case of paraWTC°, it is not even clear how one could provide such complete problems under a weak standard reduction. Thus, the field of parameterized circuit complexity should be a target of further research.

Another point is the complexity of the framework, which is quite huge. If one wants to consider a problems complexity with respect to parameterized space and circuit classes, one always has to check the para-classes, the paraW-classes, and the X-classes. It is therefore likely that one overlooks one of the classes or intersection of classes. The question thus is, if all of the classes are needed or if the size of the framework can be reduced. Since we have seen problems in all of these classes, it is unlikely that one can easily reduce the size of the framework by removing some classes. Thus, one maybe needs another point of view for some aspects of the framework. Such new aspects can work, as we have seen with the classes $\text{paraL}^{\text{para}\beta\text{L}}$ and $\text{paraWL}^{\text{para}\beta\text{L}}$, although our results did not reduce the size of the framework.

REGISTER OF PROBLEMS

Problem	Parameter	Complexity	Reference
p-DFA-INTERSECTION	k	XNL-complete	9 on Page 12
p-DFA-INTERSECTION	m	paraWL $\cap$ XAC$^\circ$	18 on Page 19
p-DFA-INTERSECTION	k, m	paraWL $\cap$ XAC$^\circ$	22 on Page 20
p-DFA-INTERSECTION	q, m	paraWNC^1 $\cap$ XAC$^\circ$	23 on Page 21
p-DFA-INTERSECTION	q, k	paraAC$^\circ$	26 on Page 23
p-FDFA-INTERSECTION	k	XNL$_{\text{FPT}}$-complete	13 on Page 16
p-NFA-INTERSECTION	k	XNL-complete	12 on Page 16
p-NFA-INTERSECTION	m	paraWNL $\cap$ XAC$^\circ$	19 on Page 20
p-NFA-INTERSECTION	k, m	paraWNL $\cap$ XAC$^\circ$	22 on Page 20
p-NFA-INTERSECTION	q, m	paraWNC1 $\cap$ XAC$^\circ$	23 on Page 21
p-NFA-INTERSECTION	q, k	paraAC$^\circ$	26 on Page 23
p-δ_{C}DFA-INTERSECTION	k	XNL$_{\text{FPT}}$	28 on Page 26
p-δ_{H}DFA-INTERSECTION	k	XNL$_{\text{FPT}}$	29 on Page 26
p-δ_{C}DFA-INTERSECTION	m	paraWAC$^\circ$	30 on Page 26
p-δ_{H}DFA-INTERSECTION	m	paraWAC$^\circ$	30 on Page 26
p-δ_{C}DFA-INTERSECTION	m, s	paraAC$^\circ$	31 on Page 27
p-δ_{H}DFA-INTERSECTION	m, s	paraAC$^\circ$	31 on Page 27
p-δ_{C}DFA-INTERSECTION	k, r	paraNL	32 on Page 27
p-δ_{H}DFA-INTERSECTION	k, r	paraNL	32 on Page 27
p-δ_{C}DFA-INTERSECTION	k, m	paraβL $\cap$ paraWAC$^\circ$	33 on Page 28
p-δ_{H}DFA-INTERSECTION	k, m	paraβL $\cap$ paraWAC$^\circ$	33 on Page 28
p-BANDWIDTH	k	XNL$_{\text{FPT}}$	36 on Page 32
p-CUTWIDTH	k	XNL$_{\text{FPT}}$ $\cap$ XL$_{\text{NU}}$	53 on Page 39
p-IMBALANCE	k	XNL$_{\text{FPT}}$	56 on Page 40
p-COLORED$_1$-VERTEX-SELECTION-UREACH	k	paraWL-complete	63 on Page 46
p-COLORED-EDGE-DELETION-UREACH	k	paraWL-complete	66 on Page 47
p-COLORED$_1$-EDGE-DELETION-UREACH	k	paraWL-complete	68 on Page 49
p-COLORED-VERTEX-DELETION-UREACH	k	paraWL-complete	69 on Page 50
p-COLORED$_1$-VERTEX-DELETION-UREACH	k	paraWL-complete	70 on Page 50
p-VERTEX-DELETION-UREACH	k	paraL-complete	73 on Page 51

p-WARDEN$^{r=1}$-UREACH	k	paraWL-complete	77 on Page 54
WARDEN$^{k=0}$-UREACH	–	NL-complete	78 on Page 55
p-WARDEN$^{k=0}$-UREACH	r	paraNL ∩ XL	79 on Page 56
p-WARDEN-UREACH	k, r	paraWNL ∩ XL	80 on Page 56
p-LANTERN$^{r=1}$-UREACH	k	paraWL-complete	83 on Page 57
LANTERN$^{k=0}$-UREACH	–	NL-complete	84 on Page 58
p-LANTERN$^{k=0}$-UREACH	r	paraNL ∩ XL	85 on Page 58
p-LANTERN-UREACH	k, r	paraWNL ∩ XL	86 on Page 58
p-WARDEN$^{k=0}$-LANTERN$^{k=0}$-UREACH	r	paraL$^{\text{para}\beta\text{L}}$-complete	91 on Page 62
p-WARDEN-LANTERN-UREACH	k, r	paraWL$^{\text{para}\beta\text{L}}$-complete	94 on Page 65
p-WARDEN-TREACH	k, r	paraL-complete	96 on Page 69
p-LANTERN-TREACH	k, r	paraWL	100 on Page 72
p-COLORED-EDGE-DELETION-REACH	k	paraWNL-complete	101 on Page 72
p-COLORED$_1$-EDGE-DELETION-REACH	k	paraWNL-complete	101 on Page 72
p-COLORED$_1$-VERTEX-SELECTION-REACH	k	paraWNL-complete	101 on Page 72
p-COLORED$_1$-VERTEX-DELETION-REACH	k	paraWNL-complete	101 on Page 72
p-WARDEN$^{r=1}$-REACH	k	paraWNL-complete	102 on Page 72
p-LANTERN$^{r=1}$-REACH	k	paraWNL-complete	102 on Page 72
p-WARDEN$^{k=0}$-REACH	r	paraNL-complete	103 on Page 73
p-LANTERN$^{k=0}$-REACH	r	paraNL-complete	104 on Page 73
p-WARDEN-REACH	k, r	paraWNL-complete	105 on Page 73
p-LANTERN-REACH	k, r	paraWNL-complete	106 on Page 73
p_k-VERTEX-DELETION-REACH	k	paraNL-complete	107 on Page 74
p-CUTTING-ℓ-VERTICES	k	paraWL	110 on Page 79
p-CUTTING-ℓ-VERTICES	ℓ	paraWTC°	111 on Page 80
p-CUTTING-ℓ-VERTICES	k, ℓ	paraWAC°	112 on Page 80
p-VERTEX-BISECTION	k	paraWL	115 on Page 81
p-EDGE-BISECTION	k	paraWL	117 on Page 82
p-WARDEN-REGION-UNCOVER	k	paraWL-complete	120 on Page 83

REFERENCES

[1] D. Adolphson and T. C. Hu. Optimal linear ordering. *SIAM Journal on Applied Mathematics*, 25(3):403–423, November 1973.

[2] D. A. M. Barrington, K. Compton, H. Straubing, and D. Thérien. Regular languages in $\mathcal{NC}^1$. *Journal of Computer and System Sciences*, 44(3):478–499, 1992.

[3] T. Biedl, T. Chan, Y. Ganjali, M. T. Hajiaghayi, and D. R. Wood. Balanced vertex-orderings of graphs. *Discrete Applied Mathematics*, 148(1):27–48, December 2005.

[4] G. Blin, G. Fertin, D. Hermelin, and S. Vialette. Fixed-parameter algorithms for protein similarity search under mRNA structure constraints. *Journal of Discrete Algorithms*, 6(4):618–626, December 2008.

[5] H. L. Bodlaender. Classes of graphs with bounded tree-width. Technical Report 22, Department of Computer Science, Utrecht University, December 1986.

[6] H. L. Bodlaender. A linear time algorithm for finding tree-decompositions of small treewidth. In *Proceedings of the Twenty-Fifth Annual ACM Symposium on Theory of Computing, May 16-18, 1993, San Diego, CA, USA*, STOC '93, pages 226–234. ACM, New York, USA, May 1993.

[7] H. L. Bodlaender. A partial k-arboretum of graphs with bounded treewidth. *Theoretical Computer Science*, 209(1–2):1–45, December 1998.

[8] R. A. Botafogo. Cluster analysis for hypertext systems. In *Proceedings of the Sixteenth Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, June 27 – July 01, 1993, Pittsburgh, Pennsylvania, USA*, SIGIR '93, pages 116–125. ACM, New York, USA, July 1993.

[9] T. Bui, C. Heigham, C. Jones, and T. Leighton. Improving the performance of the kernighan-lin and simulated annealing graph bisection algorithms. In *Proceedings of the Twenty-Sixth ACM/IEEE Design Automation Conference, June 25 - 28, 1989, Las Vegas, NV, USA*, DAC '89, pages 775–778. ACM, New York, USA, June 1989.

[10] T. N. Bui, S. Chaudhuri, and M. S. F. T. Leighton. Graph bisection algorithms with good average case behavior. *Combinatorica*, 7(2):171–191, June 1987.

[11] T. N. Bui and L. C. Strite. An ant system algorithm for graph bisection. In *Proceedings of the Genetic and Evolutionary Computation Conference, July 9–13, 2002, New York, USA*, GECCO '02, pages 43–51. Morgan Kaufmann Publishers Inc., Burlington, Massachusetts, USA, July 2002.

[12] L. Cai, J. Chen, R. G. Downey, and M. R. Fellows. Advice classes of parameterized tractability. *Annals of Pure and Applied Logic*, 84(1):119–138, March 1997.

[13] M. Cesati. Compendium of parameterized problems. Department of Computer Science, Systems, and Industrial Engineering, University of Rome "Tor Vergata", `http://cesati.sprg.uniroma2.it/research/compendium/`, September 2006.

[14] Y. Chen, J. Flum, and M. Grohe. Bounded nondeterminism and alternation in parameterized complexity theory. In *Proceedings of the Eighteenth IEEE Conference on Computational Complexity, July 7–10, 2003, Aarhus, Denmark*, CCC '03, pages 13–29. IEEE Computer Society, Los Alamitos, California, July 2003.

[15] P. Z. Chinn, J. Chvátalová, A. K. Dewdney, and N. E. Gibbs. The bandwidth problem for graphs and matrices – a survey. *Journal of Graph Theory*, 6(3):223–254, Autumn 1982.

[16] S. A. Cook and P. McKenzie. Problems complete for deterministic logarithmic space. *Journal of Algorithms*, 8(5):385–394, September 1987.

[17] B. Courcelle. Graph rewriting: An algebraic and logic approach. In *Formal Models and Semantics*, volume B of *Handbook of Theoretical Computer Science*, pages 193–242. Elsevier, Amsterdam, Netherlands and MIT Press, Cambridge, Massachusetts, 1990.

[18] B. Courcelle and J. Engelfriet. *Graph Structure and Monadic Second-Order Logic*. Cambridge University Press, Cambridge, England, 2012.

[19] M. Cygan, D. Lokshtanov, M. Pilipczuk, M. Pilipczuk, and S. Saurabh. Minimum bisection is fixed parameter tractable. In *Proceedings of the Forty-Sixth Annual ACM Symposium on Theory of Computing, May 31 – June 3, 2014, New York, USA*, STOC '14, pages 323–332. ACM, New York, USA, 2014.

[20] M. Cygan, D. Lokshtanov, M. Pilipczuk, M. Pilipczuk, and S. Saurabh. On cutwidth parameterized by vertex cover. *Algorithmica*, 68(4):940–953, November 2014.

[21] D. Delling, A. V. Goldberg, T. Pajor, and R. F. Werneck. Customizable route planning. In *Proceedings of the Tenth International Symposium on Experimental Algorithms, May 5–7, 2011, Crete, Greece*, SEA '11, pages 376–387. Springer, Heidelberg, Germany, May 2011.

[22] R. Diestel. *Graphentheorie*. Springer, Heidelberg, Germany, 4 edition, 2010.

[23] M. Elberfeld, A. Jakoby, and T. Tantau. Logspace versions of the theorems of bodleander and courcelle. In *Proceedings of the Annual IEEE Symposium on Foundations of Computer Science, October 23–26, 2010, Las Vegas, USA*, FOCS '10, pages 143–152. IEEE Computer Society, Los Alamitos, California, October 2010.

[24] M. Elberfeld, A. Jakoby, and T. Tantau. Algorithmic meta theorems for circuit classes of constant and logarithmic depth. In *Proceedings of the Twenty-Ninth International Symposium on Theoretical Aspects of Computer Science, February 29 – March 3, 2012, Prais, France*, STACS '12, pages 66–77. Schloss Dagstuhl-Leibniz-Zentrum für Informatik, March 2012.

[25] M. Elberfeld, C. Stockhusen, and T. Tantau. On the space complexity of parameterized problems. In *Proceedings of Seventh International Symposium of Parameterized and Exact Computation, September 12–14, 2012, Ljubljana, Slovenia*, IPEC '12, pages 206–217. Springer, Heidelberg, Germany, 2012.

[26] M. Elberfeld, C. Stockhusen, and T. Tantau. On the space complexity of parameterized problems: Classes and completness. *Algorithmica*, Online First:1–41, October 2014.

[27] M. Elberfeld and J. Textor. Negative selection algorithms on strings with efficient training and linear-time classification. *Theoretical Computer Science*, 412(6):534–542, August 2011.

[28] U. Feige and R. Krauthgamer. A polylogarithmic approximation of the minimum bisection. *SIAM Journal on Computing*, 31(4):1090–1118, April 2002.

[29] U. Feige, R. Krauthgamer, and K. Nissim. Approximating the minimum bisection size (extended abstract). In *Proceedings of the Thirty-Second Annual ACM Symposium on Theory of Computing, May 21–23, 2000, Portland, Oregon, USA*, STOC '00, pages 530–536. ACM, New York, USA, May 2000.

[30] M. Fellows and M. A. Langston. On search, decision, and the efficiency of polynomial-time algorithms. *Journal of Computer and System Sciences*, 49(3):769–779, December 1994.

[31] M. Fellows, D. Lokshtanov, N. Misra, M. Minch, F. A. Rosamond, and S. Saurabh. The complexity ecology of parameters: An illustration using bounded max leaf number. *Theory of Computing Systems*, 45(4):822–848, January 2009.

[32] M. R. Fellows and M. A. Langston. On well-partial-order theory and its application to combinatorial problems of VLSI design. *SIAM Journal on Discrete Mathematics*, 5(1):117–126, February 1992.

[33] M. R. Fellows, D. Lokshtanov, N. Misra, F. A. Rosamond, and S. Saurabh. Graph layout problems parameterized by vertex cover. In *Proceedings of the Nineteenth International Symposium on Algorithms and Computation, December 15–17, 2008, Gold Coast, Australia*, ISAAC '2008, pages 294–305. Springer, Heidelberg, Germany, December 2008.

[34] J. Flum and M. Grohe. Describing parameterized complexity classes. In *Proceedings of the Nineteenth Annual Symposium on Theoretical Aspects of Computer Science, March 14–16, 2002, Antibes Juan-les-Pins, France*, STACS '02, pages 359–371. Springer, Heidelberg, Germany, March 2002.

[35] J. Flum and M. Grohe. *Parameterized Complexity Theory*. Springer, Heidelberg, Germany, 2006.

[36] M. R. Garey and D. S. Johnson. *Computers and Intractability – A Guide to the Theory of NP-Completeness*. W. H. Freeman, USA, 1979.

[37] S. Gaspers, M.-E. Messinger, R. J. Nowakowski, and P. Prałat. Clean the graph before you draw it! *Information Processing Letters*, 109(10):463–467, April 2009.

[38] S. Guillemot. Parameterized complexity and approximability of the SLCS problem. In *Proceedings of the Third International Workshop on Parameterized and Exact Computation, May 14–16, 2008, Victoria, Canada*, IWPEC '08, pages 115–128. Springer, Heidelberg, Germany, May 2008.

[39] G. Gutin, M. Jones, and A. Yeo. Kernels for below-upper-bound parameterizations of the hitting set and directed dominating set problems. *Theoretical Computer Science*, 412(41):5744–5751, September 2011.

[40] J. Hao. Maximum cutwidth problem for graphs. *Applied Mathematics – A Journal of Chinese Universities*, 18(2):235–242, April 2003.

[41] N. Immerman. Nondeterministic space is closed under complementation. *SIAM Journal of Computing*, 17(5):935–938, October 1988.

[42] A. Jakoby and T. Tantau. Logspace algorithms for computing shortest and longest paths in series-parallel graphs. In *Proceedings of the Twenty-Seventh International Conference of Foundations of Software Technology and Theoretical Computer Science, December 12–14, 2007, New Delhi, India*, FSTTCS '07, pages 216–227. Springer, Heidelberg, Germany, December 2007.

[43] M. Jünger, G. Reinelt, and G. Rinaldi. The traveling salesman problem. In *Handbook on Operations Research and Managment Sciences*, volume 7: Network Models, pages 225–330. Elsevier, Amsterdam, Netherlands, 1995.

[44] G. Kant. Drawing planar graphs using the canonical ordering. *Algorithmica*, 16(1):4–32, July 1996.

[45] G. Kant and X. He. Regular edge labeling of 4-connected plane graphs and its applications in graph drawing problems. *Theoretical Computer Science*, 172(1–2):175–193, February 1997.

[46] J. Kára, J. Kratochvíl, and D. R. Wood. On the complexity of the balanced vertex ordering problem. In *Proceedings of the 11th Annual International Conference of Computing and Combinatorics, August 16–19, 2005, Kunming, China*, COCOON '05, pages 849–858. Springer, Heidelberg, Germany, August 2005.

[47] D. R. Karger. A randomized fully polynomial time approximation scheme for the all-terminal network reliability problem. *SIAM Journal on Computing*, 29(2):492–514, October 1999.

[48] S. A. Khot and N. K. Vishnoi. The unique games conjecture, integrality gap for cut problems and embeddability of negative type metrics into ℓ_1. In *Proceedings of the Forty-Sixth Annual IEEE Symposium on Foundations of Computer Science, October 23–25, 2005, Pittsburgh, USA*, FOCS '05, pages 53–62. IEEE Computer Society, Los Alamitos, California, October 2005.

[49] D. Kozen. Lower bounds for natural proof systems. In *Proceedings of the Eighteenth Annual IEEE Symposium on Foundations of Computer Science, October 31 – November 1, 1977, Providence, USA*, FOCS '77, pages 254–266. IEEE Computer Society, Los Alamitos, California, November 1977.

[50] J. B. Kruskal. The theory of well-quasi-ordering: A frequently discovered concept. *Journal of Combinatorial Theory*, 13(3):297–305, November 1972.

[51] V. Kwatra, A. Schödl, I. Essa, G. Turk, and A. Bobick. Graphcut textures: image and video synthesis using graph cuts. In *Proceedings of the Thirtieth International Conference on Computer Graphics and Interactive Techniques, July 27–31, 2003, San Diego, USA*, SIGGRAPH '03, pages 277–286. ACM, New York, USA, July 2003.

[52] N. Linial, E. London, and Y. Rabinovich. The geometry of graphs and some of its algorithmic applications. *Combinatorica*, 15(2):215–245, June 1995.

[53] R. J. Lipton and R. E. Tarjan. Applications of a planar seperator theorem. *SIAM Journal on Computing*, 9(3):615–627, August 1980.

[54] M. Liśkiewicz and J. Textor. Negative selection algorithms without generating detectors. In *Proceedings of the Twelfth Annual Conference on Genetic and Evolutionary Computation, July 7–11, 2010, Portland, Oregon, USA*, GECCO '10, pages 1047–1054. ACM, New York, USA, July 2010.

[55] D. Lokshtanov, N. Misra, and S. Saurabh. Imbalance is fixed parameter tractable. *Information Processing Letters*, 113(19–21):714–718, September–October 2013.

[56] L. Lovász. Graph minor theory. *Bulletin of the American Mathematical Society*, 43(1):75–86, October 2005.

[57] F. S. Makedon and I. H. Sudborough. Minimizing width in linear layouts. In *Proceedings of the Tenth Colloquium on Automata, Languages and Programming, July 18–22, 1983, Barcelona, Spain*, ICALP '83, pages 478–490. Springer, London, UK, July 1983.

[58] D. Marx. Parameterized graph separation problems. *Theoretical Computer Science*, 351(3):394–406, February 2006.

[59] P. Mutzel. A polyhedral approach to planar augmentation and related problems. In *Proceedings of the Third Annual European Symposium on Algorithms, September 25–27, 1995, Corfu, Greece*, ESA '95, pages 494–507. Springer, Heidelberg, Germany, September 1995.

[60] I. Newman, P. Ragde, and A. Wigderson. Perfect hashing, graph entropy, and circuit complexity. In *Proceedings of the Fifth Annual Structure in Complexity Theory Conference, July 8–11, 1990, Barcelona, Spain*, pages 91–99. IEEE Computer Society, Los Alamitos, California, July 1990.

[61] A. Papakostas and I. G. Tollis. Algorithms for area-efficient orthogonal drawings. *Computational Geometry*, 9(1–2):83–110, January 1998.

[62] H. Räcke. Optimal hierarchical decompositions for congestion minimization in networks. In *Proceedings of the Fortieth Annual ACM Symposium on Theory of Computing, May 17–20, 2008, Victoria, Canada*, STOC '08, pages 255–264. ACM, New York, USA, May 2008.

[63] O. Reingold. Undirected connectivity in log-space. *Journal of the ACM*, 55(4):17:1–17:24, September 2008.

[64] K. R. Reischuk. *Komplexitätstheorie Band I: Grundlagen: Maschinenmodelle, Zeit- und Platzkomplexität, Nichtdeterminismus.* Teubner Verlag, Leipzig, Germany, 1999.

[65] N. Robertson and P. D. Seymour. Graph Minors. XIII. the disjoint paths problem. *Journal of Combinatorial Theory*, 63(1):65–110, Graph Minors. XIII. The Disjoint Paths Problem 1995.

[66] N. Robertson and P. D. Seymour. Graph Minors. XXIII. Nash-Williams' immersion conjecture. *Journal of Combinatorial Theory*, 100(2):181–205, March 2010.

[67] M. Saks and R. Statman. An intersection problem for finte automata. *Discrete Applied Mathematics*, 21(3):245–255, October 1988.

[68] J. Textor. *Search and Learning in the Immune System: Models of Immune Surveillance and Negative Selection.* PhD thesis, Institute for Theoretical Computer Science of the University of Lübeck, July 2011.

[69] D. M. Thilikos, M. Serna, and H. L. Bodlaender. Cutwidth I: A linear time fixed parameter algorithm. *Journal of Algorithms*, 56(1):1–24, July 2005.

[70] R. van Bevern, A. E. Feldmann, M. Sorge, and O. Suchá. On the parameterized complexity of computing balanced partitions in graphs. In *Proceedings of the Thirty-Ninth International Workshop on Graph-Theoretic Concepts in Computer Science, June 19–21, 2013, Lübeck, Germany*, WG '13, pages 76–87. Springer, Heidelberg, Germany, June 2013.

[71] K. Wagner. Über eine Eigenschaft der ebenen Komplexe. *Mathematische Annalen*, 114(1):570–590, January 1937.

[72] H. T. Wareham. The parameterized complexity of intersection and composition operations on sets of finte-state automata. In *Proceedings of the Fifth International Conference of Implementation and Application of Automata, July 24–25 , 2000, London, Ontario, Canada*, CIAA '00, pages 302–310. Springer, Heidelberg, Germany, July 2000.

[73] D. R. Wood. Optimal three-dimensional orthogonal graph drawing in the general position model. *Theoretical Computer Science*, 299(1–3):151–178, April 2003.

[74] D. R. Wood. Minimising the number of bends and volume in 3-dimensional orthogonal graph drawings with a diagonal vertex layout. *Algorithmica*, 39(3):235–253, March 2004.