Electronic Colloquium on Computational Complexity, Revision 1 of Report No. 150 (2012)

On the Space Complexity of Parameterized Problems*

Michael Elberfeld Christoph Stockhusen Till Tantau

Institute for Theoretical Computer Science
Universität zu Lübeck
D-23538 Lübeck, Germany
{elberfeld,stockhus,tantau}@tcs.uni-luebeck.de

November 28, 2012

Abstract

Parameterized complexity theory measures the complexity of computational problems predominantly in terms of their parameterized *time* complexity. The purpose of the present paper is to demonstrate that the study of parameterized *space* complexity can give new insights into the complexity of well-studied parameterized problems like the feedback vertex set problem. We show that the undirected and the directed feedback vertex set problems have different parameterized space complexities, unless L = NL. For a number of further natural parameterized problems, including the longest common subsequence problem, the acceptance problem for multi-head automata, and the associative generability problem we show that they lie in or are complete for different parameterized space classes. Our results explain why previous attempts at proving completeness of different problems for parameterized *time* classes have failed.

Keywords: parameterized complexity theory, logarithmic space, fixed-parameter tractability, feedback vertex set

1 Introduction

When designing classical or parameterized algorithms, the focus often lies on the *time* complexity of a problem rather than its *space* complexity. Nevertheless, the study of space classes like logarithmic space is an integral part of classical complexity theory since many natural problems (like reachability and distance problems in graphs or satisfiability problems for

*This is the technical report version of the conference paper [11].

ISSN 1433-8092

powerful logics) do not appear to be complete for time classes like P or NP, rather they turn out to be complete for space classes like L, NL, or PSPACE.

It stands to reason that we may also expect some parameterized problems to be complete not for standard parameterized time classes like FPT or W[1] or XP, but rather for parameterized space classes. Furthermore, by analogy to findings from classical complexity theory, we may expect that parameterized problems with low space complexity can be solved quickly. A typical result supporting this reasoning is that the parameterized vertex cover problem has low space complexity—it lies in the class para-L [5, 12]—and, indeed, this problem can be solved *very* quickly in the parameterized setting.

In the present paper we investigate the space complexity of a number of natural parameterized problems whose space complexity has not yet been studied in this context. We will argue that the differences in their space complexities are the reasons why the problems could not be shown to be complete for standard parameterized time classes. Moreover, our results suggest a possible explanation of why the problems vary with respect to how quickly they can be solved in practice even though they lie in the same parameterized time class. As prominent examples, we compare the directed and the undirected version of the feedback vertex set problem as well as the treewidth problem. All of these problems are indistinguishable with respect to the parameterized time classes they belong to (namely FPT), but with regard to their space complexity they can be classified using new natural complexity classes that reflect their different complexities. As a corollary of these efforts we obtain that the directed and the undirected feedback vertex set problem have different parameterized space complexity, unless L = NL.

Previous research on parameterized space complexity [5, 7, 12] was directed at the logspace analogues para-L and para-NL of para-P, which is commonly known as FPT, as well as at the logspace analogues XL and XNL of XP. When one tries to determine the parameterized space complexity of natural parameterized problems, it quickly turns out, however, that these space classes do not suffice to paint a complete picture of the complexity landscape of the parameterized world. For this reason, we introduce new classes that are motivated by parameterized versions of natural reachability problems. An overview of the classes and our containment and completeness results is given in Figure 1.

Related Work. In 1997, Cai et al. [5] introduced the class UNIFORM-LOGSPACE+ADVICE, which is the same as para-L, and they also considered its nondeterministic version. They showed that several problems in para-P also belong to para-L or para-NL, including the naturally parameterized vertex cover problem, which lies in para-L, and the parameterized k-leaf spanning

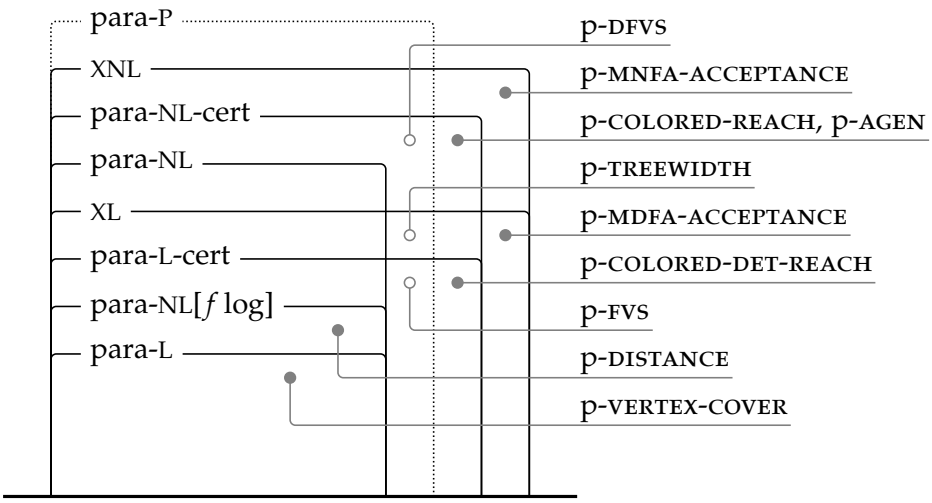


Figure 1: Overview of the parameterized space classes studied in this paper and parameterized problems that lie in them. A filled circle indicates that the problem does not only lie in the class, but is even complete for it under para-L-reductions.

tree problem, which lies in para-NL. In 2003, Flum and Grohe [12] continued the study of para-L and para-NL by showing that the parameterized model-checking problem of first-order formulas on graphs of bounded degree lies in para-L. This implies that many standard parameterized graph problems belong to para-L when we restrict our attention to bounded-degree graphs. In this context they asked, which other problems in para-P also belong to para-NL or para-L, and whether there are problems that are contained in para-P but are not contained in para-L or para-NL under the assumption that para-P $\neq$ para-L. In this paper, we answer this question affirmatively by presenting natural problems that belong to para-P, but are not contained in para-L under standard assumptions.

Organization of this Paper. In Section 2 on preliminaries we fix the terminology and review the definitions of parameterized space classes from the literature as well as of parameterized reductions. In Section 3 we investigate parameterized problems that are known to be *tractable* (that is, they are known to lie in para-P $=$ FPT), but whose space complexity has not yet been investigated. The objective of this section is to get a deeper understanding of why some problems inside para-P appear more difficult than others even though they are all tractable. Here we focus on the distance problem in directed graphs, and the directed and the undirected version of the feedback vertex set problem. We show that the distance problem characterizes a parameterized complexity class that

captures the complexity of many well-known problems of the class NL with respect to natural parameters. For the feedback vertex set problems we are able to show that the directed and the undirected version are of provable different complexity, unless L = NL, which reflects the different approaches required to prove that these problems are fixed parameter tractable. For this, we introduce two new parameterized complexity classes, para-L-cert and para-NL-cert. In Section 4 we turn our attention to parameterized problems that presumably lie outside of para-P. This time, our findings on the space complexity of natural problems provide explanations why researchers have failed to prove completeness of these problems for natural parameterized time classes like W[1] or XP: The problems turn out to lie in or to be complete for parameterized space classes. To be more concrete, we show that the longest common subsequence problem parameterized via the number of given strings is contained in XNL and therefore will not be complete for XP. For both XL and XNL we present natural complete problems, namely the problem of deciding whether a multi-head (nondeterministic) finite automaton accepts a given word. Moreover, we show that the problem of finding a set of colors of an edge-colored directed graph such that there is an *s*-*t* path made up from these colors is complete for para-NL-cert (or para-L-cert if we require the *s*-*t* path to be deterministic). Furthermore, we prove para-NL-cert-completeness for the associative generability problem, i.e. given a binary and associative operation, a universe, and a natural number k, answering the question whether there is a set of k elements of the universe such that the whole universe can be generated from these elements by the given operation.

2 Preliminaries

A *parameterized problem* is a pair (Q, κ) of a language $Q \subseteq \Sigma^*$ and a parameterization $\kappa\colon \Sigma^* \to \mathbb{N}$ that maps input instances to natural numbers, their parameter values. In the classical definition, Downey and Fellows [9] require the parameterization to be computable, while Flum and Grohe [13] require it to be computable in polynomial time. In this paper we need the binary representation of the parameter number to be computable in logarithmic space. This is the case, in particular, when the parameter is specified explicitly within the input.

For a classical complexity class C, a parameterized problem (Q, κ) belongs to the *para-class* para-C if there is an alphabet Π, a computable function $\pi\colon \mathbb{N} \to \Pi^*$, and a language $A \subseteq \Sigma^* \times \Pi^*$ with $A \in C$ such that for all $x \in \Sigma^*$ we have $x \in Q \iff \big(x, \pi\big(\kappa(x)\big)\big) \in A$. Flum and Grohe [13] phrase this as *"(Q, κ) is in C after a precomputation on the parameter"*. In particular, FPT is the same as para-P and all fixed-parameter tractable

problems are in P after a precomputation on the parameter. Analogously, we can define the classes para-L and para-NL as the family of problems that are in L and in NL after a precomputation on the parameter, respectively. Equivalently, a parameterized problem (Q, κ) over Σ is in para-L (para-NL), if there is a function $f\colon \mathbb{N} \to \mathbb{N}$ such that the question $x \in Q$ can be decided within (nondeterministic) space $f\big(\kappa(x)\big) + O\big(\log |x|\big)$.

A different way of defining a parameterized version of a classical complexity class C is to consider its so-called X-class [9]. A problem (Q, κ) is in XC if for every number $w \in \mathbb{N}$ the slice $Q_w = \{\, x \mid x \in Q \text{ and } \kappa(x) = w \,\}$ lies in C. It is immediate from the definition that para-$C \subseteq$ XC holds. The class XP is widely used in parameterized complexity theory; the logarithmic space classes XL and XNL have previously been studied by Chen, Flum, and Grohe [7, 12].

In order to compare the complexity of parameterized problems, we use parameterized logspace reductions, called para-L-reductions in the following. Following Flum and Grohe, we define them similarly to FPT-reductions: Let (Q_1, κ_1) and (Q_2, κ_2) be parameterized problems over some alphabets Σ_1 and Σ_2. A para-L-*reduction* is a mapping $R\colon \Sigma_1^* \to \Sigma_2^*$ such that

1. for all $x \in \Sigma_1^*$ we have $x \in Q_1 \iff R(x) \in Q_2$,
2. $\kappa_2\big(R(x)\big) \leq g\big(\kappa_1(x)\big)$ for some computable function $g\colon \mathbb{N} \to \mathbb{N}$,
3. R is para-L-computable with respect to κ_1.

By standard arguments one can show that all classes in this paper are closed with respect to para-L-reductions.

3 Space Complexity of Tractable Problems

From the perspective of classical FPT-theory, the complexity of a parameterized problem is "settled in principle" once it has been shown to be solvable in time $f\big(\kappa(x)\big) \cdot O(n^c)$; further research then focuses on making f as slowly growing as possible. In the following, instead of trying to differentiate between problems in para-P through their f-functions, we try to determine differences caused by their different space complexities. For this we can use complexity classes that are contained in para-P, but also the intersections with classes that are not fully contained in para-P. In the following, we give examples for such classes and their applications.

3.1 Parameterized Distance Problems

We start our exploration with a deceptively simple problem, namely the distance problem with the distance as the parameter:

Problem 1 (p-DISTANCE)**.**

Instance. A directed graph G, two vertices s and t of G, and a natural number l.

Parameter. l.

Question. Is there a path from s to t in G of length at most l?

This problem is clearly in para-P and, since the distance problem lies in NL, also in para-NL. It might also seem like a good candidate for a para-NL-complete problem. Flum and Grohe [12] showed that classes like para-P, para-NL, and para-L are "uninteresting" from the parameterized point of view in the sense that complete problems for the underlying classical complexity classes are always complete for the parameterized versions when considering the trivial parameterization. In particular, the standard distance problem for directed graphs is complete for para-NL with the trivial parameterization $\kappa(x) = 1$. However, this argument does not carry over to the above version of the distance problem with its more natural parameterization.

In order to describe the complexity of p-DISTANCE precisely, it turns out that a new class is needed that is derived from transferring the definition of the class W[P] to the space setting in a certain way. A classical definition of W[P] is as the para-P-reduction closure of the weighted circuit satisfiability problem [1, 9]. Alternatively, W[P] is also the class of all problems (Q, κ) that are solvable by NTMs deciding Q in para-P-time, making at most $f(\kappa(x)) \cdot O(\log |x|)$ nondeterministic steps on any input x. Finally, W[P] also contains exactly the problems decidable via DTMs that are provided with a proof certificate of length $f(\kappa(x)) \cdot O(\log |x|)$ (the certificate is on a special read-only tape and the machine must accept all $x \in Q$ for some certificate and must reject $x \notin Q$ for all certificates).

In the context of parameterized time complexity all of the three characterizations of W[P] yield the same classes. When we look at the space analogues of these classes, the situation is somewhat different: First, the para-L-reduction closure (rather than the para-P-reduction closure) of the parameterized weighted circuit satisfiability problem is still W[P], which can be seen from a result of Chen et al. [7]. Second, when we replace the para-P time bound in the second characterization by a para-L space bound, a subclass of para-NL arises. Third, when we make the same replacement in the third definition, a presumably different class arises that is a subclass of XNL.

Definition 2. A parameterized problem (Q, κ) is in the class para-NL[$f \log$] if it can be decided by a para-NL Turing machine that makes at most $f(\kappa(x)) \cdot O(\log |x|)$ many nondeterministic steps.

Definition 3. A parameterized problem (Q, κ) is in the class para-L-cert if it can be decided by a para-L Turing machine which is provided with proof certificates of length $f\big(\kappa(x)\big) \cdot O\big(\log |x|\big)$.

It turns out that para-NL[f log] is precisely what we need to characterize the complexity of the distance problem with its natural parameterization:

Theorem 4. p-DISTANCE *is complete for* para-NL[f log] *with respect to* para-L *reductions.*

Proof. To see that p-DISTANCE $\in$ para-NL[f log] consider the following algorithm: Starting from s, guess l times a node v that is reachable from the currently considered node. Accept if after at most l steps the target node is reached, otherwise reject. This algorithm is correct because it only accepts if there is a path from s to t of length at most l, and if there is such a path, it will be found by the nondeterministic guessing.

A Turing machine that computes this algorithm needs logarithmic space to store the currently considered node and the next node. Checking whether two nodes are connected can also be done in logarithmic space. For storing the current value of the counter for the at most l steps, the machine needs polynomial many bits (if l is encoded in binary in the input) which is captured by the para-L space bound $f(l) + O\big(\log |x|\big)$ if f is chosen accordingly. Furthermore, the machine has to guess l times a next node which results in $O\big(l \cdot \log |x|\big)$ nondeterministic steps. Altogether, this is a para-NL[f log] Turing machine.

For hardness, let (Q, κ) be a parameterized problem that is contained in para-NL[f log] via a machine M. Using standard techniques, we may assume that M has exactly one accepting and one rejecting configuration, respectively. Furthermore, we can assume that the configuration graph is a directed acyclic graph. The configuration graph of M on an input x has the size

$$O(1)^{f\left(\kappa(x)\right)+O\left(\log |x|\right)} = f'\big(\kappa(x)\big) \cdot |x|^c$$

for some function f' and some constant c. In order to transform the configuration graph into an instance for p-DISTANCE, we must tackle the problem that the length of the path from the initial configuration to the accepting configuration is at most the size of the configuration graph and, therefore, not exclusively bound by the parameter, but also by the length of the input x. However, the number of nondeterministic steps of M is bounded by $f\big(\kappa(x)\big) \cdot O\big(\log |x|\big)$, on every path from the initial configuration, so there are at most this many nodes with out-degree greater than 1. Let us call the nodes with out-degree greater than 1 and the nodes corresponding to the initial and the accepting configuration the *red nodes* and the other nodes the *black nodes* of the graph.

In a first processing step, we iterate over the nodes of the configuration graph and replace every outgoing edge of every red node u_r that points to a black node u_b by a new edge from u_r to the first red node v_r that is reachable from u_b. Now we drop the black nodes. In the resulting graph, every path from the initial to the final configuration has length at most $f\big(\kappa(x)\big) \cdot O\big(\log|x|\big)$. In a second step, we shorten the paths in the resulting graph by augmenting the graph with edges from nodes u to v if there is a path from u to v of length at most $\log|x|$. Since the outdegree of the nodes is bounded by some constant that only depends on M, every path of length $\log|x|$ can be described by $O\big(\log|x|\big)$ bits. The augmentation can then be done by an iteration over all the nodes v of the graph and enumerating all paths of length $\log|x|$ starting from v. In the resulting graph, there is a path of length $f\big(\kappa(x)\big)$ from the initial configuration to the accepting configuration if, and only if, there is a path from the initial configuration to the accepting configuration in the original configuration graph. Hence, we obtain the desired para-L-reduction, as the processing steps can be done by para-L Turing machines.

□

In general, para-NL[f log] seems to contain many parameterized versions of problems that are contained in NL in the classical sense and that follow the guess-and-forget approach, i.e. guessing an element from the input, processing this element, and guessing the next element while forgetting the previous one. With this observation, many NL-complete problems can be shown to be complete for para-NL[f log] with natural parameterizations, e.g. many problems from [18]. On the other hand, para-NL[f log] and para-NL do not coincide under standard assumptions. This can be shown with the observation that every NL-complete problem is complete for para-NL with respect to the trivial parameterization.

Theorem 5. para-NL[f log] = para-NL *if, and only if,* L = NL.

Proof. Recall that the reachability problem for directed graphs is complete for NL and therefore its parameterized version is complete for para-NL together with the trivial parameterization $\kappa_0(x) = 0$. First assume that L = NL holds. From the definition of the para-classes we then have para-L = para-NL which implies para-NL[f log] = para-NL. Now assume para-NL[f log] = para-NL. Then the reachability problem can be solved by a para-NL[f log] Turing machine within space $f\big(\kappa(x)\big) + O\big(\log|x|\big)$ making only $f\big(\kappa(x)\big) \cdot O\big(\log|x|\big)$ nondeterministic choices. For the trivial parameterization this is a constant amount which can be simulated within logarithmic space. This implies L = NL. □

3.2 Feedback Vertex Set Problems

While it is useful to know that para-NL[f log] has many natural complete problems, our original goal for looking at parameterized space classes was to study the parameterized space complexity of well-studied problems in para-P that do not come from the logspace world in the way p-DISTANCE does. We now look at the problem of identifying a set S of vertices of a given graph G such that $G - S$ has treewidth at most w for small natural numbers w. For $w = 0$ this is the vertex cover problem. For $w = 1$ we obtain the (undirected) feedback vertex set problem p-FVS. We also look at the problem of deciding whether a given graph has treewidth w for a given w.

The first problem of this "treewidth hierarchy" has been studied in the context of parameterized space complexity theory by Cai et al. [5], who showed that p-VERTEX-COVER $\in$ para-L. We look at the next problem of this hierarchy, the feedback vertex set problem p-FVS. It does not seem to lie in para-L nor even in para-NL. However, it turns out that the class para-L-cert contains it:

Theorem 6. p-FVS $\in$ para-L-cert.

Proof. On input of an undirected graph G, our para-L-cert Turing machine M interprets its certificate of length $\kappa(x) \cdot \lceil \log |V| \rceil$ as the description of the k vertices of the feedback-vertex set. Then M removes these vertices and its incident edges from G and checks whether the resulting graph G' is acyclic. Both, removal and test for acyclicity can be done by logspace Turing machines [8]. With the fact that para-L-cert is closed with respect to para-L-reductions, we obtain the result. □

While the classes para-NL[f log] and para-NL are contained in para-P, this is not the case for para-L-cert under the standard assumption para-P $\subsetneq$ W[SAT] since the weighted satisfiability problem for propositional formulas belongs to para-L-cert.

Theorem 7. *If* para-L-cert $\subseteq$ para-P, *then* W[SAT] = para-P.

Proof. Recall that W[SAT] is defined as the para-P-reduction closure of the weighted satisfiability problem for propositional formulas [9]. Also note that the weighted satisfiability problem for propositional formulas itself is contained in para-L-cert: For a given formula F and a parameter k, a weight-k satisfying assignment can be encoded within a certificate of length $k \cdot \lceil \log n \rceil$ if F has n variables. Therefore, a para-L-cert Turing machine only has to verify the given certificate by evaluating F which can be done in logarithmic space [19]. Clearly, if p-WEIGHTED-SAT $\subseteq$ para-P, we also have W[SAT] = para-P. □

In particular, p-FVS is not complete for para-L-cert, unless para-P $=$ W[SAT], which is considered to be unlikely (para-L-cert does, however, have natural complete problems as shown in Section 4). Nevertheless, para-L-cert turns out to be useful for contrasting the complexity of the feedback vertex set problem for undirected graph to the version for directed graphs.

The undirected version was shown to be fixed-parameter tractable in 1993 by Bodlaender [2]. The fixed-parameter tractability of the directed version remained an open problem until Chen et al. [6] presented a para-P-algorithm in 2008. So far, these apparently different complexities could only be felt by looking at the complexity of the proofs. In the setting of parameterized logarithmic space, the different complexities are not only mirrored, we are even able to prove that p-FVS and p-DFVS are of different complexity (under standard assumptions):

Theorem 8. p-DFVS $\in$ para-L-cert *if, and only if,* L $=$ NL.

Proof. To prove the forward direction, we show that the parameterized problem p-UNREACH, the complement of the reachability problem where we ask whether there does *not* exist a path from a given vertex s to another vertex t in a directed graph parameterized by the trivial parameterization, para-L-reduces to p-DFVS. The reduction works in two steps: The first step takes the n-vertex input graph and makes it acyclic by producing n copies of its vertices, conceptually arranging them as n layers from left to right, and drawing edges between consecutive layers from left to right in the same way as they are present in the original graph. The second step inserts an edge from t's copy in the last layer to s's copy in the first layer. The resulting directed graph remains acyclic (has a feedback vertex set of size 0) exactly if there is no path from s to t in the input graph.

Now assume p-DFVS $\in$ para-L-cert. This implies the existence of a para-L Turing machine that, provided with an appropriate certificate of length $f(\kappa(x)) \cdot O(\log |x|)$, decides p-UNREACH. But because p-UNREACH is parameterized in the trivial way, this Turing machine is in fact a deterministic logspace Turing machine that decides the NL-complete unreachability problem in directed graphs. This implies L $=$ NL.

For the backward direction assume L $=$ NL. Then the algorithm used above for p-FVS also works for p-DFVS because cycle detection in directed graphs is then possible in deterministic logarithmic space. □

This raises the question, what class captures p-DFVS? The central difference between p-FVS and p-DFVS is the complexity of the underlying cycle detection problem: In the undirected case this can be done in deterministic logarithmic space, in the directed case nondeterministic logarithmic space is required. Therefore, the natural approach is to consider a nondeterministic version of para-L-cert, namely para-NL-cert: This class is defined in

a similar way as para-L-cert, but here we allow para-NL Turing machines. We immediately obtain the following result:

Theorem 9. p-DFVS $\in$ para-NL-cert.

For the same reasons as in the undirected case, we will not be able to prove that p-FVS is complete for para-L-cert.

Many more problems from para-P are contained in para-L-cert and its nondeterministic version. In fact, every problem that can be decided via a bounded search tree algorithm [9], where the aim is to find a root-to-leaf path within a search tree whose depth depends on the parameter, is a promising candidate to be contained in para-L-cert, because the certificate of a para-L-cert Turing machine can describe the desired root-to-leaf path.

3.3 The Treewidth Problem and Slicewise Logarithmic Space

We conclude this section with a study of the treewidth problem: Given a graph G and a natural number w as the parameter, the question is whether G has treewidth at most w. This problem is fixed-parameter tractable [3], and a recent result [10] showed that for every fixed w this problem is decidable in deterministic logarithmic space. These theorems immediately give us the following result:

Theorem 10. p-TREEWIDTH $\in$ XL.

Note that we will not be able to show that p-TREEWIDTH is complete for XL unless para-P $=$ W[SAT]. The following theorem relates XL and XNL to the surrounding complexity classes.

Theorem 11.

1. para-L-cert $\subseteq$ XL *and* para-NL-cert $\subseteq$ XNL,
2. para-NL-cert $\subseteq$ XL *implies* L $=$ NL.

Proof. For the first statement consider any para-L-cert Turing machine M and note that this machine can be simulated by an XL Turing machine via enumeration of all possible certificates. The same argument works in the nondeterministic case.

For the second statement just note that the NL-complete reachability problem for directed graphs together with the trivial parameterization is contained in para-NL. Therefore, para-NL $\subseteq$ XL implies L $=$ NL, which is actually a stronger statement than the one stated above. □

4 Space Complexity of Intractable Problems

Most of the classes discussed in the previous section are not known to be contained in para-P. In fact, these classes contain problems that are far

above of para-P inside the W-hierarchy, for example the clique problem or the weighted satisfiability problem for propositional formulas, both of which are contained in para-L-cert. Therefore, a natural question is to ask for complete problems for these classes. In this section, we will give natural complete problems for the classes that are not fully contained in para-P, that is, para-L-cert, para-NL-cert, XL, and XNL.

4.1 Slicewise Logarithmic Space

So far, only few problems are known to be complete for XL or XNL. Chen, Flum, and Grohe gave the first complete problems for these classes based on Turing machine simulations. They studied the compact Turing machine computation problem for deterministic and nondeterministic Turing machines, p-CTMC and p-CNTMC, respectively [7]. The p-CTMC problem is defined as follows: On input of an encoding of a deterministic Turing machine M, a string x over M's alphabet, and a natural number k in binary, the question is if there is an accepting computation of M on x that uses at most k work tape cells, where k is the parameter. The problem p-CNTMC is defined in the same way, but with respect to nondeterministic Turing machines.

We add the following natural automata evaluation tasks to the list of problems complete for XL and XNL.

Problem 12 (p-MDFA-ACCEPTANCE)**.**

Instance. The code of a deterministic finite two-way automaton A with k heads and an input string x.
Parameter. k.
Question. Does A accept x?

The problem p-MNFA-ACCEPTANCE is defined in the same way, but with respect to nondeterministic automata.

Theorem 13.

1. p-MDFA-ACCEPTANCE *is complete for* XL *with respect to* para-L *reductions.*
2. p-MNFA-ACCEPTANCE *is complete for* XNL *with respect to* para-L *reductions.*

Proof. Hartmanis showed that $A \in$ L holds if, and only if, there is a deterministic multi-head automaton that decides A [15]. The basic idea is that the position of an automata's head on the input tape can be used to store a number between 0 and n, where n is the input tape length, and, thus, the position of a head can store up to $\log n$ bits of information. A fixed number of heads hence suffice to store the information of the work tape of a machine using $O(\log n)$ space and modifications of this work tape correspond to appropriate sequences of auxiliary heads computing the right number of steps to be made by one of the heads. The details of the

construction are not important for proving the theorem; it suffices to note that p-MDFA-ACCEPTANCE lies in XL because the construction of Hartmanis is uniform. To show hardness, given a parameterized problem $(Q, \kappa) \in \text{XL}$ that is solved by a machine in space $f(\kappa(x)) \cdot O(\log |x|) + f(\kappa(x))$, by Hartmanis's result there is a multi-head two-way automaton deciding Q whose number of heads depends only on $f(\kappa(x))$. A para-L-reduction must simply map the input x to this automaton together with x. The proof for the nondeterministic version is analogous. □

The next results show that, outside of para-P, the classes XL and XNL enable us to make statements about problems that could not be classified within the W-hierarchy. Consider the following well-known problem of finding a longest common subsequence of k strings, parameterized by k:

Problem 14 (p-LCS)**.**

Instance. A set $\{s_1, \ldots, s_k\}$ of strings over an alphabet encoded in the input and a natural number l.
Parameter. k.
Question. Is there a subsequence of the given strings with length at least l?

Bodlaender et al. showed that p-LCS is hard for $\text{W}[t]$ for every level t [4]. Pietrzak showed that the variant p-FLCS, where the alphabet is fixed, is hard for $\text{W}[1]$ [20]. Pietrzak also conjectured that an exact classification of this problem within the parameterized hierarchy is not possible, because it seems not to be contained in $\text{W}[\text{P}]$, the highest class of the W-hierarchy. While classical dynamic-programming approaches show that the longest common subsequence problem is contained in XP, completeness for XP could not be shown. We give a strong argument that this is not possible using the class XNL:

Theorem 15. p-LCS $\in$ XNL.

Proof. To decide on a given input of k strings whether there exists a longest common subsequence of length l, the XNL Turing machine scans the first string from left to right and guesses the subsequence, using a pointer to a symbol in the first string. Using $k - 1$ additional pointers on the remaining strings, the machine verifies that the guessed subsequence is also contained within the $k - 1$ remaining strings. □

By definition, X-classes inherit their inclusion structure from their underlying complexity classes. From this, we immediately get the following theorem:

Theorem 16. p-LCS *is not complete for* XP, *unless* NL = P.

Because p-LCS is hard for every level of the W-hierarchy it is now tempting to conclude $W[t] \subseteq XNL$ for every t, but this is not the case. The error in this argument is that the hardness of p-LCS has been shown with respect to para-P reductions, but XNL is only closed with respect to para-L reductions. On the other hand, this shows that under standard assumptions it will not be possible to prove that p-LCS is hard for the levels of the W-hierarchy using only para-L reductions, because this would imply NL = P. However, it is an open question whether one can show that p-LCS is complete for XL or XNL. Recently, Guillemot showed that p-LCS is complete for the class WNL, a superclass of $W[t]$ for every t [14]. However, this class is somehow orthogonal to XL and XNL since WNL is defined as the para-P reduction closure of an underlying machine acceptance problem and therefore not contained in XL or XNL. Nevertheless, the presented reductions are in fact para-L reductions and, hence, Guillemot also shows completeness for the para-L version of WNL, where we consider the para-L reduction closure of the underlying problem.

4.2 Problems for Parameterized Logspace with Certificates

In the previous section we used the classes para-L-cert and para-NL-cert to upper bound the parameterized space requirements of the fixed-parameter tractable problems p-FVS and p-DFVS, respectively. These problems are not complete for the corresponding classes under standard assumptions; in the present section we identify natural complete problems for the classes para-L-cert and para-NL-cert.

The reachability problem, which asks whether there exists a path from a start to a target vertex in a given directed graph, is among the most prominent problems in the study of logarithmic space-bounded computations. Its general version is NL-complete [16] and becomes L-complete if the path we ask for is *deterministic* [8]; that means, every vertex on the path has only a single outgoing edge in the graph. In the same way as one can understand the transition from the classical complexity class P to its parameterized version W[P] by considering the complete circuit evaluation problem for the former and the complete parameterized weighted circuit satisfiability problem for the later class, we use the following variant of the reachability problem to better understand the transition from the classical classes L and NL to their certificate-based parameterized counterparts para-L-cert and para-NL-cert, respectively.

Problem 17 (p-COLORED-REACH)**.**

Instance. A vertex set V, two vertices $s, t \in V$, a set of multi-colored edges $E \subseteq V \times V$, and a natural number k.

Parameter. k.

Question. Is there a set of k colors, such that there is a path from s to t that uses only edges that have at least one of the chosen colors.

Let p-COLORED-DET-REACH denote the variant where the colors induce a path from s to t whose inner nodes all have out-degree 1.

Theorem 18. p-COLORED-REACH *is complete for* para-NL-cert *with respect to* para-L *reductions.*

Proof. To prove p-COLORED-REACH $\in$ para-NL-cert, we use a Turing machine that is given a binary certificate of length $k \cdot O\big(\lceil \log n \rceil\big)$ along with its input. It solves p-COLORED-REACH by interpreting the certificate as a description of how to choose k colors and nondeterministically guesses a path from s to t in the graph whose edges have at least one of these colors.

For proving hardness of p-COLORED-REACH for para-NL-cert, let M be a para-NL-cert machine that, in addition to its input x, accesses a certificate of length $f\big(\kappa(x)\big) \cdot O\big(\log |x|\big)$ on a certificate tape. Since our para-L reduction does not know the right certificate in advance nor can enumerate all possible certificates due to its limited amount of space, it is not able to build a configuration graph for M. Instead, it constructs a *partial configuration graph* in the form of a p-COLORED-REACH instance. For this graph, choosing a certificate for M that leads to an accepting computation corresponds to choosing k colors, such that there is a path from s to t.

The instance of p-COLORED-REACH is constructed as follows: The set of vertices V is made up by all *partial configurations*, each describing the positions of M's heads on the tapes and its contents, while we leave out the content of the certificate tape. To encode the behavior of M on different certificates we use different colored edges. We divide the certificate into $f\big(\kappa(x)\big)$ blocks of logarithmic length and assign to each block b and each block content c a new color $\mathrm{col}(b, c)$. Now we insert directed, colored edges into the partial configuration graph depending on the transitions of M: Assume that M makes a transition from the configuration corresponding to v to the one corresponding to w and that M requires the i-th symbol of the certificate to be σ for this transition. Then, we insert an edge from v to w that is colored with all the colors $\mathrm{col}(b, c)$ such that the i-th position of the certificate is contained in block b and the block content c has symbol σ at the according position. We proceed like this for all pairs of partial configurations and its corresponding blocks and colors.

From this construction we have that if there is a certificate such that M accepts on its input, then there is also a set of $f\big(\kappa(x)\big)$ colors that encodes

the certificate and that makes up a path from the vertex s that corresponds to the initial configuration to the vertex t that corresponds to the accepting configuration. (Note that we may assume that M has exactly one initial and exactly one accepting partial configuration.) For the backward direction we have to do some more work, because now there might be two colors $\mathrm{col}(b, c)$ and $\mathrm{col}(b, c')$ such that there is an accepting path in the graph if we choose these colors, but these colors do not correspond to a single certificate since this would mean that the certificate consists of two ore more contents of a block. To tackle this, we insert a path of vertices $v_1, \ldots, v_{f(\kappa(x))+1}$ into the graph and identify $v_{f(\kappa(x))+1}$ and s. Moreover, we connect the vertices v_i and v_{i+1} with an edge and color the edges (v_i, v_{i+1}) with the colors $\mathrm{col}(i, c)$ for every possible block content c. Finally, we make v_0 our new starting node. In order to get from v_0 to t, we now have to choose colors such that we first get a path from v_0 to s. These colors have to be colors $\mathrm{col}(i, c_i)$ for every $i \in \{1, \ldots, f(\kappa(x))\}$. By this, in the constructed graph there is a path from v_0 to t that requires $f(\kappa(x))$ colors if, and only if, there is a certificate such that M accepts its input.

Since all partial configurations can be enumerated by a para-L machine and there are only a polynomial number of possible strings for each block of logarithmic length, the whole reduction can be computed by a para-L machine. □

The following theorem immediately follows from the proof of the previous one and the fact that we consider deterministic instead of nondeterministic Turing machines for para-L-cert.

Theorem 19. p-COLORED-DET-REACH *is complete for* para-L-cert *with respect to* para-L *reductions.*

Note that, by slightly changing the reduction given above, we can show that even the restrictions of p-COLORED-REACH and p-COLORED-DET-REACH to graphs where every edge has exactly one color is hard for para-NL-cert and para-L-cert respectively. To see this, observe that we only have to replace every edge $u \to w$ with colors $c_1, \ldots, c_l$ by a family of l nodes and connect them with edges $u \to v_i$ and $v_i \to w$ where we color every edge pair $u \to v_i$ and $v_i \to w$ with color c_i. Also note that this construction does not affect the property that in the para-L-cert case all nodes have outdegree 1 for an appropriate choice of colors such that there is a path from s to t, because every edge has only colors that correspond to the content of a specific block and no two colors from the same block are chosen due to the construction above. Therefore, we can conclude that the problems p-COLORED-REACH and p-COLORED-DET-REACH are complete for para-NL-cert and para-L-cert respectively when restricting the family of input graphs to graphs where every edge has exactly one color.

4.3 Parameterized Generator Problems

The colored reachability problem introduced above is a convenient tool to understand the difference between para-NL-cert and para-L-cert by considering the same problem with different kinds of paths. In the present section study a problem whose variants characterize the computational power of W[P] and para-NL-cert, the parameterized version of the generability problem.

Problem 20 (p-GEN)**.**

Instance. A finite set U, a binary operation $\circ\colon U \times U \to U$, and a natural number k.
Parameter. k.
Question. Is there a subset $G \subseteq U$ of size k, such that every element $e \in U$ can be generated by applying $\circ$ to elements of G.

The non-parameterized version of this problem where G is part of the input and does not need to be computed, is central to the study of P and its subclass NL: it is P-complete in general [17], and its restriction to associative $\circ$, the problem AGEN, is NL-complete [18]. The parameterized version of the generability problem exhibits a similar behaviour. Flum and Grohe [13] showed that p-GEN is complete for W[P] and we show that its restriction to associative operations, the parameterized problem p-AGEN, is complete for para-NL-cert. Note that the non-parameterized generability problem where k is part of the input, but not G, is NP-complete; both in the general and the associative variant.

Theorem 21. p-AGEN *is complete for* para-NL-cert *with respect to* para-L *reductions.*

Proof. A para-NL-cert Turing machine for this problem interprets the $k \cdot O(\lceil \log n \rceil)$ bits of its certificate as the description of the k elements of G, the set of generators. The machine iterates over all elements of the universe and for each element e it successively guesses a sequence $g_1, \ldots, g_l$, such that $g_1 \circ g_2 \circ \cdots \circ g_l = e$. Note that the maximum length of such a sequence is at most $|U|$, because otherwise we could shorten the sequence. Therefore, p-AGEN $\in$ para-NL-cert.

For hardness, we para-L-reduce p-COLORED-REACH to p-AGEN. The basic idea is, like in [18], to encode the transitive closure of the given graph into the universe of the p-AGEN instance. If it is possible to select k colors such that there exists a path from s to t in the given graph that is made up from these colors, then there is a set of k elements in the universe of the p-AGEN instance that represent these colors and that enable us to create all the edges that have at least one of these colors. These edge elements can then be used to generate transitive edges. Moreover, if we are able

to generate the edge from s to t, then we can generate all elements of the universe. Now the crucial part is to force that the k elements have to be k elements of the universe, that encode the corresponding colors and that it is not possible to just choose the edge element from s to t that can be used to generate the whole universe. To achieve this, we first apply some preprocessing to the input graph and, then, reduce from the preprocessed instance.

Recall that we can restrict our attention to graphs where every edge has exactly one color. Let G be the currently considered input graph with the two vertices s and t. We first add two vertices s' and t' with edges $s' \to s$ and $t \to t'$. Then, for every vertex v of $G - \{s, t, s', t'\}$ we add edges $t \to v$ and $v \to s$ if they are not already present. Moreover, we color all edges of these two types and the edges $s' \to s$, $t \to t'$ with a new, special color c_+, where we overwrite existing colors. Let us denote the resulting graph by G'. With this construction there is a path from s to t in G if, and only if, there is a path from s' to t' in G' and the transitive closure of the edge relation of G' consists of the complete graph for the vertices of $G - \{s', t'\}$, edges from s' to every other node, and edges from all nodes to t'. This construction raises the number of colors that have to be chosen to obtain a path by 1, i.e. this increases the parameter value only by a constant. In the following, we only consider instances of p-COLORED-REACH whose graphs are given in this normal form, since the preprocessing steps can be computed in logarithmic space.

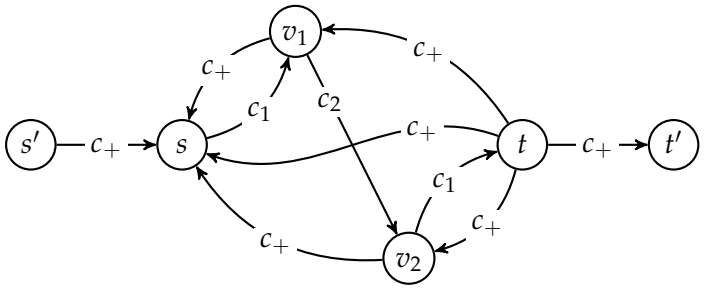
We now list the basic elements of the p-AGEN instance and the results from applying binary operations to them; this will define the remaining elements and results implicitly.

1. The *error element* `error`. We will use this to create dead ends in the evaluation of expressions. Any expression that contains the error element is evaluated to the error element. Moreover, any expression that does not make sense is evaluated to `error`, e.g. an expression that contains two end symbols (see below).
2. The *end* symbol $\lhd$. No expression can be evaluated to $\lhd$. Therefore, $\lhd$ has to be an element of any generating set G. We require this element for technical reasons that we will discuss later in the proof.
3. We define $|$ to be the *counter element*. Like in the case of the end symbol, this symbol cannot be generated by any expression and has to be an element of any generating set.
4. The elements $s_{i,j}$ for $i \in \{1, \ldots, k\}$ and every color c_j, except c_+, are called *selectors* and are conceptually arranged in k blocks. Here we enumerate the colors starting with 0 while we enumerate the blocks of selectors starting with 1 for technical reasons. We will enforce that, in order to be able to generate every element of U, we have to pick one element $s_{i,j}$ from every block i. For the color c_+ we define a special

selector element s_+. Like for the counter element, no expression will evaluate to s_+ and therefore this element has to be an element of the generating set, too. We will discuss the generation of the other selector elements later on.

5. For every pair u, v of vertices with $u \neq v$ of the given graph, we have an element $\langle u \to v \rangle$. We simply call these elements *edges*. Let us conceptually enumerate the l edges that share the same color c_j (except c_+) by $u_i \to v_i$ for $i \in \{1, \ldots, l\}$. Then we define $s_{i,j} \circ \mid \circ \mid \circ \cdots \circ \mid = \langle u_k \to v_k \rangle$ for every j and k counter elements with $k \leq l$. For $k > l$ we define that this expression evaluates to `error`. We define the analogous operation for the special color c_+ and its selector s_+. This means that we can generate an edge of a chosen color c_j by applying the appropriate amount of counters $\mid$ to an element $s_{i,j}$ or s_+. Moreover, we define $\langle v_1 \to v_2 \rangle \circ \langle v_2 \to v_3 \rangle = \langle v_1 \to v_3 \rangle$, that is, concatenating two edges e_1 and e_2, where e_1 points to the same vertex where e_2 starts from, results in the transitive edge from v_1 to v_3.
6. Because of the preprocessing, we do not need the edge element $\langle s' \to t' \rangle$ to generate any transitive edge and we can use it for some special applications. Hence, we refer to this edge also as the *star element* $\star$. We will use this element to generate the remaining selector elements that have not been chosen into the generating set by defining $s_{i,j} \circ \star = s_{i,(j+1 \bmod c)}$ where c is the number of colors in the graph, without c_+. This will be the only way to generate the selectors $s_{i,j}$ and therefore, we have to choose one of the selectors for each of the k blocks in order to be able to generate all of them. By setting the parameter to $k + 3$ we then may only choose exactly k normal selectors, the special selector s_+, the basic counter element $\mid$, and the end symbol $\lhd$. Otherwise we will not be able to generate the universe.

Let us take a look at an example. Consider the following graph that has the discussed normal form and whose underlying input graph contains a path from s to t, namely s, v_1, v_2, t, that is made up from choosing the colors c_1 and c_2. Let the original parameter, i.e. the number of colors that can be chosen, be 2. Moreover, assume that the edge $\langle s \to v_1 \rangle$ is the conceptually first and $\langle v_2 \to t \rangle$ is the second edge with color c_1, and that the edge $\langle s' \to s \rangle$ is the first and $\langle t \to t' \rangle$ is the second edge with the special color c_+.



For this graph, the expression $s_+|s_{1,1}|s_{2,2}|s_{1,1}||s_+|| \lhd$ evaluates to the transitive edge $\langle s' \to t' \rangle$. Note that the result of this expression is unambiguous. Let us consider another, more involved example:

$$\underbrace{s_{1,1}\ \underbrace{s_+|}\ \underbrace{s_{1,1}|}\ \underbrace{s_{2,2}|}\ \underbrace{s_{1,1}||}\ \underbrace{s_+||}\ \underbrace{s_+|s_{1,1}|s_{2,2}|s_{1,1}||s_+||}_{\star}\ \lhd}_{s_{1,1+2 \bmod 2}}$$
$$\underbrace{\langle s' \to s \rangle\ \langle s \to v_1 \rangle\ \langle v_1 \to v_2 \rangle\ \langle v_2 \to t \rangle\ \langle t \to t' \rangle}_{\star}$$

Also in this example we have that the result of the expression is well-defined and unambiguous. This is due to the special role of the edges $\langle s' \to s \rangle$ and $\langle t \to t' \rangle$, since they can not be used to create a new transitive edge in the graph.

Unfortunately, our operator $\circ$ is binary and, therefore, expressions like $s_{1,1}||$ are not evaluated in one step. Moreover, because of the required associativity of $\circ$, it has to be possible to completely evaluate an expression like $|s_{1,1}|s_{2,2}$. For this, we introduce *subexpression elements*, denoted by $[\![\ ]\!]$. These new elements of our universe encode expressions that are evaluated as far as possible. For example, the expression $|s_{1,1}|s_{2,2}$ evaluates (independently of the evaluation order) to $[\![|\langle s \to v_1 \rangle s_{2,2}]\!]$. And, as another example, the concatenation of $[\![|\langle s \to v_1 \rangle s_{2,2}]\!]$ and $[\![|s_{1,1}|]\!]$ results in the new subexpression $[\![|\langle s \to v_2 \rangle s_{1,1}|]\!]$. This is also the reason why we require the end symbol $\lhd$: The end symbol marks the end of the considered expression and allows the "internal" evaluation of the very last subexpression. For instance, consider the following end of an expression and its result after applying the binary operator to the subexpression a:

$$\cdots \overbrace{\underbrace{|s_{1,1}|s_{2,2}|}_{b}\ \underbrace{s_{1,1}|}_{c}}^{a} \quad \mapsto \quad \cdots [\![|\langle s \to v_2 \rangle s_{1,1}|]\!]$$

While the subexpression b can be evaluated unambiguously, this is not the case for subexpression c. The reason for this is, that b is followed by a selector element that implicitly marks the end of the sequence of counter elements $|$ that may follow $s_{2,2}$. This is not the case for subexpression c.

In the expression above, it is indistinguishable whether c is followed by another counter element or c is at the very end of the expression. To fix this, we use the end symbol $\triangleleft$. It marks the right end of the expression and enforces the unambiguous evaluation of the very last subexpression, and therefore the whole expression.

Note that the required number of subexpression elements is polynomial in the number of colors, vertices, and edges, and, therefore, the universe and the table describing the results of applying the operator $\circ$ is computable in para-L.

For correctness first consider a graph such that there are k colors that make up a path from s to t. Then we can choose our generating set to contain the counter element, a selector from each block such that these selectors correspond to the k colors, the special selector element s_+, and the end element $\triangleleft$. By appropriately using these elements we can generate the complete transitive closure, the star element, and with this element all the remaining selectors. Being able to generate these elements, we can also generate every subexpression element. For the backward direction assume that we can generate the whole universe from a generating set of size $k + 3$. By the construction above, this set has to contain the counter element, the end symbol, the special selector, and a selector element from each of the k blocks of selectors. Being able to generate the whole universe implies that we are able to generate the star element and therefore generate the transitive edge from s to t, too. This implies the existence of a path from s to t that is made up by choosing k colors. □

5 Conclusion

We have introduced new parameterized space classes that capture the complexity of parameterized problems and help in understanding the complexity of fixed-parameter tractable problems. We have seen that, under the assumption L $\neq$ NL, well-known problems like p-FVS and p-DFVS are separated. Moreover, classes like para-L-cert and para-NL-cert are not fully contained in para-P, but capture natural problems that could not be classified in an exact way before.

The next step is to reconsider other parameterized problems (both fixed-parameter tractable and intractable under standard assumptions like W[1] $\neq$ para-P) with the presented framework in mind: How can these problems be classified within this framework? What techniques from classical space complexity help in the parameterized setting and vice versa? In particular, it might be interesting to use the framework to better understand the complexity of problems whose unparameterized

versions are known to lie in P and, thus, are normally not studied from the parameterized point of view.

One might even better understand the complexity of parameterized problems whose slices are not P-hard by considering parameterized versions of classical complexity classes based on parallel and circuit computations like NC, NC^1, TC^0, and AC^0.

References

[1] K. A. Abrahamson, R. G. Downey, and M. R. Fellows. Fixed-parameter tractability and completeness IV: On completeness for W[P] and PSPACE analogues. *Annals of Pure and Applied Logic*, 73(3):235–276, 1995. doi: 10.1016/0304-3975(94)00097-3.

[2] H. L. Bodlaender. On Linear Time Minor Tests with Depth-First Search. *Journal of Algorithms*, 14(1):1–23, 1993. doi:10.1006/jagm.1993.1001.

[3] H. L. Bodlaender. A linear time algorithm for finding tree-decompositions of small treewidth. *SIAM Journal on Computing*, 25(6):1305–1317, 1996. doi:10.1137/S0097539793251219.

[4] H. L. Bodlaender, R. D. Downey, M. R. Fellows, and H. T. Wareham. The Parameterized Complexity of Sequence Alignment and Consensus. *Theoretical Computer Science*, 147(1–2):31–54, 1995. doi: 10.1016/0304-3975(94)00251-D.

[5] 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, 1997. doi:10.1016/S0168-0072(95)00020-8.

[6] J. Chen, Y. Liu, S. Lu, B. O'Sullivan, and I. Razgon. A Fixed-Parameter Algorithm for the Directed Feedback Vertex Set Problem. *Journal of the ACM*, 55(5):21:2–21:19, 2008. doi:10.1145/1411509.1411511.

[7] Y. Chen, J. Flum, and M. Grohe. Bounded Nondeterminism and Alternation in Parameterized Complexity Theory. In *Proceedings of the 18th IEEE Annual Conference on Computational Complexity (CCC 2003)*, pages 13–29, 2003. doi:10.1109/CCC.2003.1214407.

[8] S. A. Cook and P. McKenzie. Problems complete for deterministic logarithmic space. *Journal of Algorithms*, 8(3):385–394, 1987. doi:10.1016/0196-6774(87)90018-6.

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

[10] M. Elberfeld, A. Jakoby, and T. Tantau. Logspace Versions of the Theorems of Bodlaender and Courcelle. In *Proceedings of the 51st Annual IEEE Symposium on Foundations of Computer Science (FOCS 2010)*, pages 143–152, 2010. doi:10.1109/FOCS.2010.21.

[11] M. Elberfeld, C. Stockhusen, and T. Tantau. On the Space Complexity of Parameterized Problems. In *Proceedings of the 7th International Symposium on Parameterized and Exact Computation (IPEC 2012)*. Springer, 2012.

[12] J. Flum and M. Grohe. Describing parameterized complexity classes. *Information and Computation*, 187:291–319, 2003. doi:10.1016/S0890-5401(03)00161-5.

[13] J. Flum and M. Grohe. *Parameterized Complexity Theory*. Springer, 2006. doi:10.1007/3-540-29953-X.

[14] S. Guillemot. Parameterized complexity and approximability of the Longest Compatible Sequence problem. *Discrete Optimization*, 8(1):50–60, 2011. doi:10.1016/j.disopt.2010.08.003.

[15] J. Hartmanis. On Non-Determinancy in Simple Computing Devices. *Acta Informatica*, 1:336–344, 1972. doi:10.1007/BF00289513.

[16] N. D. Jones. Space-bounded reducibility among combinatorial problems. *Journal of Computer and System Sciences*, 11(1):68–85, 1975. doi:10.1016/S0022-0000(75)80050-X.

[17] N. D. Jones and W. T. Laaser. Complete problems for deterministic polynomial time. *Theoretical Computer Science*, 3(1):105–117, 1976. doi:10.1016/0304-3975(76)90068-2.

[18] N. D. Jones, Y. E. Lien, and W. T. Laaser. New Problems Complete for Nondeterministic Log Space. *Mathematical Systems Theory*, 10(1):1–17, 1976. doi:10.1007/BF01683259.

[19] N. Lynch. Log Space Recognition and Translation of Parenthesis Languages. *Journal of the ACM*, 24:583–590, 1977. doi:http://doi.acm.org/10.1145/322033.322037.

[20] K. Pietrzak. On the parameterized complexity of the fixed alphabet shortest common supersequence and longest common subsequence problems. *Journal of Computer and System Science*, 67(4):757–771, 2003. doi:10.1016/S0022-0000(03)00078-3.

ECCC ISSN 1433-8092

http://eccc.hpi-web.de