Logspace Versions of the Theorems of Bodlaender and Courcelle

Michael Elberfeld
Institut für Theoretische Informatik
Universität zu Lübeck
elberfeld@tcs.uni-luebeck.de

Andreas Jakoby
Institut für Theoretische Informatik
Universität zu Lübeck
jakoby@tcs.uni-luebeck.de

Till Tantau
Institut für Theoretische Informatik
Universität zu Lübeck
tantau@tcs.uni-luebeck.de

***Abstract*—Bodlaender's Theorem states that for every k there is a linear-time algorithm that decides whether an input graph has tree width k and, if so, computes a width-k tree composition. Courcelle's Theorem builds on Bodlaender's Theorem and states that for every monadic second-order formula ϕ and for every k there is a linear-time algorithm that decides whether a given logical structure $\mathcal{A}$ of tree width at most k satisfies ϕ. We prove that both theorems still hold when "linear time" is replaced by "logarithmic space." The transfer of the powerful theoretical framework of monadic second-order logic and bounded tree width to logarithmic space allows us to settle a number of both old and recent open problems in the logspace world.**

***Keywords*-deterministic logarithmic space, tree width, partial k-trees, monadic second-order logic.**

I. Introduction

For graphs of bounded tree width, a concept introduced by Robertson and Seymour [38] and known under several different names such as *partial k-trees*, the computational complexity of many difficult problems drops significantly compared to the same problem for general graphs. For instance, for every k the NP-complete problem HAMILTONICITY can be solved in linear sequential and in logarithmic parallel time when restricted to graphs of tree width at most k. The same is true for many other NP-complete problems, see [10] for an overview. To achieve these time bounds, algorithms for problems on graphs of bounded tree width need access to a tree decomposition of the input graph. Bodlaender's Theorem [8] states that for every fixed k, on input of a graph G of tree width at most k such a tree decomposition can be computed in linear time. Note that k must, indeed, be a fixed constant since it is NP-complete [2] to decide on input (G,k) whether G has tree width at most k.

These results have inspired researchers to investigate whether graphs of bounded tree width might also be helpful in the study of logarithmic space. Here, "difficult" problems include normally easy ones like reachability or matching. The hope is that these problems might be decidable by deterministic logspace Turing machines (logspace DTMs) for graphs of bounded tree width. Only partial results were obtained, for instance for graphs of tree width 2 or for k-trees, and two 2010 papers [19], [20] identify an analogue of Bodlaender's Theorem for logarithmic space as the central piece missing in recent advances in the study of logarithmic space. Our first main result is such an analogue:

Theorem I.1. *For every $k \geq 1$, there is a logspace* DTM *that on input of any graph G of tree width at most k outputs a width-k tree decomposition of G.*

The design of early efficient algorithms for problems on graphs of bounded tree width was a laborious process involving complex, problem-dependent arguments. A breakthrough came with Courcelle's Theorem [18], which in conjunction with Bodlaender's Theorem yields that all graph properties expressible in monadic second-order logic (MSO-logic) can be solved in linear time on graphs of bounded tree width. Since many graph properties are easily expressible in this logic, we get a simple, unified framework for showing that all of these problems are efficiently solvable.

In the logspace world, the situation resembles the one before Courcelle's work: each paper uses similar, but still problem-dependent arguments to establish membership in L or at least in LOGCFL. Our second main result is that Courcelle's Theorem also holds for logarithmic space, enabling us to apply the same unifying framework as for linear time:

Theorem I.2. *For every $k \geq 1$ and every* MSO*-formula ϕ, there is a logspace* DTM *that on input of any logical structure $\mathcal{A}$ of tree width at most k decides whether $\mathcal{A} \models \phi$ holds.*

Courcelle's original theorem has been generalized in different ways (known as *functional, optimization, counting,* and other versions). We also prove such a generalized version, which we call the *cardinality version* and which allows a wider range of applications than Theorem I.2. For its formulation we introduce the notion of *solution histograms:* Let $\phi(X_1,\dots,X_d)$ be an MSO-formula whose free predicate variables are exactly the X_i and let $\mathcal{A}$ be a logical structure with universe A. Then $\text{histogram}(\mathcal{A},\phi)$ is the d-dimensional integer array whose entry at the d-dimensional index $(i_1,\dots,i_d) \in \{0,\dots,|A|\}^d$ tells us how *many* subsets $S_1,\dots,S_d \subseteq A$ exist with $|S_1| = i_1, \dots, |S_d| = i_d$ and $\mathcal{A} \models \phi(S_1,\dots,S_d)$.

Theorem I.3 (Logspace Cardinality Version of Courcelle's Theorem)**.** *For every $k \geq 1$ and every* MSO*-formula*

$\phi(X_1,\ldots,X_d)$*, there is a logspace* DTM *that on input of any logical structure* $\mathcal{A}$ *of tree width at most* k *outputs* histogram$(\mathcal{A},\phi)$.

Observe that histogram$(\mathcal{A},\phi)$ stores a lot of information about ϕ and $\mathcal{A}$, including the number of satisfying assignments of ϕ in the form of the sum of all entries. Theorem I.2 is a special case of Theorem I.3 for $d=0$ since a 0-dimensional histogram is a scalar that is 1 if $\mathcal{A}\models\phi$, and 0 otherwise.

The above theorems make no claim concerning the behavior of the machines on inputs that have a tree width larger than k, but the following lemma shows that this could easily be remedied:

Lemma I.4. *For every* $k\geq 1$ *the language* TREE-WIDTH-k*, which contains exactly the graphs of tree width at most* k*, is* L*-complete under first-order reductions.*

Our main technical contributions are the following: For the proof of the logspace version of Bodlaender's Theorem, the main difficulty lies in coming up with an appropriate notion of graph separators and in showing how the recursive decomposition can be done in logarithmic space. We sidestep the recursion by reducing to a special version of the reachability problem for mangrove graphs, which we show to lie in L. For the logspace cardinality version of Courcelle's Theorem, we show how computing the number of satisfying assignments relates to tree automata (a standard tool in proofs of Courcelle's Theorem) and how it can be reduced to evaluating an arithmetic tree whose entries are tensors that are added and convoluted. This problem, in turn, can be reduced to evaluating a normal arithmetic tree over addition and multiplication, a problem known to be logspace solvable. We remark that our techniques are tailored to make our algorithms use only logarithmic space, at the cost of increasing their runtime to *polynomial* time instead of the *linear* time bound that lies at the heart of the original theorem of Bodlaender.

A. Applications

Our results can be applied in a number of areas. Since in the present paper we focus on proving the main theorems, we will not explore these applications in more detail. Nevertheless, we below try to sketch their impact on some of these areas.

First, for many NP-hard problems like 3-COLORABLE or HAMILTONICITY one can show that they are linear time solvable on graphs of bounded tree width by expressing the problems in MSO-logic and applying Courcelle's Theorem. By Theorem I.2, all of these results can be transferred to the logspace setting. Problems like DOMINATING-SET are also expressible as MSO-formulas, but now $\phi(X)$ holds if X is a dominating set and the question is whether ϕ can be satisfied by an X having a certain size. The logspace *cardinality* version of Courcelle's Theorem shows that this problem can also be decided in logarithmic space. Thus, for every k the languages $\{G \mid \mathrm{tw}(G)\leq k \text{ and } G \text{ is 3-colorable}\}$ and $\{(G,s) \mid \mathrm{tw}(G)\leq k \text{ and } G \text{ has a dominating set of size at most } s\}$ lie in L and this is the case for many other problems. Even when a problem is not directly expressible in MSO-logic, it may still be possible to use Courcelle's Theorem inside a larger algorithm. A simple example is computing the chromatic number of tree width bounded graphs in logarithmic space: no MSO-formula $\phi(X)$ is known that expresses that a graph has chromatic number $|X|$, but it is easily seen [22] that graphs of tree width k have chromatic number at most $k+1$ and we can successively test whether a graph is 1-, 2-, …, $(k+1)$-colorable to compute its chromatic number.

Second, a number of graph properties, such as reachability, can be expressed in MSO-logic, but they can already be checked efficiently on *arbitrary* graphs and applying Courcelle's classical theorem yields no new insights. From a logspace perspective, the situation is different: Applying Theorem I.3 to the formula $\phi(X)$ expressing that X is a simple path from s to t shows that the problem $\{(G,s,t) \mid \mathrm{tw}(G)\leq k$, there is a path from s to t in $G\}$ lies in L. Indeed, on input of (G,s,t,d) we can even compute in logarithmic space the exact number of simple paths from s to t of length exactly d. Another example is the matching problem, where the MSO-characterization allows us to compute the exact number of perfect matchings of graphs of bounded tree width in logarithmic space.

Third, the logspace version of Courcelle's Theorem has applications in the study of pseudopolynomial NP-complete problems: The classical NP-complete problem KNAPSACK is well-known to become tractable when input numbers are coded in unary: UNARY-KNAPSACK $\in$ NL. Inspired by Cook's conjecture [16] that "a problem in NL which is probably not complete is the knapsack problem with unary weights," a line of research began to capture its complexity with specialized complexity classes lying between L and NL [27], [15], [31], see also [34]. Our Theorem I.3 shows that UNARY-SUBSETSUM $\in$ L: Given unary-coded numbers a_1, …, a_n and a target sum s, construct the graph consisting of n stars, where the ith star has a_i vertices, and the formula $\phi(X)$ expressing that X must always contain a star completely or not at all. Then there is a satisfying assignment of cardinality s if, and only if, there is a subset $I\subseteq\{1,\ldots,n\}$ with $\sum_{i\in I} a_i = s$. Similar, but slightly more complex arguments show that UNARY-KNAPSACK $\in$ L and also that UNARY-k-KNAPSACK $\in$ L for every k, where there are k knapsacks.

B. Related Work

The research on the difficulty of computing width-k tree decompositions was originally focused on time complexity. The linear time bound of Bodlaender's Theorem [8] is the

best possible result on the sequential time complexity and improves on previous results [2], [36]. Similarly, the parallel time complexity was reduced in a line of papers [13], [6], [32], [9] to $O(\log n)$. Concerning the space complexity, an algorithm of Gottlob et al. [23], which extends an algorithm of Wanke [39], shows that computing width-k tree decompositions lies in the class $\text{LOGCFL} = \text{SAC}^1 \subseteq \text{AC}^1$. For the special case of computing width-2 tree decompositions a logspace algorithm can be deduced from [28], [29], together with Reingold's algorithm [37].

Algorithmic variants of Courcelle's Theorem also solve MSO-definable counting and optimization problems [3], [11], [17], similar to the cardinality version studied in the present paper. A recent survey [10] spans the time complexity of tree width related problems. The best result concerning the space complexity of analogues of Courcelle's Theorems is due to Wanke [39]. It places MSO-definable decision problems for graphs of bounded tree width in LOGCFL. Additionally, MSO-definable optimization problems like VERTEX-COVER on bounded tree width graphs lie in LOGCFL.

Only few problems were known to lie in L when restricted to graphs of bounded tree width. In [30] we showed that the reachability problem, the shortest path, and also the longest path problems lie in L when restricted to graphs of tree width 2. Das et al. [19] study k-trees, a special case of graphs of tree width k, and show that for these graphs the reachability and the perfect matching problem and, if the graph is a directed acyclic graph (DAG), also the shortest and longest path problems lie in L. All of these results are special cases of the logspace cardinality version of Courcelle's Theorem.

Concerning our proof techniques, the idea of using separators in the construction of tree decompositions is used in many other papers [32], [36]. Mangroves are also used in the study of the isomorphism problem for k-trees [4]. Our reduction in the logspace cardinality version of Courcelle's Theorem is a strong generalization of the reduction to $\{\max,+\}$-trees that we first proposed in [30] and was later also used in [19].

C. Organization of This Paper

In Section III we show that given a graph of tree width at most k, we can compute a tree decomposition of width $4k+3$, called an *approximate* tree decomposition, in logarithmic space. In Section IV we prove the logspace cardinality version of Courcelle's Theorem. The algorithms of this section work on approximate tree decompositions. In Section V we prove the L-completeness of TREE-WIDTH-k and the logspace version of Bodlaender's Theorem by showing that approximate tree decompositions can be turned into optimal ones in logarithmic space. Due to lack of space, proofs are omitted or sketched; detailed proofs can be found in the ECCC technical report version of this paper [21].

II. Preliminaries

In the following we shortly describe notations and concepts that are used in the present paper. For a detailed discussion, we refer to the textbook of Flum and Grohe [22].

A. Graphs and Trees

We consider undirected graphs as special cases of directed graphs, namely as directed graphs $G = (V,E)$ where $E \subseteq V \times V$ is symmetric. We write $V(G)$ for G's vertex set and $E(G)$ for the edge set. A *subgraph* of a graph G is a graph G' with $V(G') \subseteq V(G)$ and $E(G') \subseteq E(G)$. The *subgraph of G induced on a set $U \subseteq V(G)$* is the graph $G[U]$ with $V(G[U]) = U$ and $E(G[U]) = E(G) \cap (U \times U)$. The *children* of a vertex v in a directed graph G are all vertices u with $(v,u) \in E(G)$.

We consider trees as special directed graphs, but call their vertices *nodes*. In a *binary* tree, every node has zero or two children. In a *balanced* tree, all paths from the root to leafs have the same length. Note that balanced binary trees have logarithmic depth. A *labeled tree* is a tree T together with a mapping $l\colon V(T) \to \Sigma$, where Σ is the labeling alphabet.

B. Monadic Second-Order Logic

Monadic second-order logic (MSO-logic) is the fragment of second-order logic where all variables are either first-order variables x_1, x_2, ... (also called *element variables*) or unary second-order variables X_1, X_2, ... (also called *set variables*). The MSO-*formulas* over a vocabulary τ are inductively defined as follows: The *atomic formulas* are of the forms $x = y$, $X(z)$, $R(x_1,\ldots,x_r)$, where $x,y,z,x_1,\ldots,x_r$ are element variables, X is a set variable, and $R^r \in \tau$. Formulas are build from atomic formulas by *connectives* ($\neg$, $\wedge$, $\vee$, $\rightarrow$, $\leftrightarrow$), *element quantifiers* ($\exists x$, $\forall x$), and *set quantifiers* ($\exists X, \forall X$). *Bound* and *free* variables are defined as usual. We write $\phi(X_1,\ldots,X_d)$ for a formula ϕ with free variables $X_1,\ldots,X_d$. The *model relation* $\mathcal{A} \models \phi(S_1,\ldots,S_d)$ for a τ-structure $\mathcal{A}$ with universe A, a τ-formula $\phi(X_1,\ldots,X_d)$, and assignments of subsets $S_1,\ldots,S_d \subseteq A$ to the free variables $X_1,\ldots,X_d$ is defined as usual [22]. Since we study structures over arbitrary vocabularies τ, there is no need to distinguish between MSO_1- and MSO_2-logic [26].

C. Tree Decompositions

The concept of tree decompositions of graphs was introduced by Robertson and Seymour [38]; we use a generalized definition for logical structures [22]:

Definition II.1 (Tree Decomposition)**.** A *tree decomposition* of a τ-structure $\mathcal{A}$ is a labeled tree T whose labeling function $B\colon V(T) \to \{X \mid X \subseteq A\}$ has the following properties:

1) For all $a \in A$, the induced subgraph $T\big[\{n \in V(T) \mid a \in B(n)\}\big]$ is nonempty and connected.
2) For every $R^r \in \tau$ and every tuple $(a_1,\ldots,a_r) \in R^{\mathcal{A}}$, there is an $n \in V(T)$ with $\{a_1,\ldots,a_r\} \subseteq B(n)$.

The sets $B(n)$ are called *bags*. The *width* of a tree decomposition T is $\max_{n\in V(T)} |B(n)| - 1$. The *tree width* of a structure $\mathcal{A}$, denoted by $\mathrm{tw}(\mathcal{A})$, is the minimum width over all its tree decompositions. A class of τ-structures has *bounded tree width* if the tree width of all its elements is bounded by a constant.

We make no special assumptions concerning bags of adjacent nodes, like their symmetric difference containing exactly one vertex.

The *Gaifman graph* of a structure $\mathcal{A}$ is an undirected graph that describes how the elements of $\mathcal{A}$ are connected by relations: Its vertex set is A and there is an edge $(a,a') \in A \times A$ if, and only if, one of the relations $R^{\mathcal{A}}$ contains a tuple $(a_1,\dots,a_r) \in R^{\mathcal{A}}$ with $a,a' \in \{a_1,\dots,a_r\}$. Since a tuple of r elements from the structure gives rise to a clique of size r in the Gaifman graph and since in a tree decomposition every clique is completely contained in some bag, every tree decomposition of a structure $\mathcal{A}$ is also a tree decomposition of its Gaifman graph and vice versa.

D. Tree Automata

A (binary, bottom-up) *tree automaton* is a tuple $M = (Q, Q_a, q_0, \Sigma, \delta)$, where Q is the set of *states*, $Q_a \subseteq Q$ is the set of *accepting states*, q_0 is the *initial state*, Σ is the *alphabet*, and $\delta\colon Q \times Q \times \Sigma \to Q$ is the *state transition function*. A tree automaton inductively assigns a state to a labeled binary tree T with distinguished left and right children as follows: The empty tree has state q_0. The state of a nonempty tree with root r and root label $l(r)$, left subtree state q_{left}, and right subtree state q_{right} is $\delta(q_{\text{left}}, q_{\text{right}}, l(r))$. The tree automaton *accepts* all trees to which it assigns a state from Q_a. We only consider deterministic tree automata $M = (Q, Q_a, q_0, \Sigma, \delta)$ with alphabets $\Sigma = \{0,1\}^m$.

III. Computing Approximate Tree Decompositions in Logarithmic Space

Bodlaender's Theorem is typically proved in two steps: First, a linear-time algorithm is presented that on input of a graph of tree width at most k computes a tree decomposition of width at most $O(k)$, called an *approximate* tree decomposition. Second, another linear-time algorithm is used to turn the approximate tree decomposition into an optimal one. We proceed similarly: The present section is devoted to a proof of Lemma III.1 below, which states that approximate tree decompositions can be computed in logarithmic space. In Section V we will show how optimal tree decompositions can be computed.

Lemma III.1. *For every $k \geq 1$, there is a logspace* DTM *that on input of any structure $\mathcal{A}$ with $\mathrm{tw}(\mathcal{A}) \leq k$ outputs a tree decomposition for $\mathcal{A}$*

1) *whose width is at most $4k+3$ and*
2) *whose decomposition tree is binary and balanced.*

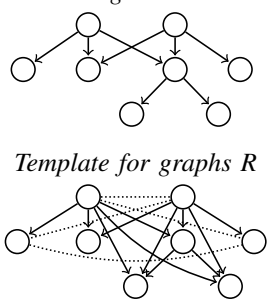
For many applications, it suffices to have access to tree decompositions satisfying part 1 of the lemma. However, for the proof of the logspace cardinality version of Courcelle's Theorem in Section IV we need access to tree decompositions of constant degree and logarithmic depth. Part 2 shows that such tree decompositions can be obtained in logarithmic space.

In the following, for the proof of Lemma III.1 we only consider *undirected, connected graphs* instead of arbitrary logical structures, because a structure and its Gaifman graph have the same tree decompositions [22] and because different components of a graph can be decomposed independently.

Algorithms for constructing tree decompositions often employ a specific notion of *separators*, which are used to split a graph into smaller subgraphs for which tree decompositions can be computed recursively. When one wants to transfer this idea to logarithmic space, one faces the problem that both the recursion stack and the intermediate subgraphs are too large to store. We overcome these problems in two ways: First, instead of avoiding deep recursions, we show how a special version of the reachability problem in mangrove graphs can be used to identify the desired tree decomposition. Second, we pick a notion of separators that allows us to represent subgraphs by descriptors that can be stored in logarithmic space.

A. Transitive Closures of Mangroves

In our tree decomposition algorithm the decomposition tree will be an induced subgraph of a larger graph in which it is "hidden." In order to recover the tree, we need to compute the set of vertices reachable from a given vertex. For the *mangrove graphs* that arise in our proofs the best upper space bound on the reachability problem is $O(\log^2 n / \log\log n)$, see [1], which is far from logarithmic. So, our algorithms will need access to some kind of additional information. This information will be in the form of what we call *transitive closures of related vertices*.



A *mangrove* [1] is a DAG in which there is at most one path between any two vertices. In a mangrove, the subgraph T_r induced on all vertices reachable from a given vertex r is a tree rooted at r. Let us say that two vertices a and b of a mangrove are *related* if they are both present in some T_r, that is, they are both reachable from some vertex r. We say that a graph R is *a transitive closure of the related vertices of M* if the following holds: Whenever a and b are related in M, then there is an edge from a to b in R if, and only if, there is a non-empty path from a to b in M.

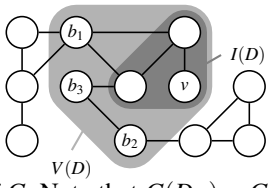
The example above shows a mangrove M at the top. All transitive closures R of M's related vertices can be obtained by arbitrarily adding edges in the lower "template" graph along the dotted lines, which connect exactly the unrelated vertices of M.

Lemma III.2. *There is a logspace* DTM *that on input of any pair (M,R) consisting of a mangrove M and a transitive closure R of M's related vertices outputs the transitive closure of M.*

Sketch of proof: To test whether b is reachable from a in M, set *current* to a and then repeatedly update *current* to the unique vertex v that is a child of *current* in M and for which there is an edge from v to b in R. If there is no unique such vertex, reject. When b becomes a child of *current*, accept. □

B. Descriptors and Descriptor Decompositions

Let $G=(V,E)$ be a connected undirected graph. A *descriptor D in G* is either (a) just a bag $B \subseteq V$ and called *simple* or (b) a pair (B,v) consisting of a bag $B \subseteq V$ and a *component selector* $v \in V-B$. We write $B(D)$ for the bag of D. We say that D *describes* the following graph $G(D)$: If D is simple, $G(D)=G[B]$. Otherwise let $G(D)=G\big[V(C)\cup B\big]$ where C is the component of $G[V-B]$ that contains v. We write $V(D)$ for the vertex set of the graph $G(D)$. The *interior* $I(D)$ of $G(D)$ is $V(D)-B(D)$. Let D_G denote the descriptor $(\emptyset, v_0)$ where v_0 is a fixed vertex of G. Note that $G(D_G)=G$ since we assumed G to be connected. An example of a graph G, a descriptor $D=(\{b_1,b_2,b_3\},v)$, and the sets $V(D)$ and $I(D)$ is shown above. The graph $G(D)$ is the induced subgraph $G[V(D)]$.

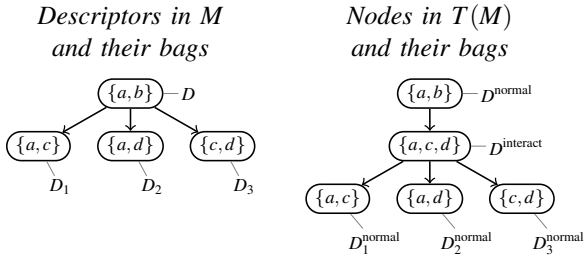


Definition III.3 (Descriptor Decomposition)**.** Let G be a connected undirected graph. A *descriptor decomposition* of G is a directed graph M whose vertices are descriptors in G, one of them is D_G, and where for every node D of M with children $D_1, \ldots, D_m$ the following holds:

1) For each child D_i, we have $V(D_i) \subseteq V(D)$ and $I(D_i) \subseteq I(D)$ and at least one inclusion is proper.
2) For each child D_i, the set $V(D_i)$ contains at least one vertex from $I(D)$.
3) For each child D_i, the set $I(D_i)$ is disjoint from all $V(D_j)$ for $j \neq i$.
4) Each edge in $G(D)$ that is not between two vertices in $B(D)$ must be present in some $G(D_i)$.

Observe that property 1 implies that all descriptor decompositions are DAGs.

We next prove that given a descriptor decomposition M of a graph G, the graph $M[W]$ where W is the set of all vertices reachable from D_G in M "nearly forms a tree decomposition of G": We only need to add an internal vertex between each node and its children whose bag contains all "interactions" between the children of the node. Formally, we define a graph $T(M)$ as follows: For each descriptor D reachable from D_G in M, it contains two vertices D^{normal} and D^{interact}. If $D_1, \ldots, D_m$ are the children of D in M, then there are edges from D^{normal} to D^{interact} and from D^{interact} to each D_i^{normal}. Label D^{normal} with the bag $B(D)$. Label D^{interact} with the bag that contains all vertices present in at least two of the sets in $\{B(D),B(D_1),\ldots,B(D_m)\}$. Below, we show an example of a descriptor D, its children D_1, D_2, and D_3 in M, and their bags; as well as the resulting nodes and bags in $T(M)$.



Lemma III.4. *If M is a descriptor decomposition of G, then $T(M)$ is a tree decomposition of G.*

Lemma III.5. *There is a logspace* DTM *that on input of any graph G together with a descriptor decomposition M of G outputs $T(M)$.*

Sketch of proofs: To show that $T(M)$ is a tree, one shows that M is a mangrove. Property 4 of descriptor compositions ensures that the bags of $T(M)$ cover all edges of G. Properties 1 and 3 ensure that each vertex v of G enters the set of bags of $T(M)$ containing v at a unique node. One then shows that the graph R, whose vertex set is $V(M)$ and where there is an edge from D to D' if, and only if, property 1 applies to them, is a transitive closure of M's related vertices. Then Lemma III.2 allows us to recover the tree $T(M)$ in logarithmic space. □

By Lemmas III.4 and III.5, in order to compute a tree decomposition of a graph, it suffices to compute a descriptor decomposition. We next show that such a descriptor decomposition can be obtained in logarithmic space by defining appropriate descriptors and their child descriptors.

As remarked earlier, algorithms for computing tree decompositions internally use different kinds of *separators*. The ones we use are also known as *balanced separators.*

Definition III.6. Let G be an undirected graph and let $U \subseteq V(G)$. A *separator $S \subseteq V(G)$ separates U in G* if each component of $G[V(G)-S]$ contains at most $|U|/2$ vertices of U. An *s-separator* is a separator of size at most s.

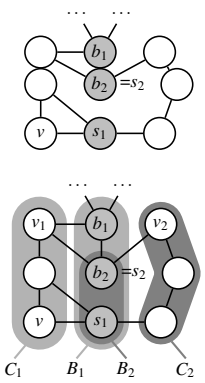
It is a folklore fact that for all G with $\mathrm{tw}(G) \leq k$ every $U \subseteq V(G)$ has a $(k+1)$-separator S in G.

Whenever we write that a DTM should "choose" some vertex or set, we mean that a deterministic choice is made.

For instance, we can always choose the lexicographically first vertex or set.

Definition III.7 (Child descriptors)**.** Let G be a connected undirected graph with $\mathrm{tw}(G) \leq k$. We define the *child descriptors* of a descriptor D in G as follows: If D is simple, it has no child descriptors. Otherwise we call D *small* if $|B(D)| \leq 2k+2$ and choose a $(k+1)$-separator S of $V(D)$ in the graph $G(D)$; and we call D *large* if $|B(D)| > 2k+2$ and choose a $(k+1)$-separator S of $B(D)$ in $G(D)$. Let C_1 to C_m be the components of $G[I(D)-S] = G\big[V(D)-(S\cup B(D))\big]$. For each $i \in \{1,\dots,m\}$ choose a vertex $v_i \in C_i$ and let B_i be the set of all vertices in $B(D)\cup S$ that are adjacent in $G(D)$ to a vertex from C_i. Then the descriptors (B_i, v_i) are the child descriptors of D and, unless $S \subseteq B(D)$, additionally the simple descriptor $D_0 = B(D)\cup S$.

To the right, we show how child descriptors are constructed. The first graph is $G(D)$ for the small descriptor $D = (\{b_1, b_2\}, v)$. The set $S = \{s_1, s_2\}$ is a separator of $V(D)$ in $G(D)$. Removing the highlighted set $B(D)\cup S$ yields the two components C_1 and C_2. The sets B_1 and B_2 contain the vertices from $B(D)\cup S$ adjacent to C_1 and C_2, respectively. The child descriptors of D are $D_1 = (\{b_1, b_2, s_1\}, v_1)$, $D_2 = (\{b_2, s_1\}, v_2)$, and $D_0 = \{b_1, b_2, s_1\}$.



Lemma III.8 (Size Lemma)**.** *Let D be a non-simple descriptor and let D' be a child descriptor of D.*

1) *If D' is simple, then $|B(D')| \leq |B(D)| + k + 1$.*
2) *If D is small, then $|V(D')| \leq |V(D)|/2 + 3k + 3$ and $|B(D')| \leq 3k+3$.*
3) *If D is large, then $|B(D')| < |B(D)|$.*

Sketch of proof: If D is small, then S separates $V(D)$. Thus, each component C_i can contain at most $|V(D)|/2$ vertices and $B(D)\cup S$ can contain at most $(2k+2)+(k+1)$ vertices. If D is large, then S separates $B(D)$. Then each B_i can contain at most $|B(D)|/2$ of the vertices in $B(D)$. Since $|S| \leq k+1$ and $|B(D)| > 2k+2$, we get $|S| < |B(D)|/2$ and, thus, $|B_i| < |B(D)|/2 + |B(D)|/2 = |B(D)|$. □

We are now ready to define the desired descriptor decomposition and to show that it is, indeed, a descriptor decomposition, has logarithmic depth, and is logspace-computable.

Definition III.9. Let G be an undirected, connected graph with $\mathrm{tw}(G) \leq k$. Let $M(G)$ be the graph whose vertex set contains all descriptors D in G with $|B(D)| \leq 3k+3$ for non-simple D and $|B(D)| \leq 4k+4$ for simple D and where there are edges from each descriptor exactly to its child descriptors.

Lemma III.10. *The graph $M(G)$ is a descriptor decomposition of G.*

Lemma III.11. *The tree decomposition $T(M(G))$ has width at most $4k+3$ and depth at most $c\log_2 n$, where c is a constant depending only on k and n is the number of vertices of G.*

Lemma III.12. *For every $k \geq 1$, there is a logspace* DTM *that on input of any graph G with $\mathrm{tw}(G) \leq k$ outputs $M(G)$.*

Sketch of proofs: It is straightforward to show that $M(G)$ satisfies the four properties of a descriptor decomposition. To see that the tree decomposition $T(M(G))$ has depth $O(\log n)$, observe that by the size lemma on any path from the root to a leaf either the bag size decreases or the size of the described graph halves. The graph $M(G)$ is logspace-computable from G because the separators S can be found by brute force. □

C. Computing Balanced Binary Tree Decompositions

In order to finish the proof of Lemma III.1, it remains to turn high-degree tree decompositions $T(M(G))$ into balanced binary tree decompositions.

Lemma III.13. *There is a logspace* DTM *that on input of any logarithmic depth tree decomposition of a graph G outputs a balanced binary tree decomposition of G of the same width.*

Sketch of proof: Replace every node of degree more than 2 by a binary tree and then fill up the tree so that it becomes balanced. However, if each node of high degree is naïvely replaced by a near-balanced binary tree, the height of the resulting tree will be $O(\log^2 n)$. The trick is to apply a more careful balancing operation where nodes with many children are placed higher in the tree. □

IV. Logspace Cardinality Version of Courcelle's Theorem

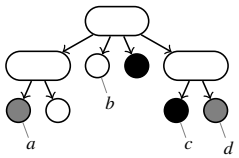
In this section we prove Theorem I.3, the logspace cardinality version of Courcelle's Theorem. We follow a classical method of proving Courcelle's Theorem: Starting with a tree decomposition of a structure $\mathcal{A}$ and a formula ϕ, we construct a labeled tree $\mathcal{T}$ and a formula ψ such that $\mathcal{A} \models \phi$ if, and only if, $\mathcal{T} \models \psi$. Using standard arguments, one can then construct a tree automaton M that decides whether $\mathcal{T} \models \psi$ holds. The main new problem is showing how runs of the automaton can be used to determine the desired solution histograms. This is done by constructing special arithmetic trees that we call *convolution trees* and by showing that they can be evaluated in logarithmic space.

An *s-tree structure* is a $\tau_{s\text{-tree}}$-structure $\mathcal{T} = (V, E^{\mathcal{T}}, P_1^{\mathcal{T}}, \dots, P_s^{\mathcal{T}})$ over the vocabulary $\tau_{s\text{-tree}} = \{E^2, P_1^1, \dots, P_s^1\}$ where $(V, E^{\mathcal{T}})$ is a tree. An s-tree structure is *binary* or *balanced*, if $(V, E^{\mathcal{T}})$ is binary or balanced, respectively.

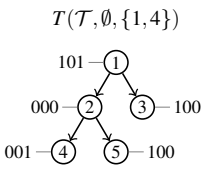
Lemma IV.1. *Let $k \geq 1$ and let $\phi(X_1, \dots, X_d)$ be an* MSO*-τ-formula. Then there are a number $s \geq 1$, an* MSO*-$\tau_{s\text{-tree}}$-formula $\psi(X_1, \dots, X_d)$, and a logspace* DTM *that on input*

of any τ-structure $\mathcal{A}$ with universe A and $\mathrm{tw}(\mathcal{A}) \leq k$ *outputs a balanced binary s-tree structure $\mathcal{T}$ such that for all indices $i \in \{0,\dots,|A|\}^d$ we have* $\mathrm{histogram}(\mathcal{A},\phi)[i] = \mathrm{histogram}(\mathcal{T},\psi)[i]$.

Sketch of proof: The proof is a variation of the one by Arnborg et al. [3]. We first construct a tree decomposition of width $k' = 4k+3$ using Lemma III.1 and turn it into an s-tree structure $\mathcal{T}$ as follows: We attach *element nodes* to each node of the tree decomposition, one for each element of the node's bag. Then we color the element nodes using up to $2k'+2$ colors so that if two elements a and b in $\mathcal{A}$'s universe are present in two adjacent bags, then the corresponding element nodes get the same color if, and only if, $a = b$. An example of this transformation for the tree decomposition $\{a,b\} \leftarrow \{b,c\} \rightarrow \{c,d\}$ is shown below right. Unary predicates on the element nodes are used to locally encode the relations of $\mathcal{A}$ at the bags. Another unary predicate singles out a set of *representative* element nodes that are in one-to-one correspondence to elements of $\mathcal{A}$'s universe. In the right example, representatives element nodes for a, b, c, and d are indicated. By allowing only these nodes to be used for the assignments to free and bound variables, we obtain a histogram-preserving reduction. □

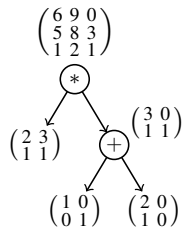


The next step is to establish a relationship between s-tree structures and tree automata. For this, given a binary s-tree structure $\mathcal{T} = (V, E^{\mathcal{T}}, P_1^{\mathcal{T}}, \dots, P_s^{\mathcal{T}})$ and sets $S_1,\dots,S_d \subseteq V$ let us write $T(\mathcal{T},S_1,\dots,S_d)$ for the tree $(V,E^{\mathcal{T}})$ where we label each node $v \in V$ with the bitstring $l_1 \dots l_s x_1 \dots x_d \in \{0,1\}^{s+d}$ with $l_i = 1 \iff v \in P_i^{\mathcal{T}}$ and $x_i = 1 \iff v \in S_i$. An example is shown right for the 1-tree structure $\mathcal{T}$ with $V = \{1,2,3,4,5\}$, $E^{\mathcal{T}} = \{(1,2),(1,3),(2,4),(2,5)\}$, $P_1^{\mathcal{T}} = \{1,3,5\}$, $S_1 = \emptyset$, and $S_2 = \{1,4\}$. We write $T(\mathcal{T})$ in case $d = 0$. A tree automaton can work on $T(\mathcal{T},S_1,\dots,S_d)$ if we choose a distinguished left child for each inner node. Different versions of the following fact are used in many versions of Courcelle's Theorem, the below formulation is implicitly shown in [3]. To prove it, one inductively constructs tree automata for subformulas and combines them to automata for composed formulas.



Fact IV.2. *For every $s \geq 0$ and every* MSO-$\tau_{s\text{-tree}}$*-formula $\phi(X_1,\dots,X_d)$, there is a tree automaton M with alphabet $\Sigma = \{0,1\}^{s+d}$ such that for all binary s-tree structures $\mathcal{T}$ and all subsets $S_1,\dots,S_d \subseteq V$ we have $\mathcal{T} \models \phi(S_1,\dots,S_d)$ if, and only if, M accepts $T(\mathcal{T},S_1,\dots,S_d)$.*

In view of Fact IV.2 and Lemma IV.1, we need to determine how many sets $S_1,\dots,S_d \subseteq V$ of certain sizes make M accept $T(\mathcal{T},S_1,\dots,S_d)$. If the formula ϕ has no free predicate variables and, hence, there are no sets S_i to choose, this problem can be solved by a simulation of the automaton M that traverses the tree $T(\mathcal{T})$ in a depth first recursion. Since $T(\mathcal{T})$ is balanced, binary, and for every node on the stack we only use a constant amount of bits that depends on M, this gives a logspace procedure. If ϕ has free predicate variables, we reduce the problem to evaluating *convolution trees.* In the following, an $[r]^d$*-array* is a d-dimensional array of non-negative integers where all indices $i = (i_1,\dots,i_d)$ are elements of the index set $[r]^d = \{0,\dots,r-1\}^d$. We call r the *range*. The *addition* of two arrays is defined component-wise in the obvious way. The *convolution* of an $[r]^d$-array A and an $[s]^d$-array B is the $[r+s-1]^d$-array $C = A * B$ defined by $C[i] = \sum_{j \in [r]^d, k \in [s]^d, j+k=i} A[j] \cdot B[k]$. A *convolution tree* T is a tree whose leafs are labeled by arrays of appropriate sizes and whose inner nodes are labeled by $+$ or $*$. Its *value* $\mathrm{val}(T)$ is the array resulting from recursively applying the operation in each node to the values of the child trees. In the example above the (2×2)-matrices at the leafs are $[2]^2$-arrays; position $(0,0)$ lies at the upper left corner.

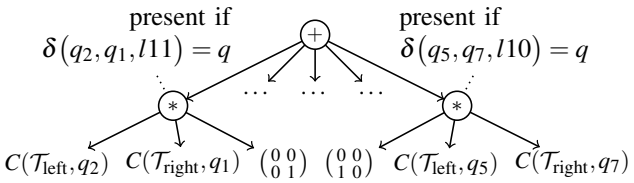


Lemma IV.3. *Let M be a tree automaton with alphabet $\{0,1\}^{s+d}$. Then there exists a logspace* DTM *that maps every binary balanced s-tree structure $\mathcal{T}$ to a convolution tree T where $\mathrm{val}(T)[i_1,\dots,i_d]$ is exactly the number of sets $S_1,\dots,S_d \subseteq V$ with $|S_j| = i_j$ for which M accepts $T(\mathcal{T},S_1,\dots,S_d)$.*

Sketch of proof: Let $\mathcal{T}$ be a binary balanced s-tree structure with node set V. To simplify the induction later on, we also allow the "empty" s-tree structure $\mathcal{T}$ with $V = \emptyset$. The first observation is that in order to count how many sets S_i of certain sizes make M accept $T(\mathcal{T},S_1,\dots,S_d)$, we can add the number of sets of these sizes that make M reach its different accepting states. Thus, we focus on computing how many sets S_i make M reach a particular state q. For this, we construct convolution trees $C(\mathcal{T},q)$ such that $\mathrm{val}\big(C(\mathcal{T},q)\big)[i_1,\dots,i_d]$ is exactly the number of sets $S_1,\dots,S_d \subseteq V$ with $|S_j| = i_j$ for which M reaches q on input $T(\mathcal{T},S_1,\dots,S_d)$.

The trees $C(\mathcal{T},q)$ are constructed inductively: If $\mathcal{T}$ is empty, then $C(\mathcal{T},q)$ is a single leaf whose attached array is a single entry, which we set to 1 for $q = q_0$ and to 0 otherwise. For $\mathcal{T}$ with a root r and left and right child trees $\mathcal{T}_{\text{left}}$ and $\mathcal{T}_{\text{right}}$, let V_{left} and V_{right} be their node sets, respectively, and let l be the label of r in $T(\mathcal{T})$. Then the root of the tree $C(\mathcal{T},q)$ is a $+$-node that has one child node for each triple $(q_{\text{left}},q_{\text{right}},x) \in Q \times Q \times \{0,1\}^d$ with $\delta(q_{\text{left}},q_{\text{right}},lx) = q$. Each child node is a $*$-node (a convolution node) that has three children: One child is $C(\mathcal{T}_{\text{left}},q_{\text{left}})$, one child is $C(\mathcal{T}_{\text{right}},q_{\text{right}})$, and one child is a leaf to which we attach

the $[2]^d$-array with binary entries that has a 1-entry exactly at index $(x_1,\ldots,x_d)$. For example, for a tree $\mathcal{T}$ with subtrees $\mathcal{T}_{\text{left}}$ and $\mathcal{T}_{\text{right}}$, the convolution tree looks like this:



One then shows that the construction is correct: The sets S_i that make M reach state q on input $\mathcal{T}$ can be partitioned according to the different triples $(q_{\text{left}},q_{\text{right}},x)$ with $\delta(q_{\text{left}},q_{\text{right}},lx)=q$. Thus, it is correct to add the values for the different triples that lead to q. Next, the number of sets S_i that make M reach state q via fixed q_{left}, q_{right} and x with, say, $|S_1|=15$ can be calculated as follows: If $|S_1\cap\{r\}|=0$, it equals the number of ways we can reach q_{left} on the left subtree with $|S_1\cap V_{\text{left}}|=0$ times the number of ways we can reach q_{right} on the right subtree with $|S_1\cap V_{\text{right}}|=15$, plus the number of ways with $|S_1\cap V_{\text{left}}|=1$ times the number of ways with $|S_1\cap V_{\text{right}}|=14$, and so on. If $|S_1\cap\{r\}|=1$, the sum of products is taken for the child sets whose sizes add up to 14 instead of 15. In both cases, the sum of products is exactly the convolution of the child values.

The tree T can be computed recursively starting from the root of a given binary balanced s-tree structure. For every node we only have to store a constant number of bits that depend on the transition function of M. Together with the fact that the input tree has logarithmic depth, the overall stack space is bounded by a function of M times the logarithm of the input size. □

All that remains to be done is to evaluate convolution trees in logarithmic space. For this we introduce a bit of terminology. Let N and R be two fixed bases, both to be chosen later. Given an $[r]^d$-array A, let

$$\text{num}(A)=\sum_{i\in[r]^d}A[i]N^{i_1+i_2R+\cdots+i_dR^{d-1}}.$$

Trivially, $\text{num}(A+B)=\text{num}(A)+\text{num}(B)$. For an $[r]^d$-array A and an $[s]^d$-array B we have $\text{num}(A*B)=\text{num}(A)\cdot\text{num}(B)$, because equation (1) holds.

Lemma IV.4. *There exists a logspace* DTM *that on input of any convolution tree outputs its value.*

Sketch of proof: The machine first determines two numbers N and R as follows: N is chosen to be 2^p where p is upper-bounded by a polynomial in the length of the input and R is the sum of all ranges of all leaf arrays. Using structural induction, one then shows that these numbers are large enough such that for all $i\in[r]^d$ the integer $\text{val}(T)[i_1,\ldots,i_d]$ can be found at the p bits prior to position $i_1+i_2R+\cdots+i_dR^{d-1}$ from the right (least-significant) end of the binary representation of $\text{num}(\text{val}(T))$. As an example, for $N=2^4$ and $R=3$, the entry 3 at index $(0,1)$ of the $[2]^2$-array $A=\left(\begin{smallmatrix}1&3\\2&4\end{smallmatrix}\right)$ can be found at the underlined bits of

$$\begin{aligned}\text{num}(A)&=1N^{0\cdot R^0+0\cdot R^1}+2N^{1\cdot R^0+0\cdot R^1}+\\&\quad 3N^{0\cdot R^0+1\cdot R^1}+4N^{1\cdot R^0+1\cdot R^1}\\&=274465=0100\underline{0011}0000010000\underline{1}_2.\end{aligned}$$

The machine maps the convolution tree T to an arithmetic tree T' whose leafs are labeled with ordinary integers by replacing every leaf array A by $\text{num}(A)$ and every convolution node by a multiplication node. Then, since $\text{num}(A+B)=\text{num}(A)+\text{num}(B)$ and $\text{num}(A*B)=\text{num}(A)\cdot\text{num}(B)$, we immediately get that the value of the resulting ordinary arithmetic $\{+,\times\}$-tree is $\text{num}(\text{val}(T))$. Since evaluating $\{+,\times\}$-trees can be done in logarithmic space [5], [12], [14], [25], we get the claim. □

V. Logspace Version of Bodlaender's Theorem

In this last section we prove Theorem I.1, the logspace version of Bodlaender's Theorem. For the proof, we first need to show that TREE-WIDTH-$k\in L$ holds for all k. Then, we reapply the ideas from Section III, only we make sure that in the constructed descriptor decomposition M the implicit tree decomposition $T(M)$ has width k. The first step toward proving Theorem I.1 is thus proving Lemma I.4.

Sketch of proof of Lemma I.4: To show that TREE-WIDTH-$k\in$ L holds, on input G we first apply Lemma III.1 and test whether the result is a tree decomposition of width $4k+3$. If not, we reject. Otherwise, observe that for every k the graph property "the graph has tree width k" can be characterized constructively by a finite set of forbidden minors [33] and that the question of whether a graph contains

$$\begin{aligned}\text{num}(A)\cdot\text{num}(B)&=\left(\sum_{j\in[r]^d}A[j]N^{j_1+j_2R+\cdots+j_dR^{d-1}}\right)\left(\sum_{k\in[s]^d}B[k]N^{k_1+k_2R+\cdots+k_dR^{d-1}}\right)\\&=\sum_{j\in[r]^d,k\in[s]^d}A[j]B[k]N^{(j_1+k_1)+(j_2+k_2)R+\cdots+(j_d+k_d)R^{d-1}}\\&=\sum_{i\in[r+s-1]^d}\left(\sum_{j\in[r]^d,k\in[s]^d,j+k=i}A[j]B[k]\right)N^{i_1+i_2R+\cdots+i_dR^{d-1}}=\text{num}(A*B)\end{aligned}\tag{1}$$

a given minor is expressible using an MSO-formula ϕ [3]. Use Theorem I.2 to test whether $G \models \phi$. Since the algorithm of Theorem I.2 internally accesses the tree decomposition produced by Lemma III.1, the result will be correct. For hardness, reduce ACYCLICITY to TREE-WIDTH-k by replacing each vertex with a size-k clique and each edge by a set of edges that mimic the construction of a k-path between the "incident cliques". □

Sketch of proof of Theorem I.1: We reuse the machinery developed in Section III for the construction of approximate tree decompositions. By Lemma III.5 it suffices to construct a descriptor decomposition M of the input graph G in logarithmic space. The idea is that for a given descriptor D we can use Lemma I.4 to tell us which child descriptors should be picked in order to allow tree decompositions of width k for the smaller child graphs and, consequently, a tree decomposition of width k for $G(D)$. In detail, given a descriptor D we search for a bag $B' \subseteq V(D)$ with the following properties:

1) $|B'| = k+1$ and $B' \cap I(D) \neq \emptyset$.
2) There is no edge between a vertex in $B(D) - B'$ and a vertex in $I(D)$.
3) Let $C_1, \ldots, C_m$ be the components of the graph $G[I(D) - B']$. Then for each $i \in \{1, \ldots, m\}$ the graph $G[V(C_i) \cup B'] \cup K_{B'}$ must have tree width at most k. Here, $K_{B'}$ is the clique on B'.

One now shows that such a B' always exists. To find it in logarithmic space, a machine can iterate over all possible B' and each time invoke Lemma I.4 on $G[V(C_i) \cup B'] \cup K_{B'}$. Once B' is found, for each $i \in \{1, \ldots, m\}$ choose v_i in C_i and let B' and all (B', v_i) be the child descriptors of D. □

VI. Conclusion

Like the classical theorems of Bodlaender and of Courcelle, their logspace versions are useful tools in classifying the complexity of problems on graphs of bounded tree width. We have sketched a number of applications; indeed our own proof of the logspace version of Bodlaender's Theorem makes heavy use of the logspace version of Courcelle's Theorem. We are confident that applications beyond the ones indicated will be found.

There are three intriguing open problems that we would like to point out:

First, can we generalize the logspace algorithms for UNARY-SUBSETSUM and UNARY-KNAPSACK to solve unary variants of other, more general, pseudopolynomial problems?

Second, what is the complexity of the graph isomorphism problem on graphs of bounded tree width? Researches have steadily lowered the complexity bound from P [7] to TC^1 [24] and to LOGCFL [20]. While one bottleneck in the latest paper [20], namely the construction of tree decompositions in logarithmic space, is removed by the present paper, it is still unclear whether the complexity can be lowered to L. One promising approach may be to first study the graph automorphism problem.

Third, can one devise logspace algorithms for more general width parameters [26]? One example is *clique width,* whose defining decompositions, the *k-expressions*, can be approximated in polynomial time [35].

References

[1] E. Allender and K.-J. Lange, "RUSPACE($\log n$) $\subseteq$ DSPACE($\log^2 n / \log\log n$)," *Theory of Computing Systems*, vol. 31, no. 5, pp. 539–550, 1998.

[2] S. Arnborg, D. G. Corneil, and A. Proskurowski, "Complexity of finding embeddings in a k-tree," *SIAM Journal on Algebraic and Discrete Methods*, vol. 8, no. 2, pp. 277–284, 1987.

[3] S. Arnborg, J. Lagergren, and D. Seese, "Easy problems for tree-decomposable graphs," *Journal of Algorithms*, vol. 12, no. 2, pp. 308–340, Jun. 1991.

[4] V. Arvind, B. Das, and J. Köbler, "The space complexity of k-tree isomorphism," in *Proceedings of the 18th International Symposium on Algorithms and Computation (ISAAC 2007)*, ser. LNCS, vol. 4835. Springer, 2007, pp. 822–833.

[5] M. Ben-Or and R. Cleve, "Computing algebraic formulas using a constant number of registers," *SIAM Journal on Computing*, vol. 21, no. 1, pp. 54–58, 1992.

[6] H. L. Bodlaender, "NC-algorithms for graphs with small treewidth," in *Proceedings of the 14th International Workshop on Graph-Theoretic Concepts in Computer Science (WG 1988)*, ser. LNCS, vol. 344. Springer, 1989, pp. 1–10.

[7] ——, "Polynomial algorithms for graph isomorphism and chromatic index on partial k-trees," *Journal of Algorithms*, vol. 11, no. 4, pp. 631–643, Dec. 1990.

[8] ——, "A linear-time algorithm for finding tree-decompositions of small treewidth," *SIAM Journal on Computing*, vol. 25, no. 6, pp. 1305–1317, 1996.

[9] H. L. Bodlaender and T. Hagerup, "Parallel algorithms with optimal speedup for bounded treewidth," *SIAM Journal on Computing*, vol. 27, no. 6, pp. 1725–1746, 1998.

[10] H. L. Bodlaender and A. M. C. A. Koster, "Combinatorial optimization on graphs of bounded treewidth," *The Computer Journal*, vol. 51, no. 3, pp. 255–269, 2008.

[11] R. B. Borie, R. G. Parker, and C. A. Tovey, "Automatic generation of linear-time algorithms from predicate calculus descriptions of problems on recursively constructed graph families," *Algorithmica*, vol. 7, no. 1–6, pp. 555–581, Jun. 1992.

[12] S. Buss, S. Cook, A. Gupta, and V. Ramachandran, "An optimal parallel algorithm for formula evaluation," *SIAM Journal on Computing*, vol. 21, no. 4, pp. 755–780, 1992.

[13] N. Chandrasekharan and S. T. Hedetniemi, "Fast parallel algorithms for tree decomposing and parsing partial k-trees," in *Proceedings of the 26th Annual Allerton Conference on Communication, Control, and Computing*, 1988, pp. 283–292.

[14] A. Chiu, G. Davida, and B. Litow, "Division in logspace-uniform NC^1," *Theoretical Informatics and Applications*, vol. 35, no. 3, pp. 259–275, May–June 2001.

[15] S. Cho and D. T. Huynh, "On a complexity hierarchy between L and NL," *Information Processing Letters*, vol. 29, no. 4, pp. 177–182, Nov. 1988.

[16] S. A. Cook, "A taxonomy of problems with fast parallel algorithms," *Information and Control*, vol. 64, no. 1–3, pp. 2–22, Jan.–Mar. 1985.

[17] B. Courcelle and M. Mosbah, "Monadic second-order evaluations on tree-decomposable graphs," *Theoretical Computer Science*, vol. 109, no. 1–2, pp. 49–82, Mar. 1993.

[18] B. Courcelle, "Graph rewriting: An algebraic and logic approach," in *Handbook of Theoretical Computer Science, Volume B: Formal Models and Semantics*, J. van Leeuwen, Ed. Elsevier and MIT Press, 1990, pp. 193–242.

[19] B. Das, S. Datta, and P. Nimbhorkar, "Log-space algorithms for paths and matchings in k-trees," in *Proceedings of the 27th International Symposium on Theoretical Aspects of Computer Science (STACS 2010)*, ser. LIPIcs, vol. 5. Schloss Dagstuhl LZI, 2010, pp. 215–226.

[20] B. Das, J. Torán, and F. Wagner, "Restricted space algorithms for isomorphism on bounded treewidth graphs," in *Proceedings of the 27th International Symposium on Theoretical Aspects of Computer Science (STACS 2010)*, ser. LIPIcs, vol. 5. Schloss Dagstuhl LZI, 2010, pp. 227–238.

[21] M. Elberfeld, A. Jakoby, and T. Tantau, "Logspace versions of the theorems of bodlaender and courcelle," Electronic Colloquium on Computational Complexity, Tech. Rep. ECCC-TR10-062, 2010.

[22] J. Flum and M. Grohe, *Parameterized Complexity Theory*. Springer, 2006.

[23] G. Gottlob, N. Leone, and F. Scarcello, "Computing LOGCFL certificates," *Theoretical Computer Science*, vol. 270, no. 1–2, pp. 761–777, Jan. 2002.

[24] M. Grohe and O. Verbitsky, "Testing graph isomorphism in parallel by playing a game," in *Proceedings of the 33rd International Colloquium on Automata, Languages and Programming (ICALP 2006)*, ser. LNCS, vol. 4051. Springer, 2006, pp. 3–14.

[25] W. Hesse, E. Allender, and D. A. Mix Barrington, "Uniform constant-depth threshold circuits for division and iterated multiplication," *Journal of Computer and System Sciences*, vol. 65, no. 4, pp. 695–716, Dec. 2002.

[26] P. Hliněný, S.-i. Oum, D. Seese, and G. Gottlob, "Width parameters beyond tree-width and their applications," *The Computer Journal*, vol. 51, no. 3, pp. 326–362, 2008.

[27] O. H. Ibarra, T. Jiang, B. Ravikumar, and J. H. Chang, "On some languages in NC," in *VLSI Algorithms and Architectures, Proceedings of the 3rd Aegean Workshop on Computing*, ser. LNCS. Springer, 1988, vol. 319, pp. 64–73.

[28] A. Jakoby and M. Liśkiewicz, "Paths problems in symmetric logarithmic space," in *Proceedings of the 29th International Colloquium on Automata, Languages and Programming (ICALP 2002)*, ser. LNCS, vol. 2380. Springer, 2002, pp. 269–280.

[29] A. Jakoby, M. Liśkiewicz, and R. Reischuk, "Space efficient algorithms for series-parallel graphs," *Journal of Algorithms*, vol. 60, no. 2, pp. 85–114, Aug. 2006.

[30] A. Jakoby and T. Tantau, "Logspace algorithms for computing shortest and longest paths in series-parallel graphs," in *Proceedings of the 27th Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2007)*, ser. LNCS. Springer, 2007, vol. 4855, pp. 216–227.

[31] B. Jenner, "Knapsack problems for NL," *Information Processing Letters*, vol. 54, no. 3, pp. 169–174, May 1995.

[32] J. Lagergren, "Efficient parallel algorithms for tree-decomposition and related problems," in *Proceedings of the 31st Annual IEEE Symposium on Foundations of Computer Science (FOCS 1990)*. IEEE Computer Society, 1990, pp. 173–182 vol.1.

[33] J. Lagergren and S. Arnborg, "Finding minimal forbidden minors using a finite congruence," in *Proceedings of the 18th International Conference on Automata, Languages and Programming (ICALP 1991)*, ser. LNCS. Springer, 1991, vol. 510, pp. 532–543.

[34] B. Monien, "On a subclass of pseudopolynomial problems," in *Proceedings of the 9th Symposium on Mathematical Foundations of Computer Science 1980 (MFCS 1980)*, ser. LNCS, vol. 88, 1980, pp. 414–425.

[35] S.-i. Oum and P. Seymour, "Approximating clique-width and branch-width," *Journal of Combinatorial Theory, Series B*, vol. 96, no. 4, pp. 514–528, Jul. 2006.

[36] B. A. Reed, "Finding approximate separators and computing tree width quickly," in *Proceedings of the 24th Annual ACM Symposium on Theory of Computing (STOC 1992)*. ACM, 1992, pp. 221–228.

[37] O. Reingold, "Undirected connectivity in log-space," *Journal of the ACM*, vol. 55, no. 4, pp. 1–24, Sep. 2008.

[38] N. Robertson and P. D. Seymour, "Graph minors. II. Algorithmic aspects of tree-width," *Journal of Algorithms*, vol. 7, no. 3, pp. 309–322, Sep. 1986.

[39] E. Wanke, "Bounded tree-width and LOGCFL," *Journal of Algorithms*, vol. 16, no. 3, pp. 470–491, May 1994.