Theoretical Computer Science 982 (2024) 114251

Contents lists available at ScienceDirect

Theoretical Computer Science

journal homepage: www.elsevier.com/locate/tcs

Space efficient algorithm for solving reachability using tree decomposition and separators ☆

Rahul Jain [a,*], Raghunath Tewari [b]

[a] Fernuniversität in Hagen, Germany
[b] Indian Institute of Technology Kanpur, India

ARTICLE INFO

Communicated by R. Klasing

Keywords:
Graph reachability
Simultaneous time-space upper bound
Tree decomposition

ABSTRACT

To solve reachability is to determine whether there is a path from one vertex to the other in a graph. Standard graph traversal algorithms such as DFS and BFS take linear time to solve reachability; however, their space complexity is also linear. On the other hand, Savitch's algorithm takes quasipolynomial time, although the space-bound is $O(\log^2 n)$. In this paper, we study space-efficient algorithms for deciding reachability that runs in polynomial time.

We show a polynomial-time algorithm that solves reachability in directed graphs using $O(w \log n)$ space. Our algorithm requires access to a tree decomposition of width w for the underlying undirected graph of the input. This requirement can be waived for graphs for which recursive balanced vertex separators can be computed space-efficiently.

1. Introduction

Suppose we have a graph G and two vertices u and v in G; the reachability problem is the problem of determining if there exists a path from u to v in G. This problem is NL-complete for directed graphs and Ł-complete for undirected graphs [1]. Hence its study gives essential insight into space-bounded computations. The famous open question $\text{Ł} \stackrel{?}{=} \text{NL}$ essentially asks if there is a deterministic logspace algorithm for reachability in directed graphs or not. Reachability can be solved in linear space and time using standard graph traversal algorithms such as DFS and BFS. We also know, due to Savitch, that we can solve it in $\Theta(\log^2 n)$ space [2]. However, Savitch's algorithm requires $n^{\Theta(\log n)}$ time. Wigderson surveyed reachability problems in which he asked if there is an algorithm for reachability that runs simultaneously in $O(n^{1-\epsilon})$ space (for any $\epsilon > 0$) and polynomial-time [3]. Here, we make some partial progress toward answering this question for directed graphs.

In 1998 Barnes et al. made progress in answering Wigderson's question for general graphs by presenting an algorithm for reachability that runs simultaneously in $n/2^{\Theta(\sqrt{\log n})}$ space and polynomial time [4]. For several other topologically restricted classes of graphs, there has been significant progress in giving polynomial-time algorithms for reachability that run simultaneously in sublinear space. For grid graphs a space-bound of $O(n^{1/2+\epsilon})$ was first achieved [5]. The same space-bound was then extended to all planar graphs by Imai et al. [6]. Later for planar graphs, the space-bound was improved to $\tilde{O}(n^{1/2})$ space by Asano et al. [7]. For graphs of higher genus, Chakraborty et al. gave an $\tilde{O}(n^{2/3}g^{1/3})$ space algorithm, which additionally requires, as an input, an embedding of the graph on a surface of genus g [8]. They also gave an $\tilde{O}(n^{2/3})$ space algorithm for H minor-free graphs which requires tree

☆ This article belongs to Section A: Algorithms, automata, complexity and games, Edited by Paul Spirakis.

* Corresponding author.
E-mail addresses: rahul.jain@fernuni-hagen.de (R. Jain), rtewari@cse.iitk.ac.in (R. Tewari).

https://doi.org/10.1016/j.tcs.2023.114251
Received 7 November 2021; Received in revised form 21 July 2023; Accepted 9 October 2023

Available online 18 October 2023

decomposition of the graph as an input and $O(n^{1/2+\epsilon})$ space algorithm for $K_{3,3}$-free and K_5-free graphs. For layered planar graphs, Chakraborty and Tewari showed that for every $\epsilon > 0$, there is an $O(n^{\epsilon})$ space algorithm [9].

Treewidth is a well-studied property of graphs. The value of treewidth can range from 1, for a tree, to $n-1$, for a complete graph on n vertices. The computational complexity of many difficult problems becomes easy for bounded treewidth graphs. We can solve classical problems such as the Hamiltonian circuit, vertex cover, Steiner tree, and vertex coloring in linear time for bounded treewidth graphs [10]. We can solve the weighted independent set problem in $O(2^w n)$ time [11]. It is NP-complete to find on given input $\langle G, k\rangle$ if G has treewidth k [12]. However, an $O(\sqrt{\log n})$-factor approximation algorithm is known [13].

When solving reachability in a directed graph, we consider the treewidth of its underlying undirected graph. Series-parallel graphs are equivalent to graphs of treewidth 2. For them, Jakoby and Tantau showed a logspace algorithm for reachability [14]. Das et al. extended the logspace bound to graphs when the input contains a tree decomposition of constant width [15]. Elberfeld et al. showed a logspace algorithm for any monadic second-order property of a logical structure of bounded treewidth [16].

1.1. Our result

Here, we present a polynomial-time algorithm with improved space-bound for deciding reachability in directed graphs using its tree decomposition. In particular, we show the following result.

Theorem 1. *Given a directed graph G, a tree decomposition $\mathcal{T}$ of G of width w, and two vertices u and v in G, there is an $O(w \log n)$ space and polynomial-time algorithm that decides if there is a path from u to v in G.*

Das et al. presented a logspace algorithm to solve reachability in graphs given a tree decomposition of constant width as input [15]. A simple analysis would show that their algorithm requires $\Omega(w^2 \log n)$ space and $n^{\Omega(w \log w)}$ time.

A constant-size MSO formula can express the graph reachability problem. Hence the result by Elberfeld et al. [16] solves a more general problem and implies a logspace algorithm for reachability in bounded treewidth graphs. However, the space required by their algorithm is $\Omega(p(w) \log n)$ and time required is $\Omega(n^{q(w)})$ where p and q are super-linear polynomials.

It is worth noting that both the algorithms of Elberfeld et al. [16] and Das et al. [15] cease to run in polynomial time if the treewidth w of the input graph is not a constant. These include a large number of exciting classes of graphs mentioned in the introduction. Our algorithm requires $O(w \log n)$ space and $O(\mathsf{poly}(n, w))$ time. Thus, it has a better time and space complexity for solving the reachability problem when compared to the results of [16] and [15].

There is a connection between the notion of treewidth and the notion of vertex separators in a graph. We know that if a graph has treewidth w, then the graph contains vertex separators of size $w+1$. To prove Theorem 1, we proceed in the following way:

- We first show that we can construct vertex separators of size $w+1$ of the input graph in $O(w \log n)$ space and polynomial time using the input tree decomposition.
- Using the algorithm for vertex separator as a subroutine, we construct a new binary balanced tree decomposition of G having $O(w)$ width and logarithmic depth.
- We use the new tree decomposition to solve the reachability problem.

To solve the reachability problem in the last step, we use the universal sequences of Asano et al. [7] to determine an appropriate order to process the vertices of the input graph.

Note that we use the input tree decomposition only to compute vertex separators in the graph. There are several classes of graphs for which we can recursively find vertex separators of size $O(w)$ in polynomial-time using $O(w \log n)$ space. For such classes of graphs, the method presented here gives a slightly stronger result. We can waive the requirement of additional tree decomposition in the input for them and still get similar space and time complexity. We state this more formally in Theorem 2.

Theorem 2. *Let $\mathcal{G}$ be a class of graphs, and w be a parameter such that there exists an $O(w \log n)$ space and polynomial-time algorithm that given a graph G of $\mathcal{G}$ and a set U of $V(G)$, outputs a separator of U in G of size w. There exists a polynomial-time algorithm to decide reachability in $\mathcal{G}$ that uses $O(w \log n)$ space.*

1.2. Consequences of our result

Our result has been used to solve the reachability problem in polynomial time with better space complexity for various classes of graphs without the requirement of tree decomposition in input. For some classes, like constant-genus graphs and chordal graphs, the requirement of tree decomposition was waived completely. For other classes, like surface-embedded graphs and intersection graphs of Jordan regions, the requirement of tree decomposition was changed to a weaker requirement.

The intersection graph of a set of n Jordan regions with m intersection points has a separator of size $O(m^{1/2})$. Given suitably-encoded Jordan regions as input, it was shown that the separator of its intersection graph could be found in $O(m^{1/2} \log n)$ space and polynomial time [17]. This separator was used with our algorithm to obtain a polynomial-time algorithm to solve the reachability problem in intersection graphs of Jordan regions using $O(m^{1/2} \log n)$ space.

Graphs that have genus g have treewidth $O((gn)^{1/2})$. It was shown that given a combinatorial embedding of such a graph, its separator could be found in $O((gn)^{1/2} \log n)$ space. This separator algorithm was combined with our algorithm to give a $O((gn)^{1/2} \log n)$

space and polynomial-time algorithm for reachability [18]. The requirement for the combinatorial embedding as an input can be waived for a constant genus graph, as it can be constructed in logspace [19]. Thus, reachability in constant genus graphs can be solved in $O(n^{1/2}\log n)$ space and polynomial time.

A chordal graph on n vertices with m edges has a separator of size $O(m^{1/2})$ [20]. It was shown that there exists an $O(m^{1/2}\log n)$ space and a polynomial-time algorithm for constructing it [17]. This separator was used with our algorithm to obtain a polynomial-time algorithm to solve the reachability problem in chordal graphs using $O(m^{1/2}\log n)$ space *without* the input tree decomposition.

Izumi and Otachi [21, Lemma 13] presented a polynomial time algorithm that can compute separator of size $O(wn^{1/2})$ using $O(wn^{1/2}\log n)$-space for a graph of treewidth w. We can combine our result with their algorithm to solve reachability using $O(wn^{1/2}\log n)$-space and polynomial time without the input tree decomposition.

Izumi and Otachi [21] also use our reachability algorithm to show construction of a lexicographic Defth First Search sequence of vertices in a directed graph. Their construction requires $n^{O(1/\epsilon)}$-time and $O(\epsilon^{-1}wn^{\epsilon}\log n)$-space

1.3. Organization of the paper

In Section 2, we give the definitions, notations and previously known results that we use in this paper. In Section 3, we show how to efficiently compute a logarithmic depth binary tree decomposition of G, which has a similar width to the input tree decomposition. In Section 4, we give the reachability algorithm and prove its correctness and complexity bounds.

2. Preliminaries

For a graph G on n vertices, we denote its vertex and edge sets as $V(G)$ and $E(G)$, respectively. Let W be a subset of $V(G)$. We denote the subgraph of G induced by the vertices in W by $G[W]$. Let $[n]$ denote the set $\{1,2,\ldots,n\}$ for $n \geq 1$. We assume that the vertices of an n vertex graph are indexed by integers from 1 to n.

In a graph G, a path of length k is a sequence of edges $e_1, e_2, \ldots, e_k$ such that there exist distinct vertices $v_1, v_2, \ldots, v_{k+1}$ such that $e_i = (v_i, v_{i+1})$ for all i in $[k]$. The vertices v_1 and v_{k+1} are the *endpoints* of this path.

We next define the terminology and notations related to tree decomposition used in this paper. For tree decomposition, we will treat the graph as an undirected graph by ignoring the direction of its edges.

For a graph G, a *tree decomposition* is a tuple $\mathcal{T} = (T, \mathcal{B})$ where T is a tree and $\mathcal{B}$ is a collection of subsets of $V(G)$. Every node t of the tree T is assigned to an element B_t of $\mathcal{B}$ such that the following conditions hold: (i) $\bigcup_{t \in V(T)} B_t = V(G)$, (ii) for every edge $\{v, w\}$ in $E(G)$, there exists t in $V(T)$ such that v and w are in B_t, and (iii) if t_3 is on the path from t_1 to t_2 in T, then $B_{t_1} \cap B_{t_2} \subseteq B_{t_3}$. The *width* of a tree decomposition T is $\max_{t \in V(T)}(|B(t)| - 1)$. Finally, the *treewidth* of a graph G is the minimum width over all tree decompositions of G. We refer to an element t of $V(T)$ as a *node* and the set B_t to be the *bag* corresponding to t. For clarity, we will use the notation $B(t)$ instead of B_t in the rest of this article.

We assume that in a *binary tree*, every node has zero or two children. Moreover, in a *balanced tree*, all paths from the root to the leaves have the same length.

The next tool that we would be using is that of separators in graphs. Let λ be a non-negative weight function on the vertices of G. For a set of vertices U, we define $\lambda(U) = \sum_{v \in U} \lambda(v)$. Let $\alpha \in (0, 1)$ be a constant. We say that a set S of vertices of G is an α-balanced separator in G if for every connected component C of $G \setminus S$, we have $\lambda(C) \leq \alpha \cdot \lambda(V(G))$.

Let G be a graph and U be a subset of $V(G)$. Consider a weight function which assigns the weight 1 to all the vertices in U and 0 to all other vertices of G. A $\frac{1}{2}$-balanced separator of G under such a weight assignment will be called a separator of U in G.

We state in Lemma 1 the commonly known result about vertex separators. For its proof, see Cygan et al. [22, Lemma 7.19]

Lemma 1. *Let G be a graph, $(T, \mathcal{B})$ be a tree decomposition of G and λ be a non-negative weight function on the vertices of G. There exists a node t of T such that the bag $B(t)$ is $\frac{1}{2}$-balanced separator in G.*

We use a multi-tape Turing machine model to discuss the space-bounded polynomial-time algorithms. A multi-tape Turing machine consists of a read-only input tape, a write-only output tape, and a work tape. We measure the space complexity of a multi-tape Turing machine by the total number of bits used in the work tape.

If we compose two polynomial-time algorithms A_1 and A_2, requiring space $S_1(n)$ and $S_2(n)$ respectively, such that the output of A_1 is used as an input to A_2, then the total space used in composing A_1 and A_2 is $S(n) = O(S_1(n) + S_2(n))$. To see this note that whenever A_2 queries for an input bit, we simulate A_1 until it yields the desired bit, and we then resume the simulation of A_2. The total time would remain polynomial as it would be a product of two polynomials.

3. Finding a tree decomposition of small depth

Consider a path from the root of the tree decomposition to an arbitrary leaf. Our main algorithm for reachability (Algorithm 4) might potentially store *reachability information* for all vertices corresponding to the bags of tree nodes in this path. We thus require a tree decomposition with a small depth. This section shows how to compute a binary balanced tree decomposition (say $\mathcal{T}'$) with logarithmic depth and width $O(w)$. Once the depth is reduced to $O(\log n)$ with bag size being $O(w)$, the algorithm will only need to store reachability information of $O(w \log n)$ vertices.

Thus, we prove the following Theorem.

Theorem 3. *Given as input $\langle G, \mathcal{T} \rangle$ where G is a graph and $\mathcal{T}$ is a tree decomposition of G with width w, there exists an algorithm working simultaneously in $O(w \log n)$ space and polynomial time which outputs a binary tree decomposition $\mathcal{T}'$ of G which has width $6w+5$ and depth $O(\log n)$.*

Various algorithms for transforming the input tree decomposition to a binary tree decomposition of small depth are known. Bodlaender and Hagerup [23] describes how to solve this problem on EREW PRAM model using $O(\log n)$ time, $O(n)$ operations and $O(n)$ space. However, they do not investigate how the complexity of their algorithm is parameterized on w. EREW PRAM can be characterized in terms of unambiguous unbounded fan-in, bounded fan-out uniform circuit families [24]. It is not known if such a circuit family of $O(\log n)$ depth can be simulated in logspace. Chatterjee et al. [25] present an algorithm to compute such a tree decomposition in $O(b)$ time, where b is the number of bags in the tree decomposition. Their algorithm requires $O(b)$ space as well. Since the number of bags can be $O(n)$, the space required by their algorithm is quite large. Other algorithms [26,27] do not use an input tree decomposition and try computing a tree decomposition from scratch. It is NP-complete to compute the treewidth of a graph, and these algorithms either do not work in polynomial-time for non-constant treewidth or use a heuristic. To the best of our knowledge, no previous result gives a polynomial-time algorithm which runs space efficiently for classes of graphs with non-constant treewidth.

We will now develop a framework that will help us to prove Theorem 3. We first show how to compute a $\frac{1}{2}$-balanced separator of G under any weight assignment in polynomial time and $O(w \log n)$ space using its tree decomposition $\mathcal{T}$. We cycle through every node in the tree T and store the set of vertices in $B(t)$. Doing this requires $O(w \log n)$ space. Then using Reingold's algorithm [1], we determine the weight of each of the components of $G[V(G) \setminus B(t)]$. By Lemma 1, at least one of these sets $B(t)$ would act as the required separator. Its size will be the size of $B(t)$ for some tree node t. Hence it can be at most $w+1$. We summarize this procedure in Lemma 2.

Lemma 2. *Given as input $\langle G, \mathcal{T} \rangle$ where G is a graph, $\mathcal{T}$ is a tree decomposition of G with width w, there exists an $O(w \log n)$ space and polynomial-time algorithm that computes a $\frac{1}{2}$-balanced separator of G of size at most $w+1$ under any non-negative weight function on the vertices of G.*

Let G be a graph, $\mathcal{T}$ be a tree decomposition of G, and U be a subset of $V(G)$. Using Lemma 2, we see that there exists an algorithm that takes as an input $\langle G, \mathcal{T}, U \rangle$ and finds a separator of U in G. We will refer to the separator returned by this algorithm as $\mathsf{sep}(U)$.

3.1. Constructing a recursive decomposition

As an intermediate step, we construct a *recursive decomposition* of the graph, which is a tree whose nodes represent a subgraph of G. The root node represents the entire G. We then remove a separator from it. We assume, inductively, that each connected component has its recursive decomposition, and we connect the root node to the roots of these recursive decompositions of connected components. We select a separator such that a small number of bits can encode each node. This recursive decomposition acts as an intermediate to our tree decomposition. Once we have a recursive decomposition of the graph, we will assign to each node a subset of G such that they would satisfy the properties of tree decomposition.

Definition 1. Let $Z \subseteq V(G)$ and a vertex $r \in (V(G) \setminus Z)$. Define $G_{\langle Z,r \rangle}$ to be the subgraph of G induced by the set of vertices in the connected component of $G[V(G) \setminus Z]$ which contains r. Define the tree $\mathsf{rdtree}(Z, r)$ which we call *recursive decomposition* as follows:

- The root of $\mathsf{rdtree}(Z, r)$ is $\langle Z, r \rangle$.
- Let $Z' = Z \cup \mathsf{sep}(Z) \cup \mathsf{sep}(V(G_{\langle Z,r \rangle}))$, k be the number of connected components of $G[(V(G_{\langle Z,r \rangle}) \setminus Z']$, and let $r_1, \ldots, r_k$ be the lowest indexed vertices in each of these connected components. The children of the root are roots of the recursive decompositions $\mathsf{rdtree}(Z'_i, r_i)$ for each $i \in \{1, \ldots, k\}$, where Z'_i is the set of vertices in Z' that are adjacent to at least one vertex of $V(G_{\langle Z', r_i \rangle})$ in G.

Observe that for the graph G, the recursive decomposition tree structure has logarithmic depth, and we can encode a node $\langle Z, r \rangle$ using $O(|Z| \log n)$ bits.

Lemma 3. *Let v_0 be a vertex in G. Then the depth of the recursive decomposition $\mathsf{rdtree}(\emptyset, v_0)$ is at most $\log n$. Moreover, for a node $\langle Z, r \rangle$ in $\mathsf{rdtree}(\emptyset, v_0)$, we have $|Z| \leq 4w+4$.*

Proof. We prove a more general result that for any set of vertices $Z \subseteq V(G)$ and a vertex $r \in (V(G) \setminus Z)$, the depth of $\mathsf{rdtree}(Z, r)$ is at most $\log n$. Let Z' be as in Definition 1. By Definition 1, the set $\mathsf{sep}(V(G_{\langle Z,r \rangle}))$ is a subset of Z'. Hence removal of Z' divides the graph $G_{\langle Z,r \rangle}$ into components, each of which is of size at most half that of the size of $G_{\langle Z,r \rangle}$. Since $r_1, \ldots, r_k$ are chosen from these components, it follows that the size of $G_{\langle Z', r_i \rangle}$ is at most half of $G_{\langle Z,r \rangle}$. Additionally, in Definition 1 the sets Z'_i are chosen in such a manner that the graphs $G_{\langle Z'_i, r_i \rangle}$ and $G_{\langle Z', r_i \rangle}$ are equivalent. This proves that the size of the graph $G_{\langle Z,r \rangle}$ halves at each level of the recursive decomposition. Hence $\mathsf{rdtree}(Z, r)$ would have at most $\log n$ depth.

We prove the second part of the lemma by induction on the depth of the node $\langle Z, r\rangle$ in $\mathsf{rdtree}(\emptyset, v_0)$. This is trivially true for the root. Now let $\langle Z'_i, r_i\rangle$ be a child of $\langle Z, r\rangle$. Let Z_i be the set of vertices of $Z \setminus \mathsf{sep}(Z)$ which are adjacent to at least one of the vertices of $V(G_{\langle Z', r_i\rangle})$ in G, and let C_i be the unique connected component of $G[V(G) \setminus \mathsf{sep}(Z)]$ whose intersection with $G_{\langle Z', r_i\rangle}$ is not empty. Since $\mathsf{sep}(Z)$ is a separator of Z in G, C_i will contain at most $|Z|/2$ vertices of Z. This shows that $|Z_i| \leq |Z|/2$. By Definition 1, we know that $|Z'_i| \leq |Z_i| + |\mathsf{sep}(Z)| + |\mathsf{sep}(V(G_{\langle Z,r\rangle}))|$. The size of $\mathsf{sep}(V(G_{\langle Z,r\rangle})) \leq w+1$ and $\mathsf{sep}(Z) \leq w+1$ by Lemma 2. Lastly by induction $|Z|/2 \leq (4w+4)/2$. Hence it follows that $|Z'_i| \leq 4w+4$. □

We now show that we can compute the recursive decomposition tree corresponding to G efficiently as well. To prove this, we present procedures to efficiently compute the parent and children of a given node in the recursive decomposition tree.

Algorithm 1: Computes the children of the node $\langle Z, r\rangle$ in $\mathsf{rdtree}(\emptyset, v_0)$.

Input: $\langle G, \mathcal{T}, v_0, Z, r\rangle$
Output: Children of the node $\langle Z, r\rangle$ in $\mathsf{rdtree}(\emptyset, v_0)$
1 Compute $\mathsf{sep}(Z)$ using Lemma 2
2 Compute $\mathsf{sep}(V(G_{\langle Z,r\rangle}))$ using Lemma 2
3 Let $Z' := Z \cup \mathsf{sep}(Z) \cup \mathsf{sep}(V(G_{\langle Z,r\rangle}))$
4 **for** $v \in V(G)$ **do**
5 **if** $v \in V(G_{\langle Z,r\rangle})$ ***and*** *v is smallest indexed vertex in* $G_{\langle Z',v\rangle}$ **then**
6 Let $\hat{Z} := \{u \in Z' \mid u$ is adjacent to $V(G_{\langle Z',v\rangle})$ in $G\}$
7 Output $\langle \hat{Z}, v\rangle$
8 **endif**
9 **endfor**

Algorithm 1 outputs the children of $\langle Z, r\rangle$ in $\mathsf{rdtree}(\emptyset, v_0)$. Note that we do not explicitly store $V(G_{\langle Z,r\rangle})$ but compute it whenever required. That is, whenever we need to check if a vertex belongs to $V(G_{\langle Z,r\rangle})$, we check if it is connected to r in the underlying undirected graph of $G \setminus Z$ using Reingold's algorithm. The separators in line 1 and 2 both have cardinality at most $w+1$ and can be computed in $O(w \log n)$ space and polynomial time by Lemma 2. The cardinality of Z is at most $4w+4$ by Lemma 3. Therefore $|Z'|$ is at most $6w+6$. The size of $\hat{Z}$ computed is $4w+4$ by Lemma 3. Thus the space required by Algorithm 1 is $O(w \log n)$.

Algorithm 2: Computes the parent of the node $\langle Z, r\rangle$ in $\mathsf{rdtree}(\emptyset, v_0)$.

Input: $\langle G, \mathcal{T}, v_0, Z, r\rangle$
Output: parent of the node $\langle Z, r\rangle$ in $\mathsf{rdtree}(\emptyset, v_0)$
1 Set *current* $:= \langle \emptyset, v_0\rangle$
2 **while** $\langle Z, r\rangle$ *is not a child of current* **do**
3 Let $\langle Z', r'\rangle$ be the child of *current* such that $G_{\langle Z',r'\rangle}$ contains r
4 Set *current* $:= \langle Z', r'\rangle$
5 **end**
6 Output *current*

Algorithm 2 outputs the parent of $\langle Z, r\rangle$ in $\mathsf{rdtree}(\emptyset, v_0)$. It uses Algorithm 1 as a subroutine to get the children of a node in $\mathsf{rdtree}(\emptyset, v_0)$. Hence we can traverse the tree $\mathsf{rdtree}(\emptyset, v_0)$ in $O(w \log n)$ space and polynomial time. We summarize the above in Lemma 4.

Lemma 4. *Let G be a graph, $\mathcal{T}$ be a tree decomposition of G with width w and v_0 be a vertex in G. Given $\langle G, \mathcal{T}, v_0\rangle$ and the node $\langle Z, r\rangle$ in $\mathsf{rdtree}(\emptyset, v_0)$, there exist algorithms that use $O(w \log n)$ space and polynomial time, and output the children and parent of $\langle Z, r\rangle$ respectively. As a consequence $\mathsf{rdtree}(\emptyset, v_0)$ can be traversed in $O(w \log n)$ space and polynomial time as well.*

3.2. Constructing a new tree decomposition

We now construct a new tree decomposition of G from the recursive decomposition defined earlier. The new tree decomposition will have the same tree structure as the recursive decomposition. We will assign to each node of this tree a subset of vertices of G. The subgraph that a node of the recursive decomposition represents is a connected component obtained after removing a set of separators from G. The assigned subset for the corresponding node in the tree decomposition is simply the set of separator vertices in the boundary of this subgraph together with the separator required to subdivide this subgraph further. We formalize this in Definition 2.

Definition 2. Let $\hat{T}$ be the tree corresponding to the recursive decomposition $\mathsf{rdtree}(\emptyset, v_0)$. For a node $\langle Z, r\rangle$ in $\mathsf{rdtree}(\emptyset, v_0)$, we define $\hat{B}(\langle Z, r\rangle)$ as $\hat{B}(\langle Z, r\rangle) := Z \cup ((\mathsf{sep}(V(G_{\langle Z,r\rangle})) \cup \mathsf{sep}(Z)) \cap V(G_{\langle Z,r\rangle}))$. Moreover, we define $\hat{B}$ as $\hat{B} := \{\hat{B}(\langle Z, r\rangle) \mid \langle Z, r\rangle$ is a node in $\mathsf{rdtree}(\emptyset, v_0)\}$

We first show that $\langle \hat{T}, \hat{B}\rangle$ is a tree decomposition of G.

Lemma 5. *$\langle \widehat{T}, \widehat{B} \rangle$ is a tree decomposition of G of width $6w+5$. Moreover, the depth of $\widehat{T}$ is at most $\log n$.*

Proof. We claim that for a node v in $G_{\langle Z,r\rangle}$, there exists a vertex $\langle Z', r' \rangle$ in $\mathsf{rdtree}(Z,r)$ such that $\widehat{B}(\langle Z', r' \rangle)$ contains v. We prove this by induction on the depth of the recursive decomposition $\mathsf{rdtree}(Z,r)$. If $\mathsf{rdtree}(Z,r)$ is just a single node, then v is in $\mathsf{sep}(V(G_{\langle Z,r\rangle}))$ by construction. Otherwise v is either in $(\mathsf{sep}(Z) \cup \mathsf{sep}(V(G_{\langle Z,r\rangle})))$ or in one of the connected components of $G[V(G_{\langle Z,r\rangle}) \setminus (\mathsf{sep}(Z) \cup \mathsf{sep}(V(G_{\langle Z,r\rangle})))]$. If v is in $(\mathsf{sep}(Z) \cup \mathsf{sep}(V(G_{\langle Z,r\rangle})))$, then v is in $\widehat{B}(\langle Z,r\rangle)$ and we are done. Otherwise one of the children of $\langle Z,r\rangle$ will be $\langle \tilde{Z}, \tilde{r} \rangle$ such that v is in $G_{\langle \tilde{Z}, \tilde{r}\rangle}$. Now by induction hypothesis, there exists a vertex $\langle Z', r' \rangle$ in $\mathsf{rdtree}(\tilde{Z}, \tilde{r})$ such that $\widehat{B}(\langle Z', r' \rangle)$ contains v. It follows that the first property of tree decomposition is satisfied.

We claim that for any edge (u,v) in G such that $\{u,v\} \subseteq V(G_{\langle Z,r\rangle}) \cup Z$, either both u and v are in $\widehat{B}(\langle Z,r\rangle)$ or there exists a child $\langle Z'_i, r_i \rangle$ of $\langle Z,r\rangle$ such that $\{u,v\} \subseteq V(G_{\langle Z'_i,r_i\rangle}) \cup Z'_i$. Since u and v are connected by an edge, there cannot exist any set of vertices $\widehat{Z}$ such that u and v are in different connected components of $G[V(G) \setminus \widehat{Z}]$. Let $Z' = Z \cup \mathsf{sep}(Z) \cup \mathsf{sep}(V(G_{\langle Z,r\rangle}))$. If both u and v are in Z', then they are in $\widehat{B}(\langle Z,r\rangle)$. Otherwise, let r_i be the lowest indexed vertex in the connected component of $G[(V(G_{\langle Z,r\rangle}) \setminus Z']$ which contains either of u or v. Let Z'_i be the set of vertices in Z' that are adjacent to at least one of the vertices of $V(G_{\langle Z',r_i\rangle})$ in G. Now, if both u and v are not in $V(G_{\langle Z'_i,r_i\rangle})$, then one of them has to be in Z'_i. Hence in all cases, u and v are contained in $V(G_{\langle Z'_i,r_i\rangle}) \cup Z'_i$. Hence by induction on the depth of the tree decomposition $\widehat{T}$ we have that there exists a node in $\widehat{T}$ whose bag contains both u and v, satisfying the second property of tree decomposition.

To establish the third property of tree decomposition we first show that if v is not in $Z \cup V(G_{\langle Z,r\rangle})$, then for no descendant $\langle \tilde{Z}, \tilde{r} \rangle$ of $\langle Z,r\rangle$ will $\widehat{B}(\langle \tilde{Z}, \tilde{r} \rangle)$ contain v. We show this by induction on the depth of the recursive decomposition. If there is only one node in $\mathsf{rdtree}(Z,r)$, then $\widehat{B}(\langle Z,r\rangle)$ does not contain v by definition. Otherwise, no connected component of $G[V(G_{\langle Z,r\rangle}) \setminus Z']$ contains v. Also, Z'_i for any of its children will not contain v as claimed.

Now let $\langle Z,r\rangle$ be a node in $\widehat{T}$. We claim that for any child $\langle Z'_i, r_i \rangle$ of $\langle Z,r\rangle$ if a vertex v is in $\widehat{B}(\langle Z,r\rangle)$, then either v is also in $\widehat{B}(\langle Z'_i, r_i \rangle)$ or no descendant of $\langle Z'_i, r_i \rangle$ has a bag corresponding to it which contains v. Since any connected component of $G[V(G_{\langle Z,r\rangle}) \setminus \widehat{B}(\langle Z,r\rangle)]$ cannot contain v, v is not in $V(G_{\langle Z'_i,r_i\rangle})$ for any child $\langle Z'_i, r_i \rangle$ of $\langle Z,r\rangle$. Now if v is not in $\widehat{B}(\langle Z'_i, r_i \rangle)$, then it implies that v is not in $Z'_i \cup V(G_{\langle Z'_i,r_i\rangle})$ as well. Hence the third property of tree decomposition is satisfied as well.

For a vertex $\langle Z,r\rangle$ in $\mathsf{rdtree}(\emptyset, v_0)$, we have $|Z| \leq 4w+4$, $\mathsf{sep}(Z) \leq w+1$ and $\mathsf{sep}((V(G_{\langle Z,r\rangle})) \leq w+1$ as well. Hence $\widehat{B}(\langle Z,r\rangle) \leq 6w+6$.

Since the tree $\widehat{T}$ and $\mathsf{rdtree}(\emptyset, v_0)$ have the same structure, the bounds on their depths are the same. □

Next, we observe that given $\langle Z,r\rangle$, we can compute $\widehat{B}(\langle Z,r\rangle)$ in $O(w \log n)$ space and polynomial time. Hence we have the following lemma.

Lemma 6. *Given a graph G and a tree decomposition $\mathcal{T}$ of G with width w, there is an algorithm that can compute a new tree decomposition $\widehat{\mathcal{T}} = \langle \widehat{T}, \widehat{B} \rangle$ of G having width at most $6w+5$ and depth at most $\log n$, using $O(w \log n)$ space and polynomial time. Moreover, the tree $\widehat{T}$ can be traversed in $O(w \log n)$ space and polynomial time as well.*

Note that the tree $\widehat{T}$ might not be a binary tree since a separator might disconnect the graph into more than two components. However, to decide reachability in the latter part of this paper, we require the tree decomposition to have a bounded degree as well. We achieve this by using the following Lemma from Elberfeld et al. to get the required tree decomposition $\mathcal{T}'$.

Lemma 7. *[16] Let G be a graph and $\mathcal{T}$ be a tree decomposition of G with logarithmic depth. There exists a logspace algorithm that takes $\mathcal{T}$ as an input and outputs a logarithmic depth, binary balanced tree decomposition of G having the same width.*

Now combining Lemma 6 and Lemma 7 we get the proof of Theorem 3.

We observe that the input tree decomposition $\mathcal{T}$ is used only to compute a vertex separator in G. For those classes of graphs in which a vertex separator can be constructed in a space-efficient manner, we can use their separator algorithms as subroutines instead of the algorithm of Lemma 2 in lines 1 and 2 of the Algorithm 1. We thus get the following theorem.

Theorem 4. *Let $\mathcal{G}$ be a class of graphs such that there exists an $O(s(n))$ space and polynomial-time algorithm that given a graph G of $\mathcal{G}$ and a set U of $V(G)$, outputs a separator of U in G of size $f(n)$. There exists an algorithm working simultaneously in $O(s(n) + f(n) \log n)$ space and polynomial time which outputs a binary balanced tree decomposition $\mathcal{T}'$ of G which has width $O(f(n))$ and depth $O(\log n)$*

Theorem 4 can be of independent interest. Izumi and Otachi [21, Lemma 13], for example, showed that there exists a polynomial-time algorithm that takes as an input a graph G and an integer k and either outputs a separator of size $O(wn^{1/2})$ or decides that the treewidth of G is more than w. We can combine 4 with their result to get the following corollary.

Corollary 1. *There exists an algorithm that takes as an input an n-vertex graph G and $w \leq n^{1/2}$ and either outputs a tree decomposition of width $O(wn^{1/2})$ or correctly decides that the treewidth of G is more than w. This algorithm runs in a polynomial time and uses $O(wn^{1/2} log n)$-bit space.*

Table 1
Universal sequence σ_s for various values of s.

s	σ_s
0	$\langle 1 \rangle$
1	$\langle 1, 2, 1 \rangle$
2	$\langle 1, 2, 1, 4, 1, 2, 1 \rangle$
3	$\langle 1, 2, 1, 4, 1, 2, 1, 8, 1, 2, 1, 4, 1, 2, 1 \rangle$
4	$\langle 1, 2, 1, 4, 1, 2, 1, 8, 1, 2, 1, 4, 1, 2, 1, 16, 1, 2, 1, 4, 1, 2, 1, 8, 1, 2, 1, 4, 1, 2, 1 \rangle$

Thus our results can be used to slightly improve the results of Izumi and Otachi [cf. 21, Theorem 2 and Lemma 14].

4. Deciding reachability using a binary balanced tree decomposition

In this section, we show that there exists a polynomial-time algorithm that uses $O(w \log n)$ space to decide reachability in G. This algorithm also requires a binary balanced tree decomposition $\mathcal{T}$ whose depth is $O(\log n)$ as part of the input. In particular, we show the following Theorem.

Theorem 5. *Given $\langle G, \mathcal{T}, u, v \rangle$ as input, where G is a graph on n vertices, u and v are two vertices of G, and $\mathcal{T}$ is a binary balanced tree decomposition of G having depth h and width w, there exists an $O(wh + \log n)$ space and $O(\mathsf{poly}(2^h, w, n))$ time algorithm that solves reachability in G.*

We first state the notation required to prove Theorem 5. This notation is commonly used to describe dynamic programming algorithms which use tree decomposition. Let T be a rooted binary tree. We denote $\mathsf{root}(T)$ to be the root of T and for a node $t \in T$, we denote $\mathsf{left}(t)$ and $\mathsf{right}(t)$ to be the left and right child of t respectively (the value is NULL if a child does not exist). For two nodes t and t' in T, if t' lies in the path from $\mathsf{root}(T)$ to t, then we say that t' is an *ancestor* of t and t is a *descendant* of t'. Note that each node is also its own ancestor and descendant. For a node t, let $B_e(t)$ denote the set of edges of G whose both endpoints are in $B(t)$. We define a subgraph of G with respect to the node t consisting of the ancestor vertices of t. Formally, the vertex set is $V_t^{\mathsf{anc}} = \bigcup_{\{t' \text{ is an ancestor of } t\}} B(t')$, and the edge set is $E_t^{\mathsf{anc}} = \bigcup_{\{t' \text{ is an ancestor of } t\}} B_e(t')$ and the graph $G_t^{\mathsf{anc}} = (V_t^{\mathsf{anc}}, E_t^{\mathsf{anc}})$. Now, we define a subgraph of G with respect to the node t consisting of the ancestor as well as descendant vertices of t. Formally, the vertex set is $V_t = \bigcup_{\{t' \text{ is an ancestor or descendant of } t\}} B(t')$, the edge set is $E_t = \bigcup_{\{t' \text{ is an ancestor or descendant of } t\}} B_e(t')$ and the graph $G_t = (V_t, E_t)$.

We assume that the vertices u and v are in $\mathsf{root}(T)$. Otherwise, we can add them to all of the bags of the given tree decomposition. Also, we assume that n is a power of 2. Otherwise, we simply add dummy vertices to the graph. This addition does not affect the asymptotic bounds we wish to prove and makes the explanation simpler.

We now explain our reachability algorithm. For a node t in the tree decomposition $\mathcal{T}$ consider the graph G_t. Let P be a path of length d from a vertex of V_t^{anc} to another (assume without loss of generality that d is a power of 2). We define a sequence of leaves $\mathsf{SEQ}_{t,d}$ of T (see Section 4.1). Each leaf f in this sequence corresponds to a set of at most wh vertices V_f. Now subdivide the path P into subpaths $P_1, P_2, \ldots, P_k$ such that each P_i completely lies either in $G_{\mathsf{left}(t)}$ or in $G_{\mathsf{right}(t)}$. We now use the sequence $\mathsf{SEQ}_{t,d}$ to give an iterative procedure to combine the results of the subpaths P_i's to determine the path P. In Algorithm 4, we show how to use the sequence $\mathsf{SEQ}_{t,d}$ to simulate the described method. We show in Lemma 11 that processing $\mathsf{SEQ}_{t,d}$ is sufficient to determine a path of length at most d between two vertices in the graph G_t.

4.1. Constructing the sequence $\mathsf{SEQ}_{t,d}$

We will be using *universal sequences* and the following lemma about it from Asano et al. to construct the sequence of leaves (see Table 1).

For every integer $s \geq 0$, a *universal sequence* σ_s of length $2^{s+1} - 1$ is defined as follows:

$$\sigma_s = \begin{cases} \langle 1 \rangle & s = 0 \\ \sigma_{s-1} \diamond \langle 2^s \rangle \diamond \sigma_{s-1} & s > 0 \end{cases}$$

where $\diamond$ is the concatenation operation.

Lemma 8. *[7] The universal sequence σ_s satisfies the following properties:*

- *Let $\sigma_s = \langle c_1, \ldots, c_{2^{s+1}-1} \rangle$. Then for any positive integer sequence $\langle d_1, \ldots, d_x \rangle$ such that $\Sigma d_i \leq 2^s$, there exists a subsequence $\langle c_{i_1}, \ldots, c_{i_x} \rangle$ such that $d_j \leq c_{i_j}$ for all $j \in [x]$.*
- *The sequence σ_s contains exactly 2^{s-i} appearances of the integer 2^i and nothing else.*
- *The sequence σ_s is computable in $O(2^s)$ time and $O(s)$ space.*

Definition 3. Let T be a binary balanced tree. Let t be a node in T and d be a positive power of 2. We define a sequence $\mathsf{SEQ}_{t,d}$ consisting of leaves of T in the following way: If t is not a leaf, then $\mathsf{SEQ}_{t,d} = \mathsf{SEQ}_{\mathsf{left}(t),c_1} \diamond \mathsf{SEQ}_{\mathsf{right}(t),c_1} \diamond \mathsf{SEQ}_{\mathsf{left}(t),c_2} \diamond \mathsf{SEQ}_{\mathsf{right}(t),c_2} \diamond$

$\cdots \diamond \mathsf{SEQ}_{\mathsf{right}(t),c_{2d-1}}$ where c_i is the i-th integer in $\sigma_{\log d}$. Otherwise, if t is a leaf, $\mathsf{SEQ}_{t,d}$ is $\langle t \rangle$ concatenated with itself d times. We also define $\mathsf{SEQ}_{t,d}(r)$ to be the leaf at the index r in the sequence $\mathsf{SEQ}_{t,d}$. The length of $\mathsf{SEQ}_{t,d}$ is the number of leaves in $\mathsf{SEQ}_{t,d}$.

We show in Algorithm 3 how to construct the sequence $\mathsf{SEQ}_{\mathsf{root}(T),d}$ in $O(h + \log d)$ space. In Lemma 9, we give a closed form expression for the length of the sequence.

Algorithm to compute $SEQ_{t,d}$

Algorithm 3: Computes the r-th element of the sequence $\mathsf{SEQ}_{t,d}$.

Input: $\langle t, d, r \rangle$
while *t is not a leaf* **do**
  Let m be the depth of the subtree of T rooted at t
  Let i^* be the smallest integer such that $(r - 2\sum_{i=1}^{i^*} L(m/2, c_i)) \leq 0$ where c_i is the i'th integer in the sequence $\sigma_{\log d}$
  if $r - 2\sum_{i=1}^{i^*-1} L(m/2, c_i) - L(m/2, c_{i^*}) \leq 0$ **then**
    $r \leftarrow r - 2\sum_{i=1}^{i^*-1} L(m/2, c_i)$
    $t \leftarrow \mathsf{left}(t)$
  else
    $r \leftarrow r - 2\sum_{i=1}^{i^*-1} L(m/2, c_i) - L(m/2, c_{i^*})$
    $t \leftarrow \mathsf{right}(t)$
  endif
  $d \leftarrow c_{i^*}$
end
return t

Lemma 9. *Let T be a binary balanced tree. Let t be a node in T, d be a positive power of 2 and h be the depth of subtree of T rooted at t. Then, the length of sequence $SEQ_{t,d}$ is $2^h d \binom{h+\log d}{\log d}$.*

Proof. Let $L(h, d)$ be the length of the sequence $\mathsf{SEQ}_{t,d}$. By definition of $\mathsf{SEQ}_{t,d}$, we have

$$L(h,d) = \begin{cases} 2\sum_{c \in \sigma_{\log d}} L(h-1, c) & h > 0 \\ d & h = 0 \end{cases}$$

From Lemma 8, we get that $\sigma_{\log d}$ contains exactly $\frac{d}{2^i}$ occurrences of the integer 2^i. Thus we have:

$$L(h,d) = \begin{cases} \sum_{i=0}^{\log d} \frac{d}{2^{i-1}} L(h-1, 2^i) & h > 0 \\ d & h = 0 \end{cases}$$

We claim that $L(h, d) = 2^h d \binom{h+\log d}{\log d}$ and we prove this by induction on h. For $h = 0$, we see that

$$\begin{aligned} 2^h d \binom{h + \log d}{\log d} &= d \binom{\log d}{\log d} \\ &= d \end{aligned}$$

Now, we assume the statement to be true for smaller values of h. We see that:

$$L(h,d) = \sum_{i=0}^{\log d} \frac{d}{2^{i-1}} L(h-1, 2^i)$$

$$L(h,d) = \sum_{i=0}^{\log d} \frac{d}{2^{i-1}} 2^{h-1} 2^i \binom{h+i-1}{i}$$

$$L(h,d) = 2^h d \sum_{i=0}^{\log d} \binom{h+i-1}{i}$$

using $\binom{a}{r} = \binom{a+1}{r} - \binom{a}{r-1}$

$$L(h,d) = 2^h d \sum_{i=0}^{\log d} \left(\binom{h+i}{i} - \binom{h+i-1}{i-1} \right)$$

$$L(h,d) = 2^h d \binom{h + \log d}{\log d} \quad \square$$

Lemma 10. *Let T be a binary balanced tree of depth at most h. Let t be a node of T and d be a power of 2. The sequence $\mathsf{SEQ}_{t,d}$ can be constructed in space $O(h + \log d)$.*

Proof. We see that $L(m,d)$ is bounded by a polynomial in m and d. For a given integer r, let i^* be the smallest integer such that $r - 2\sum_{i=1}^{i^*} L(m/2, c_i) \leq 0$. By the definition, $\mathsf{SEQ}_{t,d}(r) = \mathsf{SEQ}_{\mathsf{left}(t),c_{i^*}}(r - 2\sum_{i=1}^{i^*-1} L(m/2,c_i))$ if $r - 2\sum_{i=1}^{i^*-1} L(m/2,c_i) - L(m/2,c_{i^*}) \leq 0$ and $\mathsf{SEQ}_{t,d}(r) = \mathsf{SEQ}_{\mathsf{right}(t),c_{i^*}}(r - 2\sum_{i=1}^{i^*-1} L(m/2,c_i) - L(m/2,c_{i^*}))$ otherwise.

The length of the sequence $\mathsf{SEQ}_{t,d}$ is at most $2^h d\binom{h+\log d}{\log d}$. Hence the number of bits required to store any index of the sequence is at most $\log(2^h d\binom{h+\log d}{\log d}) = O(h + \log d)$. This gives the space-bound of Algorithm 3. □

4.2. Algorithm to solve reachability

Algorithm 4: Reach(G, $\mathcal{T}$, u, v).

```
   Input: ⟨G, T, u, v⟩
1  Let R_0 be and R_1 be two wh bit-vectors
2  Initialize t_0 and t_1 by two arbitrary leaves of T
3  Initialize all the bits of R_0 with 0 and mark u (by setting the bit at position pos_{t_0}(u) to 1)
4  for every leaf f in SEQ_{root(T),n} in order do
5      Let i be the iteration number
6      Reset all the bits of R_{i mod 2} to 0
7      Let t_{i mod 2} ← f
8      for all x marked in R_{(i−1) mod 2} and all y in V_f do
9          if (x, y) is an edge in G OR x = y then
10             Mark y in R_{i mod 2} (by setting the bit at position pos_{t_{i mod 2}}(y) to 1)
11         endif
12     endfor
13 endfor
14 If v is marked return 1; otherwise return 0.
```

Let G be a graph and $\mathcal{T} = \langle T, \mathcal{B}\rangle$ be a binary balanced tree decomposition of G. For a leaf t of T and a vertex v of G we use $\mathsf{pos}_t(v)$ for the position of v in an arbitrarily fixed ordering of the vertices of G_t. This position can be found in logspace. For example, consider the vertices ordered by their indices in ascending order. For this ordering, $\mathsf{pos}_t(v)$ can be found by cycling through all vertices and counting the number of vertices in G_t whose index is less than that of v.

Lemma 11. *Let G be a graph and $\mathcal{T} = \langle T, \mathcal{B}\rangle$ be a binary balanced tree decomposition of G of width w and depth h. Let t be a node of T and d be a power of 2. For each vertex $y \in V_t^{\mathsf{anc}}$, y is marked after the execution of iterations in lines 4 to 13 of Algorithm 4 with values of f in $\mathsf{SEQ}_{t,d}$ if there is a marked vertex x in V_t^{anc} and a path from x to y in G_t of length at most d.*

Proof. We prove this by induction on the depth of subtree rooted at t. The base case is trivial. Let p be the path of length at most d from x to y such that x is marked and x, y is in V_t^{anc}. We see that the edges of path p will belong to either $E_{\mathsf{left}(t)}$ or $E_{\mathsf{right}(t)}$ (or both). We label an edge of p as 0 if it belongs to $E_{\mathsf{left}(t)}$, else label it as 1. Break down p into subpaths $p_1, \ldots, p_k$ such that the edges in p_i are all labeled with the same value and label of edges in p_{i+1} is different from label of p_i. The endpoints y_i of these subpaths will belong to V_t^{anc}, for otherwise, y_i will not be in $B(t)$ but since y_i has edges of both labels incident on it, it will be in bags of both subtrees rooted at $\mathsf{left}(t)$ and $\mathsf{right}(t)$ contradicting the third property of tree decompositions. Let l_i be the length of path p_i. Since $l_1 + l_2 + \cdots + l_k \leq d$, by Lemma 8, there exists a subsequence $\langle c_{i_1}, c_{i_2}, \ldots, c_{i_k}\rangle$ of $\sigma_{\log d}$ such that $l_j \leq c_{i_j}$.

Consider the subsequence $\mathsf{SEQ}_{\mathsf{left}(t),c_{i_1}} \diamond \mathsf{SEQ}_{\mathsf{right}(t),c_{i_1}} \diamond \mathsf{SEQ}_{\mathsf{left}(t),c_{i_2}} \diamond \mathsf{SEQ}_{\mathsf{right}(t),c_{i_2}} \diamond \mathsf{SEQ}_{\mathsf{left}(t),c_{i_3}} \diamond \mathsf{SEQ}_{\mathsf{right}(t),c_{i_3}} \diamond \cdots \diamond \mathsf{SEQ}_{\mathsf{left}(t),c_{i_k}} \diamond \mathsf{SEQ}_{\mathsf{right}(t),c_{i_k}}$ of $\mathsf{SEQ}_{t,d}$. We claim that y_j is marked after the iterations with the value of f in $\mathsf{SEQ}_{\mathsf{left}(t),c_{i_j}} \diamond \mathsf{SEQ}_{\mathsf{right}(t),c_{i_j}}$. Since y_{j-1} is marked before the iterations and the path p_j is either the subgraph $G_{\mathsf{left}(t)}$ or $G_{\mathsf{right}(t)}$ having length at most c_{i_j}, y_j will be marked by induction hypothesis. We see that any vertex present in V_t^{anc} is present in $G_{t'}$ for all leaves t' that is present in $\mathsf{SEQ}_{t,d}$. Therefore, once such a vertex is marked, it remains marked for the rest of these iterations. Hence, y_j is marked before the iterations with the value of f in $\mathsf{SEQ}_{\mathsf{left}(t),c_{i_{j+1}}} \diamond \mathsf{SEQ}_{\mathsf{right}(t),c_{i_{j+1}}}$ □

Lemma 12. *On input of a graph G with n vertices and its binary balanced tree decomposition $\mathcal{T} = \langle T, \mathcal{B}\rangle$ with width w and depth h; Algorithm 4 solves reachability in G and requires $O(wh + \log n)$ space and time polynomial in $2^h, n$ and w.*

Proof. Algorithm 4 marks a vertex only if it is reachable from u. The proof of correctness of the algorithm follows from Lemma 11 and the fact that u and v are both present in $B(\mathsf{root}(T))$ and u is marked before the first iteration of the for-loop in line 4.

We first analyze the space required. The size of bit-vectors R_0 and R_1 is wh. t_0 and t_1 are indices of nodes of T. The space required to store the index of a vertex of T is $O(h)$. Space required to store a vertex of G is $O(\log n)$, and $\mathsf{pos}_t(x)$ for a node t and a vertex x can be found in $O(\log n + h)$ space. Hence the total space required is $O(wh + \log n)$.

We now analyze the time-bound. By Lemma 9, the size of $\mathsf{SEQ}_{t,d}$ is polynomial in 2^h and d, the number of iterations in the for-loop of line 4 is thus a polynomial. The other lines do trivial stuff, and hence, the total running time of the algorithm is polynomial. □

Theorem 5 follows from Lemma 12. Combining Theorem 5 and Theorem 3 we get the proof of Theorem 1. Similarly, combining Theorem 5 and Theorem 4 we get the proof of Theorem 2.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

No data was used for the research described in the article.

References

[1] O. Reingold, Undirected connectivity in log-space, J. ACM 55 (4) (2008) 17.
[2] W.J. Savitch, Relationships between nondeterministic and deterministic tape complexities, J. Comput. Syst. Sci. 4 (2) (1970) 177–192.
[3] A. Wigderson, The complexity of graph connectivity, in: Proceedings of the 17th International Symposium on Mathematical Foundations of Computer Science (MFCS 1992), Springer, 1992, pp. 112–132.
[4] G. Barnes, J.F. Buss, W.L. Ruzzo, B. Schieber, A sublinear space, polynomial time algorithm for directed s-t connectivity, SIAM J. Comput. 27 (5) (1998) 1273–1282.
[5] T. Asano, B. Doerr, Memory-constrained algorithms for shortest path problem, in: Proceedings of the 23rd Annual Canadian Conference on Computational Geometry (CCCG 2011), 2011.
[6] T. Imai, K. Nakagawa, A. Pavan, N.V. Vinodchandran, O. Watanabe, An $O(n^{\frac{1}{2}+\epsilon})$-space and polynomial-time algorithm for directed planar reachability, in: Proceedings of the 28th Conference on Computational Complexity, CCC 2013, 2013, pp. 277–286.
[7] T. Asano, D. Kirkpatrick, K. Nakagawa, O. Watanabe, $\tilde{O}(\sqrt{n})$-space and polynomial-time algorithm for planar directed graph reachability, in: Proceedings of the 39th International Symposium on Mathematical Foundations of Computer Science (MFCS 2014), 2014, pp. 45–56.
[8] D. Chakraborty, A. Pavan, R. Tewari, N.V. Vinodchandran, L.F. Yang, New time-space upperbounds for directed reachability in high-genus and h-minor-free graphs, in: Proceedings of the 34th Annual Conference on Foundation of Software Technology and Theoretical Computer Science (FSTTCS 2014), 2014, pp. 585–595.
[9] D. Chakraborty, R. Tewari, An $O(n^{\epsilon})$ space and polynomial time algorithm for reachability in directed layered planar graphs, ACM Trans. Comput. Theory 9 (4) (2017) 19.
[10] S. Arnborg, A. Proskurowski, Linear time algorithms for np-hard problems restricted to partial k-trees, Discrete Appl. Math. 23 (1) (1989) 11–24.
[11] H.L. Bodlaender, A.M.C.A. Koster, Combinatorial optimization on graphs of bounded treewidth, Comput. J. 51 (3) (2008) 255–269.
[12] S. Arnborg, D.G. Corneil, A. Proskurowski, Complexity of finding embeddings in a k-tree, SIAM J. Algebraic Discrete Methods 8 (2) (1987) 277–284.
[13] U. Feige, M. Hajiaghayi, J.R. Lee, Improved approximation algorithms for minimum weight vertex separators, SIAM J. Comput. 38 (2) (2008) 629–657.
[14] A. Jakoby, 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), 2007, pp. 216–227.
[15] B. Das, S. Datta, P. Nimbhorkar, Log-space algorithms for paths and matchings in k-trees, Theory Comput. Syst. 53 (4) (2013) 669–689.
[16] M. Elberfeld, A. Jakoby, T. Tantau, Logspace versions of the theorems of Bodlaender and Courcelle, in: Proceedings of the 51st Annual Symposium on Foundations of Computer Science (FOCS 2010), IEEE Computer Society, 2010, pp. 143–152.
[17] S. Bhore, R. Jain, Space-efficient algorithms for reachability in directed geometric graphs, in: 32nd International Symposium on Algorithms and Computation (ISAAC 2021), in: Leibniz International Proceedings in Informatics (LIPIcs), vol. 151, Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, Dagstuhl, Germany, 2021.
[18] R.J. Chetan Gupta, R. Tewari, Time space optimal algorithm for computing separators in bounded genus graphs, in: 41st IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science, (FSTTCS 2021), in: LIPIcs, vol. 152, Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021.
[19] M. Elberfeld, K.-i. Kawarabayashi, Embedding and canonizing graphs of bounded genus in logspace, in: Proceedings of the 46th Annual ACM Symposium on Theory of Computing (STOC 2014), ACM, 2014, pp. 383–392.
[20] J. Gilbert, D. Rose, A. Edenbrandt, A separator theorem for chordal graphs, SIAM J. Algebraic Discrete Methods 5 (3) (1984) 306–313.
[21] T. Izumi, Y. Otachi, Sublinear-space lexicographic depth-first search for bounded treewidth graphs and planar graphs, in: A. Czumaj, A. Dawar, E. Merelli (Eds.), 47th International Colloquium on Automata, Languages, and Programming (ICALP 2020), in: Leibniz International Proceedings in Informatics (LIPIcs), vol. 168, Schloss Dagstuhl–Leibniz-Zentrum für Informatik, Dagstuhl, Germany, 2020, pp. 67:1–67:17, https://drops.dagstuhl.de/opus/volltexte/2020/12474.
[22] M. Cygan, F. Fomin, Ł. Kowalik, D. Lokshtanov, D. Marx, M. Pilipczuk, M. Pilipczuk, S. Saurabh, Parameterized Algorithms, Springer International Publishing, 2015, https://books.google.de/books?id=Frg0CgAAQBAJ.
[23] H.L. Bodlaender, T. Hagerup, Parallel algorithms with optimal speedup for bounded treewidth, SIAM J. Comput. 27 (6) (1998) 1725–1746, https://doi.org/10.1137/S0097539795289859.
[24] I. Niepel, P. Rossmanith, Uniform circuits and exclusive read prams, in: S. Biswas, K.V. Nori (Eds.), Foundations of Software Technology and Theoretical Computer Science, Springer Berlin Heidelberg, Berlin, Heidelberg, 1991, pp. 307–318.
[25] K. Chatterjee, R. Ibsen-Jensen, A. Pavlogiannis, Optimal tree-decomposition balancing and reachability on low treewidth graphs, https://doi.org/10.15479/AT:IST-2014-314-V1-1, 2014.
[26] B.A. Reed, Finding approximate separators and computing tree width quickly, in: Proceedings of the Twenty-Fourth Annual ACM Symposium on Theory of Computing, STOC'92, Association for Computing Machinery, New York, NY, USA, 1992, pp. 221–228.
[27] F. Wei-Kleiner, Tree decomposition-based indexing for efficient shortest path and nearest neighbors query answering on graphs, J. Comput. Syst. Sci. 82 (1, Part A) (2016) 23–44, https://doi.org/10.1016/j.jcss.2015.06.008, https://www.sciencedirect.com/science/article/pii/S0022000015000707.