On Space Efficiency of Algorithms Working on Structural Decompositions of Graphs

MICHAŁ PILIPCZUK and MARCIN WROCHNA, Institute of Informatics, University of Warsaw

Dynamic programming on path and tree decompositions of graphs is a technique that is ubiquitous in the field of parameterized and exponential-time algorithms. However, one of its drawbacks is that the space usage is exponential in the decomposition's width. Following the work of Allender et al. [5], we investigate whether this space complexity explosion is unavoidable. Using the idea of reparameterization of Cai and Juedes [18], we prove that the question is closely related to a conjecture that the Longest Common Subsequence problem parameterized by the number of input strings does not admit an algorithm that simultaneously uses **XP** time and **FPT** space. Moreover, we extend the complexity landscape sketched for pathwidth and treewidth by Allender et al. by considering the parameter *tree-depth*. We prove that computations on tree-depth decompositions correspond to a model of non-deterministic machines that work in polynomial time and logarithmic space, with access to an auxiliary stack of maximum height equal to the decomposition's depth. Together with the results of Allender et al., this describes a hierarchy of complexity classes for polynomial-time non-deterministic machines with different restrictions on the access to working space, which mirrors the classic relations between treewidth, pathwidth, and tree-depth.

CCS Concepts: • **Theory of computation** → **Complexity classes**; *Models of computation*; *Problems, reductions and completeness*; *Parameterized complexity and exact algorithms*; Fixed parameter tractability;

Additional Key Words and Phrases: Complexity classes, complexity theory, time-space tradeoff, treewidth, pathwidth, tree-depth, completeness, CSP, LCS

1 INTRODUCTION

Treewidth is a graph parameter that measures how much a graph resembles a tree. Intuitively, a graph has treewidth s if it can be decomposed into parts (called *bags*) of size at most $s + 1$ so that this decomposition, called a *tree decomposition*, has the shape of a tree. While initially introduced by Robertson and Seymour in their Graph Minors project [57], treewidth has found numerous applications in the field of algorithms. This is mostly due to the fact that many problems that

are intractable on general graphs, become efficiently solvable on graphs of small treewidth using the principle of dynamic programming. Theorems of Courcelle [22] and of Arnborg et al. [6] explain that every decision or optimization problem expressible in Monadic Second Order logic can be solved in time $f(s) \cdot n$ on graphs of treewidth s and size n, for some function f. While f can be non-elementary, for many classic problems, like Vertex Cover, 3Coloring, or Dominating Set, the natural dynamic programming approach yields an algorithm with running time $O(c^s \cdot n)$ for some typically small constant c. This is important from the point of view of applications, because dynamic programming procedures working on tree decompositions often serve as critical subroutines in more complex algorithms, such as subexponential algorithms derived using the technique of *bidimensionality* [24] or approximation schemes that use Baker's approach [9]. Algorithms working on tree decompositions are usually analyzed in the paradigm of *parameterized complexity*, where the width of the decomposition is the considered parameter. Nowadays, the study of problems parameterized by structural measures of the input, such as treewidth, is a classic topic in this field. We refer to textbooks [23, 26, 32] for a broad introduction, and to a recent survey of Langer et al. [42] for more specific results.

A certain limitation of dynamic programming on a tree decomposition is that it uses space exponential in its width. In practical applications, this is often a prohibitive factor, because the machine is likely to simply run out of space much before the elapsed time exceeds tolerable limits [4, 12, 41]. Therefore, recently there is much focus on reducing the space complexity of exponential-time algorithms to polynomial, even at the cost of slightly worsening the time complexity [8, 13, 33, 34, 46, 49]. Here, the usage of algebraic tools proved to be an extremely useful approach. Unfortunately, algorithms working on tree decompositions of graphs remain a family where virtually no progress has been achieved in this matter. Therefore, a natural question arises: Can we reduce the space complexity of algorithms working on tree decompositions while keeping (or moderately worsening) their time complexity? The first explicit statement of this question known to us is due to Lokshtanov et al. [45], who sketched how using a simple tradeoff one can obtain polynomial space complexity while increasing the time complexity to $2^{O(s^2)} + O(n^2)$. Later, the same question was asked by Langer et al. [42].

Following early completeness results of Monien and Sudborough [48] on graph problems in small bandwidth and of Gottlob et al. [37] on conjunctive queries of bounded treewidth, Allender et al. [5] recently initiated the systematic study of satisfaction complexity in small, but non-constant pathwidth and treewidth. Essentially, they observe that CSP-like problems—say, 3Coloring for concreteness[1]—when limited to instances of small treewidth or pathwidth, are complete for certain complexity classes under logspace reductions.

More precisely, let the input graph be equipped with a path decomposition of width at most $s(n) \geq \log n$, for some fixed constructible function s of the input size. Then 3Coloring (denoted in this case as pw-3Coloring $[s]$) is complete for the class of problems that admit non-deterministic algorithms working simultaneously in polynomial time and space $O(s(\text{poly}(n)))$, which we denote by $\mathbf{N}[\underset{time}{\text{poly}}, \underset{space}{s(\text{poly})}]$.

Similarly, tw-3Coloring $[s]$, where $s(n)$ bounds the width of a given tree decomposition, is complete for the class $\mathbf{NAuxPDA}[\underset{time}{\text{poly}}, \underset{space}{s(\text{poly})}]$, defined just as $\mathbf{N}[\underset{time}{\text{poly}}, \underset{space}{s(\text{poly})}]$, but the algorithm can use an auxiliary push-down of unlimited size, to which read/write access is only from the top (details of this computation model will be given in Section 4).

---

[1]Allender et al. use SAT parameterized by treewidth/pathwidth of its primal graph as an exemplary problem, but SAT and 3Coloring can be easily seen to be equivalent under parameter preserving logspace reductions; see Lemma 2.7. In this article, we prefer to use 3Coloring as an exemplary hard CSP-like problem.

We remark that Allender et al. focus on a different characterization by *semi-unbounded fan-in* (SAC) circuits. In particular, they show that $\mathbf{NAuxPDA}[\underset{time}{\text{poly}}, \underset{space}{s(\text{poly})}]$ is exactly characterized by SAC circuits of logarithmic depth and size $2^{s(\text{poly})}$. This implies that all the classes we consider have relatively small circuits of only logarithmic depth. In order to not use too many models, we omit the discussion of circuit classes from now on and refer to Allender et al. instead. While they state their results only for $s(n) = \log^k n$, their proof works in the more general setting given below.

THEOREM 1.1 ([5]). *Let $s(n) \geq \log n$ be a nice function.*[2] *Then,* pw-3COLORING $[s]$ *is complete for* $\mathbf{N}[\underset{time}{\text{poly}}, \underset{space}{s(\text{poly})}]$ *under logspace reductions, whereas tw-3COLORING $[s]$ is complete for* $\mathbf{NAuxPDA}[\underset{time}{\text{poly}}, \underset{space}{s(\text{poly})}]$.

Thus, the feasibility of various space-time tradeoffs for algorithms working on tree or path decompositions is equivalent to inclusions of corresponding complexity classes. For instance, assume for conciseness that $s(n^c) = O(s(n))$ for each constant c, e.g., $s(n) = \log^k n$ for some k. Then pw-3COLORING $[s]$ is solvable in:

- time $2^{o(s(n)\log n)}$ and space $2^{o(s(n))}$ if and only if $\mathbf{N}[\underset{time}{\text{poly}}, \underset{space}{s}] \subseteq \mathbf{D}[\underset{time}{2^{o(s \cdot \log)}}, \underset{space}{2^{o(s)}}]$;
- time $2^{O(s(n))}$ and space poly(n) if and only if $\mathbf{N}[\underset{time}{\text{poly}}, \underset{space}{s}] \subseteq \mathbf{D}[\underset{time}{2^{O(s)}}, \underset{space}{\text{poly}}]$.

Similar statements can be inferred for treewidth. In contrast, the best known determinization results come from a brute-force approach or from Savitch's theorem[3] [61], yielding, respectively, (for constructible $t(n) \geq n$ and $s(n) \geq \log t(n)$):

$$\mathbf{N}[\underset{time}{t}, \underset{space}{s}] \subseteq \mathbf{D}[\underset{time}{2^{O(s)}}] = \mathbf{D}[\underset{time}{2^{O(s)}}, \underset{space}{2^{O(s)}}];$$

$$\mathbf{N}[\underset{time}{t}, \underset{space}{s}] \subseteq \mathbf{D}[\underset{space}{s \cdot \log t}] = \mathbf{D}[\underset{time}{2^{O(s \cdot \log t)}}, \underset{space}{s \cdot \log t}].$$

In this manner, Allender et al. conclude that, intuitively speaking, achieving better time-space tradeoffs for algorithms working on path and tree decompositions of small width would require developing a general technique of improving upon the tradeoff of Savitch. As Lipton [43] phrased it, "One of the biggest embarrassments of complexity theory is the fact that Savitch's theorem has not been improved [...]. Nor has anyone proved that it is tight."

Allender et al. argue that such an improvement would contradict certain rescaled variants of known conjectures about the containment of time- and space-constrained classes, in particular the assumption that $\mathbf{NL} \nsubseteq \mathbf{SC}$; more precisely that the rescaled hierarchies built on top of these two classes are somehow orthogonal—see Reference [5] for details. We consider the study of Allender et al. not as a definite answer in the topic, but rather as an invitation to a further investigation of links between the introduced conjectures and more well-established complexity assumptions.

Our Contribution. We show that two seemingly unrelated questions on time-space trade-offs are in fact a common barrier, which reinforces the conjecture that no better determinization result is possible. Let us first state the conjecture precisely in terms of a well-studied problem. In the LONGEST COMMON SUBSEQUENCE problem (LCS), we are given an alphabet Σ and k strings

---

[2]By a *nice* function we mean a function s that is constructible and such that $s(n)/\lg n$ is non-decreasing.

[3]There is also a simple simulation approach that tries each possible sequence of non-deterministic choices, proving $\mathbf{N}[\underset{time}{t}, \underset{space}{s}] \subseteq \mathbf{D}[\underset{space}{t}]$, but in our context it is always worse than Savitch's theorem.

$s_1, s_2, \ldots, s_k$ over Σ, and the task is to compute the longest sequence of symbols of Σ that appears as a subsequence in each s_i. The applicability of the LCS problem in, e.g., computational biology, motivated many to search for faster or more space-efficient algorithms, as the classical dynamic programming solution, running in time and space $O(n^k)$ (where n is the length of each string and k the number of strings) is often far from practical. From the point of view of parameterized complexity, LCS parameterized by k is W$[t]$-hard for every level t [16], remains W$[1]$-hard even for a fixed-sized alphabet [55], and is W$[1]$-complete when parameterized jointly by k and ℓ, the target length of the subsequence [38]. In a recent breakthrough, Abboud et al. [1] proved that the existence of an algorithm with running time $O(n^{k-\varepsilon})$, for any $\varepsilon > 0$, would contradict the Strong Exponential Time Hypothesis. As far as the space complexity is concerned, only modest progress has been achieved: The best known result is an algorithm of Barsky et al. [11], which improves the space complexity to $O(n^{k-1})$. This motivates us to formulate the following conjecture.

CONJECTURE 1.2. *There is no algorithm for LCS that works in time $n^{f(k)}$ and space $f(k)\mathrm{poly}(n)$ for a computable function f, where k is the number of input strings and n the total input length.*[4]

The conjecture is equivalent to a natural separation of parameterized complexity classes, which can be viewed as a statement about the impossibility of determinization results improving upon Savitch's theorem (namely, Conjecture 1.2 is equivalent to $\mathbf{N}[f\mathrm{poly}, f\log] \not\subseteq \mathbf{D}[n^f, f\mathrm{poly}]$; we defer the definitions and detailed discussion to Section 3). This follows from a completeness result for LCS by Elberfeld et al. [31].

Among other implications, we prove that, quite surprisingly, Conjecture 1.2 is closely related to the question about time-space tradeoffs for algorithms working on path decompositions of small width:[5]

THEOREM 1.3. *The following statements hold:*

- *Conjecture 1.2 implies the following: There is no algorithm for* pw-3COLORING *working in time $2^{O(s)}\mathrm{poly}(n)$ and space $2^{o_{eff}(s)}\mathrm{poly}(n)$ (for all values of width s).*
- *Conjecture 1.2 is implied by the following: There is an unbounded, computable function g, for which* pw-3COLORING *cannot be solved in time $2^{s \cdot g(s)}\mathrm{poly}(n)$ and space $2^{o_{eff}(s)}\mathrm{poly}(n)$ (for all values of width s).*

Thus, Conjecture 1.2 is sandwiched between a weaker statement: that it is impossible to achieve subexponential space complexity while keeping single exponential time complexity, and a stronger statement: that this holds even if we allow the time complexity exponent to increase by some arbitrarily slowly growing function of the width.

To prove Theorem 1.3, we rely on the results of Elberfeld et al. [31] and use the ideas of Cai and Juedes [18] that connected subexponential complexity to fixed-parameter tractability, considering a reparameterized version of pw-3COLORING. This allows us to compare questions concerning time-space tradeoffs for pw-3COLORING and determinization of $\mathbf{N}[\underset{time}{t}, \underset{space}{s}]$ classes to those concerning parameterized classes and the complexity of LCS. In particular, we show that Conjecture 1.2 implies $\mathbf{NL} \not\subseteq \mathbf{D}[\underset{time}{\mathrm{poly}}, \underset{space}{\mathrm{poly}\log}]$ (the latter class being usually called $\mathbf{SC}$) and is implied by a rescaled version of the following stronger variant: $\mathbf{NL} \not\subseteq \mathbf{D}[\underset{time}{2^{o(\log^2 n)}}, \underset{space}{n^{o(1)}}]$. We recall how

---

[4]Note f does not depend on n. Computability is a technical uniformity condition needed in some reductions: see Section 3 for details and [25] for a discussion of uniform versus non-uniform definitions.

[5]Here and throughout the article, $o_{\mathrm{eff}}()$ is an effective variant of the $o()$ notation. Formally, for $f, h : \mathbb{N} \to \mathbb{N}$, we write $f = o_{\mathrm{eff}}(h)$ if there is a non-decreasing, unbounded, computable function $g(n)$ such that $f = O(\frac{h}{g})$.

these statements relate to research on algorithms for reachability in directed graphs in Section 3.4, giving further arguments supporting the conjecture.

In the second part of this work, we extend the findings of Allender et al. [5] by considering the graph parameter *tree-depth*. For a graph G, its tree-depth is equal to the minimum height of a rooted forest $\mathcal{T}$ whose ancestor-descendant closure contains G as a subgraph; the forest $\mathcal{T}$ is then called a *tree-depth decomposition* of G. Tree-depth of a graph is lower bounded by its pathwidth and upper bounded by its treewidth times $\lg n$. Our motivation for considering this parameter is twofold. First, recent advances have uncovered a wide range of topics where tree-depth appears to be fundamental. For instance, tree-depth plays an important role in the theory of sparse graphs developed by Nešetřil and Ossona de Mendez [51], it is the key factor in classification of homomorphism problems that can be solved in logspace [20], and corresponds to classes of graphs where the expressive power of First-Order logic and Monadic Second-Order logic coincides [28]. It was also rediscovered several times in different contexts and under different names: *minimum elimination tree height* [56], *ordered chromatic number* [39], *vertex ranking* [15], or the maximum number of introduce nodes on a root-to-leaf path in a tree decomposition of a graph [34].

Second, algorithms working on tree-depth decompositions of small height can model generic exponential-time Divide&Conquer algorithms. In this approach, after finding a small, usually balanced separator S in the graph, the algorithm tries all possible ways a solution can interact with S, and solves connected components of $G - S$ recursively. A run of such an algorithm naturally gives rise to a tree-depth decomposition of the graph, where S is placed on top of the decomposition, and decompositions of the components of $G - S$ are attached below it as subtrees. The maximum total number of separator vertices handled at any moment in the recursion corresponds to the height of the decomposition. Thus, many classic Divide&Conquer algorithms, including the ones derived for planar graphs using the Lipton-Tarjan separator theorem [44], can be reinterpreted as first building a tree-depth decomposition of the graph using a separator theorem, and then running the algorithm on this decomposition.

Most importantly for us, when implemented using recursion, the algorithms working on tree-depth decompositions run in polynomial space. For instance, such an algorithm for 3Coloring on a tree-depth decomposition of depth s runs in time $3^s \cdot \text{poly}(n)$ and space $O(s + \log n)$ (see Lemma 4.5), which places td-3Coloring $[s]$ in $\mathbf{D}[\underset{time}{2^{O(s)}\text{poly}}, \underset{space}{s + \log}] = \mathbf{D}[\underset{space}{s + \log n}]$. This is immediate for CSP-like problems like 3Coloring, but recently Fürer and Yu [34] showed that algebraic transforms can be used to reduce the space usage to polynomial in n also for other problems, like counting perfect matchings or dominating sets. In Section 5, we define the problem and describe how this approach gives an $3^s \cdot \text{poly}(n)$-time poly(n)-space algorithm for Dominating Set in more detail, and then improve the space usage to $O(s \cdot \log n)$ using Fourier transforms and the Chinese remainder theorem. This means that the reduction of space complexity that is conjectured to be impossible for treewidth and pathwidth, actually is possible for tree-depth. Therefore, we believe that it is useful to study the model of computations standing behind low tree-depth decompositions, to understand how it differs from the models for treewidth and pathwidth.

Consequently, mirroring Theorem 1.1, we prove that computations on tree-depth decompositions exactly correspond to the class $\mathbf{NAuxSA}[\underset{time}{\text{poly}}, \underset{space}{\log}, \underset{height}{s}]$: problems that can be decided by a non-deterministic Turing Machine that uses polynomial time and logarithmic working space, but also has access to an auxiliary stack of maximum height s. The stack can be freely read by the machine, just like the input tape, but the write access to it is only via push and pop operations.

THEOREM 1.4. *Let $s(n) \geq \log^2 n$ be a nice function. Then td-3Coloring $[s]$ is complete*

for **NAuxSA[**poly, log, s(poly)**]** *under logspace reductions.*
         $\underset{time}{}$ $\underset{space}{}$ $\underset{height}{}$

Thus, computations on tree-depth and path decompositions differ by the access restrictions to $O(s)$ space used by the machine. While for pathwidth this space can be accessed freely, for tree-depth all of the space apart from an $O(\log n)$ working buffer has to be organized in a stack.

The proof of Theorem 1.4 largely follows the approach of Akatov and Gottlob [3], who proved a different completeness result for the class **NAuxSA[**poly, log, $\log^2$**]**, which they call $\mathbf{DC}^1$. The
                                                        $\underset{time}{}$ $\underset{space}{}$ $\underset{height}{}$
main idea is to regularize the run of the machine so that the push-pop tree has the rigid shape of a full binary tree. Then, we can use this concrete structure to "wrap around" gadgets encoding an accepting run of a regularized NAuxSA machine. However, the motivation in the work of Akatov and Gottlob was answering conjunctive queries in a hypergraph by exploiting a kind of balanced decomposition, and hence the problem proven to be complete for $\mathbf{DC}^1$ is a quite general and expressive problem originating in database motivations; see References [2] and [3] for details. In our setting, to get a reduction to 3Coloring, we need to work more to encode an accepting run. In particular, to encode each part of the computation where no push or pop is performed, instead of producing a single atom in a conjunctive query, we use computation gadgets that originate in Cook's proof of the NP-completeness of SAT. The assumption that the computation has a polynomial number of steps is essential here for bounding the tree-depth of each such gadget. This way, Theorem 1.4 presents a more natural complete problem for $\mathbf{DC}^1$.

Another difference is that Theorem 1.4 works for any well-behaved function $s(n) \geq \log^2 n$, as opposed to the bound $s(n) = \log^2 n$ inherent to the problem considered by Akatov and Gottlob. For this, the crucial new idea is to increase the working space of the machine to $s(n)/\log n$ in order to be able to perform regularization—a move that looks dangerous at first glance, but turns out not to increase the expressive power of the computation model. This proves the following interesting by-product of our work.

THEOREM 1.5. *Let* $s(n) \geq \log^2 n$ *be a nice function. Then,*

$$\mathbf{NAuxSA[}\underset{time}{\text{poly}}, \underset{space}{\log}, \underset{height}{s(\text{poly})}\mathbf{]} = \mathbf{NAuxSA[}\underset{time}{\text{poly}}, \underset{space}{s(\text{poly})/\log}, \underset{height}{s(\text{poly})}\mathbf{]}.$$

The following determinization result for NAuxSA machines follows from the observation that td-3Coloring [s] can be solved in $\mathbf{D[}\underset{time}{2^{O(s)}\text{poly}}, \underset{space}{s + \log}\mathbf{]} = \mathbf{D[}\underset{space}{s + \log n}\mathbf{]}$.

THEOREM 1.6. *Let* $s(n) \geq \log^2(n)$ *be a nice function. Then,*

$$\mathbf{NAuxSA[}\underset{time}{\text{poly}}, \underset{space}{\log}, \underset{height}{s(\text{poly})}\mathbf{]} \subseteq \mathbf{D[}\underset{space}{s(\text{poly})}\mathbf{]}.$$

Theorem 1.6 for $s(n) = \log^2 n$ also follows from the work of Akatov and Gottlob [3]. Observe that now the justification for the assumption $s(n) \geq \log^2 n$ becomes apparent: for, say, $s(n) = \log n$, the theorem would state that $\mathbf{L} = \mathbf{NL}$, a highly unexpected outcome. We find Theorem 1.6 interesting, because a naïve simulation of the whole configuration space for NAuxSA would require space exponential in s. It appears, however, that the exponential blow-up of the space complexity can be avoided. We do not see any significantly simpler way to prove this result other than going through the td-3Coloring [s] problem, and hence it seems that the tree-depth view gives a valuable insight into the computation model of NAuxSA.

The classic relations between treewidth, pathwidth, and tree-depth are, through completeness results, mirrored in a hierarchy between NAuxPDA, N, and NAuxSA classes, as detailed in the

concluding section. In particular, this answers a question of Akatov and Gottlob [2, 3] about the relation of other classes in **NP** to **NAuxSA**[poly, log , poly log].
$\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad{}_{time}\ {}_{space}\quad{}_{height}$

Finally, using Theorem 1.4 we also give an alternative view on NAuxSA computations using alternating Turing machines and *treesize*, a notion introduced by Ruzzo [60] (see Sections 2 and 4.2 for the definitions). Ruzzo showed the following, among other relations that give a unified view of some complexity classes:

$$\mathbf{NAuxPDA}[\mathrm{poly},\ \underset{space}{s}\ ] = \mathbf{A}[\ \underset{space}{s}\ ,\ \underset{treesize}{\mathrm{poly}}\ ].$$

Answering another question of Akatov and Gottlob, we show the following:

THEOREM 1.7. *Let $s(n) \geq \log^2(n)$ be a nice function. Then,*

$$\mathbf{NAuxSA}[\mathrm{poly},\ \underset{space}{\log}\ ,\ \underset{height}{s(\mathrm{poly})}] = \mathbf{A}[\underset{time}{s(\mathrm{poly})},\ \underset{treesize}{\mathrm{poly}}\ ].$$

In other words, bounding the time (as opposed to space) of a polynomial treesize ATM, leads to the classes corresponding to small tree-depth (as opposed to small treewidth). The determinization in Theorem 1.6 can also be seen as a corollary of this characterization via ATMs.

Outline. In Section 2, we give preliminaries. In Section 3, we prove Theorem 1.3 and related results. In Section 4, we prove Theorem 1.4 and derive corollaries from it (Theorems 1.5 and 1.6). Section 5 describes the algebraic algorithm for DOMINATING SET in graphs of low tree-depth. In Section 6, we give a summary of the hierarchy of containments studied here and some concluding remarks. The discussion of how standard NP-hardness reductions preserve parameters linearly is deferred to Appendix A.

2 PRELIMINARIES

2.1 Reductions and Complexity Classes

For two languages P, Q, we write $P \leq_L Q$ when P is logspace reducible to Q (see [7, 4.4]). Most of the complexity classes we consider are closed under logspace reductions.

Because we handle various measures of complexity and compare a wide array of classes that bound two measures simultaneously, we introduce the following notation. A complexity class is first described by the machine model: **D**, **N**, **A** denote deterministic, non-deterministic, and alternating (see Reference [60]) Turing machines, respectively. Then, in square brackets, bounds on complexity measures are described (up to constant factors) as a list of functions with the name of the measure it bounds underneath. All functions except the symbol f (which we reserve for classes in parameterized complexity) are functions of the input size n. For example, for $t, s : \mathbb{N} \to \mathbb{N}$, $\mathbf{D}[\ \underset{space}{s}\ ]$ denotes the class of languages recognizable by deterministic Turing machines using at most $O(s(n))$ space, usually known as DSpace($s(n)$); similarly $\mathbf{N}[\ \underset{time}{t}\ ,\ \underset{space}{s}\ ]$ is often denoted NTiSp($t(n), s(n)$). We write lg for the logarithm with base 2, $\log(n)$ when the base is irrelevant and poly(n) for $n^{O(1)}$. As customary for the O-notation, a complexity class stated with a bound that is a family of functions (instead of a single function) is defined as the sum of classes over all functions in the family. For example, $\mathbf{D}[\underset{time}{\mathrm{poly}}] = \bigcup_{k \in \mathbb{N}} \mathbf{D}[\ \underset{time}{n^k}\ ] = \mathbf{P}$ and $\mathbf{N}[\ \underset{space}{\log}\ ] = \mathbf{NL}$.

An auxiliary push-down or stack is denoted as AuxPDA or AuxSA, respectively: the difference is that a push-down can only be read at the top, while a stack can be read just as a tape (both can be written to only by pushing and popping symbols at the top); see e.g., Reference [62] (we describe

some details of this computation model at the beginning of Section 4). The measure named *height* is the maximum height of the push-down or stack.

We say a function $s : \mathbb{N} \to \mathbb{N}$ is *constructible* if there is a Turing machine that given a number n in unary outputs $s(n)$ in unary using logarithmic space; in particular, this implies $s(n) \leq \text{poly}(n)$. A function s is *nice* if it is constructible and $\frac{s(n)}{\lg n}$ is non-decreasing; note that this implies that $s(n)$ itself is also non-decreasing. For simplicity, we will assume all functions $s : \mathbb{N} \to \mathbb{N}$ describing complexity bounds to be nice.

Note that logspace reductions can blow-up instance sizes polynomially; hence, the closure of $\mathbf{N}[\underset{time}{\text{poly}}, \underset{space}{s}]$ under such reductions is $\mathbf{N}[\underset{time}{\text{poly}}, \underset{space}{s(\text{poly})}]$, for example. These are equal for functions $s(n)$ such that $s(\text{poly}(n)) \leq O(s(n))$ (i.e., if for every $c > 0$ there is a $d > 0$ such that $s(n^c) \leq ds(n)$). This includes $\lg^k(n)$ for any $k \geq 1$ and $\lg n \lg \lg n$, for example; however, one can construct artificial examples that are polylogarithmically bounded but fail to have this property.

2.2 Structural Parameters

We recall the definitions of treewidth, pathwidth and tree-depth. For conciseness, we will refer to the certifying structures defined below as *decompositions* for all three parameters.

Definition 2.1 (treewidth). A *tree decomposition* of an undirected graph G is a tree $\mathcal{T}$ together with a collection of sets of vertices of G (called *bags*) X_t indexed by nodes $t \in \mathcal{T}$, such that:

- every vertex of G is in at least one bag;
- for every edge uv of G, there is a bag containing both u and v; and
- for every vertex v of G, the set $\{t \in \mathcal{T} : v \in X_t\}$ induces a connected subtree of $\mathcal{T}$.

The width of a tree decomposition is defined as $\max_{t \in \mathcal{T}} |X_t| - 1$. The *treewidth* of G is the minimum width over all possible tree decompositions of G.

Definition 2.2 (pathwidth). A *path decomposition* of an undirected graph G is a tree decomposition $(\mathcal{T}, (X_t)_{t \in \mathcal{T}})$ in which $\mathcal{T}$ is a path. The *pathwidth* of G is the minimum width over all possible path decompositions of G.

Definition 2.3 (tree-depth). A *tree-depth decomposition* of an undirected graph G is a rooted forest $\mathcal{T}$ (a disjoint union of rooted trees) together with a bijective mapping μ from the vertices of G to the nodes of $\mathcal{T}$, such that for every edge uv of G, $\mu(u)$ is an ancestor of $\mu(v)$ or $\mu(v)$ is an ancestor of $\mu(u)$ in $\mathcal{T}$. The *depth* of a rooted forest is the largest number of nodes on a path between a root and a leaf. The *tree-depth* of G is the minimum depth over all possible tree-depth decompositions of G.

Without loss of generality, we assume that in all given tree and path decompositions, $|\mathcal{T}| \leq 2|V(G)|^2$; standard methods allow to prune any decomposition to this size in logspace (see Reference [40, Lemma 13.1.2]).

For a graph problem, such as 3Coloring, a structural parameter $\pi \in \{\text{td}, \text{pw}, \text{tw}\}$, and a nice function $s : \mathbb{N} \to \mathbb{N}$, we define π-3Coloring $[s]$ to be the decision problem where given an instance G of 3Coloring and a π-decomposition of G, we ask whether the decomposition has width at most $s(|V(G)|)$ and G is a yes-instance of 3Coloring. The assumption that a decomposition is given on input is to factor away the complexity of finding it, which is a problem not directly relevant to our work. Note that the validity and width/depth of a decomposition given in any natural encoding can easily be checked in logarithmic space.

Observe that for any $c > 0$, π-3Coloring $[s(n)]$ is equivalent to π-3Coloring $[s(n^c)]$ under logspace reductions. Namely, from π-3Coloring $[s(n)]$ to π-3Coloring $[s(n^c)]$ a reduction is

trivial, while the reverse reduction follows easily by padding: adding isolated vertices up to size n^c that do not change the answer nor the value of π. Since we assume s to be nice, we have $\frac{s(n)}{\lg n} \leq \frac{s(n^c)}{\lg n^c}$; hence, $c \cdot s(n) \leq s(n^c)$ for any $c \geq 1$. This implies that π-3COLORING $[c \cdot s(n)]$ is equivalent to π-3COLORING $[s(n)]$.

A hierarchy between these problems immediately follows from well-known inequalities between the parameters td, pw, tw, stated in the next lemma and corollary. Every tree-depth decomposition can be turned into a path decomposition by taking bags to be the vertex sets of all the root-to-leaf paths, and ordering them as in a left-to-right scan of the tree. Every path decomposition is trivially a tree decomposition. Every tree decomposition allows us to find small separators, which can be used to recursively build a tree-depth decomposition, yielding the last inequality—details can be found, e.g., in Reference [50, Corollary 2.5]. We show how to execute them effectively below for completeness.

LEMMA 2.4. *There is a constant $c \in \mathbb{N}$ such that for any graph G,*

$$\mathsf{td}(G) \geq \mathsf{pw}(G) \geq \mathsf{tw}(G) \geq \mathsf{td}(G)/(c \cdot \log |V(G)|).$$

Furthermore, each inequality is certified by an algorithm that transforms the respective graph decompositions in logspace.

COROLLARY 2.5. *Let $s : \mathbb{N} \to \mathbb{N}$ be a nice function. Then,*

$$\text{td-}3COLORING[s] \leq_L \text{pw-}3COLORING[s] \leq_L \text{tw-}3COLORING[s] \leq_L \text{td-}3COLORING[s \cdot \log]$$

OF LEMMA 2.4. The algorithms for the first two inequalities are trivial. For the third inequality, a straightforward implementation would be problematic because of recursion and the need to remember a subset of vertices. We now show how to circumvent these issues.

Let $(\mathcal{T}, (X_t)_{t \in \mathcal{T}})$ be a given tree decomposition of a graph G of width k. Elberfeld et al. [30] showed that there is a constant $c \in \mathbb{N}$ and a logspace algorithm that given a tree $\mathcal{T}$, outputs a width-3 tree decomposition $(\mathcal{S}, (Y_s)_{s \in \mathcal{S}})$ of $\mathcal{T}$ such that $\mathcal{S}$ is a full binary tree of depth $c \cdot \log |V(\mathcal{T})|$ (their implementation in fact uses a circuit model even more restrictive than logspace). Let $Z_s = \bigcup_{t \in Y_s} X_t$; then it is easy to check that $(\mathcal{S}, (Z_s)_{s \in \mathcal{S}})$ is a tree decomposition of G of width at most $4k + 3$, also computable in logspace. Note that $\mathcal{S}$ is rooted, so we can consider the ancestor relation on it.

Since $\mathcal{S}$ has logarithmic depth, we can construct the following tree-depth decomposition of G. For $s \in V(\mathcal{S})$, let $\widetilde{Z}_s$ be the set of those vertices of Z_s, for which s is the top-most node of $\mathcal{S}$ to whose bag s belongs. Observe that $\{\widetilde{Z}_s : s \in V(\mathcal{S})\}$ is a partition of $V(G)$. Let $\mathcal{S}'$ be the tree obtained by replacing every node s of $\mathcal{S}$ by a path P_s of $|\widetilde{Z}_s|$ nodes, respecting the ancestor relation (so that the last vertex of P_s becomes the parent of the first vertex of $P_{s'}$ for every child s' of s). Consider any mapping $\mu : V(G) \to V(\mathcal{S}')$, which bijectively assigns vertices in each $\widetilde{Z}_s$ to nodes of P_s in an arbitrary order. We claim that $(\mathcal{S}, \mu)$ defines a tree-depth decomposition. To see this, consider any edge uv of G. The vertices u, v must be contained in some common bag X_t and hence in some bag Z_s, $s \in V(\mathcal{S})$. Let $Z_{s(u)}, Z_{s(v)}$ be the topmost bags containing u, v, respectively, then $u \in \widetilde{Z}_{s(u)}, v \in \widetilde{Z}_{s(v)}$. Both $s(u)$ and $s(v)$ must be ancestors of s in $\mathcal{S}$, and hence they are themselves related by the ancestor relation. Since the ancestor relation was preserved by the construction, $\mu(u)$ is related to $\mu(v)$. This shows correctness.

Membership in $\widetilde{Z}_s$, as well as $|\widetilde{Z}_s|$ can be calculated on the fly in logspace; hence, it is straightforward to perform the whole construction in logspace. The depth of $\mathcal{S}'$ is at most $3k \cdot c \cdot \log |\mathcal{T}|$. Since we assumed that in all given decompositions $|\mathcal{T}| \leq 2|V(G)|^2$, the depth is $O(k \cdot \log |V(G)|)$. □

2.3 Equivalence of Problems

We say that a reduction between two graph problems *preserves structural parameters* (linearly) if for each parameter $\pi \in \{\texttt{tw}, \texttt{pw}, \texttt{td}\}$ there is an integer $c \in \mathbb{N}$ such that for any instance with graph G, the graph H produced by the reduction satisfies $\pi(H) \leq c \cdot \pi(G)$, and moreover a decomposition of G of width/depth at most s can be transformed in logspace into a decomposition of H of width/depth at most $c \cdot s$. Many known NP-hardness reductions can be shown to have this property, in particular those that replace each vertex or edge with a gadget of bounded size (see the descriptions of "local replacement" and "component design" methods in the classical work of Garey and Johnson [35]). For example, 3COLORING and variants of SAT are equivalent in all our theorems, while VERTEX COVER or DOMINATING SET (defined in Reference [35]) are at least as hard. The proofs are deferred to Appendix A.

Definition 2.6. Let ϕ be a CNF formula. The *primal* (or *Gaifman*) *graph of ϕ* is the graph with a vertex for each variable of ϕ and an edge between every pair of variables that appear together in some clause (see, e.g., Reference [59]). The *incidence graph of ϕ* is the bipartite graph with a vertex for each clause and each variable of ϕ and an edge between each clause and every variable contained in that clause.

LEMMA 2.7. *The following problems are equivalent under logspace reductions that preserve structural parameters: 3COLORING, CNF-SAT (using a decomposition of the primal graph), k-SAT (using a decomposition of either the primal or incidence graph) for each $k \geq 3$.*

Furthermore, the following problems admit logspace reductions that preserve structural parameters from the above problems: VERTEX COVER, INDEPENDENT SET, DOMINATING SET.

We will often consider problems like π-3-SAT or π-CNF-SAT, for $\pi \in \{\texttt{tw}, \texttt{pw}, \texttt{td}\}$, in which case we always mean the width/depth of a given decomposition of the primal graph of the formula.

2.4 Cook's Theorem with Bounded Space

In our reductions, we will need to describe Turing machine computations using CNF formulas, just as in Cook's theorem on the NP-completeness of CNF-SAT. It has already been observed by Monien and Sudborough [48] that Cook's reduction applied to machines with bounded space yields formulas of bounded width (see also References [5] and [53] for an independent proof). The only difference is that in this setting a machine's worktape space bound can be significantly shorter than the input word—the read-only tape on which the input is placed must be treated differently. One standard solution would be to modify the machine to make it oblivious, i.e., simulate tape operations so that head movements are independent of the input; the reduction can then encode appropriate input symbols exactly where they would be read. We employ a different approach by encoding the reading process directly into the formula, providing the input to each computation step with a copy of the following simple *random access gadget*. This has the advantage of making our reductions slightly more explicit and adaptable.

LEMMA 2.8 (RANDOM ACCESS GADGET). *For every $n \in \mathbb{N}$, there is a 3-CNF formula including named variables: $x_0, \ldots, x_{n-1}$ ("input"), $y_0, \ldots, y_{\lceil \lg n \rceil - 1}$ ("index"), and z ("output"), such that: every assignment of the named variables can be extended to a satisfying assignment if and only if it satisfies $z = x_{\bar{y}}$. Here, $\bar{y}$ is the number encoded in binary by the index variables (we require $z = 0$ if $\bar{y} \geq n$). The formula has $O(n)$ variables in total, tree-depth $O(\log n)$ (of the primal graph), and can be constructed using $O(\log n)$ space, given n.*

PROOF. Construct a full binary tree of variables, of depth $\lceil \lg n \rceil$. Let the root be z and name the leaves $x_0, x_1, \ldots, x_{2^{\lceil \lg n \rceil} - 1}$. Add clauses of size 1 requiring $x_i = 0$ for $i \geq n$. Introduce new variables

$y_0, \ldots, y_{\lceil \lg n \rceil - 1}$. For each internal variable v at level j of the tree, with children v_0, v_1, add clauses enforcing that $v = v_0$ if y_j is false and $v = v_1$ otherwise (four clauses of size 3 on the variables $\{v, v_0, v_1, y_j\}$). This enforces that any assignment to the input and index variables can be extended to a satisfying assignment in exactly one way, in which furthermore the root variable z has the same assignment as the leaf variable x. indexed by $\bar{y}$, that is, $z = x_{\bar{y}}$. A tree-depth decomposition of logarithmic depth is obtained from the full binary tree simply by attaching a new path of length $\lceil \lg n \rceil$ above the root of the tree, moving the root to the other end of the path, and mapping the index variables nodes of this path, in an arbitrary order.   □

The following lemma shows more precisely (for our needs) how a Turing machine computation can be encoded in a formula (a *computation gadget*)—the crucial part of Cook's theorem. In reductions involving stack machines it will also describe fragments of computation without any push/pop operation. The contents of the stack will be considered as a separate read-only input tape, which we treat differently because, while smaller in size, the content is not given to the reduction.

Note that Lemmas 2.9 and 2.7, applied for $h = 0$, immediately give the first part of Theorem 1.1: for nice $s(n) \geq \log n$, pw-3COLORING $[s]$ is complete for $\mathbf{N}[\underset{time}{\text{poly}}, \underset{space}{s(\text{poly})}]$.

LEMMA 2.9 (COMPUTATION GADGET). *Let M be a non-deterministic Turing machine over alphabet Σ with two read-only input tapes and one work tape. Given an input word α over Σ of length n and integers s, t, h such that $\lg n + \lg h \leq O(s)$, one can, using at most $O(\log(n + s + t + h))$ space, output a CNF formula such that:*

- *The formula has $O(t \cdot (s + h + n))$ variables, including named variables $u_1, \ldots, u_{s'}$, $v_1, \ldots, v_{s'}, w_1, \ldots, w_{h'}$, where $s' \in \Theta(s)$ and $h' = h \cdot \lceil \lg |\Sigma| \rceil$. These variables, respectively, describe two configurations $\mathbf{u}, \mathbf{v}$ of M (up to s symbols of the working tape, heads' positions encoded in binary, and the state), and a word $\bar{w}$ over Σ of length h.*
- *Any assignment to the named variables can be extended to a satisfying assignment iff the computation of M on inputs α and $\bar{w}$ can lead (by some sequence of non-deterministic choices) from the configuration $\mathbf{u}$ to the configuration $\mathbf{v}$, using at most t steps and s space.*
- *The formula's primal graph has pathwidth $O(s + h)$ and tree-depth $O(s \cdot \log(n + s + t + h) + h)$. Moreover, appropriate decompositions can be output within the same space bound.*

PROOF. For simplicity of presentation, we assume input tapes use the binary alphabet. A larger alphabet can be reduced by encoding each symbol using a block of $\lceil \lg |\Sigma| \rceil$ bits and applying straightforward modifications to the machine M.

As in Cook's original proof, we create t blocks of $s' \in \Theta(s)$ variables each, describing the configuration at each step. The first and last blocks contain variables $(u_i)_{1 \leq i \leq s'}$ and $(v_i)_{1 \leq i \leq s'}$, respectively, to encode configurations $\mathbf{u}$ and $\mathbf{v}$. Additionally, variables $(w_i)_{1 \leq i \leq h}$ are created.

We may assume that machine M always keeps track of the indices on which the heads of the input tapes are placed. These indices are encoded in binary in pre-defined buffers on the working tape (*index buffers*), and each time the head on an input tape is moved, the machine updates the index. To encode reading access to the input tapes in the formula, for each step and each input tape, we create a copy of the random access gadget of Lemma 2.8. For the first input tape, the gadget has its input variables fixed with the bits of the word α (given to the reduction). For the second input tape, the gadget has its input variables identified with $w_1, \ldots, w_h$. The index variables are identified with the variables of the block that encode the contents of respective index buffers. The machine behavior is then encoded with clauses binding variables of two consecutive blocks, including the output bit z of each random-access gadget, exactly as in Cook's proof: the clauses verify the correctness of the transition. In doing this, we allow at each step a transition that does

$$\mathbf{N}[f\mathrm{poly}, f\log] \not\subseteq \mathbf{D}[n^{f+\log}, f\mathrm{poly}] \overset{\text{(Lemma 3.7)}}{\Longleftrightarrow}$$

There is no algorithm for pw-3Coloring working in time $2^{o_{\mathrm{eff}}(s^2)}n^{O(\log n)}$ and space $2^{o_{\mathrm{eff}}(s)}\mathrm{poly}(n)$

rescaling

$\Downarrow$ (trivial)

$$\mathbf{NL} \not\subseteq \mathbf{D}[\underset{time}{2^{o_{\mathrm{eff}}(\log^2 n)}}, \underset{space}{n^{o_{\mathrm{eff}}(1)}}]$$

For some computable unbounded function g there is no algorithm for pw-3Coloring working in time $2^{s \cdot g(s)}\mathrm{poly}(n)$ and space $2^{o_{\mathrm{eff}}(s)}\mathrm{poly}(n)$

$\Downarrow$ (Lemma 3.6)

Conjecture 1.2 (Thm 3.3)

(Thm 3.1) $\updownarrow$ $\Longleftrightarrow$

$$\mathbf{N}[f\mathrm{poly}, f\log] \not\subseteq \mathbf{D}[n^f, f\mathrm{poly}]$$

There is no algorithm for pw-3Coloring working in time $n^{f(s/\lg n)}$ and space $f(s/\lg n)\mathrm{poly}(n)$ for a computable f

$\Downarrow$ (Lemma 3.4)

There is no algorithm for pw-3Coloring working in time $2^{O(s)}\mathrm{poly}(n)$ and space $2^{o_{\mathrm{eff}}(s)}\mathrm{poly}(n)$

$\Downarrow$ (trivial)

(Lemma 3.5)

$$\mathbf{NL} \not\subseteq \mathbf{SC} \Longleftrightarrow$$

There is no algorithm for pw-3Coloring working in time $2^{O(s)}\mathrm{poly}(n)$ and space $\mathrm{poly}(s, \log n)$
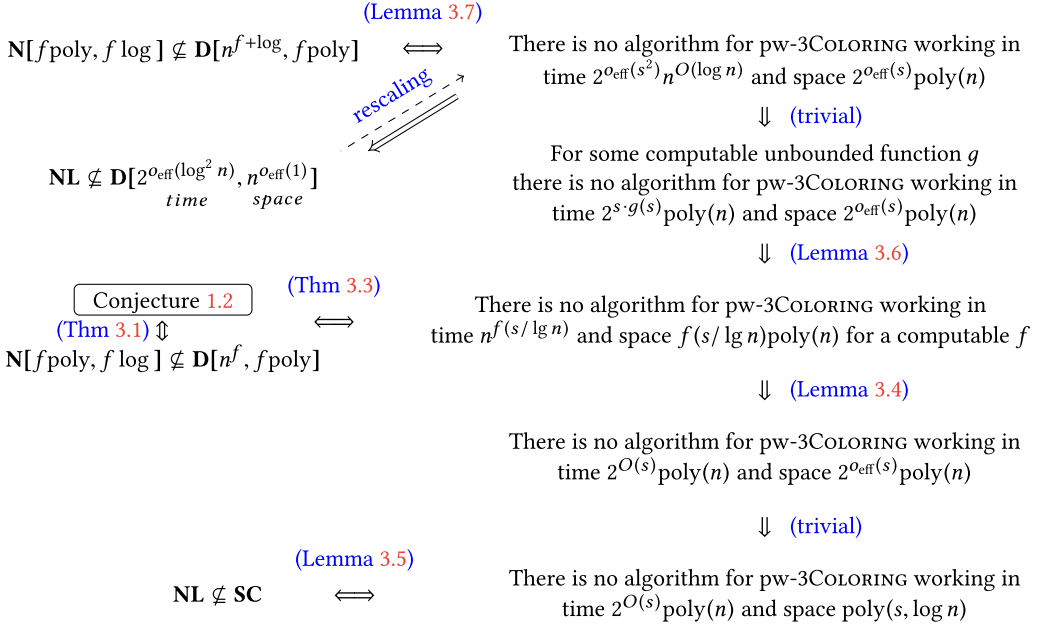
Fig. 1. A summary of the relationships between various statements related to Conjecture 1.2.

not change the configuration in any way. Transitions that would move the working tape's head outside the first s symbols are not allowed. This construction makes the formula satisfiable exactly with assignments in which the tth block describes a configuration reachable in at most t steps and s space (from the configuration described by u variables).

It remains to bound the pathwidth and the tree-depth of the constructed formula's primal graph. To construct a path decomposition, create $t - 1$ bags $A_1, A_2, \ldots, A_{t-1}$, where each A_i contains a pair of consecutive configuration blocks i and $i + 1$ (i.e., variables describing the configuration just before and after a single computation step) and all the variables $(w_i)_{1 \leq i \leq h}$. Consider one of the two random-access gadgets created for step i, and let $B_1, \ldots, B_b$ be the bags of the provided path decomposition of this gadget ($|B_j| = O(\log n + \log h) \leq O(s)$ for each $1 \leq j \leq b$). Similarly, let $B'_1, \ldots, B'_{b'}$ be the bags of the provided path decomposition of the second gadget ($|B'_j| \leq O(s)$). Then, the final decomposition is obtained by adding bags $A_i \cup B_1, \ldots, A_i \cup B_b, A_i \cup B'_1, \ldots, A_i \cup B'_{b'}$ immediately after A_i. From the construction it follows that each bag contains at most $O(s' + h + s) = O(s + h)$ variables and each clause binds a set of variables contained in one of the bags. For the tree-depth bound, consider the above decomposition with the variables $w_1, \ldots, w_h$ removed. Its width is $O(s)$, and hence using Lemma 2.4, we can obtain a tree-depth decomposition of the formula's primal graph of depth $O(s \cdot \log(n + s + t + h))$ if we removed the w variables. The variables can then be reintroduced by placing them atop all others in the decomposition, in any order, which increases the tree-depth by at most h. □

3 CONNECTIONS WITH TRADEOFFS FOR LCS

In this section, we relate Conjecture 1.2 to statements of varying strength concerning complexity class containments, or time-space tradeoffs for pathwidth-constrained problems. The results are summarized in Figure 1. Here, we consider algorithms that work uniformly for all values of

pathwidth, with complexity depending on both the input size n and the pathwidth s of a given decomposition.

We use the notion of pl-reduction between parameterized problems: it is an algorithm that transforms an instance of one problem with parameter k into an equivalent instance of another problem with parameter $k' \leq f(k)$, working in space $f(k) + O(\log n)$, for some computable function f. Following Elberfeld et al. [31] we define[6] $\mathbf{N}[f\text{poly}, f \log]$ as the class of parameterized problems that can be solved in non-deterministic time $f(k)\text{poly}(n)$ and space $f(k)\log(n)$ for some computable function f, where k is the parameter. Similarly, $\mathbf{D}[n^f, f\text{poly}]$ is the class of parameterized problems that can be solved in deterministic time $n^{f(k)}$ and space $f(k)\text{poly}(n)$ for some computable function f. Further classes $\mathbf{D}[t, s]$ will be defined analogously for different expressions t, s. All those mentioned in the article are closed under pl-reductions.

The reason we do not use the better known fpt-reductions is that $\mathbf{N}[f\text{poly}, f \log]$ is not expected to be closed under such reductions; its closure under fpt-reductions has been called WNL by Guillemot [38], though Elberfeld et al. [31] argue that a different parameterized class should have this name.

Recall that we use the notation $o_{\text{eff}}(h(n))$ as an effective variant of $o(h(n))$; formally, for $f, h : \mathbb{N} \to \mathbb{N}$, we write $f = o_{\text{eff}}(h)$ if there is a non-decreasing, unbounded, computable function $g(n)$ such that $f = O(\frac{h}{g})$.

The *inverse* of a function f is the function $f^{-1}(n) := \max\{i : f(i) \leq n\}$; observe that $f(f^{-1}(n)) \leq n \leq f^{-1}(f(n))$. Cai et al. [17] showed that whenever a computable function f is only used as an arbitrarily quickly growing upper bound, it can be assumed without loss of generality that $f(n)$ is computable in space $O(\log f(n))$ and its inverse is computable in space $O(\log n)$. Formally (we give here slightly stronger statements that follow from exactly the same proofs as in Reference [17]):

LEMMA 3.1 ([17]). *For any computable function f, there is an increasing function f' satisfying $f'(n) \geq f(n)$ and $f'(n) > n$ that is computable in space $O(\log f'(n))$.*

LEMMA 3.2 ([17]). *For any unbounded, non-decreasing function $f(n) > n$ computable in space $O(\log f(n))$, the inverse $f^{-1}(n)$ is computable in space $O(\log n)$.*

COROLLARY 3.3. *For any unbounded, computable function g, there is an unbounded, computable, non-decreasing function g' satisfying $g'(n) \leq g(n)$ and $g'(n) < n$ that is computable in space $O(\log n)$.*

PROOF. We apply Lemma 3.1 to $f := g^{-1}$ to obtain an unbounded, (strictly) increasing function f' satisfying $f'(n) \geq g^{-1}(n)$. Let $g' := f'^{-1}$. By definition, g' is non-decreasing and since $g'(f'(n)) = f'^{-1}(f'(n)) \geq n$, it is unbounded. By Lemma 3.2, $g'(n)$ is computable in space $O(\log n)$. Furthermore, observe that $f'(g(n)) \geq g^{-1}(g(n)) \geq n$. Since f' is increasing, for $i > g(n)$ we have $f'(i) > n$. Thus $g'(n) = \max\{i : f'(i) \leq n\} \leq g(n)$. Finally since $f'(n) > n$, we have $g'(n) = \max\{i : f'(i) \leq n\} < n$. □

3.1 Completeness Results and Statements Equivalent to Conjecture 1.2

Conjecture 1.2 refers to the following parameterized problem.

LCS (longest common subsequence)	**Parameter:** k

Input: A finite alphabet Σ, k strings $s_1, s_2, \ldots, s_k$ over Σ, and an integer ℓ.
Question: Is there a common subsequence of $s_1, s_2, \ldots, s_k$ of length at least ℓ?

---

[6]Throughout this section, in classes $\mathbf{N}[\cdot, \cdot]$ and $\mathbf{D}[\cdot, \cdot]$, we always use time and space as the first and the second argument, respectively; hence, we drop the subscripts for readability.

We stress that in this variant, the alphabet Σ is not of fixed size, but it is given on the input and can be arbitrarily large. Under standard fpt-reductions, LCS is known to be W[t]-hard for every level t [16]. Moreover, there is some fixed alphabet size for which the problem remains W[1]-hard [55], and the problem is also W[1]-complete when parameterized jointly by k and ℓ [38]. This makes it very hard from the parameterized perspective. From the point of view of pl-reductions, Elberfeld et al. [31], drawing on the work of Guillemot [38], pinpointed the exact complexity of the LCS problem.

THEOREM 3.4 ([31]). *LCS is complete for* $\mathbf{N}[f\,\mathrm{poly}, f\log\,]$ *under pl-reductions.*

Thus Conjecture 1.2 is in fact a general statement about parameterized complexity classes.

COROLLARY 3.5. *Conjecture 1.2 holds if and only if* $\mathbf{N}[f\,\mathrm{poly}, f\log\,] \not\subseteq \mathbf{D}[n^f, f\,\mathrm{poly}]$.

Similarly, as described in the introduction, the best known determinization results (a brute-force approach and Savitch's theorem) imply only that $\mathbf{N}[f\,\mathrm{poly}, f\log\,]$ is contained in classes $\mathbf{D}[n^f, n^f]$ (commonly known as **XP**) and $\mathbf{D}[n^{f(k)\cdot\log n}, f(k)\cdot\log^2 n]$.

To relate parameterized tractability bounds to subexponential bounds, we use the following tool from Cai and Juedes [18]. For a parameterized problem Π, its reparametrization (or *extended version*) $\Pi^{\log n}$ is defined as the same problem parameterized by $s/\lg n$, where s was the old parameter[7]. In particular, pw-3COLORING $^{\log n}$ is the following parameterized problem:

---

pw-3COLORING$^{\log n}$ **Parameter:** k
Input: A graph G with a path decomposition of width $k \cdot \lg n$
Question: Is G 3-colorable?

Similarly, as in Theorem 1.1, pathwidth-constrained problems turn out to be complete for non-deterministic computation with simultaneous time and space bounds.

THEOREM 3.6. *The problem* pw-3COLORING$^{\log n}$ *is complete for* $\mathbf{N}[f\,\mathrm{poly}, f\log]$ *under pl-reductions.*

PROOF. To show containment, we give a non-deterministic algorithm solving pw-3COLORING in time $\mathrm{poly}(n)$ and space $O(s + \log n)$, where s is the width of the path decomposition given on the input. The algorithm proceeds on consecutive bags of the path decomposition by guessing the color of every vertex when it is introduced for the first time and comparing it with previously guessed colors of adjacent vertices in the bag. At any moment, only the colors of all vertices in the current bag and the position in the decomposition need to be remembered; hence, $O(s + \log n)$ space suffices. Correctness follows, since every color is guessed exactly once and every edge is checked, because every pair of adjacent vertices is contained in some bag. Since $k = \frac{s}{\lg n}$ is the parameter of the rescaled problem, the same algorithm works in time $\mathrm{poly}(n)$ and space $O(k \log n + \log n)$, and hence in time $f(k)\mathrm{poly}(n)$ and space $O(f(k)\log n)$ for $f(k) = k$.

To show hardness, consider any non-deterministic Turing machine M solving a problem with input size n and parameter k in time $t(n, k) = f(k)\mathrm{poly}(n)$ and space $s(n, k) = O(f(k)\log n)$, for some computable function f. By Lemma 3.1, we can assume without loss of generality that $f(k)$ is computable in space $O(\log f(k))$. Given an instance α of this problem with size n and parameter k, the reduction of Lemma 2.9, with $h = 0$, $t = t(n, k)$, $s = s(n, k)$, will output a CNF-SAT instance φ, using $O(\log(n + s + t))) = O(\log f(k) + \log n)$ space. Note we need to compute t and s using the computability of f. We then add $O(s') = O(s)$ clauses on the named variables $u_1, \ldots, u_{s'}$ and

---

[7]A careful reader probably noticed that some floors/ceilings are formally necessary here. For the sake of readability, in this section, we ignore such straightforward details.

$v_1, \ldots, v_{s'}$ that enforce the two configurations $\mathbf{u}, \mathbf{v}$ they describe to be the initial configuration and an accepting configuration, respectively. Hence, the modified formula φ' is satisfiable if and only if M accepts α. By Lemma 2.9, we further obtain a path decomposition of the primal graph of φ of width $O(s) = O(f(k) \log n)$; by adding all the named variables $u_1, \ldots, u_{s'}$ and $v_1, \ldots, v_{s'}$ to all bags of the decomposition, we obtain a path decomposition of the primal graph of φ' of width $O(s + s') = O(f(k) \log n)$. Using Lemma 2.7, we can then reduce this instance φ' of CNF-SAT to an instance of pw-3COLORING with a path decomposition of width $O(f(k) \log n)$, that is, to pw-3COLORING $^{\log n}$ with parameter $k' = O(f(k))$. □

Conjecture 1.2 is thus equivalent to the statement that pw-3COLORING$^{\log n}$ is not in $\mathbf{D}[n^f, f \operatorname{poly}]$, in other words, that pw-3COLORING cannot be solved deterministically in time $n^{f(s/\lg n)}$ and space $f(s/\lg n)\operatorname{poly}(n)$ for any computable function f, where s is the width of the input path decomposition. To contrast pathwidth with tree-depth, one may easily observe (see Lemma 4.5) that an instance of 3COLORING with a tree-depth decomposition of depth s can be solved deterministically using $O(s + \log n)$ space. This places td-3COLORING$^{\log n}$ in $\mathbf{D}[n^f, f \log]$, a class usually known as **XL**.

3.2 Statements Weaker than Conjecture 1.2

Similarly, as in the work of Cai and Juedes [18], we show that the parameterized complexity of the reparameterized problem pw-3COLORING$^{\log n}$ is related to subexponential bounds in the complexity of pw-3COLORING. Recall that we always assume that instances of pw-3COLORING come with appropriate path decompositions of the graph.

LEMMA 3.7. *Assuming Conjecture 1.2, there is no algorithm for* pw-3COLORING *working in time* $2^{O(s)}\operatorname{poly}(n)$ *and space* $2^{o_{\mathrm{eff}}(s)}\operatorname{poly}(n)$ *(for all values of pathwidth s).*

PROOF. We show the contrapositive. Suppose that pw-3COLORING can be solved in time $2^{O(s)}\operatorname{poly}(n)$ and space $2^{o_{\mathrm{eff}}(s)}\operatorname{poly}(n)$. We show that $\mathbf{N}[f\operatorname{poly}, f\log] \subseteq \mathbf{D}[n^f, f\operatorname{poly}]$, negating Conjecture 1.2. The assumption implies that pw-3COLORING$^{\log n}$ can be solved in time $2^{O(k \cdot \lg n)} = n^{O(k)}$ and space $2^{k \cdot \lg n / g(k \cdot \lg n)}$ for some unbounded and non-decreasing computable function $g(\cdot)$. If $k \leq g(k \cdot \lg n)$, then the bound on space is bounded by n. Otherwise, if $k > g(k \lg n) \geq g(\lg n)$, then $n \leq 2^{g^{-1}(k)}$. In this case, the bound on space is bounded by a computable function of k, namely $2^{k \cdot g^{-1}(k)}$, because g is computable. Hence, in each case, the same algorithm solves pw-3COLORING$^{\log n}$ in time $n^{O(k)}$ and space $n + 2^{k \cdot g^{-1}(k)}$. By Theorem 3.6, this implies $\mathbf{N}[f\operatorname{poly}, f\log] \subseteq \mathbf{D}[n^f, f\operatorname{poly}]$. □

An even weaker statement is equivalent to $\mathbf{NL} \nsubseteq \mathbf{SC}$ by a simple padding argument.

LEMMA 3.8. *There is no algorithm for* pw-3COLORING *working in time* $2^{O(s)}\operatorname{poly}(n)$ *and space* $\operatorname{poly}(s, \log n)$ *(for all values of pathwidth s) if and only if* $\mathbf{NL} \nsubseteq \mathbf{SC}$.

PROOF. If there was such an algorithm, then it would solve pw-3COLORING[$\log n$] in polynomial time and polylogarithmic space. However, from Theorem 1.1 it follows that pw-3COLORING[$\log n$] is complete for $\mathbf{NL}$, so this would imply that $\mathbf{NL} \subseteq \mathbf{SC}$.

For the other direction, suppose $\mathbf{NL} \subseteq \mathbf{SC}$. Then, pw-3COLORING on instances with path decompositions of width at most $\lg n$ can be solved in polynomial time and space $O(\log^c n)$ for some constant c. Thus, an instance of size n and a path decomposition of width s can be padded to size $n' = \max(n, 2^s)$ (by adding isolated vertices—neither the answer nor the width changes) and solved by this algorithm in time $\operatorname{poly}(n')$ and space $O(\log^c n')$. This solves pw-3COLORING in time at most $\max(\operatorname{poly}(n), \operatorname{poly}(2^s)) \leq 2^{O(s)}\operatorname{poly}(n)$ and space $\max(\log^c n, s^c) \leq \operatorname{poly}(s, \log n)$. □

3.3 Statements Stronger than Conjecture 1.2

Contrary to the results of Cai and Juedes [18], in the considered context, our methods seem insufficient to prove the exact converse of Lemma 3.7. However, we can get arbitrarily close to it, in a sense.

LEMMA 3.9. *If Conjecture 1.2 fails, then for every arbitrarily slowly growing, unbounded, computable function g, pw-3COLORING can be solved in time $2^{s \cdot g(s)}\mathrm{poly}(n)$ and space $2^{o_{\mathit{eff}}(s)}\mathrm{poly}(n)$ (for all values of pathwidth s).*

PROOF. The assumption is equivalent to $\mathbf{N}[f\mathrm{poly}, f\log] \subseteq \mathbf{D}[n^f, f\mathrm{poly}]$, which in turn implies that pw-3COLORING can be solved in time $n^{f(s/\lg n)}$ and space $f(s/\lg n) \cdot \mathrm{poly}(n)$ for some computable, increasing function f. By Lemma 3.1, we can assume without loss of generality that $f(n)$ is computable in space $O(\log f(n))$. Let $g(s)$ be an arbitrarily slowly growing, unbounded, computable function. By Corollary 3.3, we can assume without loss of generality that $g(s)$ is non-decreasing, computable in space $O(\log s)$ and that $g(s) < s$. Let $n' = 2^{s/f^{-1}(g(s))}$. Observe that $f(s/\log n') = f(f^{-1}(g(s))) \le g(s)$ and that $f^{-1}(g(s))$ is unbounded and computable in space $O(\log s)$, by Lemma 3.2.

Consider an instance of pw-3COLORING of size n with a path decomposition of width s. If $s \le \lg n$, then the assumed algorithm runs in time $n^{f(s/\lg n)} \le n^{f(1)} = \mathrm{poly}(n)$ and space $f(1) \cdot \mathrm{poly}(n) = \mathrm{poly}(n)$. If $\lg n < s \le f^{-1}(g(s)) \cdot \lg n$, then $f(s/\lg n) \le g(s)$ and hence the assumed algorithm runs in time $n^{f(s/\lg n)} \le n^{g(s)} = 2^{\lg n \cdot g(s)} \le 2^{s \cdot g(s)}$ and space $f(s/\lg n) \cdot \mathrm{poly}(n) \le g(s) \cdot \mathrm{poly}(n) = \mathrm{poly}(n)$ (the last equality follows from $g(s) \le s$ and $s \le n$). Finally, if $s > f^{-1}(g(s)) \cdot \lg n$, then $n' > n$; hence, we can pad the instance (by adding isolated vertices—neither the answer nor the width changes) to size n'. Applying the assumed algorithm to the padded instance solves the problem in time $n'^{f(s/\lg n')} \le 2^{s \cdot \frac{g(s)}{f^{-1}(g(s))}} \le 2^{s \cdot g(s)}$ and space $f(s/\lg n') \cdot n'^{O(1)} \le g(s) \cdot 2^{O(s/f^{-1}(g(s)))} = 2^{o_{\mathit{eff}}(s)}$. Note that to compute the padding length n' from s we use the computability of $f^{-1}(g(s))$ in logarithmic space. □

For a somewhat less natural, stronger variant of Conjecture 1.2, we can show a similar, but exact correspondence (note the quasi-polynomial factor on both sides). The proof is very similar.

LEMMA 3.10. *There is no algorithm for pw-3Coloring with $2^{o_{\mathit{eff}}(s^2)}n^{O(\log n)}$ time and $2^{o_{\mathit{eff}}(s)}\mathrm{poly}(n)$ space complexity (working for all values of pathwidth s) iff $\mathbf{N}[f\mathrm{poly}, f\log] \nsubseteq \mathbf{D}[n^{f+\log}, f\mathrm{poly}]$.*

PROOF. Suppose first there is an algorithm for pw-3COLORING working in time $2^{s^2/g(s)}n^{O(\log n)}$ and space $2^{s/g(s)}\mathrm{poly}(n)$, for some unbounded, non-decreasing, computable function g. Consider an instance of pw-3COLORING$^{\log n}$ of size n, parameter k, and hence equipped with a path decomposition of the graph of width $k \cdot \lg n$. If $k^2 < g(\lg n)$, then the algorithm solves the instance in time at most $2^{\frac{k^2 \lg^2 n}{g(k \lg n)}}n^{O(\log n)} \le 2^{O(\log^2 n)}$ and space $2^{\frac{k \lg n}{g(k \lg n)}} \le \mathrm{poly}(n)$. Otherwise, if $g(\lg n) \le k^2$, then n is bounded by a computable function of k, namely $2^{g^{-1}(k^2)}$. Hence, the algorithm solves the instance in time and space bounded by a computable function f of k. Therefore, in any case, the same algorithm solves pw-3COLORING$^{\log n}$ in time $f(k) + 2^{O(\log^2 n)}$ and space $f(k) + \mathrm{poly}(n)$, which implies $\mathbf{N}[f\mathrm{poly}, f\log] \subseteq \mathbf{D}[n^{f+\log}, f\mathrm{poly}]$.

For the converse, suppose now that $\mathbf{N}[f\mathrm{poly}, f\log] \subseteq \mathbf{D}[n^{f+\log}, f\mathrm{poly}]$. Then, there is an algorithm for pw-3COLORING working in time $n^{f(s/\lg n)+O(\log n)}$ and space $f(s/\lg n)\mathrm{poly}(n)$ on instances of size n and with path decompositions of width s, for some computable, increasing function f (by Lemmas 3.1 and 3.2, we can assume that $f^{-1}(n)$ is computable in space $O(\log n)$). Given such an instance of pw-3COLORING, let $n' = 2^{s/f^{-1}(\lg n)}$. Note that the value n' can be computed from s and n in space $O(s + \log n)$. If $n' < n$, then $\frac{s}{\lg n} < f^{-1}(\lg n)$; hence, $f(s/\lg n) < f(f^{-1}(\lg n)) \le \lg n$

and the assumed algorithm works in time $n^{O(\log n)}$ and space $\text{poly}(n)$. Otherwise, if $n' \geq n$, then we can pad the instance to size n' (by adding isolated vertices—neither the answer nor the width changes). Then, since $\lg n' = s/f^{-1}(\lg n) \leq s/f^{-1}(\lg s) = o_{\text{eff}}(s)$, the padded instance is solved in time $2^{\lg n' \cdot (\lg n + O(\lg n'))} = 2^{O(\lg^2 n')} = 2^{o_{\text{eff}}(s^2)}$ and space $\lg n \cdot \text{poly}(n') = 2^{O(\lg n')} = 2^{o_{\text{eff}}(s)}$. □

3.4 A Summary

Theorem 1.3 follows from Lemmas 3.7 and 3.9. We summarize the relationships around Conjecture 1.2 in Figure 1. The weakest statement there is $\mathbf{NL} \not\subseteq \mathbf{SC}$, a widely explored hypothesis in complexity theory (see, e.g., References [10], [19], [52], [63]). Since DIRECTED (s, t)-REACHABILITY (asking given a directed graph and two nodes s, t, whether is t reachable from s) is an $\mathbf{NL}$-complete problem, this is also equivalent to the question of whether this problem can be decided in polynomial time and polylogarithmic space simultaneously. However, even this weakest statement is not known to be implied by better known conjectures such as the Exponential Time Hypothesis. It seems that the simultaneous requirement on bounding two complexity measures—time and space—has a nature that is independent of the usual time complexity considerations. Hence, new complexity assumptions may be needed to explore this paradigm, and we hope that Conjecture 1.2 may serve as a transparent and robust example of such.

In a certain restricted computation model (allowing operations on graph nodes only, not on individual bits), unconditional tight lower bounds have been proved by Edmonds et al. [27]: it is impossible to decide DIRECTED (s, t)-REACHABILITY in time $2^{o(\log^2 n)}$ and space $O(n^{1-\varepsilon})$ (for any $\varepsilon > 0$), even if randomization is allowed. Essentially all known techniques for solving DIRECTED (s, t)-REACHABILITY are known to be implementable in this model [47] (including Depth- and Breadth-First Search, as well as the well-known theorems of Savitch, of Immerman and Szelepcsényi, and Reingold's breakthrough); therefore, this strongly suggests that no algorithm running in time $2^{o_{\text{eff}}(\log^2 n)}$ and space $n^{o_{\text{eff}}(1)}$ is possible, that is, $\mathbf{NL} \not\subseteq \mathbf{D}[\underset{time}{2^{o_{\text{eff}}(\log^2 n)}}, \underset{space}{n^{o_{\text{eff}}(1)}}]$.

By Theorem 1.1, this is equivalent to saying that pw-3COLORING[log] cannot be solved in these time and space bounds. The strongest statement in Figure 1 is a rescaling of this, that is, it implies $\mathbf{NL} \not\subseteq \mathbf{D}[\underset{time}{2^{o_{\text{eff}}(\log^2 n)}}, \underset{space}{n^{o_{\text{eff}}(1)}}]$ by a trivial padding argument, but the reverse implication is also probable in the sense that any proof of the latter would likely scale to prove the former. However, it is still possible that an algorithm working in polynomial space refutes the stronger statement even though $\mathbf{NL} \not\subseteq \mathbf{D}[\underset{time}{2^{o_{\text{eff}}(\log^2 n)}}, \underset{space}{n^{o_{\text{eff}}(1)}}]$.

Let us finally mention that $\mathbf{NL} \not\subseteq \mathbf{SC}$ would imply that $\mathbf{NC} \not\subseteq \mathbf{SC}$ (where $\mathbf{NC}$ stands for polynomial size circuits of poly-logarithmic depth), but even for this far weaker separation there are few technical results indicating that it is true. One natural conjecture that is known to imply it, but not known to imply $\mathbf{NL} \not\subseteq \mathbf{SC}$, is the *depth irreducibility hypothesis* of Reference [54], saying intuitively that a super-constant decrease in circuit depth requires a super-quasi-polynomial increase in size.

4 TREEDEPTH

4.1 Characterization via NTMs with a Small Auxilliary Stack

In this section, we prove a completeness theorem for small tree-depth computations, i.e., Theorem 1.4. Let $s : \mathbb{N} \to \mathbb{N}$ be a nice function. Before we proceed, we discuss more precisely the model of machines used to define the class $\mathbf{NAuxSA}[\underset{time}{\text{poly}}, \underset{space}{\log}, \underset{height}{s}]$. The machine has three tapes: a read-only input tape, a working tape of length $O(\log n)$, and a stack tape of length $s(n)$. Each

tape contains symbols from some fixed, finite alphabet Σ. Initially both the working tape and the stack tape are empty, i.e., filled with blank symbols. On each of the tapes there is a head, and the transitions of the machine depend on its state and the triple of symbols under the heads. The access restraints to each of the tapes are as follows: The input tape is a read-only tape. The working tape can be both read and written on by the machine. The stack tape can be read but not freely written on; instead, the transitions of the machine may contain instructions of the form $\mathsf{push}\,\sigma$ or pop, where σ is some non-blank symbol of Σ. In the case of $\mathsf{push}\,\sigma$, the first blank symbol of the tape is replaced by σ. In the case of pop, the last non-blank symbol of the tape is replaced by a blank. Since s is nice, $s(n) \leq \mathrm{poly}(n)$, so within the working tape the machine can keep track of the current height of the stack and the index on which the stack's head is positioned. The machine accepts if it can reach an accepting state through a sequence of transitions, and for a problem in $\mathbf{NAuxSA}[\underset{time}{\mathrm{poly}}, \underset{space}{\log}, \underset{height}{s}]$, there must always be an accepting run where the number of transitions is bounded by a polynomial in n.

In this section, we show that, for any nice function $s(n) \geq \log^2 n$, td-3COLORING $[s]$ is complete for $\mathbf{NAuxSA}[\underset{time}{\mathrm{poly}}, \underset{space}{\log}, \underset{height}{s}]$ and $\mathbf{NAuxSA}[\underset{time}{\mathrm{poly}}, s/\underset{space}{\log}, \underset{height}{s}]$ under logspace reductions. In particular, this implies that these two classes are equal. We start by showing containment, exemplifying how the resources are used.

LEMMA 4.1. *For any nice function $s(n)$,* td-3Coloring[s] *is in* $\mathbf{NAuxSA}[\underset{time}{\mathrm{poly}}, \underset{space}{\log}, \underset{height}{s}]$.

PROOF. The input consists of a graph G and a mapping of its vertices into a rooted forest $\mathcal{T}$; we assume any natural encoding for which validity of the decomposition can be checked in logarithmic space. We now give a non-deterministic algorithm showing that td-3COLORING $[s]$ belongs to $\mathbf{NAuxSA}[\underset{time}{\mathrm{poly}}, \underset{space}{\log}, \underset{height}{s}]$. The algorithm considers the trees of $\mathcal{T}$ one by one, and for each tree T of $\mathcal{T}$, it explores T by a depth-first search beginning from the root. For this, it suffices to maintain in memory the identifier of the current node visited in T, since after exploring a node, the next node in DFS order (i.e., its next sibling, or its parent when no such sibling exists, or the next root if the node has no parent) can be retrieved in logarithmic space from any reasonable encoding of T.

When the depth-first search enters a node t of T, the algorithm non-deterministically guesses a color (encoded using a constant number of bits) for the vertex in G that maps to t, and pushes this color onto the stack. When the depth-first search withdraws from a vertex, the algorithm pops its color from the stack. Thus, the stack always contains the sequence of guessed colors for vertices mapped to t and its ancestors (nodes on the path from t to the root of T, in order).

To verify the correctness of the coloring, whenever the algorithm guesses the color of some vertex u, it checks that all ancestors of u in $\mathcal{T}$ that are adjacent to u in G have different colors from u and rejects if this ever fails; these colors are available on the stack, and adjacency can be checked in logarithmic space. If the whole graph is explored without rejecting, the algorithm accepts. In this way, execution paths of this non-deterministic algorithm correspond to sequences of guesses: exactly one color for each vertex of G; furthermore, an execution path accepts only if for each vertex u, all its neighbors that are ancestors received colors different from it. By definition of tree-depth decompositions, every edge uv of G has one endpoint mapped to an ancestor of the other; therefore, an accepting execution path exists if and only if it is possible to guess the colors in a way that adjacent vertices get different colors, that is, if and only if there is a proper 3-coloring. □

When $s(n) \geq \log^2 n$, $\mathbf{NAuxSA}[\underset{time}{\mathrm{poly}}, \underset{space}{\log}, \underset{height}{s}] \subseteq \mathbf{NAuxSA}[\underset{time}{\mathrm{poly}}, \underset{space}{s/\log}, \underset{height}{s}]$. The next step is to show how the stack operations of the latter class' machines can be regularized. This idea originates in the approach of Akatov and Gottlob [3]. Following their ideas, we define a regular stack machine in the following way. For any valid sequence S of push and pop operations that starts and ends with an empty stack, define the corresponding push-pop tree $\tau(S)$ to be the ordered tree (a tree with an order imposed on the children of each node) in which a depth-first search would result in the sequence S; specifically,

- $\tau(\varepsilon)$ is a single root node,
- $\tau(\mathrm{push}\ S\ \mathrm{pop})$ is a new root node with $\tau(S)$ attached as the only child subtree,
- $\tau(S_1 S_2)$ is obtained from $\tau(S_1)$ and $\tau(S_2)$ by identifying their roots, and putting all the children of the root of $\tau(S_1)$ before all children of the root of $\tau(S_2)$.

We say that a language is in reg-$\mathbf{NAuxSA}[\underset{time}{\mathrm{poly}}, \underset{space}{s/\log}, \underset{height}{s}]$ if it is recognized by an NTM with $s(n)/\log(n)$ working space and an auxiliary stack of height $s(n)$ that has the following properties:

(1) The machine pushes and pops blocks of $\lceil s(n)/\lg(n) \rceil$ symbols at a time. More precisely, the machine uses a pre-specified block size $\mathbf{b} = \lceil s(n)/\lg(n) \rceil$. The reader may imagine that the push operation causes a simultaneous push of a block of $\mathbf{b}$ symbols stored on the working tape; say, from the first $\mathbf{b}$ positions of the working tape. The pop operation causes a simultaneous pop of a block of $\mathbf{b}$ symbols from the stack (i.e., replacing them with blank symbols on the stack tape).

(2) Whenever the machine decides to push or pop, it can only change its state. That is, the heads cannot move and the content of the working tape does not change. Moreover, the decision about using a push or pop transition is done solely based on the machine's state, that is, such transitions are available if and only if the machine's state belongs to some subset of states, independently of the symbols under the machine's heads.

(3) If the machine accepts an input α, then there is a run on α where the corresponding push-pop tree (where each operation of pushing/popping a block is considered atomic) is the full binary tree of depth exactly $c\lceil \lg n \rceil$, for some fixed integer c. In particular, at the moment of accepting the stack is empty.

Obviously, the block pushes and pops described in restriction (1) can be simulated by a standard machine in $O(\mathbf{b})$ steps, so their introduction does not give additional expressive power to the computation model.

Intuitively, restriction (1) is easy to achieve, because the machine has enough working space to simulate the top $\lceil s(n)/\lg(n) \rceil$ symbols from the stack on the working tape, and group pushes and pops into blocks of size $\mathbf{b}$. Restriction (2) is a technical adjustment that will be used to streamline future constructions. The most important restriction is (3): the push-pop tree has a fixed shape of a full binary tree. This property will be essential when reducing an arbitrary problem from $\mathbf{NAuxSA}[\underset{time}{\mathrm{poly}}, \underset{space}{s/\log}, \underset{height}{s}]$ to td-3COLORING $[s]$, because the push-pop tree will form a "skeleton" of the graph output from the reduction. To achieve this property, we use the following technical result of Akatov [2], which was also used independently by Elberfeld et al. [29]. The *traversal ordering* of the nodes of an ordered tree is the linear ordering which places a parent before its children and, for two children a, b of a node, a occurring before b, places all descendants of a before all descendants of b. In other words, the traversal ordering is the ordering in which the nodes are visited in a depth-first search started at the root.

LEMMA 4.2 ([2, 29]). *Given an ordered tree T with n nodes and depth at most $\lg n$, one can in logarithmic space compute an embedding (an injection that preserves the ancestor relation and traversal ordering) into a full binary tree of depth $4 \lg n$.*

We are now ready to prove that every problem from $\mathbf{NAuxSA}[\text{poly}, s/\log, \underset{height}{s}]$ can be recognized by a regularized machine, in a similar way as was the case in Reference [3].

LEMMA 4.3. $\mathbf{NAuxSA}[\underset{time}{\text{poly}}, \underset{space}{s/\log}, \underset{height}{s}] \subseteq reg\text{-}\mathbf{NAuxSA}[\underset{time}{\text{poly}}, \underset{space}{s/\log}, \underset{height}{s}].$

PROOF. Consider a machine M placing L in $\mathbf{NAuxSA}[\underset{time}{\text{poly}}, \underset{space}{s/\log}, \underset{height}{s}]$. We modify M to comply with restrictions (1), (2), and (3). The first two are straightforward to achieve using standard techniques.

First, to achieve restriction (1), we designate the first $\mathbf{b} = \lceil s(n)/\lg(n) \rceil$ of the working tape as a buffer for simulating the top of the stack. Whenever during the run M has p symbols on the stack, then after modification the top-most $p \mod \mathbf{b}$ symbols are stored in the buffer, while the remaining symbols are stored in $\lfloor p/\mathbf{b} \rfloor$ blocks on the stack. The operations on the stack are simulated in the buffer. Whenever the size of the buffer reaches full size $\mathbf{b}$, the modified machine invokes the block push operation and clears the buffer. Whenever M wants to pop a symbol but the buffer is empty, the modified machine copies the top $\mathbf{b}$ symbols from the stack to the buffer, invokes the block pop operation, and then simulates the pop of M in the buffer. It is straightforward to simulate the read access to the stack tape with a polynomial-time overhead in the running time.

To achieve restriction (2), whenever the machine would like to push or pop, we split this transition into three. In the first transition, the machine only verifies the symbols under the heads, and enters a state "ready to push/pop" where it additionally remembers the target state and additional operations (moving the heads) to be performed after the stack operation. Then, in the second transition, it performs only the push or pop; note that this transition does not depend on the symbols under the heads, and the heads also do not move. Finally, in the third transition the machine performs the remembered head movements and reaches the target state.

Finally, we concentrate on restriction (3). Let us add a dummy symbol $\blacklozenge$ to the alphabet. Suppose that on some input α of length n the machine M (after the modifications above) has an accepting run. By a simple modification, we may assume that M always empties the stack before accepting. Since the maximum stack height is still $O(s(n))$ and stack operations are done in blocks of $\mathbf{b} = \lceil s(n)/\lg n \rceil$ operations at a time, the corresponding push-pop tree T (where blocks operations are considered as atomic) has depth at most $c_1 \lg n$ and size at most n^{c_2} for some integers c_1, c_2. By Lemma 4.2, there exists an integer c for which T can be always embedded into a tree T_0 that is a full binary tree of depth exactly $c\lceil \lg n \rceil$. We now modify M so that it has also an accepting run whose push-pop tree is T_0.

We modify machine M in the following manner. First, M will keep track of the current position of the computation in the push-pop tree T_0, encoded as binary string of length at most $c\lceil \lg n \rceil$. We add the possibility for M to guess non-deterministically, at any moment of the computation, to push a block of $\mathbf{b}$ dummy symbols without changing the machine's configuration, apart from updating the current position in T_0 (if the computation is in a leaf of T_0, then this transition is not allowed). Similarly, M can, at any moment, pop the top-most block of the stack provided it consists only of dummy symbols, update the position in T_0, without changing the configuration otherwise. The read access to the stack tape is simulated by ignoring the dummy symbols, i.e., the head always continues browsing the tape until a non-dummy symbol is found. Since the dummy symbols are effectively ignored by the computation, it is easy to see that the modified machine accepts if and only if the original one accepts.

Take now an accepting run of M on α and consider an embedding η of its push-pop tree T into T_0. Construct a run of modified M on α by adding non-deterministic pushes and pops of blocks of dummy symbols $\blacklozenge$ for all nodes of T_0 that are not in the image of η. Thus, the push-pop tree of the modified run is exactly T_0.   □

Knowing that computations for NAuxSA can be conveniently regularized, we can describe the existence of such a computation by a CNF formula "wrapped around" the rigid shape of the full binary tree that encodes the push-pop tree of the run. This was also the idea of Akatov and Gottlob [3], but our reduction needs to introduce many more elements, in particular copies of the gadget of Lemma 2.9, due to a far less expressive target language.

LEMMA 4.4. *Let* $L \in$ *reg*-**NAuxSA**[poly, $s/\log$, s]. *Then* $L \leq_L$ *td-CNF-SAT*[s].
            $\underset{time\ \ space\ \ height}{}$

PROOF. Let M be an appropriate machine recognizing L; by the assumption, M satisfies restrictions (1), (2), and (3). Consider an input word α; let $n = |\alpha|$. Let T be the push-pop tree of M on input α. By definition T is the full binary tree of depth exactly $c\lceil \lg n \rceil$ for some fixed integer c that depends on L only. We assume that the machine works in time at most $t(n)$ and uses work tape of size at most $s(n)/\log n$.

Before we proceed to the formal description, let us elaborate on the intuition; see Figure 2 for a visualization. An Euler tour of the nodes and edges of T corresponds to subsequent phases of M's execution on α. We think of the computation as starting at the root node, moving down an edge whenever a push is made and moving up an edge whenever a pop is made. Since T is fixed, the idea is to construct a gadget for each node of T and to wire the gadgets so that they encode the full computation of M. Each node gadget will contain three copies of the computation gadget of Lemma 2.9. These copies respectively encode the parts of the computation before going into the first subtree, between withdrawing from them first subtree and proceeding to the second subtree, and after withdrawing from the second subtree. Each part of the computation depends on symbols pushed onto the stack on the path to the root, but is independent of the guesses in different branches. This, together with the bound of Lemma 2.9 on the tree-depth of the computation gadget, will give rise to a natural tree-depth decomposition of depth $O(s(n))$ of the obtained graph.

We proceed to the formal description; the reader is advised to look at Figure 2 while reading. For each edge e of tree T, we create a block stack(e) of $O(\mathbf{b})$ variables, describing the values pushed on the stack when accessing the lower endpoint of e from the upper during the computation. Moreover, we create four blocks of $O(\frac{s(n)}{\log n})$ variables to describe the configurations immediately before and after the corresponding push and pop operations concerning block stack(e): before-push(e), after-push(e), before-pop(e) and after-pop(e), respectively. Each of these blocks contains the information about (a) the full content of the working tape, (b) the positions of heads on all the three tapes (encoded as indices in binary), and (c) the machine's state. Similarly, we create also two blocks of $O(\frac{s(n)}{\log n})$ variables describing the initial and final configurations: init, final. We enforce them to be the initial and accepting configurations, respectively. For init this only requires introduction of a number of clauses of size 1 that precisely describe the initial configuration. For final, we only need to verify that the final state is accepting. Since the description of the state uses a constant number of variables, this can be easily done by imposing a constant number of CNF clauses on them.

We now wire these groups of variables to simulate the machine's execution.

First, for each edge e of T, we connect before-push(e) with after-push(e) and before-pop(e) with after-pop(e) by clauses that enforce that valid push/pop transitions are being used at these points. By restriction (2), during these transitions only the machine's state can change. Hence, we
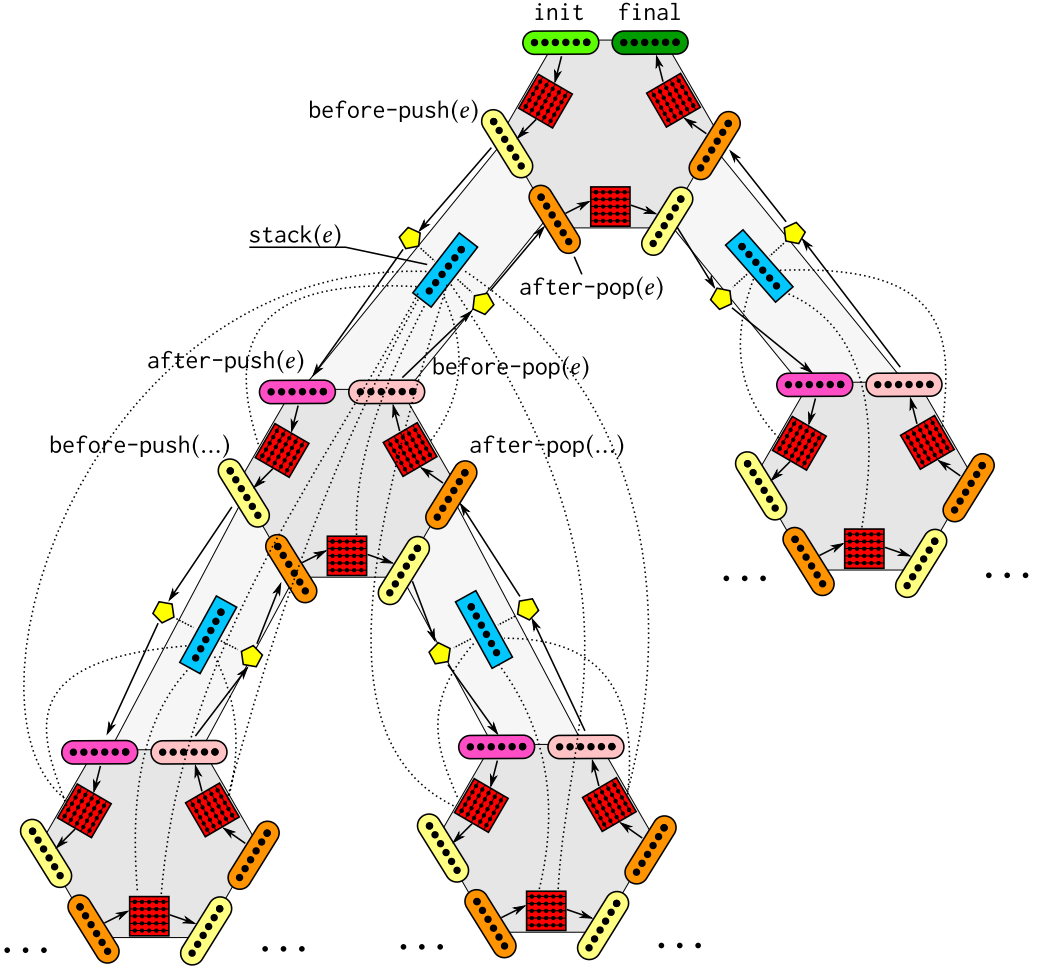
Fig. 2. The construction of Lemma 4.4. Blocks of variables are labeled, e.g., stack(e) blocks depicted as light blue rectangles. Copies of the computation gadget are shown as red squares, while clauses validating push and pop operations are depicted as yellow pentagons.

can enforce that in blocks before-push(e) and after-push(e) all the information apart from the machine's state is exactly the same, and the same holds also for before-pop(e) and after-pop(e); this can be easily done using clauses of size 2. Then, we need to verify that the performed transition was indeed available. Since the availability of a push/pop transition depends solely on the machine's state (restriction (2)), we can verify that the original and target state can be connected by a push/pop transition using a constant-size family of CNF clauses on the variables describing these states. Finally, we need to make sure that the values memorized in block stack(e) are indeed the ones that the machine intended to push onto the stack. Therefore, in the connection between before-push(e) and after-push(e) we also verify, again using clauses of size 2, that values stored in stack(e) are exactly the same as values on the **b** first positions of the working tape. Observe that no such check is needed between before-pop(e) and after-pop(e).

Then, we connect (in a manner described later) pairs of configurations that correspond to the beginning and end of a computation without stack operations. That is, for the root note q with children l, r we connect: `init` with `before-push(ql)`, `after-pop(ql)` with `before-push(qr)` and `after-pop(qr)` with `final`. For every internal, non-root node q with parent p and children l, r, we connect: `after-push(pq)` with `before-push(ql)`, `after-pop(ql)` with `before-push(qr)` and `after-pop(qr)` with `before-pop(pq)`. Finally, for every leaf node q with parent p we connect `after-push(pq)` with `before-pop(pq)`.

Each connection between two blocks is made by creating a new copy of the computation gadget of Lemma 2.9, where:

- t from the statement of Lemma 2.9 is simply $t(n)$.
- s from the statement of Lemma 2.9 is $\frac{s(n)}{\lg n}$.
- h is equal to the stack height at the corresponding moment. That is, this height is exactly $\mathbf{b}$ times the number of edges on the path from the root to the current node q of the push-pop tree T (note that $h \leq \mathbf{b} \cdot c \lceil \lg n \rceil \leq 4c \cdot s(n)$).
- The input word is α.

Variables $u_1, \ldots, u_{s'}$ and $v_1, \ldots, v_{s'}$ of the gadget are identified with the two variable blocks we connect. Variables $w_1, \ldots, w_{h'}$ of the gadget are identified with consecutive variables from `stack(e)` for all edges e on the path from root to the current node q. Note that in this manner, each computation gadget has the input string α encoded within, so that the computation simulated by the gadget has read access to α.

This concludes the construction of the formula. By Lemma 2.9 and restriction (3), it is clear that the formula has a satisfying assignment iff there is an accepting run of M. It remains to construct a tree-depth decomposition of the formula's primal graph G that has depth $O(s(n))$.

Let R be the subset of variables consisting of all the variables contained in the named blocks (i.e., `init`, `stack(e)`, `before-push(e)`, etc.). First, we create a mapping η that maps variables of R to the nodes of T. We begin by mapping the variables of `init` and `final` blocks to the root of T. For every non-root node q of T with parent p, we map the variables of `stack(pq)` to q. Additionally, if q has a parent p or children l, r, we map the variables of `after-push(pq)`, `before-push(ql)`, `after-pop(ql)`, `before-push(qr)`, `after-pop(qr)` and `before-pop(pq)` to q. To this point, η maps $O(\frac{s(n)}{\log n})$ variables to each node of T. Create a tree-depth decomposition of $G[R]$ as follows: start with T, end replace each node q of T with a path consisting of variables $\eta^{-1}(q)$, ordered arbitrarily. These paths are organized in a tree in the same way as the original nodes: the last vertex of a path corresponding to a node q becomes the parent of the first node of each path corresponding to a child of q. Observe that whenever two variables of R appear in the same clause, then the nodes to which their blocks are mapped by η are either equal, or they are in the ancestor-descendant relation. Therefore, it is easy to see that indeed we have constructed a valid tree-depth decomposition of $G[R]$. Since T has depth $O(\log n)$ and the pre-image of each node of T under η has size $O(\frac{s(n)}{\log n})$, the depth of the decomposition is $O(s(n))$.

It remains to consider the variables and clauses created in computation gadgets for each node q. By Lemma 2.9, every such gadget has a tree-depth decomposition of depth $O(\frac{s(n)}{\log n} \cdot \log n + h) = O(s(n))$, where h is the current stack depth. Observe that, among the variables of R, the clauses of each gadget connect only variables from blocks mapped to q by η, and variables from the `stack` blocks corresponding to the edges on the path from q to the root of T. Hence, we can take the tree-depth decomposition of the gadget of depth $O(s(n))$, remove from it all the variables contained in R, and attach the resulting decomposition as a new subtree below the deepest vertex of $\eta^{-1}(q)$. By performing this operation for each computation gadget, we obtain a tree-depth decomposition of

the whole prime graph G that has depth $O(s(n) + s(n)) = O(s(n))$. It is straightforward to verify, using Lemma 2.9, that all the described constructions can be performed in logspace. Therefore, $L \leq_L$ td-CNF-SAT$[c \cdot s(n)]$ for some constant c depending on the machine M; since we assume $s(n)$ to be a nice function, td-CNF-SAT$[c \cdot s] \leq_L$ td-CNF-SAT$[s]$.   □

Lemmas 4.3 and 4.4 show that td-CNF-SAT$[s]$ is hard for $\mathbf{NAuxSA}[\underset{time}{\text{poly}}, \underset{space}{s/\log}, \underset{height}{s}]$ under logspace reductions, and by Lemma 2.7, so is td-3COLORING$[s]$. Lemmas 4.1 and 4.3, and the fact that the closure of $\mathbf{NAuxSA}[\text{poly}, \log, s]$ under logspace reductions is $\mathbf{NAuxSA}[\text{poly}, \log, s(\text{poly})]$, give the following chain of containments (here $[A]^L$ is the class of problems reducible to A in logspace):

$$[\text{td-3COLORING}[s]]^L \quad \subseteq \quad \mathbf{NAuxSA}[\underset{time}{\text{poly}}, \underset{space}{\log}, \underset{height}{s(\text{poly})}] \quad \subseteq$$

$$\subseteq \mathbf{NAuxSA}[\underset{time}{\text{poly}}, \underset{space}{s(\text{poly})/\log}, \underset{height}{s(\text{poly})}] \subseteq \text{reg-}\mathbf{NAuxSA}[\underset{time}{\text{poly}}, \underset{space}{s(\text{poly})/\log}, \underset{height}{s(\text{poly})}] \subseteq$$

$$\subseteq \quad [\text{td-3COLORING}[s(\text{poly})]]^L \quad \subseteq \quad [\text{td-3COLORING}[s]]^L$$

Therefore, all containments must be equalities, which concludes the proof of Theorems 1.4 and 1.5. Note that for an unbounded (or polynomial) stack, Theorem 1.5 implies that space is unbounded too; in other words, $\mathbf{NL}$ (that is, non-deterministic logspace) machines augmented with an unbounded auxiliary stack have the same power as $\mathbf{NP}$, an observation already made by Vinay and Chandru [62].

By Theorem 1.4, to prove a determinization result for $\mathbf{NAuxSA}[\text{poly}, \log, s(\text{poly})]$, we only need such a result for td-3COLORING $[s]$.

LEMMA 4.5. *td-3Coloring$[s]$ can be solved in time $3^s \cdot \text{poly}(n)$ and space $O(s + \log n)$.*

PROOF. Let G be the input instance of td-3COLORING $[s]$, and let $(\mathcal{T}, \mu)$ be the given tree-depth decomposition of G. By abuse of notation, we identify the vertices of G with their images under μ. For every $u \in V(G)$, let tail(u) be the set of vertices on the path from u to the root of its tree in $\mathcal{T}$, excluding u, and let tree$[u]$ be the set of vertices contained in the subtree of $\mathcal{T}$ rooted at u, including u. For every $u \in V(G)$ and every proper coloring ϕ of $G[\text{tail}(u)]$ into 3 colors, let $f(u, \phi)$ be the Boolean value denoting whether ϕ can be extended to a proper 3-coloring of $G[\text{tree}[u] \cup \text{tail}(u)]$. Then, clearly $f(u, \phi)$ is true if and only if it is possible to extend ϕ to ϕ' by assigning u one of the three colors in such a manner that ϕ' remains a proper 3-coloring of $G[\text{tail}(u) \cup \{u\}]$, and $f(v, \phi')$ is true for every child v of u. Whether G has a proper 3-coloring is equivalent to the conjunction of values $f(r, \emptyset)$ over the roots r of trees in $\mathcal{T}$.

We give a recursive procedure for computing values $f(u, \phi)$; the whole problem then reduces to computing $f(r, \emptyset)$ for every root r of a tree in $\mathcal{T}$. This recursive procedure simply browses through all three possible extensions ϕ' of ϕ to a proper 3-coloring of $G[\text{tail}(u) \cup \{u\}]$, and calls itself recursively to compute $f(v, \phi')$ for all children v of u; in particular, no memoization of computed values is done. Note that the recursion tree stops in the leaves of $\mathcal{T}$. The correctness of the algorithm follows directly from the discussion of the previous paragraph. As far as the space usage is concerned, at each point the algorithm maintains identifier of the current vertex u, of logarithmic length, and a stack of $O(s)$ calls to the procedure computing $f(\cdot, \cdot)$. The data stored for each call requires constant space; note that there is no need to memorize the identifier of the vertex, because it can be recomputed when returning from a subcall. Hence, the space complexity of the algorithm is $O(s + \log n)$. To analyze the running time, observe that for each pair (u, ϕ), where ϕ is a proper 3-coloring of $G[\text{tail}(u)]$, throughout the whole algorithm there will be at most one call to $f(u, \phi)$; this is because whenever recursing, we are considering an extension of the current coloring. Thus,

the whole recursion tree will have at most $n \cdot 3^s$ nodes. Since the computations at each node are done in polynomial time, it follows that the running time of the algorithm is $3^s \cdot \mathrm{poly}(n)$.   □

Therefore, for any nice $s(n) \geq \log n$, we have td-3COLORING$[s] \in \mathbf{D}[\underset{space}{s}]$. Theorem 1.4, Lemma 4.5, and the observation that $\mathbf{D}[\underset{space}{s(\mathrm{poly})}]$ is closed under logspace reductions yield Theorem 1.6.

4.2 Characterization via Alternating Machines

The completeness result of the previous section allows to easily prove another characterization of the same complexity class, this time in terms of alternating Turing machines (ATMs). See Reference [7] for classical results on how ATMs relate to many known complexity classes. An *alternating Turing machine* is a Turing machine with a partition of states into *existential* or *universal* states. For an ATM and an input word, an *accepting tree* is a finite tree T labeled with machine configurations, such that the root is labeled with the initial configuration, every node with a configuration in an existential state in T has one child labeled with a next configuration (one reachable in one step according to the machine's transition rules), every node with a configuration in a universal state in T has all possible next configurations as children, and all leaves are accepting configurations. An ATM accepts an input word in time t, space s, and *treesize z* if there is an accepting tree with root-to-leaf distances at most t, configurations using at most s space, and at most z tree nodes.

Similarly, as ATMs proved to be a useful computational model, giving a new, unified view on various complexity issues, the notion of treesize introduced by Ruzzo [60] allowed to see various classes under a common light and simplify a few containment proofs. In particular, Ruzzo showed that $\mathbf{NAuxPDA}[\underset{time}{\mathrm{poly}}, \underset{space}{s}] = \mathbf{A}[\underset{space}{s}, \underset{treesize}{\mathrm{poly}}]$. Here we show that bounding the time (as opposed to space) of a polynomial treesize ATM leads to the classes corresponding to small tree-depth (as opposed to small treewidth). That is, we show Theorem 1.7, restated below.

THEOREM 4.6. *Let $s(n) \geq \log^2(n)$ be a nice function. Then,*

$$\mathbf{NAuxSA}[\underset{time}{\mathrm{poly}}, \underset{space}{\log}, \underset{height}{s(\mathrm{poly})}] = \mathbf{A}[\underset{time}{s(\mathrm{poly})}, \underset{treesize}{\mathrm{poly}}].$$

PROOF. For one containment ($\subseteq$), we show that td-3COLORING$[s]$, which is hard for the former class by Theorem 1.4, is contained in the latter. Indeed, a straightforward algorithm traverses a given tree-depth decomposition of the input graph top-down by existentially guessing a color of each encountered vertex, checking its compatibility with previous guesses, and universally guessing which subtree to proceed into. The bounds on time and treesize of an alternating Turing machine executing this algorithm follow directly from the bounds on the depth (s) and size (polynomial) of the decomposition's tree.

For the other containment, we simulate an alternating Turing machine with a NAuxSA machine. This is done exactly as in Ruzzo's simulation with NAuxPDA machines (Theorem 1 in Reference [60]), except that all the configurations on the current path are remembered on the stack by only writing the difference (*log of changes*) from the previous configuration: the constant-size description consists of the new state, the direction of movement for each head and the symbol written on the worktape. Observe that in this manner, the machine can within logarithmic working space retrieve all the information needed to verify availability of a transition:

- The current simulated state is on the top of the stack;
- The current head positions can be recalculating by browsing through the stack and applying consecutive moves;

- The symbol under the worktape head can be recalculated by finding on the stack the latest symbol written on its current position.

Consequently, deterministic steps of the machine can be simulated by verifying the applicability of the transition, and pushing onto the stack the constant-size log of changes leading to the new configuration. Similarly, existential guesses are simulated with the machine's own non-determinism. Universal guesses are simulated by choosing one possibility and then backtracking (popping the stack) to choose the next possibility, as described by Ruzzo. Such backtracking corresponds exactly to traversing an accepting tree of the ATM; hence, the running time is polynomial. At most $s(\text{poly}(n))$ steps are described on the stack at any time, each requiring a constant number of symbols due to keeping only the log of changes.   □

5 DOMINATING SET ON GRAPHS OF SMALL TREEDEPTH

In this section, we show how to solve DOMINATING SET, or even count the number of dominating sets of any cardinality, in time $3^s \cdot \text{poly}(n)$ and space $O(s \cdot \log n)$, given a tree-depth decomposition of depth s. Recall that for a graph G, a set $S \subseteq V(G)$ is a *dominating set* if every vertex of G either is in S, or is adjacent to some vertex of S. We first describe an algorithm working in space $\text{poly}(n)$.

LEMMA 5.1. *There exists an algorithm that, given a graph G on n vertices and its tree-depth decomposition of depth s, runs in time $3^s \cdot \text{poly}(n)$ and space $\text{poly}(n)$, and outputs a sequence $(q_i)_{0 \leq i \leq n}$, where q_i is the number of dominating sets of G of cardinality i.*

PROOF. We will work in the ring of polynomials $\mathbb{Z}[x]$, where x is a formal variable. The algorithm will compute polynomial $P(x) = \sum_{i=0}^{n} q_i x^i$, whose coefficients constitute the output.

Let $(\mathcal{T}, \mu)$ be the given tree-depth decomposition of G; recall that $\mathcal{T}$ is a rooted forest of depth at most s. By abuse of notation, we identify the vertices of G with their images under μ. Let us introduce some notation relating to $\mathcal{T}$. For $u \in V(G)$, by $\texttt{tail}[u]$, we denote the set of vertices on the path in $\mathcal{T}$ from u to the root of its tree in $\mathcal{T}$. By $\texttt{tree}[u]$, we denote the set of all the vertices contained in the subtree of $\mathcal{T}$ rooted at u, including u. Define $\texttt{tail}(u) = \texttt{tail}[u] \setminus \{u\}$ and $\texttt{tree}(u) = \texttt{tree}[u] \setminus \{u\}$. By $\texttt{chld}(u)$, we denote the set of children of u in $\mathcal{T}$. For a function h, an argument e outside the domain of h, and a value α, by $h[e \to \alpha]$, we denote the function h extended by adding e to the domain and mapping it to α.

Let $\Sigma = \{\mathbf{A}, \mathbf{F}, \mathbf{T}\}$, where $\mathbf{A}, \mathbf{F}, \mathbf{T}$ are some symbols (the reader should think of them as Allowed, Forbidden, and Taken, respectively). For any vertex u of G and function $\phi : \texttt{tail}(u) \to \Sigma$, define $f(u, \phi) \in \mathbb{Z}[x]$ as $\sum_{i=0}^{n} a_i x^i$, where a_i is the number of i-element subsets $X \subseteq \texttt{tree}[u]$ such that $X \cup \phi^{-1}(\mathbf{T})$ dominates $\texttt{tree}[u]$ and no vertex of $\phi^{-1}(\mathbf{F})$ in G. Similarly, for a function $\psi : \texttt{tail}[u] \to \Sigma$, define $g(u, \psi) \in \mathbb{Z}[x]$ as $\sum_{i=0}^{n} b_i x^i$, where b_i is the number of i-element subsets $X \subseteq \texttt{tree}(u)$ such that $X \cup \psi^{-1}(\mathbf{T})$ dominates $\texttt{tree}(u)$ and no vertex of $\phi^{-1}(\mathbf{F})$ in G. Note that if there are two adjacent vertices $v, v' \in \texttt{tail}(u)$ with $\phi(v) = \mathbf{T}$ and $\phi(v') = \mathbf{F}$, then no set X can satisfy the requirements above and hence $f(u, \phi) = 0$. Similarly for g and ψ.

We remark that values f and g are exactly what one would obtain by applying the Möbius transform to the standard definition of states for dynamic programming for DOMINATING SET (that is, we count sets that dominate any subset of $\phi^{-1}(\{\mathbf{A}, \mathbf{T}\})$ in the tail, instead of exactly specifying which vertices are to be dominated). This transform translates subset convolutions used in the standard dynamic programming to pointwise products, which is the crucial idea behind the proof. Since the algorithm is not complicated, we prefer to present it directly after applying the transform.

We now give recursive equations on the values of $f(\cdot, \cdot)$ and $g(\cdot, \cdot)$. First, observe that for each $v \in \texttt{chld}(u)$ we have $\texttt{tail}(v) = \texttt{tail}[u]$. Then, it is easy to verify that the following equation

holds for each $u \in V(G)$ with $\text{chld}(u) \neq \emptyset$ and each $\psi : \text{tail}[u] \to \Sigma$:

$$g(u, \psi) = \prod_{v \in \text{chld}(u)} f(v, \psi). \tag{1}$$

Indeed, every set $X \subseteq \text{tree}(u)$ that contributes to some coefficient of $g(u, \psi)$ can be partitioned into $\{X \cap \text{tree}[v] : v \in \text{chld}(u)\}$. Each set $X \cap \text{tree}[v]$ contributes to the coefficient by $x^{|X \cap \text{tree}[v]|}$ of $f(v, \psi)$, and hence when computing the product, the formal variable x correctly keeps track of the cardinality. When $\text{chld}(u) = \emptyset$, then $\text{tree}(u) = \emptyset$, and we can compute $g(u, \psi)$ directly from the definition:

$$g(u, \psi) = \begin{cases} 1 & \text{if there is no edge between } \psi^{-1}(\mathbf{T}) \text{ and } \psi^{-1}(\mathbf{F}), \\ 0 & \text{otherwise.} \end{cases} \tag{2}$$

We now proceed to setting up the equation for $f(\cdot, \cdot)$. Take any $u \in V(G)$ and $\phi : \text{tail}(u) \to \Sigma$. Then, it is easy to verify that the following equation holds:

$$f(u, \phi) = g(u, \phi[u \to \mathbf{A}]) - g(u, \phi[u \to \mathbf{F}]) + x \cdot g(u, \phi[u \to \mathbf{T}]). \tag{3}$$

The term $x \cdot g(u, \phi[u \to \mathbf{T}])$ counts the contribution from sets X that contain u. The term $g(u, \phi[u \to \mathbf{A}])$ counts the contribution from all sets X that do not contain u, regardless of whether they dominate u or not, whereas by subtracting the term $g(u, \phi[u \to \mathbf{F}])$, we remove the contribution from sets that do not contain or dominate u. Observe that if u has a neighbor in $\phi^{-1}(\mathbf{T})$, i.e., it is already dominated by $\phi^{-1}(\mathbf{T})$, then the subtracted term $g(u, \phi[u \to \mathbf{F}])$ will be a zero polynomial. This corresponds to the fact that in this case, we do not need to care about domination of u by X. Similarly, if u has a neighbor in $\phi^{-1}(\mathbf{F})$, then $g(u, \phi[u \to \mathbf{T}])$ will be a zero polynomial. This corresponds to the fact that in this case, it is not allowed to take u to X.

Finally, observe that (letting roots be the set of roots of the trees in forest $\mathcal{T}$)

$$P = \prod_{u \in \text{roots}} f(u, \emptyset). \tag{4}$$

We now give the algorithm that computes P. The algorithm uses two mutually recursive functions that compute the values of $f(\cdot, \cdot)$ and $g(\cdot, \cdot)$, respectively. The polynomial P is computed using Equation (4) by a sequence of calls to the procedure computing $f(\cdot, \cdot)$. The procedure computing $f(\cdot, \cdot)$ applies Equation (3) and calls $g(\cdot, \cdot)$ recursively. Similarly, the procedure computing $g(\cdot, \cdot)$ applies Equation (1) and calls $f(\cdot, \cdot)$ recursively, or uses the base case (2). In particular, no memoization of computed values is performed.

The correctness of the algorithm follows from Equations (1)–(4). Note that at each moment, the space used by the algorithm is composed of a stack of at most $2s + 1$ recursive calls to $f(\cdot, \cdot)$, $g(\cdot, \cdot)$, and the main procedure computing P, and for each of these calls we can store just the partial result of computation being one polynomial from $\mathbb{Z}[x]$ (in case of Equations (1) and (4), this will be the product calculated for a prefix). These polynomials have degrees bounded by n and their coefficients have values between 0 and 2^n; hence, the total space usage of the algorithm is $O(sn^2)$. Finally, to estimate the running time observe that for every pair (u, ϕ), where $u \in V(G)$ and $\phi : \text{tail}(u) \to \Sigma$, throughout the whole computation there will be at most one call to computing $f(u, \phi)$. This is because when recursing, the new function ϕ is always an extension of the previous one. Similarly, for every pair (u, ψ), where $u \in V(G)$ and $\psi : \text{tail}[u] \to \Sigma$, throughout the whole computation there will be at most one call to computing $g(u, \psi)$. The total number of such pairs (u, ϕ) and (u, ψ) is at most $2n \cdot 3^s$, which implies that the whole recursion tree has at most this many nodes. Since the work done at each node is polynomial in n, we conclude that the algorithm runs in time $3^s \cdot \text{poly}(n)$. □

We now show how to improve the space usage of the above algorithm to $O(s \cdot \log n)$. Given a graph G on n vertices and its tree-depth decomposition of depth s, consider again the polynomial $P(x) = \sum_{i=0}^{n} q_i x^i$ over a single variable x, with integer coefficients q_i equal to the number of dominating sets of G of cardinality exactly i (in particular, $0 \le q_i \le 2^n$). We use the fact that given a prime number $p \le 2n + 2$ and some element a of the Galois field $\mathbb{F}_p$, the value of $(P(a) \mod p)$ can be computed in time $O(3^s \cdot \mathrm{poly}(n))$ and space $O(s \cdot \log n)$. Indeed, one can recursively compute all $O(3^s \cdot n)$ values of $(f(\cdot, \cdot)(a) \mod p)$ and $(g(\cdot, \cdot)(a) \mod p)$, as described in the proof of Lemma 5.1. On each of the $O(s)$ recursion levels, we need to maintain only one number in $\mathbb{F}_p$ (describing a partial sum or product), and each value can be added or multiplied (to the partial sum or product that requires it) in time $O(\mathrm{poly}(n))$ and space $O(\log n)$.

With such a procedure in hand, the following theorem describes how to recover the exact coefficients of $P(x)$ using interpolation. This is done using the Chinese remainder theorem, and applying a number-theoretic transform, that is, the discrete Fourier transform specialized to the field of integers mod p, for prime p. Effectively, this technique boils down to evaluating the polynomial in many points and computing a weighted sum of the results. We need the following simple corollary of the prime number theorem. In fact, it can be proved for $n_0 = 21$ using explicit bounds given in Reference [58, Theorem 4] and hand computation for small n.

FACT 5.2. *There is an $n_0 \in \mathbb{N}$ such that for all $n \ge n_0$, the product of primes strictly between n and $2n$ is larger than 2^n.*

THEOREM 5.3. *Let $P(x) = \sum_{i=0}^{n} q_i x^i$ be a polynomial over one variable x, of degree at most n and with integer coefficients satisfying $0 \le q_i \le 2^n$, for $i = 0, \ldots, n$. Suppose that given a prime number $p \le 2n + 2$ and $a \in \mathbb{F}_p$, the value of $(P(a) \mod p)$ can be computed in T time and S space. Then given $k \in \{0, \ldots, n\}$, the value q_k can be computed in $O(T \cdot \mathrm{poly}(n))$ time and $O(S + \log n)$ space.*

PROOF. We first show how to compute $q_k \mod p$, given k and a prime p with $n + 1 < p < 2n + 2$. Let α be a primitive element of the field $\mathbb{F}_p$, that is, a generator of the multiplicative group $\mathbb{F}_p^*$ (in other words, $\mathbb{F}_p^* = \{\alpha^0, \alpha^1, \ldots, \alpha^{p-2}\}$). Such an element can be found in polynomial time by trying all elements of $\mathbb{F}_p^*$ and testing whether $\alpha^1, \ldots, \alpha^{p-2} \ne 1$. Compute in $\mathbb{F}_p$ the value

$$q_k' = -\sum_{i=0}^{p-2} P(\alpha^i) \cdot \alpha^{-ik}.$$

This takes $O(T \cdot n)$ time and $O(S + \log n)$ space.

We claim $q_k' = q_k \mod p$. To show this, first notice that for $a \in \mathbb{F}_p \setminus \{0\}$, we have

$$(a - 1) \cdot \sum_{i=0}^{p-2} a^i \;=\; a^{p-1} - 1 \;=\; 0 \quad (\mod p).$$

Hence, if $a \ne 1$ then $\sum_{i=0}^{p-2} a^i = 0 \pmod p$, while for $a = 1$, we have $\sum_{i=0}^{p-2} a^i = p - 1 = -1 \pmod p$. Thus, for any $k \in \mathbb{Z}$ we have the following:

$$\sum_{i=0}^{p-2} (\alpha^k)^i \;=\; [k \equiv 0 \mod (p-1)] \cdot (-1) \quad (\mod p).$$

Here, $[\cdot]$ denotes Iverson's notation: the value is 0 or 1 depending whether the predicate in the brackets is false or true, respectively. Since $n < p - 1$, for integers $j, k \in \{0, \ldots, n\}$ we have $j - k \equiv 0$

mod $(p-1)$ if and only if $j = k$. Therefore, as claimed, we conclude that

$$
q_k' = -\sum_{i=0}^{p-2}\left(\sum_{j=0}^{n} q_j \cdot (\alpha^i)^j\right) \cdot \alpha^{-ik} = -\sum_{j=0}^{n} q_j \sum_{i=0}^{p-2}(\alpha^{j-k})^i
$$

$$
= -\sum_{j=0}^{n} q_j \cdot [j = k] \cdot (-1) = q_k \pmod{p}.
$$

Therefore, for every integer p strictly between $n + 1$ and $2(n + 1)$, we can check whether it is prime (by brute-force) and compute q_k mod p in $O(T \cdot n)$ time and $O(S + \log n)$ space. From Fact 5.2 and the Chinese remainder theorem, it follows that q_k is non-zero if and only if (q_k mod p) turns out to be non-zero for at least one p (we assume $n + 1 \geq n_0$, as otherwise q_k can be computed by brute-force). Moreover, given a list of pairs $(p, q_k \bmod p)$ for the primes p between $n + 1$ and $2(n + 1)$, the exact value of q_k can be recovered with a logspace (and hence polynomial time) algorithm by Chiu et al. [21], which is an effective version of the Chinese remainder theorem. Note that we do not need to simultaneously store all values q_k mod p for different p: we use the compositionality of logspace algorithms instead, that is, values output by our algorithm are recomputed on the fly as needed by the remaindering algorithm, multiplying the running times and adding the space bounds of the two algorithms.   □

COROLLARY 5.4. *There exists an algorithm that, given a graph G on n vertices and its tree-depth decomposition of depth s, runs in time $3^s \cdot \mathrm{poly}(n)$ and space $O(s \cdot \log n)$, and outputs a sequence $(q_i)_{0 \leq i \leq n}$, where q_i is the number of dominating sets of G of cardinality i.*

6 CONCLUSIONS

In what follows, we assume for conciseness that $s(n)$ is a nice function satisfying $s(n^c) = O(s(n))$ for each constant c (this includes $s(n) = \lg^k n$ for $k \geq 1$, in particular).

The hierarchy of graph parameters of Corollary 2.5 together with Theorems 1.1, 1.4, and 1.7 implies the following hierarchy of complexity classes between **NL** and **NP**.

$$
\mathbf{NAuxSA}[\underset{time}{\mathrm{poly}}, \underset{space}{\log}, \underset{height}{s}] = [\text{td-3COLORING}[s]]^L = \mathbf{A}[\underset{time}{s}, \underset{treesize}{\mathrm{poly}}] \subseteq \mathbf{D}[\underset{space}{s}]
$$

$$
\cap
$$

$$
\mathbf{N}[\underset{time}{\mathrm{poly}}, \underset{space}{s}] = [\text{pw-3COLORING}[s]]^L = \mathbf{N}[\underset{time}{\mathrm{poly}}, \underset{space}{s}]
$$

$$
\cap
$$

$$
\mathbf{NAuxPDA}[\underset{time}{\mathrm{poly}}, \underset{space}{s}] = [\text{tw-3COLORING}[s]]^L = \mathbf{A}[\underset{space}{s}, \underset{treesize}{\mathrm{poly}}] \subseteq \mathbf{D}[2^{O(s)}]_{time}
$$

$$
\cap
$$

$$
\mathbf{NAuxSA}[\underset{time}{\mathrm{poly}}, \underset{space}{\log}, \underset{height}{s \cdot \log}] = [\text{td-3COLORING}[s \cdot \log]]^L = \mathbf{A}[s \cdot \log, \underset{treesize}{\mathrm{poly}}] \subseteq \mathbf{D}[s \cdot \log]_{space}
$$

In particular, when considering functions $s(n) = \log^k(n)$, these classes have been considered under different names in literature: $\mathbf{NAuxSA}[\underset{time}{\mathrm{poly}}, \underset{space}{\log}, \underset{height}{\log^k}]$ was named $\mathbf{DC}^{k-1}$ (for *divide and conquer*) in References [2] and [3], $\mathbf{N}[\underset{time}{\mathrm{poly}}, \underset{space}{\log^k}]$ are known as $\mathbf{NSC}^k$ (the non-deterministic

variant of *Steve's Class*), while **NAuxPDA**[poly, $\log^k$] is shown equal to a class named **SAC**$^k_{quasi}$
$\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad$ *time* *space*
in Reference [5]. This yields the following hierarchy:

$$\begin{array}{ccc} \mathbf{NL} & \mathbf{SAC}^1 & \\ \mathbf{L} \subseteq \;\;\parallel\;\; \subseteq \;\;\parallel\;\; & \subseteq \mathbf{DC}^1 \subseteq \cdots \subseteq \mathbf{DC}^{k-1} \subseteq \mathbf{NSC}^k \subseteq \mathbf{SAC}^k_{quasi} \subseteq \mathbf{DC}^k \subseteq \cdots \subseteq \mathbf{NP} \\ \mathbf{NSC}^1 & \mathbf{SAC}^1_{quasi} & \end{array}$$

We conclude with an open question stemming from this work. In Section 4, we have shown
that 3Coloring is complete for **NAuxSA**[poly, log , s] when a tree-depth decomposition of
$\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad$ *time* *space* *height*
depth $s(n)$ is given on the input. By Lemma 2.7, the same holds for equivalent CSP-like problems,
like CNF-SAT (with primal or incidence graph), whereas Independent Set and Dominating Set
are hard for the same class.

It is not hard to see that Independent Set actually can be solved in the complexity class
NAuxSA[poly, log , s], using an approach very similar to that of Lemma 4.5 as follows. The
$\quad\quad\quad\quad\quad$ *time* *space* *height*
algorithm traverses the treedepth decomposition in the prefix order, nondeterministically guess-
ing a maximum-size independent set X on the fly, and storing the following information: on the
stack we store the intersection of X with the path from the current vertex to a root of the decom-
position, whereas in the working memory, we store the number of vertices from X found so far. By
Theorem 1.6, this means that Independent Set on a graph given with tree-depth decomposition
of depth s can be solved deterministically in space $O(s + \log n)$.

As for Dominating Set, in Section 5, we demonstrated how using the algebraic approach of
Lokshtanov and Nederlof [46], and of Fürer and Yu [34], one can obtain an algorithm using $3^s \cdot$
poly(n) time and $O(s \cdot \log n)$ space. It is unclear to us whether the problem Dominating Set on
graphs with treedepth $s(n)$ belongs to **NAuxSA**[poly, log , s]. Observe that by Theorem 1.6,
$\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad$ *time* *space* *height*
it would then be solvable in space $O(s + \log n)$, whereas achieving $O(s \cdot \log n)$ was already highly
nontrivial.

APPENDIX

A   REDUCTIONS PRESERVING STRUCTURAL PARAMETERS

To capture the structural dependencies in reductions for a more uniform proof, we use the follow-
ing definition borrowed from Chen and Müller [20]. While very similar to a tree decomposition, it
is not limited to trees and allows an edge to be covered by two adjacent bags instead of one, which
turns out to give a generalization with better properties.

Definition A.1. For graphs G, H, an *H-deconstruction of G* is a family $(B_h)_{h \in V(H)}$ of subsets of
$V(G)$ (called *bags*) such that every vertex of G is in some bag, every edge of G has both endpoints
contained in one, or two adjacent (in H) bags, and for each vertex $v \in V(G)$ the subset $\{h \in V(H) :
v \in B_h\}$ is connected in H. The *width* of a deconstruction is the maximum size of a bag or a union
of two adjacent bags.

We observe that in many reductions, the output graph can be deconstructed into the input
graph (or, e.g., the incidence graph of the input formula) with constant width. We first show that
this guarantees the reduction preserves structural parameters.

Lemma A.2. *Let* $\pi \in \{\mathsf{td}, \mathsf{pw}, \mathsf{tw}\}$. *Given graphs* G, H, *an H-deconstruction of G of width w, and
a π-decomposition of H of width/depth* w_h, *one can compute in logspace a π-decomposition of G of
width/depth at most* $w \cdot (w_h + 1)$. *In particular,* $\pi(G) \leq w \cdot \pi(H)$.

PROOF. Let $(B_h)_{h \in V(H)}$ be an H-deconstruction of G of width w.

For treewidth and pathwidth, observe that if $(\mathcal{T}, (C_t)_{t \in \mathcal{T}})$ is a tree (or path) decomposition of H of width w_h, then the same tree $\mathcal{T}$ with bags defined as $C'_t = \bigcup_{h \in C_t} B_h$ is a valid decomposition of G of width at most $w \cdot (w_h + 1)$.

For tree-depth, let $(\mathcal{T}, \mu)$ define a tree-depth decomposition of H of depth w_h. Create sets M_t for nodes t of $\mathcal{T}$ and place each vertex v of G in M_t, where t is the lowest common ancestor of $\{\mu(h) : v \in B_h\}$ in $\mathcal{T}$.

Observe that by definition of a deconstruction, if $v \in B_h$ and $v \in B_{h'}$ for some $h, h' \in V(H)$, then there is a path connecting h and h' in H containing only vertices h'' such that $v \in B_{h''}$; hence, there is an $h'' \in V(H)$ such that $\mu(h'')$ is a common ancestor of $\mu(h)$ and $\mu(h')$ in $\mathcal{T}$ and $v \in B_{h''}$. Therefore, if h_0 is the lowest common ancestor of $\{\mu(h) : v \in B_h\}$ in $\mathcal{T}$, then also $v \in B_{h_0}$, and hence every vertex v is put into a set M_t such that $v \in B_{\mu^{-1}(t)}$. That is, $M_t \subseteq B_{\mu^{-1}(t)}$, which implies $|M_t| \leq w$ for all $t \in V(\mathcal{T})$.

Let us then modify $\mathcal{T}$ by replacing every node $t \in V(\mathcal{T})$ by a path of $|M_t|$ nodes, and define a bijection μ' between vertices of G and nodes of this tree that maps vertices in M_t to nodes of the path that replaced t (in any order). Let $\mathcal{T}'$ be the modified decomposition. Clearly $\mathcal{T}'$ has depth at most $w \cdot w_h$. To check that $\mathcal{T}'$ with μ' defines a valid tree-depth decomposition, consider any edge uv of G. By definition of a deconstruction there are adjacent or equal vertices h, h' in H such that $u \in B_h$ and $v \in B_{h'}$. Since $\mu(h)$ is an ancestor of $\mu(h')$ or vice versa, u was assigned to a set M_t such that t is an ancestor of $\mu(h)$ and v was assigned to a set $M_{t'}$ such that t' is an ancestor of $\mu(h')$, it must be that t is an ancestor of t' or vice versa. Hence, $\mu'(u) \in M_t$ is an ancestor of $\mu'(v) \in M_{t'}$ or vice versa. □

Many reductions between NP-complete graph problems introduce components of bounded size replacing every edge of the original graph, or more generally, attach small components to cliques of the original graph. We need the following lemma to show that such reductions also preserve structural parameters. For a graph G and a vertex set $S \subseteq V(G)$, the *subgraph of G induced by S*, denoted $G[S]$, is the graph with vertex set S and edge set $E(G) \cap (S \times S)$. We write $G - S$ for $G[V(G) \setminus S]$.

LEMMA A.3. *Let $\pi \in \{\mathtt{td}, \mathtt{pw}, \mathtt{tw}\}$. Let G be an induced subgraph of G' such that for each connected component C of $G' - V(G)$, we have that C has at most c vertices and the neighborhood of C in $V(G)$ is a clique in G, for some constant c. Then,*

$$\pi(G') \leq \pi(G) + c.$$

Furthermore, given G, G' and any π-decomposition of G, one can compute a π-decomposition of G' of width/depth larger by at most c in logspace.

PROOF. For $\mathtt{td}$, in any tree-depth decomposition, the vertices of a clique in G must be mapped to nodes fully ordered by the ancestor relation. We may thus simply take each connected component C of $G' - V(G)$, examine the placement of the clique $N(C) \subseteq V(G)$ in the given tree-depth decomposition of G, and attach the vertices of C as a path of length $|C|$ below the lowest node the clique $N(C)$ maps to. In this manner, we create a tree-depth decomposition of G' of depth larger than the input tree-depth decomposition of G by at most c.

For $\mathtt{tw}$ and $\mathtt{pw}$, we use the fact that each clique in G must be fully contained in some bag of a decomposition (e.g., Reference [14, Lemma 1]). Hence, for each connected component C of $G' - V(G)$, we find a bag that contains all the vertices of the clique $N(C) \subseteq V(G)$, and create a copy of this bag into which all the vertices of C are added. It is straightforward to arrange the new bags in the decomposition. In this manner we construct a decomposition of G' of width larger than the

original decomposition of G by at most c. For all three parameters, it is trivial to implement the described procedure in logspace. □

We are now ready to show how standard reductions for some example NP-complete problems prove them to be equivalent to 3Coloring, or at least as hard, in our setting.

LEMMA A.4. *The following problems are equivalent under logspace reductions that preserve structural parameters: 3Coloring, CNF-SAT (using a decomposition of the primal graph), k-SAT (using a decomposition of either the primal or incidence graph) for each $k \geq 3$.*

Furthermore, the following problems admit logspace reductions that preserve structural parameters from the above problems: Vertex Cover, Independent Set, Dominating Set.

PROOF. All the following reductions are standard, and hence we keep the description concise. Also, it will be straightforward to verify that they can be implemented in logspace. The only non-trivial check will be to verify, using Lemmas A.3 and A.2, that the structural parameters are preserved.

CNF-SAT (primal graph) $\leq_L$ k-SAT (primal graph) (for any $k \geq 3$):

Replace every clause $(l_1 \vee l_2 \vee \ldots \vee l_\ell)$ of length ℓ with clauses $(l_1 \vee l_2 \vee x_2)$, $(\neg x_2 \vee l_3 \vee x_3)$, $(\neg x_3 \vee l_4 \vee x_4) \ldots (\neg x_{\ell-2} \vee l_{\ell-1} \vee l_\ell)$ using new variables $x_2, \ldots, x_{\ell-2}$. Let G be the primal graph of the original formula, G'' be the primal graph of the new formula, and let $G' = G \cup G''$, i.e., a graph on the vertex set $V(G'')$ where the edge set is the union of the edge sets of G and G''. By the construction, it follows that $G'[V(G)] = G$ and that each connected component of $G' - V(G)$ has size at most $c - 3$, where c is the maximum clause size in the original formula. Moreover, the neighborhood of such a connected component is a clique in G. Since each clause induces a clique in the primal graph, it follows that $c \leq \pi(G) + 1$ for each $\pi \in \{\text{td}, \text{pw}, \text{tw}\}$. Hence, from Lemma A.3, we have that $\pi(G') \leq \max(\pi(G), 2\pi(G) - 2)$, and an appropriate decomposition can be constructed from a decomposition of G in logspace. Since G'' is a subgraph of G, it is also a decomposition of G''.

k-SAT (primal graph) (for any fixed $k \in \mathbb{N}$) $\leq_L$ k-SAT (incidence graph):

Use the same formula, bounds follow immediately from Lemma A.3 (the connected components are single vertices).

k-SAT (incidence graph) (for any fixed $k \in \mathbb{N}$) $\leq_L$ k-SAT (primal graph):

The primal graph has a natural width-k deconstruction into the incidence graph of the formula. For every variable of the formula, we create a bag containing only it. For every clause of the formula, we create a bag containing all the variables contained in this clause. It is easy to verify that this is a deconstruction.

k-SAT (primal graph) (for any fixed $k \in \mathbb{N}$) $\leq_L$ CNF-SAT (primal graph): trivial.

3-SAT (incidence graph) $\leq_L$ 3Coloring:

The reduction of Garey et al. [36] creates a pair of adjacent vertices for every variable of the formula (a variable gadget), and a 6-vertex subgraph for every clause (clause gadget). For each clause, three edges are added to connect it to gadgets for variables occurring in this clause. Then, a single triangle is created, whose one vertex is connected to all the vertices of all the variable gadgets. The graph created can easily be seen to have a width-11 deconstruction into the formula's incidence graph. Namely, a variable's (clause's) bag contains the 2 (6) corresponding vertices, and all bags contain the last triangle.

3Coloring $\leq_L$ 3-SAT (primal graph):

Create three variables x, y, z and four clauses $(x \vee y \vee z)$, $(\neg x \vee \neg y)$, $(\neg y \vee \neg z)$, $(\neg z \vee \neg x)$ for each vertex of the input graph G, describing that exactly one of the variables corresponding to this vertex is true. Then, for each edge of G add three clauses of size 2, describing that the true variable corresponding to one endpoint has a different label than for the other endpoint. It is easy to see

that the formula's primal graph has a width-6 deconstruction into the original graph. Namely, for each original vertex of G, create a bag that contains the corresponding 3 variables x, y, z.

3-SAT (incidence graph) $\leq_L$ INDEPENDENT SET:

Create two adjacent vertices $x, \neg x$ for every variable x (variable gadget), and a triangle for every clause (clause gadget). In every clause gadget label, the vertices of the triangle by the literals occurring in the clause, and connect these vertices to corresponding literals in clause gadgets. Then, the input formula is satisfiable if and only if there is an independent set in the output graph with as many vertices as there are variables and clauses in total. The output graph has a trivial width-5 deconstruction into the incidence graph of the input formula, where for every variable/clause, we create a bag containing the corresponding gadget.

INDEPENDENT SET $\leq_L$ VERTEX COVER:

Given a graph G and a number k, output G and $|V(G)| - k$.

VERTEX COVER $\leq_L$ DOMINATING SET:

Given a graph G and a number k, let G' be obtained from G by subdividing every edge once; output G' and k. Bounds follow from Lemma A.3 (the connected components are single vertices). □

ACKNOWLEDGMENTS

REFERENCES

[1] Amir Abboud, Arturs Backurs, and Virginia Vassilevska Williams. 2015. Tight hardness results for LCS and other sequence similarity measures. In *Proceedings of the IEEE 56th Annual Symposium on Foundations of Computer Science (FOCS'15)*. IEEE Computer Society, 59–78.

[2] Dmitri Akatov. 2010. *Exploiting parallelism in decomposition methods for constraint satisfaction*. Ph.D. Dissertation. University of Oxford.

[3] Dmitri Akatov and Georg Gottlob. 2010. Balanced queries: Divide and conquer. In *Proceedings of the 35th International Symposium on Mathematical Foundations of Computer Science (MFCS'10) (Lecture Notes in Computer Science)*, Vol. 6281. Springer, 42–54. DOI:http://dx.doi.org/10.1007/978-3-642-15155-2_6

[4] Jochen Alber, Frederic Dorn, and Rolf Niedermeier. 2005. Experimental evaluation of a tree decomposition-based algorithm for vertex cover on planar graphs. *Discrete Applied Mathematics* 145, 2 (2005), 219–231. DOI:http://dx.doi.org/10.1016/j.dam.2004.01.013

[5] Eric Allender, Shiteng Chen, Tiancheng Lou, Periklis A. Papakonstantinou, and Bangsheng Tang. 2014. Width-parametrized SAT: Time–space tradeoffs. *Theor. Comput.* 10 (2014), 297–339. DOI:http://dx.doi.org/10.4086/toc.2014.v010a012

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

[7] Sanjeev Arora and Boaz Barak. 2009. *Computational Complexity - A Modern Approach*. Cambridge University Press.

[8] Per Austrin, Petteri Kaski, Mikko Koivisto, and Jussi Määttä. 2013. Space-time tradeoffs for subset sum: An improved worst case algorithm. In *Proceedings of the 40th International Colloquium on Automata, Languages, and Programming (ICALP'13), Part I (Lecture Notes in Computer Science)*, Vol. 7965. Springer, 45–56.

[9] Brenda S. Baker. 1994. Approximation algorithms for NP-complete problems on planar graphs. *J. ACM* 41, 1 (1994), 153–180. DOI:http://dx.doi.org/10.1145/174644.174650

[10] Greg Barnes, Jonathan F. Buss, Walter L. Ruzzo, and Baruch Schieber. 1998. A sublinear space, polynomial time algorithm for directed *s-t* connectivity. *SIAM J. Comput.* 27, 5 (1998), 1273–1282. DOI:http://dx.doi.org/10.1137/S0097539793283151

[11] Marina Barsky, Ulrike Stege, Alex Thomo, and Chris Upton. 2007. Shortest path approaches for the longest common subsequence of a set of strings. In *Proceedings of the 7th IEEE International Conference on Bioinformatics and Bioengineering (BIBE'07)*. IEEE Computer Society, 327–333. DOI:http://dx.doi.org/10.1109/BIBE.2007.4375584

[12] Nadja Betzler, Rolf Niedermeier, and Johannes Uhlmann. 2006. Tree decompositions of graphs: Saving memory in dynamic programming. *Discrete Optim.* 3, 3 (2006), 220–229. DOI : http://dx.doi.org/10.1016/j.disopt.2006.05.008

[13] Andreas Björklund, Thore Husfeldt, Petteri Kaski, and Mikko Koivisto. 2010. Narrow sieves for parameterized paths and packings. *CoRR* abs/1007.1161 (2010). http://arxiv.org/abs/1007.1161

[14] Hans L. Bodlaender. 2005. Discovering treewidth. In *Proceedings of the 31st Conference on Current Trends in Theory and Practice of Computer Science (SOFSEM'05) (Lecture Notes in Computer Science)*, Vol. 3381. Springer, 1–16. DOI : http://dx.doi.org/10.1007/978-3-540-30577-4_1

[15] Hans L. Bodlaender, Jitender S. Deogun, Klaus Jansen, Ton Kloks, Dieter Kratsch, Haiko Müller, and Zsolt Tuza. 1998. Rankings of graphs. *SIAM J. Discrete Math.* 11, 1 (1998), 168–181. DOI : http://dx.doi.org/10.1137/S0895480195282550

[16] Hans L. Bodlaender, Rodney G. Downey, Michael R. Fellows, and Harold T. Wareham. 1995. The parameterized complexity of sequence alignment and consensus. *Theor. Comput. Sci.* 147, 1&2 (1995), 31–54. DOI : http://dx.doi.org/10.1016/0304-3975(94)00251-D

[17] Liming Cai, Jianer Chen, Rodney G. Downey, and Michael R. Fellows. 1995. On the structure of parameterized problems in NP. *Inf. Comput.* 123, 1 (1995), 38–49. DOI : http://dx.doi.org/10.1006/inco.1995.1156

[18] Liming Cai and David W. Juedes. 2003. On the existence of subexponential parameterized algorithms. *J. Comput. Syst. Sci.* 67, 4 (2003), 789–807. DOI : http://dx.doi.org/10.1016/S0022-0000(03)00074-6

[19] Diptarka Chakraborty and Raghunath Tewari. 2015. An O(n^ϵ) space and polynomial time algorithm for reachability in directed layered planar graphs. In *Proceedings of the 26th International Symposium on Algorithms and Computation (ISAAC'15) (Lecture Notes in Computer Science)*, Vol. 9472. Springer, 614–624. DOI : http://dx.doi.org/10.1007/978-3-662-48971-0_52

[20] Hubie Chen and Moritz Müller. 2014. One hierarchy spawns another: Graph deconstructions and the complexity classification of conjunctive queries. In *Proceedings of the Joint Meeting of the 23rd EACSL Annual Conference on Computer Science Logic (CSL) and the 29th Annual ACM/IEEE Symposium on Logic in Computer Science (LICS) (CSL-LICS'14)*. ACM, 32:1–32:10. DOI : http://dx.doi.org/10.1145/2603088.2603107

[21] Andrew Chiu, George I. Davida, and Bruce E. Litow. 2001. Division in logspace-uniform NC[1]. *Theoret. Informatics and Appl.* 35, 3 (2001), 259–275. DOI : http://dx.doi.org/10.1051/ita:2001119

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

[23] Marek Cygan, Fedor V. Fomin, Łukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michał Pilipczuk, and Saket Saurabh. 2015. *Parameterized Algorithms*. Springer.

[24] Erik D. Demaine, Fedor V. Fomin, Mohammad Taghi Hajiaghayi, and Dimitrios M. Thilikos. 2005. Subexponential parameterized algorithms on bounded-genus graphs and H-minor-free graphs. *J. ACM* 52, 6 (2005), 866–893.

[25] Rodney G. Downey and Michael R. Fellows. 1999. *Parameterized Complexity*. Springer. DOI : http://dx.doi.org/10.1007/978-1-4612-0515-9

[26] Rodney G. Downey and Michael R. Fellows. 2013. *Fundamentals of Parameterized Complexity*. Springer.

[27] Jeff Edmonds, Chung Keung Poon, and Dimitris Achlioptas. 1999. Tight lower bounds for st-connectivity on the NNJAG model. *SIAM J. Comput.* 28, 6 (1999), 2257–2284. DOI : http://dx.doi.org/10.1137/S0097539795295948

[28] Michael Elberfeld, Martin Grohe, and Till Tantau. 2012. Where first-order and monadic second-order logic coincide. In *Proceedings of the 27th Annual IEEE Symposium on Logic in Computer Science (LICS'12)*. IEEE Computer Society, 265–274.

[29] Michael Elberfeld, Andreas Jakoby, and Till Tantau. 2010. Logspace versions of the theorems of bodlaender and courcelle. *Electronic Colloquium on Computational Complexity (ECCC)* 17 (2010), 62. Extended abstract included in the proceedings of FOCS 2010.

[30] Michael Elberfeld, Andreas Jakoby, and Till Tantau. 2012. Algorithmic meta theorems for circuit classes of constant and logarithmic depth. In *Proceedings of the 29th International Symposium on Theoretical Aspects of Computer Science (STACS'12)*, Vol. 14. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 66–77. DOI : http://dx.doi.org/10.4230/LIPIcs.STACS.2012.66

[31] Michael Elberfeld, Christoph Stockhusen, and Till Tantau. 2015. On the space and circuit complexity of parameterized problems: Classes and completeness. *Algorithmica* 71, 3 (2015), 661–701. DOI : http://dx.doi.org/10.1007/s00453-014-9944-y

[32] J. Flum and M. Grohe. 2006. *Parameterized Complexity Theory* (1 ed.). Springer.

[33] Fedor V. Fomin, Petteri Kaski, Daniel Lokshtanov, Fahad Panolan, and Saket Saurabh. 2015. Parameterized single-exponential time polynomial space algorithm for steiner tree. In *Proceedings of the 42nd International Colloquium on Automata, Languages, and Programming (ICALP'15), Part I (Lecture Notes in Computer Science)*, Vol. 9134. Springer, 494–505.

[34] Martin Fürer and Huiwen Yu. 2014. Space saving by dynamic algebraization. In *Proceedings of the 9th International Computer Science Symposium in Russia (CSR'14) (Lecture Notes in Computer Science)*, Vol. 8476. Springer, 375–388.

[35] M. R. Garey and David S. Johnson. 1979. *Computers and Intractability: A Guide to the Theory of NP-Completeness.* W. H. Freeman.

[36] M. R. Garey, David S. Johnson, and Larry J. Stockmeyer. 1976. Some simplified NP-complete graph problems. *Theor. Comput. Sci.* 1, 3 (1976), 237–267. DOI:http://dx.doi.org/10.1016/0304-3975(76)90059-1

[37] Georg Gottlob, Nicola Leone, and Francesco Scarcello. 2001. The complexity of acyclic conjunctive queries. *J. ACM* 48, 3 (2001), 431–498. DOI:http://dx.doi.org/10.1145/382780.382783

[38] Sylvain Guillemot. 2011. Parameterized complexity and approximability of the longest compatible sequence problem. *Discrete Optim.* 8, 1 (2011), 50–60. DOI:http://dx.doi.org/10.1016/j.disopt.2010.08.003

[39] Meir Katchalski, William McCuaig, and Suzanne M. Seager. 1995. Ordered colourings. *Discrete Math.* 142, 1-3 (1995), 141–154.

[40] Ton Kloks. 1994. *Treewidth, Computations and Approximations.* Lecture Notes in Computer Science, Vol. 842. Springer. DOI:http://dx.doi.org/10.1007/BFb0045375

[41] Arie M. C. A. Koster, Stan P. M. van Hoesel, and Antoon W. J. Kolen. 2002. Solving partial constraint satisfaction problems with tree decomposition. *Networks* 40, 3 (2002), 170–180. DOI:http://dx.doi.org/10.1002/net.10046

[42] Alexander Langer, Felix Reidl, Peter Rossmanith, and Somnath Sikdar. 2014. Practical algorithms for MSO model-checking on tree-decomposable graphs. *Comput. Sci. Rev.* 13–14 (2014), 39–74. DOI:http://dx.doi.org/10.1016/j.cosrev.2014.08.001

[43] Richard J. Lipton. 2010. Savitch's theorem. In *The P=NP Question and Gödel's Lost Letter.* Springer, 135–138.

[44] Richard J. Lipton and Robert Endre Tarjan. 1980. Applications of a planar separator theorem. *SIAM J. Comput.* 9, 3 (1980), 615–627.

[45] Daniel Lokshtanov, Matthias Mnich, and Saket Saurabh. 2011. Planar k-path in subexponential time and polynomial space. In *Proceedings of the 37th International Workshop on Graph-Theoretic Concepts in Computer Science (WG'11), Revised Papers (Lecture Notes in Computer Science)*, Vol. 6986. Springer, 262–270. DOI:http://dx.doi.org/10.1007/978-3-642-25870-1_24

[46] Daniel Lokshtanov and Jesper Nederlof. 2010. Saving space by algebraization. In *Proceedings of the 42nd ACM Symposium on Theory of Computing (STOC'10).* ACM, 321–330. DOI:http://dx.doi.org/10.1145/1806689.1806735

[47] Pinyan Lu, Jialin Zhang, Chung Keung Poon, and Jin-yi Cai. 2005. Simulating undirected *st*-connectivity algorithms on uniform JAGs and NNJAGs. In *Proceedings of the 16th International Symposium on Algorithms and Computation (ISAAC'05) (Lecture Notes in Computer Science)*, Vol. 3827. Springer, 767–776. DOI:http://dx.doi.org/10.1007/11602613_77

[48] Burkhard Monien and Ivan Hal Sudborough. 1985. Bandwidth constrained NP-complete problems. *Theor. Comput. Sci.* 41 (1985), 141–167. DOI:http://dx.doi.org/10.1016/0304-3975(85)90068-4

[49] Jesper Nederlof. 2013. Fast polynomial-space algorithms using inclusion-exclusion. *Algorithmica* 65, 4 (2013), 868–884. DOI:http://dx.doi.org/10.1007/s00453-012-9630-x

[50] Jaroslav Nešetřil and Patrice Ossona de Mendez. 2006. Tree-depth, subgraph coloring and homomorphism bounds. *Eur. J. Comb.* 27, 6 (2006), 1022–1041. DOI:http://dx.doi.org/10.1016/j.ejc.2005.01.010

[51] J. Nešetřil and P. Ossona de Mendez. 2012. *Sparsity: Graphs, Structures, and Algorithms.* Algorithms and Combinatorics, Vol. 28. Springer.

[52] Noam Nisan. 1994. RL <= SC. *Computational Complexity* 4 (1994), 1–11. DOI:http://dx.doi.org/10.1007/BF01205052

[53] Periklis A. Papakonstantinou. 2009. A note on width-parameterized SAT: An exact machine-model characterization. *Inf. Process. Lett.* 110, 1 (2009), 8–12. DOI:http://dx.doi.org/10.1016/j.ipl.2009.09.012

[54] Periklis A. Papakonstantinou. 2014. The depth irreducibility hypothesis. In *Proceedings of the Electronic Colloquium on Computational Complexity (ECCC)* 21 (2014), 124. http://eccc.hpi-web.de/report/2014/124

[55] Krzysztof Pietrzak. 2003. On the parameterized complexity of the fixed alphabet shortest common supersequence and longest common subsequence problems. *J. Comput. Syst. Sci.* 67, 4 (2003), 757–771.

[56] Alex Pothen. 1988. The complexity of optimal elimination trees. (1988). Technical Report CS 88-16, Pennsylvania State University.

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

[58] J. Barkley Rosser and Lowell Schoenfeld. 1962. Approximate formulas for some functions of prime numbers. *Illinois J. Math.* 6, 1 (03 1962), 64–94.

[59] Francesca Rossi, Peter van Beek, and Toby Walsh (Eds.). 2006. *Handbook of Constraint Programming.* Foundations of Artificial Intelligence, Vol. 2. Elsevier.

[60] Walter L. Ruzzo. 1980. Tree-size bounded alternation. *J. Comput. Syst. Sci.* 21, 2 (1980), 218–235. DOI:http://dx.doi.org/10.1016/0022-0000(80)90036-7

[61] Walter J. Savitch. 1970. Relationships between nondeterministic and deterministic tape complexities. *J. Comput. Syst. Sci.* 4, 2 (1970), 177–192.

[62] V. Vinay and V. Chandru. 1990. The expressibility of nondeterministic auxiliary stack automata and its relation to treesize bounded alternating auxiliary pushdown automata. In *Proceedings of the 10th Conference on Foundations of Software Technology and Theoretical Computer Science (Lecture Notes in Computer Science)*, Vol. 472. Springer, 104–114. DOI : http://dx.doi.org/10.1007/3-540-53487-3_38

[63] Avi Wigderson. 1992. The complexity of graph connectivity. In *Proceedings of the 17th International Symposium on Mathematical Foundations of Computer Science (MFCS'92) (Lecture Notes in Computer Science)*, Vol. 629. Springer, 112–132. DOI : http://dx.doi.org/10.1007/3-540-55808-X_10