Reachability in High Treewidth Graphs

Rahul Jain
Indian Institute of Technology Kanpur, India
jain@iitk.ac.in

Raghunath Tewari
Indian Institute of Technology Kanpur, India
rtewari@iitk.ac.in

Abstract

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

In this paper, we show that given an n vertex directed graph of treewidth w along with its tree decomposition, there exists an algorithm running in polynomial time and $O(w \log n)$ space that solves the reachability problem.

2012 ACM Subject Classification Theory of computation

Keywords and phrases graph reachability, simultaneous time-space upper bound, tree decomposition

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2019.12

Funding *Rahul Jain*: Ministry of Human Resource Development, Government of India
Raghunath Tewari: DST Inspire Faculty Grant, Visvesvaraya Young Faculty Fellowship

1 Introduction

Given a graph G and two vertices u and v in G, the reachability problem is to decide if there exists a path from u to v in G. This problem is NL-complete for directed graphs and L-complete for undirected graphs [17]. Hence its study gives important insight into space bounded computations. We will henceforth refer to the problem of directed graph reachability as Reach. The famous open question $\mathsf{L} \overset{?}{=} \mathsf{NL}$ essentially asks if there is a deterministic logspace algorithm for Reach or not. Reach can be solved in $\Theta(n \log n)$ space and optimal time using standard graph traversal algorithms such as DFS and BFS. We also know, due to Savitch, that it can be solved in $\Theta(\log^2 n)$ space [18]. However, Savitch's algorithm requires $n^{\Theta(\log n)}$ time. Wigderson surveyed reachability problems in which he asked if there is an algorithm for Reach that runs simultaneously in $O(n^{1-\epsilon})$ space (for any $\epsilon > 0$) and polynomial time [19]. Here, we make some partial progress towards answering this question.

In 1998 Barnes et al. made progress in answering Wigderson's question for general graphs by presenting an algorithm for Reach that runs simultaneously in $n/2^{\Theta(\sqrt{\log n})}$ space and polynomial time [6]. For several other topologically restricted classes of graphs, there has been significant progress in giving polynomial time algorithms for Reach that run simultaneously in sublinear space. For grid graphs a space bound of $O(n^{1/2+\epsilon})$ was first achieved [4]. The same space bound was then extended to all planar graphs by Imai et al. [14]. Later for planar graphs, the space bound was improved to $\tilde{O}(n^{1/2})$ space by Asano et al. [5]. 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 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 [3]. The weighted independent set problem can be solved in $O(2^w n)$ time [7]. It is NP-complete to find on given input $\langle G, k\rangle$ if G has treewidth k [2]. However, an $O(\sqrt{\log n})$-factor approximation algorithm is known [12]. Series-parallel graphs are equivalent to graphs of treewidth 2. For them, Jakoby and Tantau showed a logspace algorithm for Reach [15]. Das et al. extended the logspace bound to bounded treewidth graphs when the input contains the tree decomposition [10]. Elberfeld et al. showed a logspace algorithm for any monadic second order property of a logical structure of bounded treewidth [11].

1.1 Our Result

In this paper, we present a polynomial time algorithm with improved space bound for deciding reachability in directed graphs of treewidth w. In particular, we show the following result.

▶ **Theorem 1.** *Given a directed graph G, a tree decomposition T of G of treewidth 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 constant treewidth graph given a tree decomposition as input [10]. Although they do not explicitly analyze the space and time bounds of their algorithm to show how it is dependent on the treewidth w of the graph, a naive analysis would show that their algorithm requires $\Omega(w^2 \log n)$ space and $n^{\Omega(w \log w)}$ time.

The graph reachability problem can also be expressed by a constant-size MSO formula. Hence the result by Elberfeld et al. [11] solves for a more general problem and implies a logspace algorithm for reachability in constant 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 here that both the algorithms of Elberfeld et al. [11] and Das et al. [10] cease to be polynomial time algorithms for classes of graphs whose treewidth w is not constant. These include a large number of interesting classes of graphs mentioned in the introduction. Our algorithm requires $O(w \log n)$ space and $O(\mathsf{poly}(n, w))$ time and therefore has a better time and space complexity for solving the reachability problem when compared to the results of [11] and [10].

The notion of treewidth is intimately connected with a well-studied notion of vertex-separators in a graph. It is known 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, using the input tree decomposition, vertex separators of size $w+1$ of input graph can be constructed in $O(w \log n)$ space and polynomial time.
- Using the algorithm for vertex separator as a subroutine, we construct a new binary balanced tree decomposition of G having $O(w)$ treewidth 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. [5] 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. Hence, the method presented here gives a slightly stronger result when dealing with classes of graph where an $O(w \log n)$ space and polynomial time algorithm for finding a vertex separator of size $O(w)$ is known. For such graphs, we can waive the requirement of additional tree decomposition in the input 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 $w : \mathbb{N} \times \mathbb{N} \to \mathbb{N}$ be a function such that for every graph $G \in \mathcal{G}$ with n vertices and m edges, treewidth of G is atmost $w(n, m)$. If there exist an $O(w(n, m) \log n)$ space and polynomial time algorithm, that given a graph $G \in \mathcal{G}$ and a set U of $V(G)$, outputs a separator of U in G of size $O(w(n, m))$, then there exists an algorithm to decide reachability in G that uses $O(w(n, m) \log n)$ space and polynomial time.*

For constant treewidth graphs, the result of Elberfeld et al. [11] is equipped with a logspace algorithm to construct a binary balanced tree decomposition of $O(w)$ treewidth whose depth is $O(w \log n)$. Since vertex separators can be constructed in logspace for constant treewidth graphs, our technique can be used to construct a binary balanced tree decomposition of $O(w)$ treewidth with $O(\log n)$ depth in logspace. The depth of our tree decomposition is independent of w.

1.2 Consequences of our Result

For graphs of treewidth $n^{1-\epsilon}$, for any $\epsilon > 0$, our algorithm gives an $O(n^{1-\delta})$ space and polynomial time algorithm (for some δ). For graphs of polylog treewidth, we show that reachability can be solved in polynomial time and polylog space.

Graphs which have genus g have treewidth $O((gn)^{1/2})$; hence our algorithm gives a $O((gn)^{1/2} \log n)$ space and polynomial time algorithm for it.

For planar graphs, our approach gives a $O(n^{1/2} \log n)$ space and polynomial time algorithm. Note that for planar graphs, a careful analysis of the separator construction algorithm of Imai et al. shows that one can construct a separator for planar graphs in polynomial time and $O(n^{1/2} \log n)$ space [14]. We can use this separator construction and waive the requirement of having the tree decomposition as input for planar graphs. As a result, we get the best known simultaneous time space bound for reachability in planar graphs.

Let H be a graph on h vertices. An H minor-free graph is also, by definition, K_h minor free where K_h is a complete graph on h vertices. Graphs which exclude a fixed minor K_h, have a treewidth of atmost $hn^{1/2}$ [1][16]. Hence, for constant h, our approach results in $O(n^{1/2} \log n)$ space and polynomial time algorithm for H minor-free graphs.

A chordal graph on n vertices with m edges have a separator of size $O(m^{1/2})$[13]. The proof of existence of such a separator in [13] directly leads to a $O(m^{1/2} \log n)$ space and polynomial time algorithm of constructing it. We can use this separator and obtain an $O(m^{1/2} \log n)$ space and polynomial time algorithm to solve the reachability problem in chordal graphs *without* the input tree decomposition.

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 having a similar width from 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.

We next define the terminology and notations related to tree decomposition that we use 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 labeled tree T where the labeling function $B : V(T) \to \{X \mid X \subseteq V(G)\}$ has the following property: (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 *treewidth* of a tree decomposition T is $\max_{t \in V(T)}(B(t) - 1)$. Finally the treewidth of a graph G is the minimum treewidth 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.

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 leaves have the same length.

The next tool that we would be using is that of separators in graphs. For a subset W of $V(G)$, a *vertex separator* of W in G, is a subset S of $V(G)$ such that every component of the graph $G[V(G) \setminus S]$ has at most $|W|/2$ vertices of W.

We state in Lemma 3 the commonly known result about vertex separators in the form that we would be using it.

▶ **Lemma 3.** *Let G be a graph and T be a tree decomposition of G. For every subset U of $V(G)$, there exists a node t in $V(T)$ such that the bag $B(t)$ is a vertex separator of U in G.*

Proof. We root the tree arbitrarily. For a node t in $V(T)$, we denote its parent by $\mathsf{parent}(t)$. Let $C(t) = B(t) \cap U$. We define weights on the nodes of T, such that each vertex of U is counted in exactly one of the weights. $\alpha(t) = |C(t) \setminus C(\mathsf{parent}(t))|$. Thus, $\sum_{t \in V(T)} \alpha(t) = |U|$. In a weighted tree, there exists a node whose removal divides the tree into components whose weights are at most half the total weight of the tree. Let this node be t^* for the weight function α. We claim that $B(t^*)$ is the vertex separator for U in G. To prove this, we will prove that for a connected component H of $G[V(G) \setminus B(t^*)]$, H is a subset of $(\cup_{t \in T_i} B(t))$ for some subtree T_i of $T \setminus \{t^*\}$. Since the total weight of this subtree is at most half the total weight on T, it would follow that the number of vertices of $U \setminus B(t^*)$ contained in this set would be at most half, thus proving the lemma.

We will now prove that $H \subseteq (\cup_{t \in T_i} B(t))$. We first observe that a vertex $v \in V(G) \setminus B(t^*)$ can be in the bag of only one of the subtree, since otherwise, it would belong to $B(t^*)$ as well, due to the third property of tree decomposition. Now, let us assume that there are two vertices of H which belong to the bags of two different subtrees, say T_i and T_j. Since they are in a connected component, there will exist a path between them. In this path, there will exist an edge, whose endpoints v_1 and v_2 would belong to different subtrees. We thus get a contradiction to the second property of tree decomposition. ◀

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

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 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

In this section, we show how to compute a binary balanced tree decomposition (say T') with logarithmic depth and treewidth $O(w)$. We require such a tree decomposition because our main algorithm for reachability (Algorithm 4) might potentially store reachability information for all vertices corresponding to the bags of treenodes in a path from the root to a leaf. 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 4.** *Given as input $\langle G, T\rangle$ where G is a graph and T is a tree decomposition of G with treewidth w, there exists an algorithm working simultaneously in $O(w \log n)$ space and polynomial time which outputs a binary tree decomposition T' of G which has treewidth $6w + 6$ and depth $O(\log n)$.*

We will now develop a framework that will help us to prove Theorem 4.

First, we show how to compute a vertex separator of a given set U in G in polynomial time and $O(w \log n)$ space. 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 undirected reachability algorithm [17], we count the number of vertices of U in each of the components of $G[V(G) \setminus B(t)]$. By Lemma 3, at least one of these sets $B(t)$ would be a separator of U in G. Its size will be the size of $B(t)$ for some treenode t. Hence it can be at most $w + 1$. We summarise this procedure in Lemma 5.

▶ **Lemma 5.** *Given as input $\langle G, T, U\rangle$ where G is a graph, T is a tree decomposition of G with treewidth w, and U is a subset of $V(G)$, there exists an $O(w \log n)$ space and polynomial time algorithm that computes a vertex separator of U in G of size atmost $w + 1$.*

When G and T are clear from the context, we will refer to the vertex separator of U in G that is returned by the algorithm of Lemma 5 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 of the connected components has its recursive decomposition and 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 add labels to each node such that it satisfies the properties of tree decomposition.

▶ **Definition 6.** *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* $Z' = Z \cup \mathsf{sep}(Z) \cup \mathsf{sep}(V(G_{\langle Z,r\rangle}))$ *and let* $r_1, \ldots, r_k$ *be the lowest indexed vertices in each of the connected components of* $G[(V(G_{\langle Z,r\rangle}) \setminus Z']$. *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 7.** *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 6. By Definition 6, 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 6 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 6, 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 5. Lastly by induction $|Z|/2 \leq (4w+4)/2$. Hence it follows that $|Z'_i| \leq 4w+4$. ◀

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

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

```
  Input: $\langle G, T, v_0, Z, r\rangle$
  Output: Children of the node $\langle Z, r\rangle$ in $\mathsf{rdtree}(\emptyset, v_0)$
1 Compute $\mathsf{sep}(Z)$ using Lemma 5
2 Compute $\mathsf{sep}(V(G_{\langle Z,r\rangle}))$ using Lemma 5
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 $\widehat{Z} := \{u \in Z' \mid u$ is adjacent to $V(G_{\langle Z',v\rangle})$ in $G\}$
7         Output $\langle \widehat{Z}, v\rangle$
8     endif
9 endfor
```

Algorithm 1 outputs the children of $\langle Z, r\rangle$ in $\mathsf{rdtree}(\emptyset, v_0)$. Note that we don't 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 5. The cardinality of Z is at most $4w+4$ by Lemma 7. Therefore $|Z'|$ is at most $6w+6$. The size of $\widehat{Z}$ computed is $4w+4$ by Lemma 7. 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: $\langle G, T, v_0, Z, r\rangle$
  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 8.

▶ **Lemma 8.** *Let G be a graph, T be a tree decomposition of G with treewidth w and v_0 be a vertex in G. Given $\langle G, T, v_0\rangle$ 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 that of the recursive decomposition. However, we will assign it a labeling function. The subgraph that a node of the recursive decomposition represents is a connected component obtained after removing a set of separators from G. The corresponding label for this node in the new 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 9.

▶ **Definition 9.** *Let $\widehat{T}$ 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 the function $\widehat{B}(\langle Z, r\rangle)$ as, $\widehat{B}(\langle Z, r\rangle) := Z \cup ((\mathsf{sep}(V(G_{\langle Z,r\rangle})) \cup \mathsf{sep}(Z)) \cap V(G_{\langle Z,r\rangle}))$.*

We first show that $\widehat{T}$ is a tree decomposition of G as well, with labeling function $\widehat{B}$.

▶ **Lemma 10.** *The tree $\widehat{T}$ defined in Definition 9 along with the labeling function $\widehat{B}$, is a tree decomposition of G of width $6w+6$. 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 every vertex v of $V(G)$ is contained in the label of at least one of the vertices of $\widehat{T}$, satisfying the first property of tree decomposition.

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 is 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 have 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 treenode 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 treenode 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 11.** *Given a graph G and a tree decomposition T of G with treewidth w, there is an algorithm that can compute a new tree decomposition $\widehat{T}$ of G having treewidth at most $6w + 6$ 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 later part of this paper, we require the tree decomposition to have bounded degree as well. We achieve this by using the following lemma from Elberfeld et al. to get the required tree decomposition T'.

▶ **Lemma 12** ([11]). *There is a logspace algorithm that on the input of any logarithmic depth tree decomposition of a graph G outputs a logarithmic depth, binary balanced tree decomposition of G having the same treewidth.*

Now combining Lemma 11 and Lemma 12 we get the proof of Theorem 4.

We observe here that the input tree decomposition T is used only to compute a vertex separator in G.

The requirement of input tree decomposition can be waived for those classes of graphs where vertex separators can be constructed in a space efficient manner. For example, in planar graphs [14] and chordal graphs [13], we can use their respective separator algorithms as subroutines instead of the algorithm of Lemma 5 in lines 1 and 2 of the Algorithm 1.

4 Deciding Reachability using a Binary Balanced Tree Decomposition

In this section, we show that given a graph G along with a binary balanced tree decomposition T whose depth is $O(\log n)$; there exists an efficient algorithm to decide reachability in G in $O(w \log n)$ space and polynomial time. In particular, we show the following theorem.

▶ **Theorem 13.** *Given $\langle G, T, u, v\rangle$ as input, where G is a graph on n vertices, u and v are two vertices of G, and T is a binary balanced tree decomposition of G having depth h and treewidth 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 13. 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 *descendent* of t'. For a treenode 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 treenode 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 treenode t consisting of the ancestor as well as descendent vertices of t. Formally, the vertex set is $V_t = \bigcup_{\{t' \text{ is an ancestor or descendent of } t\}} B(t')$, the edge set is $E_t = \bigcup_{\{t' \text{ is an ancestor or descendent of } t\}} B_e(t')$ and the graph $G_t = (V_t, E_t)$.

We assume that the vertices u and v are in $\mathsf{root}(T)$, for otherwise, we can add them in all of the bags of the given tree decomposition. Also, we assume that n is a power of 2.

We now explain our reachability algorithm. For a node t in the tree decomposition 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 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 18 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.

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 14** ([5]). *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 15.** *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 $SEQ_{t,d}$ consisting of leaves of T in the following way: If t is not a leaf then $SEQ_{t,d} = SEQ_{\mathit{left}(t),c_1} \diamond SEQ_{\mathit{right}(t),c_1} \diamond SEQ_{\mathit{left}(t),c_2} \diamond SEQ_{\mathit{right}(t),c_2} \diamond \cdots \diamond SEQ_{\mathit{right}(t),c_{2d-1}}$ where c_i is the i-th integer in $\sigma_{\log d}$. Otherwise, if t is a leaf, $SEQ_{t,d}$ is $\langle t \rangle$ concatenated with itself d times. We also define $SEQ_{t,d}(r)$ to be the leaf at the index r in the sequence $SEQ_{t,d}$. The length of $SEQ_{t,d}$ is the number of leaves in $SEQ_{t,d}$.*

We show in Algorithm 3 how to construct the sequence $\mathsf{SEQ}_{\mathsf{root}(T),d}$ in $O(h + \log d)$ space. In Lemma 16 we give a closed form expression for the length of the sequence.

Algorithm to Compute $\mathsf{SEQ}_{t,d}$

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 16.** *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 14, 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 $\binom{a}{r} = \binom{a+1}{r} - \binom{a}{r-1}$

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

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

▶ **Lemma 17.** *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

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.

▶ **Lemma 18.** *Let G be a graph and T be a binary 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.*

Algorithm 4 Reach(G, T, u, v).

```
Input: $\langle G, T, v, u\rangle$
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 $\mathsf{pos}_{t_0}(u)$
     to 1)
4  for every leaf $f$ in $\mathsf{SEQ}_{\mathsf{root}(T),n}$ in order do
5      Let $i$ be the iteration number
6      Reset all the bits of $R_{i \bmod 2}$ to 0
7      Let $t_{i \bmod 2} \leftarrow f$
8      for all $x$ marked in $R_{(i-1) \bmod 2}$ and all $y$ in $V_f$ do
9          if $(x, y)$ is an edge in $G$ OR $x = y$ then
10             Mark $y$ in $R_{i \bmod 2}$ (by setting the bit at position $\mathsf{pos}_{t_{i \bmod 2}}(y)$ to 1)
11         endif
12     endfor
13 endfor
14 If $v$ is marked return 1; otherwise return 0.
```

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 $p_1, \dots, p_k$ such that the edges in p_i all have same label and label of edges in p_{i+1} is different form 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 decomposition. Let l_i be the length of path p_i. Since $l_1 + l_2 + \dots + l_k \leq d$, by Lemma 14, there exists a subsequence $\langle c_{i_1}, c_{i_2}, \dots, c_{i_k}\rangle$ 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 19.** *On input of a graph G with n vertices and its tree decomposition T with treewidth 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 18 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 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 16, 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 13 follows from Lemma 19. Combining Theorem 13 and Theorem 4 we get the proof of Theorem 1. Theorem 2 follows in a similar way. We use the separator algorithm which exists due to the hypothesis of Theorem 2 as subroutines instead of the algorithm of Lemma 5 in lines 1 and 2 of the Algorithm 1. The rest of the analysis is similar.

References

1 Noga Alon, Paul Seymour, and Robin Thomas. A separator theorem for nonplanar graphs. *Journal of the American Mathematical Society*, 3(4):801–808, 1990.

2 Stefan Arnborg, Derek G. Corneil, and Andrzej Proskurowski. Complexity of Finding Embeddings in a k-Tree. *SIAM Journal on Algebraic Discrete Methods*, 8(2):277–284, 1987.

3 Stefan Arnborg and Andrzej Proskurowski. Linear time algorithms for NP-hard problems restricted to partial k-trees. *Discrete applied mathematics*, 23(1):11–24, 1989.

4 Tetsuo Asano and Benjamin Doerr. Memory-Constrained Algorithms for Shortest Path Problem. In *Proceedings of the 23rd Annual Canadian Conference on Computational Geometry (CCCG 2011)*, 2011.

5 Tetsuo Asano, David Kirkpatrick, Kotaro Nakagawa, and Osamu 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)*, pages 45–56, 2014.

6 Greg Barnes, Jonathan F. Buss, Walter L. Ruzzo, and Baruch Schieber. A Sublinear Space, Polynomial Time Algorithm for Directed s-t Connectivity. *SIAM Journal on Computing*, 27(5):1273–1282, 1998.

7 Hans L. Bodlaender and Arie M. C. A. Koster. Combinatorial Optimization on Graphs of Bounded Treewidth. *The Computer Journal*, 51(3):255–269, 2008.

8 Diptarka Chakraborty, Aduri Pavan, Raghunath Tewari, N. V. Vinodchandran, and Lin 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)*, pages 585–595, 2014.

9 Diptarka Chakraborty and Raghunath Tewari. An $O(n^\epsilon)$ Space and Polynomial Time Algorithm for Reachability in Directed Layered Planar Graphs. *ACM Transactions on Computation Theory (TOCT)*, 9(4):19:1–19:11, 2017.

10 Bireswar Das, Samir Datta, and Prajakta Nimbhorkar. Log-Space Algorithms for Paths and Matchings in k-Trees. *Theory of Computing Systems*, 53(4):669–689, 2013.

11 Michael Elberfeld, Andreas Jakoby, and Till Tantau. Logspace Versions of the Theorems of Bodlaender and Courcelle. In *Proceedings of the 51st Annual Symposium on Foundations of Computer Science (FOCS 2010)*, pages 143–152. IEEE Computer Society, 2010.

12 Uriel Feige, MohammadTaghi Hajiaghayi, and James R. Lee. Improved Approximation Algorithms for Minimum Weight Vertex Separators. *SIAM Journal on Computing*, 38(2):629–657, 2008.

13 J. Gilbert, D. Rose, and A. Edenbrandt. A Separator Theorem for Chordal Graphs. *SIAM Journal on Algebraic Discrete Methods*, 5(3):306–313, 1984.

14 Tatsuya Imai, Kotaro Nakagawa, Aduri Pavan, N. V. Vinodchandran, and Osamu 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*, pages 277–286, 2013.

15 Andreas Jakoby and Till 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)*, pages 216–227, 2007.

16 Ken-ichi Kawarabayashi and Bruce Reed. A separator theorem in minor-closed classes. In *Proceedings of the 51st Annual Symposium on Foundations of Computer Science (FOCS 2010)*, pages 153–162. IEEE, 2010.

17 Omer Reingold. Undirected connectivity in log-space. *Journal of the ACM (JACM)*, 55(4):17, 2008.

18 Walter J Savitch. Relationships between nondeterministic and deterministic tape complexities. *Journal of Computer and System Sciences*, 4(2):177–192, 1970.

19 Avi Wigderson. The complexity of graph connectivity. In *Proceedings of the 17th International Symposium on Mathematical Foundations of Computer Science (MFCS 1992)*, pages 112–132. Springer, 1992.