Parallel Multivariate Meta-Theorems

Max Bannach[1] and Till Tantau[2]

1 **Institute for Theoretical Computer Science, Universität zu Lübeck, Germany**
`bannach@tcs.uni-luebeck.de`

2 **Institute for Theoretical Computer Science, Universität zu Lübeck, Germany**
`tantau@tcs.uni-luebeck.de`

Abstract

Fixed-parameter tractability is based on the observation that many hard problems become tractable even on large inputs as long as certain input parameters are small. Originally, "tractable" just meant "solvable in polynomial time," but especially modern hardware raises the question of whether we can also achieve "solvable in polylogarithmic parallel time." A framework for this study of *parallel fixed-parameter tractability* is available and a number of isolated algorithmic results have been obtained in recent years, but one of the unifying core tools of classical FPT theory has been missing: algorithmic meta-theorems. We establish two such theorems by giving new upper bounds on the circuit depth necessary to solve the model checking problem for monadic second-order logic, once parameterized by the tree width and the formula (this is a parallel version of Courcelle's Theorem) and once by the tree depth and the formula. For our proofs we refine the analysis of earlier algorithms, especially of Bodlaender's, but also need to add new ideas, especially in the context where the parallel runtime is bounded by a function of the parameter and does not depend on the length of the input.

1998 ACM Subject Classification F.1.3 Complexity Measures and Classes

Keywords and phrases Parallel computation, FPT, meta-theorems, tree width, tree depth

Digital Object Identifier 10.4230/LIPIcs.IPEC.2016.4

1 Introduction

Algorithmic meta-theorems bound the computational resources needed to solve problems defined in a certain logic for inputs from a specific class of structures. The prime example is Courcelle's Theorem [5], which states that monadic second-order (MSO) definable problems can be solved in linear time on structures with bounded tree width. This yields, for instance, a linear time algorithm for the feedback vertex set problem on graphs of bounded tree width. Other examples are a "logspace version" [6] or a theorem for structures of bounded tree depth, where constant depth circuits (AC^0) suffice [7]; many more versions can be found in the surveys by Grohe and Kreutzer [10] and Kreutzer [12].

With the rise of multivariate algorithms, algorithmic meta-theorems have become useful tools for establishing parameterized upper bounds. The prime example is again Courcelle's Theorem, which actually gives a linear-time FPT-algorithm when the tree width of the input structure is the parameter. Since the tree width of a graph with a feedback vertex set of size k is at most $k + 1$, the theorem shows that the naturally parameterized feedback vertex set problem can be solved in parameterized linear time.

The field of parameterized complexity is renowned for its ability to find algorithms that solve NP- or even PSPACE-complete problems in reasonable time. Unfortunately, "reasonable time" is not quite the same as "fast" and, furthermore, the instances in typical applications for these algorithms are huge. We may thus wish to speedup the computation by taking advantage of the multiple cores and powerful GPUs present in modern hardware. In order to do so, we need *parallel* fixed-parameter algorithms. A first step in this direction was taken by us in [1], where we showed that the vertex cover problem, among several other problems, allows fast parallel fixed-parameter algorithms; but for many problems, including the feedback vertex set problem, the parallel parameterized complexity remains open. In particular, results concerning the parallel fixed-parameter tractability of problems have been obtained on a problem-by-problem basis without an overarching, unifying approach – which is exactly what the present paper tries to remedy.

Our Contributions. We formulate and prove different *parallel parameterized meta-theorems,* which unify previous results and allow us to obtain new algorithms for natural problems. Our meta-theorems are obtained by translating the logspace and circuit versions of Courcelle's Theorem from [6, 7] into parameterized counter parts, but we must point out already at this point that this is harder than one might expect: Unlike the original linear-time version of Courcelle's Theorem, which is "a theorem about parameterized complexity in disguise," the logspace and circuit versions just state that problems lie in the classes XL and XAC^0. However, these latter classes are presumably not even contained in FPT, let alone in parallel subclasses thereof and, thus, are not the classes we are looking for.

To establish the parallel parameterized meta-theorem, we need to study the parameterized parallel complexity of computing tree decomposition of parameterized width and possibly also depth. At the heart of Courcelle's Theorem and related versions are tree automata that process the tree decomposition of the input. We provide fast parallel algorithms to evaluate parameter-sized tree automata on arbitrary trees and on trees of parameterized depth. By combining these algorithms with the parallel algorithms for computing tree decompositions, we obtain parallel algorithms for monadic second-order model checking on graphs of parameterized tree width or parameterized tree depth ($p_{\phi,\mathrm{td/tw}}$-$\mathrm{MC}(\mathrm{MSO})$). The logic is defined as usual, for instance the following MSO-sentence describes that a graph is colorable with three colors:

$$\begin{aligned}\phi = {} & \exists R\,\exists G\,\exists B\,\forall x\,\big(R(x) \lor G(x) \lor B(x)\big)\\ & \land \forall x, y\,\big(E(x,y) \to (\neg R(x) \lor \neg R(y)) \land (\neg G(x) \lor \neg G(y)) \land (\neg B(x) \lor \neg B(y))\big).\end{aligned}$$

The model checking problem asks, given a logical structure (for instance a graph), and a logical formula, whether or not the structure is a model for the formula. For example, we have $\models \phi$, but $\not\models \phi$. For an introduction to the field, we refer to [8]. Our main results are stated in form of the following theorems (para-$\mathrm{AC}^{0\uparrow}$ contains problems decidable by "FPT-sized" circuits whose depth depends only on the parameter, detailed definition follow later):

▶ **Theorem 1.** $p_{\phi,\mathrm{td}}$-$\mathrm{MC}(\mathrm{MSO}) \in \text{para-}\mathrm{AC}^{0\uparrow}$.

▶ **Theorem 2.** $p_{\phi,\mathrm{tw}}$-$\mathrm{MC}(\mathrm{MSO}) \in \text{para-}\mathrm{NC}^{2+\epsilon}$.

Armed with these new meta-theorems, we settle the parallel parameterized complexity of different natural problems, including the feedback vertex set problem.

Related Work. The prime example of algorithmic meta-theorems is Courcelle's Theorem [5] which becomes powerful in combination with Bodlaender's linear-time algorithm for computing optimal tree decompositions of graphs of bounded tree width [4]. Since the release of this theorem, many other meta-theorems, which place many problems in P, were presented, see [10, 12] for surveys. For Courcelle's Theorem there are versions for other classes: a LOGCFL-version by Wanke [15], which was later improved to an L-version by Elberfeld, Jakoby, and the last author [6], who also prove an AC^0-version [7]. While most meta-theorems that place problems in P place the parameterized version of the problem in para-P = FPT, this is not the case for the last-mentioned versions: The L- and AC^0-versions of Courcelle's Theorem place problems in XL and XAC^0 and not, as we would like, in para-L and para-AC^0. Early studies on the parallel complexity of computing tree decompositions for graphs of bounded tree width where made by Bodlaender [3]. However, the algorithm does not obtain "FPT-work" in the parameterized setting and only yields a XNC-algorithm. Lagergren provided a parallel $O(\log^3 n)$ time algorithm using $O(n)$ processors for this problem in the CRCW model [13], which translates into a para-NC algorithm for parameterized problems. Bodlaender and Hagerup later provided a parallel algorithm with optimal speedup running in time $O(\log^2 n)$ using $O(n)$ operations on the EREW model [2]. This algorithm readily translates into a para-NC algorithm, but only the careful analysis done in this paper shows that it is actually a para-$\mathrm{NC}^{2+\epsilon}$ algorithm.

Organization of This Paper. In Section 2 we define our basic terminology and recap the definition of classes of fixed-parameter parallelism. In Section 3 we provide parallel algorithms to compute tree decompositions of graphs with parameterized tree width or parameterized tree depth. In Section 4 we provide parallel algorithms to evaluate tree automata on arbitrary trees and on trees of parameterized depth. Putting it all together, we provide parallel algorithms for monadic second-order model checking on graphs of parameterized tree width or depth in Section 5. We close the paper by studying the parallel complexity of certain parameterized problems with the help of these meta-theorems in Section 6. Due to lack of space, proofs have been moved to the appendix.

2 Classes of Fixed-Parameter Parallelism

We use standard terminology of parameterized complexity theory, see for instance [8]. A *parameterized problem* is a tuple (Q, κ) of a language $Q \subseteq \Sigma^*$ and a *parameterization* $\kappa\colon \Sigma^* \to \mathbb{N}$. As we deal with small parameterized circuit classes, we require the parameter to be computable in DLOGTIME-uniform AC^0 or, equally, to be first-order computable.[1] We denote parameterized problems by a leading "p-" as in p-VERTEX-COVER, and, whenever the parameter is not clear from the context, we add it as index as in p_{tw}-DISTANCE.

A parameterized problem (Q, κ) is called *fixed-parameter tractable* if there is a language R decidable in polynomial time (P) and a computable function $f\colon \mathbb{N} \to \mathbb{N}$ such that $x \in Q$ if, and only if, $(x, 1^{f(\kappa(x))}) \in R$. That is, the problem is decidable in polynomial time after an arbitrarily complex pre-computation on the parameter. The resulting complexity class is called FPT or para-P. If we replace P in this definition by subclasses of P, we obtain subclasses of FPT, which inherit their inclusion structure from their classical counter parts:

$$\text{para-AC}^0 \subsetneq \text{para-TC}^0 \subseteq \text{para-NC}^1 \subseteq \text{para-L} \subseteq \text{para-NL} \subseteq \text{para-AC}^1 \subseteq \text{para-P}.$$

In order to explicitly define what the parameterized circuit classes contain, we use the definition from [1]:

[1] Sometimes this definition is to restrictive, for instance the tree width of a graph is computable in FPT, but probably not in P, and certainly not in AC^0. In such cases we assume that the input is extended by an upper bound on the parameter, which can easily be extracted in AC^0. However, in this case any algorithm deciding the problem has to verify this parameter by itself.

▶ **Definition 3** (Classes of Parallel Fixed-Parameter Tractability)**.** Let $d\colon \mathbb{N}^2 \to \mathbb{N}$ be a *depth bounding function* and $s\colon \mathbb{N}^2 \to \mathbb{N}$ be a *size bounding function* which both map each pair of an input length and a parameter to a number. We define para-AC$[d,s]$ as the class of parameterized problems (Q,κ) for which there exists a DLOGTIME-uniform[2] family $(C_{n,k})_{n,k\in\mathbb{N}}$ of AC-circuits (only NOT-, AND-, and OR-gates are allowed, AND- and OR- gates may have unbounded fan-in) such that: (1) For all $x \in \Sigma^*$, the circuit $C_{|x|,\kappa(x)}$ evaluates to 1 on input x if, and only if, $x \in Q$. (2) The depth of each $C_{n,k}$ is at most $d(n,k)$. (3) The size of each $C_{n,k}$ is at most $s(n,k)$.

We define the classes para-AC^i as para-AC^0 = para-AC$[O(1), f(\kappa(x)) \cdot |x|^{O(1)}]$ and for $i > 0$ para-AC^i = para-AC$[O(\log^i |x|), f(\kappa(x)) \cdot |x|^{O(1)}]$ (in slight abuse of notation, as its actually the union of this class over all computable functions f). However, there are also interesting new classes, namely the "up-"classes, defined in [1]:

$$\text{para-AC}^{i\uparrow} = \text{para-AC}[f(\kappa(x)) \cdot \log^i |x|, f(\kappa(x)) \cdot |x|^{O(1)}].$$

Note that para-AC$^{0\uparrow}$ captures exactly the problems that can be solved by a circuit of depth depending only on the parameter, and "FPT"-size. Notice, furthermore, that the "up-"classes can be strictly more powerful than the underlying classes (para-AC$^0 \subsetneq$ para-AC$^{0\uparrow}$ [1]), but that a slight increase of the depth in dependence on $|x|$ compensates this effect: We have para-AC$^i \subseteq$ para-AC$^{i\uparrow} \subseteq$ para-AC$^{i+\epsilon}$. These definitions and observations can, of course, also be applied to circuits of bounded fan-in (NC), and to circuits that are equipped with threshold-gates (TC).

To get familiar with parameterized circuits, let us consider an important technique from the design of parallel algorithms: *symmetry breaking*, that is, the ability to find parts of the input that can be processed in parallel. For graph algorithms in the PRAM model, this is often achieved by computing maximal independent-sets. In the lemma, as in the rest of the paper, f is an appropriate computable function and c is an appropriate constant.

▶ **Lemma 4.** *There is a* DLOGTIME*-uniform family of* AC*-circuits of depth $f(k) + \log^* |V|$ and size $f(k) \cdot |V|^c$ that, on input of an undirected graph $G = (V,E)$ and an integer k, outputs either that the maximum degree of G exceeds k or a maximal independent set I of G.*

Notice that, in sense of circuit classes, the lemma yields a para-AC$^{0+\epsilon} \subseteq$ para-NC$^{1+\epsilon}$ circuit for computing maximal independent sets with respect to the parameter "maximum degree."

3 Parallel Computation of Tree Decompositions

In our algorithmic meta-theorems, the tree *width* and tree *depth* of the input graphs are of special interest: First, they are parameters and, second, our algorithms work on the tree decompositions underlying the input graphs. Thus, it is of particular interest how such tree decompositions can be computed in parallel.

Recall the definition of a tree decomposition (T,ι) of a graph $G = (V,E)$. It is a rooted tree T together with a mapping ι from the nodes of T to subsets of V (which we call *bags*) such that for each vertex $v \in V$ and for each edge $\{v,w\} \in E$ there is (1) at least one node n in T with $v \in \iota(n)$, (2) at least one node n in T with $\{v,w\} \subseteq \iota(n)$, and (3) the set of nodes of

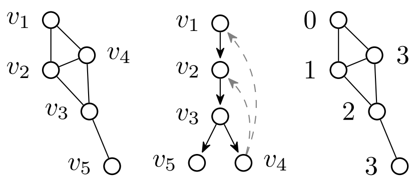
[2] In this context, this means that the circuit $C_{n,k}$ can be computed in time $f(k) + O(\log n)$ by a deterministic Turing machine that obtains $1^n\#1^k$ as input.

T that contain v in their bag is connected. The *width* of a tree decomposition is the maximum size of its bags minus 1, its *depth* is the maximum of its width and the depth of the tree T. For a graph G, we define $\mathrm{tw}(G)$ to be the minimum width each tree decomposition of G has to have, and we define $\mathrm{td}(G)$ in a similar way for the tree depth.[3] For many algorithms it is useful to have a certain form of a tree decomposition: A *nice* tree decomposition is a tuple (T, ι, η) such that (T, ι) is a tree decomposition and $\eta\colon V(T) \to \{\mathrm{leaf}, \mathrm{introduce}, \mathrm{join}, \mathrm{forget}\}$ is a labeling function of the nodes. The nodes that are labeled as *leaf* are exactly the leafs and the root of T, and the bags of these nodes are empty. *Introduce*- and *forget*-nodes n have one child x such that there is one $v \in V$ with $v \notin \iota(x)$ and $\iota(n) = \iota(x) \cup \{v\}$, or $v \in \iota(x)$ and $\iota(n) = \iota(x) \setminus \{v\}$, respectively. *Join*-nodes n have two children x and y with $\iota(n) = \iota(x) = \iota(y)$. A tree decomposition (T, ι) is called *balanced* if T is a balanced tree; a nice tree decomposition (T, ι, η) is *balanced* if the tree obtained from T by contracting introduce and forget nodes is balanced. We refer to the textbook from Flum and Grohe for a more detailed introduction into the field [8].

Computing Depth-Bounded Tree Decompositions. We first study the case that we deal with graphs parameterized by their tree depth. This class of graphs is well suited for parallel algorithms, as a parallel algorithm can traverse the whole decomposition in time depending only on the parameter. We will see in this section that we can also compute a tree decomposition of parameterized depth within this time bound.

▶ **Theorem 5.** *There is a* DLOGTIME*-uniform family of* AC*-circuits of depth $f(k)$ and size $f(k)\cdot|G|^c$ that, on input of an undirected graph $G = (V, E)$ and an integer k, either determines $\mathrm{td}(G) > k$ or outputs a tree decomposition (T, ι) of G with depth bounded by $O\big(2^{\mathrm{td}(G)}\big)$.*

In order to prove Theorem 5 we will use known facts about the relation of bounded-depth tree decompositions and depth-first search trees [14]. To use these facts, we need a representation of a depth-first search tree that is suitable for our circuit model. Let $G = (V, E)$ be a graph with $s \in V$, and let T be a depth-first search tree of G starting at s, a *depth-first search labeling* is a mapping $\lambda_s\colon V \to \mathbb{N}$ such that $\lambda_s(v)$ is the distance from s to v in T. The figure below shows from left to right: an example graph, a depth-first search tree starting at v_1, and a corresponding depth-first search labeling.



In a similar way, we can define a *breadth-first search labeling* with respect to a breadth-first search tree. Notice that in this case the labeling is actually the (path) distance from s to the other vertices.

▶ **Lemma 6.** *There is a* DLOGTIME*-uniform family of* AC*-circuits of depth $f(k)$ and size $f(k)\cdot|G|^c$ that, on input of an undirected graph $G = (V, E)$, a vertex $s \in V$, and an integer k, either correctly detects that the longest path in G is longer than 2^k, or that output a depth-first and a breadth-first search labeling starting at s.*

[3] Note that the common definition of tree depth is slightly different, but that it is an upper bound for the definition we use.

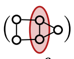
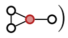
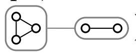
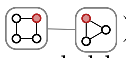
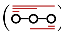
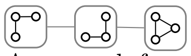
Proof of Theorem 5. It is a well-known fact [14] that the length of the longest path in a graph G is bounded by $2^{\mathrm{td}(G)}$. A direct consequence is that a depth-first search tree can be used to obtain a tree decomposition (T, ι) of width and depth bounded by $2^{\mathrm{td}(G)}$: let us assume G is connected and let T be a depth-first search tree rooted at an arbitrary start vertex $r \in V$. For all $v \in V$ define $\iota(v) = \{\, w \mid w$ lies on the unique path from v to r in $T\,\}$. The depth of T is naturally bounded by $2^{\mathrm{td}(G)}$, and, therefore, we also have $|\iota(v)| \leq 2^{\mathrm{td}(G)}$ for each $v \in V$. Since bags extend along the paths from the root to the leaves of T, all the conditions of a tree decomposition are satisfied by (T, ι).

A circuit with the desired size and depth can compute a depth-first search labeling using Lemma 6, and either conclude that the length of the longest path exceeds k, and therefore $\mathrm{td}(G) > k$, or it can compute the bags of the decomposition in parallel. For each $v \in V$ the circuit initializes the bag $\iota(v) = \{v\}$. As long as $r \notin \iota(v)$, the circuit repeats the following sequentially: let $w \in \iota(v)$ the vertex that minimizes $\lambda(w)$ in $\iota(v)$, the circuits adds the unique $w' \in N(w)$ that satisfies $\lambda(w') = \lambda(w) - 1$ to $\iota(v)$. To complete the proof, we have to handle the case that G is not connected. The circuit can compute all connected components of G using a breadth-first search labeling (Lemma 6). Afterwards, the circuit can apply the algorithm from above to each connected component. Finally, the circuit adds a new empty root bag that is connected to the roots of all constructed tree decompositions. This operation does not increase the width and increases the depth only by one. ◀

Computing Width-Bounded Tree Decompositions. We will now handle the case that the input graph is parameterized by tree width. In this case the depth of a tree decomposition is not bounded by any function in the parameter and, thus, it seems unlikely that parallel algorithms running in time depending only on the parameter exist. And, indeed, deciding if a graph has tree width at most k for a fixed k is already L-complete and, hence, the parameterized version of this problem cannot lie in para-AC^0 or para-AC$^{0\uparrow}$.

▶ **Theorem 7.** *There is a* DLOGTIME*-uniform family of* NC*-circuits with depth* $f(k) + \log^{2+\epsilon}|G|$ *and width* $f(k) \cdot |G|^c$ *that, on input of an undirected graph* $G = (V, E)$ *and an integer* k*, either determines* $\mathrm{tw}(G) > k$ *or outputs a tree decomposition of* G *of width at most* k*.*

The proof of Theorem 7 is essentially a new analysis of a parallel algorithm from Bodlaender and Hagerup [2]. They provide an $O(\log^2 n)$ time and $O(n)$ work algorithm on the EREW-PRAM model to compute optimal tree decompositions of graphs with bounded tree width, from which one can derive that the problem of computing a tree decomposition lies somewhere in para-NC. Our main contribution in the following is a careful analysis regarding the exact circuit class the algorithm achieves: It is para-NC$^{2+\epsilon}$ for all $\epsilon > 0$.

The idea of the algorithm is as follows: If $G = (V, E)$ is small enough, we can compute an optimal tree decomposition via "brute-force", otherwise we try to reduce the graph until it has a suitable size. We call two vertices $u, v \in V$ *reduction partner* if they are adjacent or twins (). We can reduce the size of G by 1 if we *contract* the two vertices, that is, if we remove v from G after connecting all neighbors of v to u (without creating multi-edges: ). Let G' be the resulting graph, and let (T', ι') be a recursively computed tree decomposition of G' of width at most k (). We can compute a tree decomposition (T, ι) of G of width at most $k+1$ by *injecting* v into (T', ι'), that is, by adding v to all bags that contain u (). The resulting tree decomposition is most likely not optimal, but its width is bounded by a function in k and we can use it to compute an implicit representation of an optimal tree decomposition of G. This implicit representation, called *path labeled tree* *representation*, is a binary tree T in which for every $v \in V$ exactly two vertices are labeled with v, i. e., the vertices of V correspond to paths in T (). If we consider the nodes as bags, each bag that lies on the unique path between two nodes labeled width v will contain v. Each node may be labeled with multiple vertices, but of course with at most $k + 1$. Given such an implicit representation, we can compute a tree decomposition of width k ().

As we seek for a circuit of polylogarithmic depth, we can not only contract one reduction pair in every round, as we would require $O(|V|)$ rounds. Fortunately, there are always multiple reduction partners that can be contracted in parallel. The correctness of the algorithm is shown in [2]. For us, it remains to show that we can implement the algorithm in para-$\mathrm{NC}^{2+\epsilon}$, a task for which we have to show that each subfunction of the algorithm can be realized by circuits of logarithmic depth and polynomial size. The following lemmas show that all parts of a *single* iteration of the algorithm can be computed by a para-$\mathrm{NC}^{1+\epsilon}$-circuit.

▶ **Lemma 8.** *There is a* DLOGTIME*-uniform family of* NC*-circuits of depth* $f(k) + \log^{1+\epsilon}|V|$ *and size* $f(k) \cdot |V|^c$ *that, on input of a graph* $G = (V, E)$ *and* $k \in \mathbb{N}$*, outputs a set* I *of* $1/g(k) \cdot |V|$ *pairs of vertices that can be contracted in parallel, or that concludes* $\mathrm{tw}(G) > k$.

▶ **Lemma 9.** *There is a* DLOGTIME*-uniform family of* NC*-circuits of depth* $f(k) \cdot \log |V|$ *and size* $f(k) \cdot |V|^c$ *that, on input of a graph* $G = (V, E)$*, a set of pairs of vertices* I*, a graph* $G' = (V', E')$ *that is obtained from* G *by contracting the pairs in* I*, and a tree decomposition* (T', ι') *of* G' *of width* k*, outputs a balanced and nice tree decomposition* (T, ι, η) *of* G *of width at most* $8k + 3$ *and depth* $(16k + 6) \cdot \log |V| + 1$.

▶ **Lemma 10.** *There is a* DLOGTIME*-uniform family of* NC*-circuits of depth* $f(k) \cdot \log |V|$ *and size* $f(k) \cdot |V|^c$ *that, on input of a graph* $G = (V, E)$*, an integer* k*, and of a balanced and nice tree decomposition* (T, ι, η) *of* G *of width at most* $\ell \le f(k)$*, outputs either* $\mathrm{tw}(G) > k$ *or a width-*k *tree decomposition of* G.

Proof of Theorem 7. The circuit first checks whether the size of the input graph is bounded by k. If this is the case, an optimal tree decomposition can be computed via "brute-force". Otherwise, the circuit computes a set of $1/f(k) \cdot |V|$ reduction pairs using Lemma 8, or concludes that the tree width of G exceeds k. The circuit reduces G to G' by contracting the reduction pairs (the lemma guarantees that this is possible in parallel) and recursively computes a tree decomposition of G'. This tree decomposition can be transformed to a nice and balanced decomposition of G of width bounded by a function in k using Lemma 9. Finally, the circuit can reduce the width of the decomposition to k or conclude $\mathrm{tw}(G) > k$ using Lemma 10.

Since Lemma 8 provides us with $1/f(k) \cdot |V|$ reduction pairs, $f(k) \cdot \log |V|$ rounds of the algorithm are sufficient to reduce the graph to a size depending only on the parameter. Considering each round as a subcircuit, each subcircuit has to execute the algorithms from the lemmas 8, 9, and 10. The most expensive part is Lemma 8, as the circuit needs depth $f(k) + \log^{1+\epsilon}$ here, for the lemmas 9 and 10 circuits of depth $f(k) \cdot \log |V| \le f(k) + \log^{1+\epsilon}|V|$ are sufficient. The complete circuit has, therefore, a total depth of $f(k) \log |V| \cdot \big(f(k) + \log^{1+\epsilon}|V|\big) \le f(k) + \log^{2+\epsilon}|V|$, and is, hence, a para-$\mathrm{NC}^{2+\epsilon}$-circuit. ◀

4 Parallel Evaluation of Tree Automata

A key aspect of modern algorithmic meta-theorems is the simulation of tree automata, since such theorems commonly translate a tree decomposition of the input structure into a labeled tree that is accepted by a certain tree automaton if, and only if, the structure was a model for the input formula. "Classical" translations produce degree-bounded trees that are then processed by classical tree automata. However, this approach may increase the depth of the tree decomposition by up to a logarithmic factor, which is unacceptable if we wish to handle the tree in parallel time depending on the depth of the tree. As a solution, the authors of [7] suggest the use of *multiset automata.* A *multiset* M is a set S together with a multiplicity function $\#_M \colon S \to \mathbb{N}$. The *multiplicity* of M is $\max_{e \in S} \#_M(e)$. We denote by $\mathcal{P}_\omega(S)$ the class of all multisets of S and by $\mathcal{P}_m(S)$ the class of all multisets of multiplicity at most $m \in \mathbb{N}$ of S. Notice that $\mathcal{P}_1(S)$ is just the standard power set of S. For a multiset $M \in \mathcal{P}_\omega(S)$ and a number $m \in \mathbb{N}$, the *capped version* $M|_m$ of M is defined by setting $\#_M(e) = \min(\#_M(e), m)$ for all $e \in S$.

▶ **Definition 11** (Multiset Automaton). A *nondeterministic* (bottom-up) multiset automaton is a tuple $\mathcal{A} = (\Sigma, Q, Q_a, \Delta, m)$ consisting of an *alphabet* Σ, a *state set* Q with *accepting states* $Q_a \subseteq Q$, a *state transition relation* $\Delta \subseteq \Sigma \times \mathcal{P}_m(Q) \times Q$, and a *multiplicity bound* $m \in \mathbb{N}$. The automaton is *deterministic* if for every $\sigma \in \Sigma$ and every $M \in \mathcal{P}_m(Q)$ there is exactly one $q \in Q$ with $(\sigma, M, q) \in \Delta$; in this case we can view Δ as *state transition function* $\delta \colon \Sigma \times \mathcal{P}_m(Q) \to Q$.

▶ **Definition 12** (Computation of a Multiset Automaton). Let $(\mathfrak{T}, \lambda)$ be a labeled tree, where $\lambda \colon V(\mathfrak{T}) \to \Sigma$ is the labeling function, and let $\mathfrak{A} = (\Sigma, Q, Q_a, \Delta, m)$ be a multiset automaton. A *computation* of $\mathfrak{A}$ on $(\mathfrak{T}, \lambda)$ is a partial assignment $q \colon V(\mathfrak{T}) \to Q$ such that for every node $n \in V(\mathfrak{T})$ for which $q(n)$ is defined, we have that (a) the value $q(c)$ is defined for each child c of n in $\mathfrak{T}$ and (b) for the multiset $M = \{\, q(c) \mid c \text{ is a child of } n \,\}$ we have $(\lambda(n), M \mid_m, q(n)) \in \Delta$. A computation is *accepting* if $q(r) \in Q_a$ holds for the root node r of $\mathfrak{T}$. The *tree language* $\mathrm{L}(\mathfrak{A})$ contains all labeled trees accepted by $\mathfrak{A}$.

▶ **Fact 13** ([7]). *The following statements hold and are constructive:*
1. *For all multiset automata* $\mathfrak{A}$ *and* $\mathfrak{B}$ *there is a multiset automaton* $\mathfrak{C}$ *with* $\mathrm{L}(\mathfrak{C}) = \mathrm{L}(\mathfrak{A}) \cap \mathrm{L}(\mathfrak{B})$*;*
2. *For every nondeterministic multiset automaton* $\mathfrak{A}$ *there is a deterministic multiset automaton* $\mathfrak{B}$ *with* $\mathrm{L}(\mathfrak{A}) = \mathrm{L}(\mathfrak{B})$*;*
3. *For every multiset automaton* $\mathfrak{A}$ *there is a multiset automaton* $\mathfrak{B}$ *accepting the complement of* $\mathrm{L}(\mathfrak{A})$*.*

The actual aim of this section is to study the parallel parameterized complexity of the simulation of a multiset automaton. Since we will need such simulations in different scenarios, instead of classifying the problem into complexity classes, we identify circuit families depending on different parameters.

▶ **Lemma 14.** *Let* $S_{k,d}$ *be the set of labeled trees* $(\mathfrak{T}, \lambda)$ *of maximal depth* d *and maximal degree* k*. There is a* DLOGTIME*-uniform family of circuits over the standard base (only* AND-*,* OR-*, and* NOT*-gates) with fan-in* k*, depth* $f(|\mathfrak{A}|) \cdot d$ *and size* $f(|\mathfrak{A}|) \cdot |\mathfrak{T}|^c$ *that, on input of a labeled tree* $(\mathfrak{T}, \lambda) \in S_{k,d}$ *and a multiset automata* $\mathfrak{A} = (\Sigma, Q, Q_a, \Delta, m)$*, decides whether or not* $(\mathfrak{T}, \lambda) \in \mathrm{L}(\mathfrak{A})$ *holds.*

As used later on, we will mention two special cases of Lemma 14: The simulation of multiset automata can be performed (a) in para-$\mathrm{AC}^{0\uparrow}$ for trees of depth bounded by the parameter and (b) in para-$\mathrm{NC}^{1\uparrow}$ for balanced binary trees. Here, the size of the automata is the parameter.

5 Parallel Second-Order Model Checking

The goal of this section is to actually prove Theorem 1 and Theorem 2. The classical way of proving variants of Courcelle's Theorem is as follows: On input of a logical structure $\mathcal{S}$ and a MSO-formula ϕ, we first compute a tree decomposition (T, ι) of $\mathcal{S}$. This tree decomposition is then translated into a s-tree-structure $\mathcal{T}$ and ϕ is translated to a new MSO-formula ψ such that $\mathcal{S} \models \phi \Leftrightarrow \mathcal{T} \models \psi$. To decide $\mathcal{T} \models \psi$, the s-tree-structure $\mathcal{T}$ is transformed into a labeled tree $(\mathfrak{T}, \lambda)$ and ψ is turned into a multiset automata $\mathfrak{A}$ such that $\mathcal{T} \models \psi \Leftrightarrow (\mathfrak{T}, \lambda) \in \mathrm{L}(\mathfrak{A})$. Here, an s-tree-structure is a structure $\mathcal{T} = (V, E^{\mathcal{T}}, P_1^{\mathcal{T}}, \dots, P_s^{\mathcal{T}})$ over the signature $\tau_{s\text{-tree}} = (E^2, P_1^1, \dots, P_s^1)$ where $(V, E^{\mathcal{T}})$ is a directed tree.

▶ **Fact 15** (Implicit in [7]). *There are functions h_1, h_2, h_3 and h_4 performing the following mappings:*

1. *The input for h_1 are a structure $\mathcal{S}$ together with a width-w tree decomposition (T, ι) of $\mathcal{S}$ and an MSO-formula ϕ. The output is an s-tree-structure $\mathcal{T}$.*
2. *The input for h_2 are an MSO-formula ϕ and a tree width w. The output is an MSO-formula formula ψ.*
3. *The input for h_3 are an s-tree-structure $\mathcal{T}$ and an MSO-formula ψ. The output is a labeled tree $(\mathfrak{T}, \lambda)$ of the same depth.*
4. *The input for h_4 is an MSO-formula ψ. The output is a multiset automaton $\mathfrak{A}$.*

The following holds for the values computed by these functions:

$$\mathcal{S} \models \phi \iff \mathcal{T} \models \psi \iff (\mathfrak{T}, \lambda) \in \mathrm{L}(\mathfrak{A}).$$

All h_i are computable and h_1 and h_3 are even computable by DLOGTIME*-uniform* AC*-circuits of depth $O(1)$ and size $f(\phi, w)|\mathcal{S}||T|$.*

Since the size of ϕ and the tree depth or width of the input structure are parameters in our setting, we can use Fact 15 to prove Theorem 1 and Theorem 2:

Proof of Theorem 1. On input of a logical structure $\mathcal{S}$ and an MSO-formula ϕ, a para-$\mathrm{AC}^{0\uparrow}$-circuit can compute a tree decomposition (T, ι) of the Gaifman graph of $\mathcal{S}$ (the graph that uses the universe of $\mathcal{S}$ as vertex-set and that contains an edge between two elements if, and only if, the two elements stand in any relation) using Theorem 5. Given the tuple $(\mathcal{S}, (T, \iota), \phi)$, the circuit can then compute a labeled tree $(\mathfrak{T}, \lambda)$ and a multiset automaton $\mathfrak{A}$ using Fact 15. The depth of $\mathfrak{T}$ is bounded by the depth of (T, ι) and, hence, bounded by the parameter. Furthermore, we have $|\mathfrak{A}| \leq f(|\psi| + \mathrm{td}(\mathcal{S}))$ for a computable function f. Hence, a para-$\mathrm{AC}^{0\uparrow}$-circuit can now invoke Lemma 14 and output the result. ◀

Proof of Theorem 2. The proof is almost identical to the proof of Theorem 1. On input of a logical structure $\mathcal{S}$ and a MSO-formula ϕ, a para-$\mathrm{NC}^{2+\epsilon}$-circuit can compute a tree decomposition (T, ι) of the Gaifman graph of $\mathcal{S}$ using Theorem 7. At this point a problem arises, as the depth of (T, ι) is not bounded. This can be overcome as follows: Let the width of (T, ι) be w, then a FTC^0-circuit can compute a balanced tree decomposition (T', ι') of width at most $4w + 3$ [7]. Given this decomposition, we can proceed as in the proof of Theorem 1 and compute the labeled tree $(\mathfrak{T}, \lambda)$ and a multiset automaton $\mathfrak{A}$. Since $(\mathfrak{T}, \lambda)$ is balanced, it is binary and of logarithmic depth, and, therefore, Lemma 14 can be invoked by a para-$\mathrm{NC}^{1\uparrow}$-circuit which presents the result as output. ◀

6 Applications

Most graph problems studied in complexity theory can be described in monadic second order logic, including vertex cover, dominating set, independent set, or clique, and, thus, our algorithmic meta-theorems apply to them. For instance, we get corollaries like $p_{\mathrm{tw},k}$-DOMINATING-SET $\in$ para-$\mathrm{NC}^{2+\epsilon}$ and $p_{\mathrm{td},k}$-DOMINATING-SET $\in$ para-$\mathrm{AC}^{0\uparrow}$.

It is, however, worth to take a closer look, as we are naturally interested in more precise parameterizations than in the combined parameter td/tw $+ k$. Although the tree width and depth are fairly sensible parameters, we are even more interested in the complexity of the problems without restrictions on *these* parameters (but, perhaps still with other, more natural parameters). Sometimes this is possible, as the tree width is parameterized *indirectly.* For the feedback vertex set problem (given an undirected graph $G = (V, E)$ and a parameter k, decide whether there exists a set $S \subseteq V$ with $|S| \leq k$ such that $G[V \setminus S]$ is acyclic) the existence of such a set implies that the tree width of G is at most $k + 1$: Since $G[V \setminus S]$ is a tree, adding S to each bag of a tree decomposition of $G[V \setminus S]$ yields a tree decomposition of G of width at most $k + 1$.

▶ **Corollary 16.** *p_k-FEEDBACK-VERTEX-SET $\in$ para-$\mathrm{NC}^{2+\epsilon}$.*

The above corollary is currently the best result on the parallel parameterized complexity of the feedback vertex set problem. In other scenarios the opposite is possible, i. e., the tree width indirectly parameterizes the solution size. A well known example is the clique problem, as any graph with tree width at most w can not contain a clique bigger than $w + 1$.

▶ **Corollary 17.** *p_{td}-CLIQUE $\in$ para-$\mathrm{AC}^{0\uparrow}$, p_{tw}-CLIQUE $\in$ para-$\mathrm{NC}^{2+\epsilon}$.*

On the other hand, there are problems where we can not hope for such effects. For instance, the dominating-set problem is well known to be W[2]-complete and, hence, we can not hope to get rid of the parameter tree width. Since the tree width does not bound k in this case either, we do not get rid of this parameter as well. Nevertheless, our meta-theorems at least improve the known upper bound for the dominating-set problem with combined parameter. In contrast, for the vertex cover problem we do need both parameters as well and obtain $p_{\mathrm{tw},k}$-VERTEX-COVER $\in$ para-$\mathrm{NC}^{2+\epsilon}$ ($p_{\mathrm{td},k}$-VERTEX-COVER $\in$ para-$\mathrm{AC}^{0\uparrow}$), but one can show directly [1] that p_k-VERTEX-COVER $\in$ para-AC^0 holds. In other words, our algorithmic meta-theorems do not yield an optimal bound on the vertex cover problem, a "less generic" approach yields better bounds.

Reachability Problems. The charm of studying parameterized parallel complexity is that it is not only interesting to consider NP- or even PSPACE-hard problems, but also problems that lie within P. For instance, the classical reachability problem in directed graphs REACH $= \{ (G, s, t) \mid \text{in } G \text{ is a path from } s \text{ to } t \}$ is a natural NL-complete problem. If we consider graphs with parameterized tree depth, the complexity of the problem can be lowered by Theorem 1.

▶ **Corollary 18.** *p_{td}-REACH $\in$ para-$\mathrm{AC}^{0\uparrow}$.*

From the point of view of parallel complexity, we are especially interested in P-complete problems, since it is believed that such problems are inherent sequential. A natural P-complete version of the reachability problem is the alternating reachability problem [11], which is based on the following definition of *alternating paths:* Given a directed graph $G = (V, E)$ and a partition $V = V_\exists \cup V_\forall$, an *alternating path* from s to t is a set S of paths in G, all of

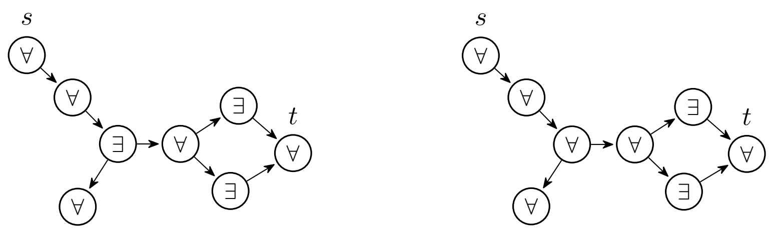


Figure 1 Examples of input graphs for the alternating reachability problem. In left graph there is an alternating path from s to t and the alternating distance is 5, in the right one there is not.

which end at t, such that (1) exactly one of them starts at s; (2) when a path in S starts at some $v \in V_\exists \setminus \{t\}$, then there is for some w with $(v, w) \in E$ a path in S starting at w; and (3) when a path in S starts at some $v \in V_\forall \setminus \{t\}$, then for all w with $(v, w) \in E$ there is a path in S starting at w (and there is at least one such w). The *length* of an alternating path is the maximum length of any path in the set S. The *alternating distance* between two vertices is the minimum distance of any alternating path between them.

▶ **Problem 19** (p_{tw}-AREACH, p_{td}-AREACH).
Instance: *A directed graph $G = (V, E)$, a partition $V = V_\exists \cup V_\forall$, and two vertices $s, t \in V$.*
Parameter: *Tree width or tree depth of G.*
Question: *Is there an alternating path from s to t in G?*

▶ **Problem 20** ($p_{\text{tw},d}$-ADISTANCE, $p_{\text{td},d}$-ADISTANCE).
Instance: *A directed graph $G = (V, E)$, a partition $V = V_\exists \cup V_\forall$, two vertices $s, t \in V$, a distance d.*
Parameter: *Tree width or tree depth of G as well as d.*
Question: *Is the alternating distance from s to t in G at most d?*

It is a standard exercise to describe the alternating reachability and distance problems using a monadic second order formula and, thus, our algorithmic meta-theorems yield the following:

▶ **Corollary 21.** p_{tw}-AREACH $\in$ para-NC$^{2+\epsilon}$, $p_{\text{tw},d}$-ADISTANCE $\in$ para-NC$^{2+\epsilon}$.

▶ **Corollary 22.** p_{td}-AREACH $\in$ para-AC$^{0\uparrow}$, $p_{\text{td},d}$-ADISTANCE $\in$ para-AC$^{0\uparrow}$.

It turns out that, as for the vertex cover problem, for the alternating distance problem we can do better, but also, that the classes we study are the "right" classes for these problems:

▶ **Theorem 23.** p_d-ADISTANCE *is complete for* para-AC$^{0\uparrow}$ *under* para-AC^0*-reduction.*

7 Conclusion

Algorithmic meta-theorems play a key role in modern complexity theory. We have seen that this powerful tool can also be applied to the study of parameterized parallel algorithms. Indeed, the results state that MSO-definable problems on graphs with parameterized tree width do not only allow linear time dynamic programs, but that these problems also allow fast parallel algorithms as well. The theorems show that problems definable in monadic second order logic can be solved in parallel time $f(k)$ or $f(k) \cdot \log n$ if a tree decomposition of parameterized depth or width is given. In the first case, we have seen that such a decomposition can be computed in the same time. However, in the second case it turns out that the bottleneck is the computation of such a decomposition, since we were only able to show that this can be done in time $f(k) + \log^{2+\epsilon} n$. A reasonable research goal is therefore to seek an algorithm between para-L and para-$\mathrm{NC}^{2+\epsilon}$ that computes a tree decomposition of a given graph with parameterized tree width. A first step would be to reduce the circuit depth to para-$\mathrm{NC}^{2\uparrow}$.

References

1 M. Bannach, C. Stockhusen, and T. Tantau. Fast parallel fixed-parameter algorithms via color coding. In *10th International Symposium on Parameterized and Exact Computation (IPEC 2015)*, pages 224–235. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2015. doi:10.4230/LIPIcs.IPEC.2015.224.

2 H. Bodlaender and T. Hagerup. Parallel algorithms with optimal speedup for bounded treewidth. *SIAM Journal on Computing*, 27(6):1725–1746, 1998. doi:10.1137/S0097539795289859.

3 Hans L. Bodlaender. NC-algorithms for graphs with small treewidth. In *Graph-Theoretic Concepts in Computer Science: International Workshop*, WG'88, pages 1–10. Springer Berlin Heidelberg, 1989. doi:10.1007/3-540-50728-0_32.

4 H. L. Bodlaender. A linear time algorithm for finding tree-decompositions of small treewidth. In *Proceedings of the Twenty-Fifth Annual ACM Symposium on Theory of Computing, May 16-18, 1993, San Diego, CA, USA*, STOC'93, pages 226–234. ACM, New York, USA, 1993. doi:10.1145/167088.167161.

5 B. Courcelle. Graph rewriting: An algebraic and logic approach. In *Formal Models and Semantics*, volume B of *Handbook of Theoretical Computer Science*, pages 193–242. Elsevier, Amsterdam, Netherlands and MIT Press, Cambridge, Massachusetts, 1990. doi:10.1016/B978-0-444-88074-1.50010-X.

6 M. Elberfeld, A. Jakoby, and T. Tantau. Logspace Versions of the Theorems of Bodleander and Courcelle. In *Proceedings of the Annual IEEE Symposium on Foundations of Computer Science, October 23–26, 2010, Las Vegas, USA*, FOCS'10, pages 143–152. IEEE Computer Society, Los Alamitos, California, 2010. doi:10.1109/FOCS.2010.21.

7 M. Elberfeld, A. Jakoby, and T. Tantau. Algorithmic meta theorems for circuit classes of constant and logarithmic depth. In *Proceedings of the Twenty-Ninth International Symposium on Theoretical Aspects of Computer Science, February 29 – March 3, 2012, Prais, France*, STACS'12, pages 66–77. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2012. doi:10.4230/LIPIcs.STACS.2012.66.

8 J. Flum and M. Grohe. *Parameterized Complexity Theory.* Springer, Heidelberg, Germany, 2006. doi:10.1007/3-540-29953-X.

9 A. Goldberg, S. Plotkin, and G. Shannon. Parallel symmetry-breaking in sparse graphs. In *Proceedings of the Nineteenth Annual ACM Symposium on Theory of Computing*, STOC'87, pages 315–324. ACM, New York, USA, 1987. doi:10.1145/28395.28429.

10 M. Grohe and S. Kreutzer. Methods for algorithmic meta theorems. In *Model Theoretic Methods in Finite Combinatorics*, pages 181–206. AMS, Contemporary Mathematics Series, 2011. doi:10.1090/conm/558/11051.

11 Neil Immerman. Languages which capture complexity classes. In *Proceedings of the Fifteenth Annual ACM Symposium on Theory of Computing*, STOC'83, pages 347–354. ACM New York, NY, 1983. doi:10.1145/800061.808765.

12 Stephan Kreutzer. Algorithmic meta-theorems. *CoRR*, abs/0902.3616, 2009. URL: http://arxiv.org/abs/0902.3616.

13 Jens Lagergren. Efficient parallel algorithms for graphs of bounded tree-width. *Journal of Algorithms*, 20:20–44, 1996. doi:10.1006/jagm.1996.0002.
14 Jaroslav Nešetřil and Patrice Ossona de Mendez. *Sparsity.* Springer Berlin Heidelberg, 2012. doi:10.1007/978-3-642-27875-4.
15 Egon Wanke. Bounded tree-width and logcfl. *Graph-Theoretic Concepts in Computer Science*, 790:33–44, 2005. doi:10.1006/jagm.1994.1022.

A Technical Appendix: Proofs

For the readers convenience, the claims of the proofs given in this appendix are repeated before the proofs.

Claim of Lemma 4. *There is a* DLOGTIME*-uniform family of* AC*-circuits of depth* $f(k) + \log^* |V|$ *and size* $f(k) \cdot |V|^c$ *that, on input of an undirected graph* $G = (V, E)$ *and an integer* k*, outputs either that the maximum degree of* G *exceeds* k *or a maximal independent set* I *of* G*.*

Proof. As the circuit may have depth $f(k)$, it can count the degree of each vertex and can directly reject if any degree exceeds k [1]. Otherwise, the circuit implements the algorithm from Goldberg, Plotkin, and Shannon to compute a maximal independent set in degree-bounded graphs [9]. The circuit interprets G as directed graph $\vec{G}$ by considering each edge $\{u, v\}$ as two directed edges (u, v) and (v, u). The edge set of this graph is partitioned into k sets $E_1, \dots, E_k$ such that each of the graphs $\vec{G_i} = (V, E_i)$ has only vertices of out-degree at most 1. This partition can be computed in depth $f(k)$ as the circuit has essentially to count up to k.

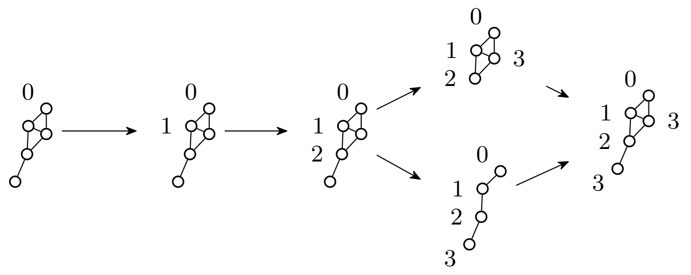
The circuit now performs the following operations on all $\vec{G_i}$ in parallel: First, in constant depth, an initial coloring of $\vec{G_i}$ is computed by assigning each vertex v_i the color $i \in \mathbb{N}$, which needs at most $\log |V|$ bits. This coloring can be improved to a coloring with $\log |V|$ colors in constant depth: Replace the color c of each vertex v by $2k + b$, where k is the position of the lowest bit on which c differs from the color of the unique successor of v, and where b is the value of this bit. Computing this improvement consecutively $\log^* |V|$ times yields a coloring with 6 colors [9].

Given the colorings of the k graphs $\vec{G_i}$, the circuit can compute a 6^k coloring of G by assigning to each vertex the k-tuple of colors that this vertex has in the different $\vec{G_i}$. Finally, the circuit initializes a set $I = \emptyset$, iterates over the colors and, in parallel, adds all vertices of the current color that do not have a neighbor in I to I. As each step can be performed in a constant number of AC-layers, the set I can be computed in $f(k)$ AC-layers. The circuit outputs I, as the final value of I is a maximal independent-set. The total depth of the circuit is $f(k) + \log^* |V|$. ◀

Claim of Lemma 6. *There is a* DLOGTIME*-uniform family of* AC*-circuits of depth* $f(k)$ *and size* $f(k) \cdot |G|^c$ *that, on input of an undirected graph* $G = (V, E)$*, a vertex* $s \in V$*, and an integer* k*, either correctly detects that the longest path in* G *is longer than* 2^k*, or that output a depth-first and a breadth-first search labeling starting at* s*.*

Proof. We first handle the breadth-first search labeling, which yields a natural parallel algorithm. Our circuit starts by assigning color 0 to s. The circuit is build up of layers, where layer $i + 1$ assigns color $i + 1$ to each vertex that is not colored yet and that has at least one vertex of color i as neighbor. The algorithm stops if all vertices are colored, or at the very last after 2^k layers. In the later case, the circuit can report that the length of the longest path exceeds 2^k. After a run of the algorithm, each vertex that has obtained a color is in the same connected component as s and, furthermore, the colors constitute a breadth-first search labeling starting at s.

Computing a depth-first search labeling turns out to be more complicated, since an AC-circuit of the desired depth cannot simply follow a path of the search tree and "backtrack" once it reaches a leaf, as the depth of the circuit would not be bounded by the longest path in this case. Instead, we have to compute the vertices which have more than one child in the depth-first search tree in advance. Once we know these vertices, we can perform a fork and compute the depth-first search labeling for all of there children in parallel. Since we have seen how the circuit can compute a breadth-first search labeling, we can assume that we have access to a subcircuit that computes the connected components of G. In order to compute the depth-first search labeling, the circuit first computes these connected components and checks if the the longest path in all these components is bounded by 2^k. Afterwards, the following algorithm, which we call a *phase*, is executed in parallel on all connected components with color $c = 0$ as argument. Each phase does nothing if all vertices are colored, this is the end of the recursion. If $c = 0$, an arbitrary vertex is selected and colored with c, otherwise an arbitrary vertex that is not colored, but that has a neighbor of color $c - 1$, is selected and colored with c. At the end of a phase the vertices of G are partitioned in colored vertices C and the uncolored vertices $V \setminus C$. The circuit computes the connected components of $G[V \setminus C]$, which we denote by $V_1, \dots, V_\ell \subseteq V \setminus C$. Afterwards, a new phase is started recursively on each graph $G[V_i \cup C]$. If all phases are completed, the coloring constitutes a depth-first search labeling starting at s.



Since this algorithm never performs backtracking, the number of consecutive phases is bounded by the length of the longest path, which is bounded by 2^k. For each phase, a circuit of depth $f(k)$ is sufficient, since the most expensive part is clearly the computation of the connected components. Thus, a depth first-search labeling can be computed by an AC-circuit of depth $f(k)$. ◀

Claim of Lemma 8. *There is a* DLOGTIME*-uniform family of* NC*-circuits of depth* $f(k) + \log^{1+\epsilon}|V|$ *and size* $f(k) \cdot |V|^c$ *that, on input of a graph* $G = (V, E)$ *and* $k \in \mathbb{N}$*, outputs a set* I *of* $1/g(k) \cdot |V|$ *pairs of vertices that can be contracted in parallel, or that concludes* $\mathrm{tw}(G) > k$.

Proof. We call two vertices u, v *reduction partners* if we have either $\{u, v\} \in E$, or if they are twins, i. e., $N(u) = N(v)$. Let us call a vertex v *d-small* if $\delta(v) \leq d$.

Let $d = 2^{k+4}(54k + 54)$ and $c = 1/\big(8(27k + 27)^2\big)$. If $\mathrm{tw}(G) \leq k$, then there are at least $c|V|/2$ distinct pairs $\{u, v\}$ of d-small vertices that are reduction partners [2]. Since a circuit of the desired size can check all pairs of vertices in parallel, it can compute in the desired depth a set S of reduction partners. Furthermore, the circuit can check whether $|S| \leq c|V|/2$ holds, and can reject otherwise.

We cannot contract all pairs in S simultaneously, as pairs may share a vertex, may be adjacent, or may have a common neighbor. Since all these properties are first-order definable, a circuit of the desired size and depth can easily check for each pair of reduction partners if they are in conflict. By doing so, the circuit can compute a *conflict graph* C whose node set is S and whose edges indicate conflicts. As the degree of each vertex appearing in a pair in S is bounded by d, the degree of C is bounded by $g(k)$ for a computable function g.

Since each maximal independent set I in a graph of maximum degree Δ has size at least $|V|/(\Delta+1)$, it is sufficient to use the reduction partners that constitute a maximal independent set in C. The circuit can compute such a set using Lemma 4. ◀

Claim of Lemma 9. *There is a* DLOGTIME*-uniform family of* NC*-circuits of depth* $f(k)\cdot\log|V|$ *and size* $f(k)\cdot|V|^c$ *that, on input of a graph* $G=(V,E)$*, a set of pairs of vertices* I*, a graph* $G'=(V',E')$ *that is obtained from* G *by contracting the pairs in* I*, and a tree decomposition* (T',ι') *of* G' *of width* k*, outputs a balanced and nice tree decomposition* (T,ι,η) *of* G *of width at most* $8k+3$ *and depth* $(16k+6)\cdot\log|V|+1$*.*

Proof. Let (T',ι') be the given tree decomposition. An AC^0-circuit can compute (T,ι) by adding for each pair $\{u,v\}\in I$ the vertex v to every bag that contains u. This can be done in parallel for all vertices and all bags. Since the number of vertices in each bag is at most doubled, (T,ι) has width at most $2k$.

This decomposition can be transformed into a balanced one of width at most $8k+3$ by a TC^0-circuit [7]. The last thing we have to do is to transform this decomposition into a nice decomposition (T,ι,η). In order to do so, the circuit first adds an empty bag to each leaf, which is labeled as *leaf node*. Then, each node n with two children x and y is replaced by nodes n, n_l, and n_r such that n_l, n_r are the children of n, x is a child of n_l, and y a child of n_r. The node n is labeled as *join node*. This operation doubles the depth of the decomposition. Finally, for every node x with child y, the circuit computes a chain of *forget nodes* from x to a new node z with $\iota(x)\cap\iota(y)=\iota(z)$, and a chain of *introduce nodes* from z to y. This will increase the depth of the decomposition at most by a factor of $8k+3$.

Since making a balanced tree decomposition nice will result in a balanced decomposition again, the above algorithm produced a nice, balanced tree decomposition of width at most $2k$ and depth at most $(16k+6)\log|V|+1$. ◀

Claim of Lemma 10. *There is a* DLOGTIME*-uniform family of* NC*-circuits of depth* $f(k)\cdot\log|V|$ *and size* $f(k)\cdot|V|^c$ *that, on input of a graph* $G=(V,E)$*, an integer* k*, and of a balanced and nice tree decomposition* (T,ι,η) *of* G *of width at most* $\ell\leq f(k)$*, outputs either* $\mathrm{tw}(G)>k$ *or a width-*k *tree decomposition of* G*.*

Proof. The original algorithm by Bodlaender and Hagerup [2] computes a path labeled tree representation of a tree decomposition of width k of G, or correctly detects $\mathrm{tw}(G)>k$. This algorithm “bubbles up” the nice tree decomposition and spends $f(k)$ time on every node. Since the depth of the tree is $f(k)\log|V|$, the desired circuit can implement this algorithm without modification.

After the execution of the above algorithm, the circuit may either reject if the algorithm reports that the tree width exceeds k, or obtains a path labeled tree representation. Recall that this implicit representation is a labeled binary tree T where exactly two nodes are labeled with each vertex of G. The idea is that the unique path between these two nodes defines the bags in which the vertex (used as label) lies. Since the "real" tree decomposition we try to extract from this implicit representation uses the same tree, the rest of the lemma boils down to the following algorithmic task: Given a tree $T = (V, E)$ and three nodes $s, x, t \in V$, decide whether or not x lies on the unique path between s and t. This property is clearly expressible in MSO and, since T is a tree (of tree width 1), decidable in NC^1 [7]. ◀

Claim of Lemma 14. *Let $S_{k,d}$ be the set of labeled trees $(\mathfrak{T}, \lambda)$ of maximal depth d and maximal degree k. There is a* DLOGTIME*-uniform family of circuits over the standard base (only* AND-*,* OR-*, and* NOT*-gates) with fan-in k, depth $f(|\mathfrak{A}|) \cdot d$ and size $f(|\mathfrak{A}|) \cdot |\mathfrak{T}|^c$ that, on input of a labeled tree $(\mathfrak{T}, \lambda) \in S_{k,d}$ and a multiset automata $\mathfrak{A} = (\Sigma, Q, Q_a, \Delta, m)$, decides whether or not $(\mathfrak{T}, \lambda) \in \mathrm{L}(\mathfrak{A})$ holds.*

Proof. Since both, the depth and the size of the circuit, depend on the size of $\mathfrak{A}$ by an arbitrary computable function f, we can assume that $\mathfrak{A}$ is deterministic, since if this is not the case we can compute an equivalent deterministic automaton using Fact 13. The circuit has d layers, each of which consists of circuits of depth $f(|\mathfrak{A}|)$. The i-th layer will assign states to the nodes of the $(d - i)$-th layer of $\mathfrak{T}$. The first layer simply assigns states to the leafs of $\mathfrak{T}$. Layer i then has access to the assigned states of layer $i - 1$. In order to compute the state $q(n)$ for a node n the circuit computes the multiset $M = \{\, q(c) \mid c \text{ is a child of } n \,\}$ using the result of the last layer. Now the circuit has to cap M to compute $M|_m$. In order to do so, the circuit has to count up to m. Since we have $m \le |\mathfrak{A}|$, the value m is bounded by the parameter and, therefore, a para-AC^0 layer is sufficient for this task [1]. Once $M|_m$ is computed, the circuit can compute $q(n)$ by a lookup of $(\lambda(n), M|_m)$ in the description of δ. The circuit outputs 1 if, and only if, after the evaluation of the d layers the root r of $\mathfrak{T}$ is assigned with $q(r) \in Q_a$.

Clearly, the depth of the circuit is bounded by $f(|\mathfrak{A}|) \cdot d$. To see that the fan-in of the circuit does only depend on the maximal degree of $\mathfrak{T}$, observe the following: The subcircuit of a layer computing $q(n)$ for a node n has size bounded by $f(|\mathfrak{A}|)$ and, hence, can be replaced by a circuit of fan-in two without violating the depth bound of $f(|\mathfrak{A}|)$. The bigger fan-in is only needed to transmit the multiset $M = \{\, q(c) \mid c \text{ is a child of } n \,\}$ to the subcircuit computing $q(n)$, but since we have $|M| \le k$ the claim follows. ◀

Claim of Theorem 23. *p_d-*ADISTANCE *is complete for* para-$\mathrm{AC}^{0\uparrow}$ *under* para-AC^0*-reduction.*

Proof. For containment consider a circuit that performs a backward breadth-first search starting at t, similar to Lemma 6. The circuit handles the graph in d layers, computing in layer i the vertices that have alternating distance i to t. In the first layer, vertex t is colored. In layer i, all vertices $x \in V_\exists$ that have one colored neighbor, and all $y \in V_\forall$ that have only colored neighbors (and at least one) are colored. There is an alternating path of distance at most d from s to t if, and only if, s is colored after d layers. The correctness of the circuit follows by a simple induction: in layer 1 we color exactly the vertices with alternating distance 1, and it is easy to see that coloring a vertex in layer i is only possible if it has a neighbor (or all its neighbors) with alternating distance $i - 1$.

For completeness let us reduce any problem $L \in$ para-$\mathrm{AC}^{0\uparrow}$ to p_d-ADISTANCE. As L is in para-$\mathrm{AC}^{0\uparrow}$, there is a fixed family of circuits deciding L. Let C be such a circuit. We may assume that C is monotone since one can always replace a non-monotone circuit by a monotone one (using the standard argument used in showing that the circuit value problem reduces to its monotone version).

We translate the monotone circuit C into an alternating graph as follows: The vertices of the graph are the gates, and the wires are edges directed from the unique output gate towards the input bits. For each input bit there is a vertex as well. We label the output gate as s, add a new vertex t, and we add edges from all input bits that are set to 1 towards t. We partition the vertices such that $V_\exists$ is the set of OR -gates joined by t and the input bits; and such that $V_\forall$ is the set of AND -gates. The constructed graph with s and t, and with d as distance, constitutes an instance of p_d-ADISTANCE. An alternating path from s to t corresponds to wires that are set to true during the evaluation of the circuit and, hence, such a path can only exist if the circuit evaluates to true. Since, furthermore, the depth of the circuit is bounded by d, such a path has length at most d as well. ◀