Compact Representation of Graphs with Small Bandwidth and Treedepth

Shahin Kamali

University of Manitoba
Winnipeg, MB, R3T 2N2, Canada
shahin.kamali@umanitoba.ca

Abstract

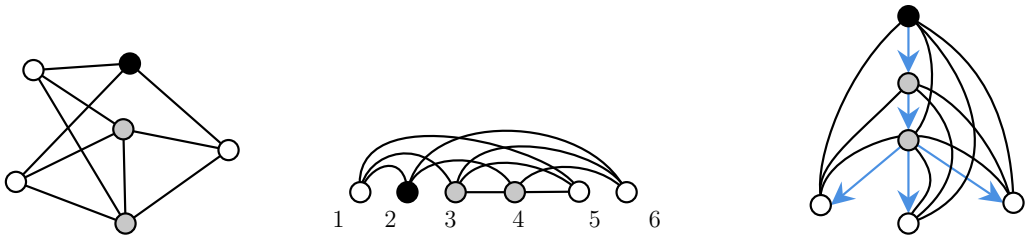
We consider the problem of compact representation of graphs with small *bandwidth* as well as graphs with small *treedepth*. These parameters capture structural properties of graphs that come in useful in certain applications. We present simple navigation oracles that support degree and adjacency queries in constant time and neighborhood query in constant time per neighbor. For graphs of bandwidth k, our oracle takes $(k+\log k)n+o(kn)$ bits. By way of an enumeration technique, we show that $(k-6\sqrt{k}-12)n-O(k^{3/2})$ bits are required to encode a graph of bandwidth k and size n. For graphs of treedepth k, our oracle takes $(k+\log k+2)n+o(kn)$ bits. We present a lower bound that certifies our oracle is succinct for certain values of $k \in o(n)$.

1 Introduction

Graphs are among the most relevant ways to model relationship between objects. Given the ever-growing number of objects to model, it is increasingly important to store the underlying graphs in a compact manner in order to facilitate designing efficient algorithmic solutions. One simple way to represent graphs is to consider them as random objects without any particular structure. There are two issues, however, with such general approach. First, many computational problems are NP-hard on general graphs and often remain hard to approximate. Second, random graphs are highly incompressible and cannot be stored compactly [1]. At the same time, graphs that arise in practice often have combinatorial structures that can -and should- be exploited to provide space-efficient representations.

Width parameters are ways to partition graphs into families that share certain structural properties. As an example, the most well-known width parameter, treewidth, measures how far a graph is from a tree. Many computationally hard problems are easy-to-solve for graphs with small treewidth, thanks to their tree-like structure. Since the successful introduction of treewidth by Robertson and Seymour [2], many other width parameters have been presented and studied. These parameters are proved useful for many applications, specially for the design of efficient algorithms. As such, it is desirable to have efficient data structure that adapt to these width parameters. In this paper, we focus on graphs of small bandwidth and graphs of small treedepth.

Given a graph $G = (V, E)$ of size n, a *bandwidth labeling* f of G is an assignment of distinct integers from 1 to n to vertices of G. The width of the labeling f is



(a) Graph G (b) A labelling with bandwidth 4 (c) A mapping with treedepth 3

Figure 1: (a) A graph G of size $n = 5$ (b) A bandwidth representation of G (b) A treedepth representation of G (c). Vertices of the same color are indistinguishable.

the maximum value of $|f(u) - f(v)|$ where $(u, v) \in E$. The bandwidth of G is the minimum bandwidth taken over all labelings. If G has bandwidth at most k, the rows and columns in its adjacency matrix can be permuted in a way that all non-zero entries lie in "band" of width k along the diagonal. Having a bounded bandwidth is a useful property with applications that range from code correction [3] to VLSI design [4] and Gaussian elimination [5].

Treedepth is another width parameter than has been studied under various names in the literature. These names include "vertex ranking number", "ordered chromatic number", and "minimum elimination tree height". Given a graph $G = (V, E)$ of size n, a *treedepth mapping* f of G maps vertices of G to nodes of a rooted tree T with the property that for any edge $(u, v) \in E$, either $f(v)$ is an ancestor of $f(u)$ in T or vice-versa. The depth of the mapping f is the height of the tree T. The treedepth of G is the minimum treedepth taken over all mappings. Intuitively speaking, treedepth measures how far a graph is to a star tree (a tree of diameter 2). Treedepth appears in mathematical applications such as non-repetitive coloring [6]; we refer the reader to [7] for a survey on treedepth. Figure 1 provides an illustration of bandwidth and treedepth.

Although related, bandwidth and treedepth are not directly related. For example, a star tree on n vertices has bandwidth $\Theta(n)$ and treedepth 2. On the other hand, a path graph has bandwidth 1 and treedepth $\Theta(\log n)$. Generally speaking, bandwidth and treedepth are more restrictive parameters than other width parameters such as treewidth. As an example, basic graph families such as stars, outerplanar graphs and series-parallel graphs have constant treewidth but upto logarithmic treedepth. In this paper, we assume the underlying graphs have small but not necessarily constant bandwidth/treedepth k such that $k \in o(n)$.

Throughout the paper, we assume the graphs are simple, undirected and unweighted. We assume a word RAM model where size of a word is $\Omega(\log n)$. This is a standard assumption that implies a vertex can be distinguished, in constant time, with a label that fits in word of RAM.

1.1 Contribution

We present simple and compact navigation oracles for graphs of bounded bandwidth and treedepth that support adjacency, degree, and neighborhood queries in constant

time. Given a pair of vertices, adjacency query reports whether u and v are connected. Degree query reports the number of neighbors of a given vertex. Neighborhood query reports all neighbors of a given vertex in constant time per neighbor.

Given a graph $G = (V, E)$ of size n and bandwidth k, our navigation oracle stores G in $(k + \log k)n + o(kn)$ bits. By way of an enumeration argument, we show that $(k - 6\sqrt{k} - 12)n - O(k^{3/2})$ bits are required to distinguish graphs of bandwidth k and size n. Hence, both our upper and lower bounds are tight within lower order term for graphs of small (but not constant) bandwidth.

For graphs of size n and treedepth k, our navigation oracle takes $(k + \log k + 2)n$ bits. Meanwhile, we show $(k-1)n - k^2 - o(kn)$ bits are required if $k \in \omega(\log n)$. When $k \in O(\log n)$, we show a lower bound of $k \log n - O(k \log k)$ bits. So, our navigation oracle is succinct for certain values of k, e.g., when k is polylogarithmic to n.

1.2 Related work

Compact graph representation. Compact representations for graphs with various combinatorial structures have been presented in the past. The studied graphs include but are not limited to: separable graphs [8, 9], planar graphs (e.g., [10, 11]), interval graphs [12], graphs of bounded treewidth [13], and graphs of bounded cliquewidth [14].

Blelloch and Farzan [9] provided a succinct representation of separable graphs that supports navigation operations in constant time. Graphs of constant bandwidth satisfy the graph separator theorem as they are closed under taking minors and have a separator of constant size. Hence, the oracle of [9] can be used to represent this family of graphs. Such oracle works by recursively representing smaller components after removing separators. In the base of the reduction, the oracle write answers to queries for all "micro-graphs" of size $O(\log n / \log \log n)$ in a look-up table. Forming such lookup table requires enumerating graphs of size $O(\log n / \log \log n)$. Such enumeration, however, is not easy in practice. In fact, it is not even clear how many such graphs exist. Note that graphs of non-constant bandwidth are not separable.

Another relevant work is that of Farzan and Kamali [13] in which a navigation oracle for graphs of bounded treewidth was presented. Their oracle answers all queries in constant time and takes $kn + O(n)$ bits. Both bandwidth and treedepth are bounded above by the pathwidth and consequently the treewidth. As such, the oracle of [13] can be used to encode bandwidth and treedepth representations. This oracle, however, is rather complex, and the constants involved in $O(n)$ makes it undesirable (certainly not succinct) for graphs of constant bandwidth or treedepth.

Bandwidth/Treedepth computation. Finding the labeling with minimum bandwidth is NP-hard [15]. On the other hand, the optimal labelling can be found in $O(n^k)$ [16]. There are also polynomial time algorithms for finding the exact or approximate bandwidth of special graph families such as interval graphs and convex-bipartite graphs [17]. Given the practical significance of the problem, there has been a rich line of research for providing algorithms that perform well in practice, e.g., Cuthill–McKee algorithm [18] and its variants. Finding the treedepth of a graph is also NP-hard [19].

There is a polynomial algorithm with approximation factor of $O(\log^2 n)$ [20]. There are also polynomial time algorithms for finding the exact or approximate treedepth of special graph families such as trees [21] and interval graphs [22]. When designing our oracles, we assume a graph G and a labeling of it with bandwidth k is provided.

2 Bandwidth representation

In this section, we consider compact representation of graphs with small bandwidth. First, we present a lower bound and later we introduce a simple navigation oracle.

Lower bound. Our lower bound argument has the following structure. First, we introduce a fixed labelled structure, named "extended-comb", that is used to define a family of graphs named "thicket graphs". We will show that these graphs have bounded bandwidth. Extended-comb is defined in a way that vertices are almost distinguishable in the unlabeled thicket graphs. As such, we can limit double-countings when studying the number of thicket graphs. The argument follows by a rather simple counting argument for the number of thicket graphs, which indeed provides a lower bound for the number of graphs with bounded bandwidth.

An *extended-comb*, simply an *excomb*, graph of size n has integer parameters (α, c) such that $\alpha > c$. Throughout, we assume n is divisible by $\alpha + 1$.

Definition 1. *An (α, c)-excomb graph $\chi = (V, E_\chi)$ of size n is a labelled graph defined as follows. We have $V = \{P \cup Q^0 \cup Q^1 \cup \ldots \cup Q^\beta\}$, where $\beta = n/(\alpha+1)$. Subset P has size β vertices, referred to as* prime *verties, which are labelled as $p_1, p_2, \ldots, p_\beta$. The subgraph induced by P is a path of length β. Any subset Q^i has size α, and the subgraph induced by each Q^i is a path in which vertices are labelled as $q^i_1, \ldots, q^i_\alpha$ from one endpoint to another. Moreover, for any $j \geq 1$ with $j \in (i-c, i+c]$ and for any $x \in Q^j$, we have $(p_i, x) \in E_\chi$.*

Figure 2 provides an illustration of excomb graphs. In an (α, c)-excomb graph, any prime vertex p_i for $i \in [c, \beta - c]$ has degree $2\alpha c + 2$; this is because any p_i is connected to all vertices in Q_j for all $j \in [i-c, i+c]$ (there are $2\alpha c$ such vertices); counting p_{i-1} and p_{i+1}, we get a degree of $2c\alpha + 2$ for p_i. We call vertices like p_i as *middle* prime vertices; this is because they are at distance i or more from the endpoints of the path induced by prime vertices. An (α, c)-thicket graph is an (α, c)-excomb graph to which a set of extra edges are added. These edges connect vertices in Q_j to those in $Q_{j'}$ only if $|j - j'| \leq c - 1$:

Definition 2. *A labelled (α, c)-thicket graph of size n is a graph $G = (V, E)$ such that I) there is a (α, c)-excomb graph $\chi = (V_\chi = \{P \cup Q^1 \cup \ldots \cup Q^\beta\}, E_\chi)$ that spans G, that is, $V = V_\chi$. II) Any edge $e \in E - E_\chi$ connects two vertices $x \in Q^j$ and $x' \in Q^{j'}$ such that $|j - j'| \in [1, c-1]$. An unlabeled graph is an (α, c)-thicket graph if there is a labeling of its vertices that satisfy the above properties.*

Lemma 1. *Any (α, c)-thicket graph G has bandwidth at most $c(\alpha + 1)$.*

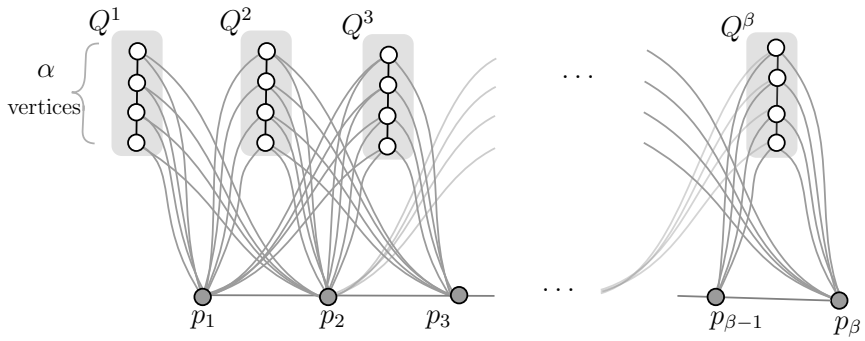


Figure 2: An (α, c)-excomb graph with $\alpha = 4$ and $c = 2$.

Proof. Let $\chi = (V = P \cup Q^1 \cup \dots Q^\beta, E_\chi)$ be any excomb graph that spans G; vertices in P are ordered as $p_1, \dots, p_\beta$ from one endpoint of their induced path to another, and similarly those of Q^i are labelled as $q^i_1, q^i_2, \dots, q^i_\alpha$. Consider a bandwidth labeling f of vertices in the following order:

$$(q^1_1, q^1_2, \dots, q_\alpha)^1, p_1, (q^2_1, q^2_2, \dots, q^2_\alpha), p_2, \dots, p_{\beta-1}, (q^\beta_1, q^\beta_2, \dots, q^\beta_\alpha)$$

So, any prime vertex p_i receives a label $f(p_i) = (\alpha + 1)i$. We show this labeling certifies a bandwidth of $c(\alpha + 1)$ for G. Let $e = (x, y)$ be an edge of G such that $f(x) < f(y)$; we have $x \in Q^i \cup \{p_i\}$ for some value of i and $y \in Q^j \cup \{p_j\}$ for some value of j such that $j \geq i$. Meanwhile, since x and y are connected, it holds that $j - i \leq c + 1$. Moreover, we have $f(x) \geq f(p_{i-1})$ and $f(y) \leq f(p_j)$ (to be consistent when $i = 1$ assume $p(0) = 0$). We can write, $f(y) - f(x) \leq f(p_j) - f(p_{i-1}) \leq f(p_{i+c-1}) - f(p_{i-1}) = (\alpha + 1)(i + c - 1) - (\alpha + 1)(i - 1) = c(\alpha + 1)$. □

Lemma 2. *Given an (α, c)-thicket graph $G = (V, E)$ of size n, there are at most $2^{n/(\alpha+1)+1}$ different labeling of vertices of G that result in a labelled thicket graph.*

Proof. We describe a process that labels all vertices of G and, on the meantime, count how many possible labeling exist.

First, consider vertices of G that have degree $d = 2c\alpha + 2$. In any labeling, middle prime vertices have such degree; this is because middle prime vertices in the excomb graph have degree d and extra edges in the ticket graph connect non-prime vertices. Moreover, any other vertex in G has degree strictly less than d: the degree of other prime vertices is equal to their degree in the excomb graph which is less than middle prime vertices. Meanwhile, any non-prime vertex x is connected to at most $2(c-1)\alpha$ other non-prime vertices and at most $2c$ prime vertices; this gives a degree of at most $2c\alpha + 2(c - \alpha)$ for x, which is less than d since $c < \alpha$.

From the above discussion, vertices in G with degree d are exactly middle prime vertices. We label these vertices as $p_c, p_{c+1}, \dots, p_{\beta-c}$ from one endpoint of the path they induce to the other endpoint. Note that there are two possible ways for such labeling. For any $i \in [c + 1, \beta - c]$, a vertex v is in Q_i iff it is a common neighbor of p_{i-c} and p_{i+c}. So, given the labelling of middle prime vertices, we can distinguish vertices in Q^i for $i \in [2c, \beta - 2c]$. For any such Q^i, we label vertices as $Q^i_1, \dots Q^i_\alpha$ from one endpoint of they path induce to another.

The remaining unlabeled vertices belong to the set $L \cup R$, where $L = \{p_1, \ldots, p_{c-1}\} \cup Q^1 \cup \ldots \cup Q^c$ and $R = \{p_{\beta-c+1}, \ldots, p_\beta\} \cup Q^{\beta-c+1} \cup Q^\beta$; these are vertices on the extreme left and right positions in an orientation of prime vertices from left to right. To label vertices in L, we apply the following process for all values of $y = \in \{c, c-1, c-2, \ldots, 1\}$; initially $y = c$. Any vertex u in Q^y is adjacent to p_{y+c}; meanwhile, by definition of excomb and thicket graphs, other vertices of L are not connected to p_{y+c}. So, we can detect and remove vertices of Q^y from L. After detecting vertices in Q^y, we label them from one endpoint of the path they induce to the other. Similarly, p_{y-1} is connected to any vertex $u \in Q^{y+x}$, while the remaining vertices of L are not connected to u. So, by checking neighbors of u, we can detect p_{y-1} and remove it from L. Repeating this process for all values of y (in the indicated order) enables us to distinguish the partition P or Q_j that each vertex of L belongs to. A similar procedure can be applied to distinguish and label vertices that belong to R.

In summary, given a graph G, there is a unique way to partition its vertices into a subsets $P \cup Q^1 \cup \ldots \cup Q^\beta$ that is consistent with an excomb labeling. For each of these partitions, there are two ways to label vertices in the partition from one endpoint of the path they induce to the other. So, there are $2^{\beta+1} = 2^{n/(\alpha+1)+1}$ ways to label vertices of G. □

Provided with Lemma 2, we can count the number of unlabeled (α, c)-thicket graphs. By Lemma 1, these graphs have bounded bandwidth, and consequently, we can find a lower bound for the number of graphs with a given bandwidth.

Lemma 3. *Assume $k \in o(n)$. In order to represent any graph of size n and bandwidth k, at least $(k - 6\sqrt{k} - 12)n - \Theta(k^{3/2})$ bits are required.*

Proof. The proof works by counting the number of (α, c)-thicket graphs of size n. In particular, we use Lemma 2 to find a lower bound for the number of unlabeled (α, c)-thicket graphs. In a special setting where α and c are both roughly equal to $\sqrt{k}$, Lemma 1 the (α, c)-thicket graphs that we count have bandwidth at most k, and this will give the desired lower bound.

Let χ be a labelled (α, c)-excomb graphs in which vertices are partitioned as $P \cup Q_1 \cup \ldots Q_\beta$, where vertices in P are labelled as $p_1, p_2, \ldots, p_\beta$, and vertices in each Q_i are labelled as $Q_i^1, Q_i^2, \ldots, Q_i^\alpha$. These orderings are consistent with the path each of these partitions, that is, vertices are labelled from one endpoint to another). In what follows, we count the number of labelled thicket graphs which have χ as their spanning excomb. Later, we count how many ways we can label an unlabeled graph.

Consider any partition Q^j for $j \geq c$. There is an edge between a vertex edge in Q_j and some other partition $j' < j$ only if $j - j' \leq c - 1$. There are $c - 1$ partitions like $Q_{j'}$, and each include α labelled vertices. So, for each vertex of Q_j, there are $(c-1)\alpha$ potential edges that find their other endpoint in some $Q_{j'}$ where $j' < j$. Over all vertices of Q_j, there will be $(c-1)\alpha^2$ edges. Note that these edges are distinct from each other in the labelled graph. So, in total, the presence or absence of edges between vertices of Q_j and partitions like Q'_j where $j' < j$ define $2^{(c-1)\alpha^2}$ possibilities labelled graphs. There are $\beta - c + 1$ partitions like Q_j (since $j \geq c$). So, overall partitions, there are $2^{(c-1)\alpha^2(\beta-c+1)}$ graphs. By Lemma **??**, each unlabelled graph has

$2^{\beta+1}$ different labeling. So, the number of unlabeled (α, c)-thicket graphs will be at least $2^{(c-1)\alpha^2(\beta-c+1)-(\beta+1)}$. So, in order to distinguish (α, c)-thicket graphs, we need $B = (c-1)\alpha^2(\beta-c+1)-(\beta+1)$ bits.

Let $k' \leq k$ be the largest value that can be written as $w(w+2)$ for some integer w; note that $k < (w+1)(w+3) < (w+2)^2$, that is $w > (\sqrt{k}-2)$. Let $c = w$ and $\alpha = w+1$. By Lemma 1, the pathwidth of the (α, c)-graph is $c(\alpha+1) = k' \leq k$. Meanwhile, $\beta = n/(\alpha+1) = n/(w+2)$. Replacing in B, we conclude that the number of bits to represent thicket graphs of bandwidth k is:

$$
\begin{aligned}
B &= (c-1)\alpha^2(\beta-c+1)-(\beta+1) \\
&= (w^2-1)n - \frac{w^2 n}{w+2} - \frac{(w-1)^2(w+1)^2}{w+2} - 1 \\
&= (w^2-1)n - \frac{w^2 n}{w+2} - \Theta(k^{3/2}) && \text{because } w = \Theta(\sqrt{k}) \\
&> (k-4w-4)n - \frac{w^2 n}{w+2} - \Theta(k^{3/2}) && k < (w+1)(w+3) \text{ or } w^2 > k-4w-3 \\
&> kn - 6w - \Theta(k^{3/2}) > (kn - 6\sqrt{k} - 12)n - \Theta(k^{3/2})
\end{aligned}
$$

□

Navigation oracle. we provide a simple oracle that takes $O(kn)$ bits to represent a graph G of size n and bandwidth k, and supports navigation queries in constant time. Given a binary matrix M, *access(i, j)* returns the entry at index (i, j), *r-successor(i, j)* returns the index of the column that contains the next '1' after column j in row i, and *c-successor*(i, j) is defined identically on columns. Farzan and Munro [23] provide a representation of an $n \times n$ matrix in $n^2 + o(n^2)$ that supports *access(i, j)*, *r-successor(i, j)* and *c-successor*(i, j) in constant time and *r-rank(i, j)* in constant time, $(1+\epsilon)kn$ bits are required. Their approach in presenting these matrices is efficient and easy-to-code. The same approach can be applied to represent a $k \times n$ matrix.

Lemma 4. *[23] A $k \times n$ binary matrix can be presented in $kn + o(kn)$, with the support of the following queries in constant time:* access(i, j), r-successor(i, j) *and* c-successor(i, j).

Let f be a function that maps vertices of G to integers 1 to n and certifies a bandwidth of at most k. We refer to each vertex with its index in the bandwidth labeling. Let $S(u) = \{x | u - x \leq k\}$, that is, S is the set of vertices that precede u in the ordering and have distance at most k with u in that labeling. Let $y \in S(u)$ be a vertex such that $y \mod k = i$. Note that there is exactly one such y. We call y the i'th *forerunner* of u.

Our navigation oracle has two components. (I) a binary matrix of size $k \times n$, in which the u'th column is associated with vertex u. The entry (i, u) in M is '1' iff there is an edge between u and its i'th forerunner. We store M using the structure of Lemma 4 (II) a simple array of size $n \log k + n$ in which there is an entry of size $\log 2k$ for each vertex of u, indicating the degree of u. Note that the degree of each vertex is at most $2k$ and hence $\log k + 1$ bits are sufficient to store it. Provided with this array,

degree query becomes trivial. Next, we describe how other queries are supported:
adjacency: recall that vertices are referred with their labels in bandwidth ordering. Let u and v be labels of two vertices. W.l.o.g. assume $u > v$. If $u - v > k$, then there is no vertex between u and v. Otherwise, v is the $q = v \mod k$ ancestor of u. In order to answer adjacency request, it suffices to return *access*(q, u).
Neighborhood: we want to report neighbors of a vertex u. First, we report neighbors of u that are in $S(u)$. This can be done in $O(1)$ time per neighbor by successively applying *c-successor* query on the u'th column of M. Let (i, u) be a '1'-entry, that is, u is connected to its i'th forerunner. Let $q = u \mod k$. If $q > i$, then we report vertex $x = u - (q - i)$; note that $x \in S(u)$ and $x \mod k = q'$. Otherwise, x will come after u in the bandwidth labeling; in this case, we report $x - k$, which is indeed in $S(u)$. Next, we how to report neighbors of u that come after u in the bandwidth ordering. Let $q = u \mod k$. Any '1'-entry at index (q, j) of M is associated with a neighbor of u if $j \leq u + k$. Any such neighbor j can be reported in $O(1)$ by successive application *r-successor* query on the row q of M.

From the above discussion, we conclude the following:

Theorem 1. *Given a graph of size n and treewidth k, an oracle is constructed to answer degree, adjacency, and neighborhood queries in constant time. The storage requirement of the oracle is $(k + \log k)n + o(kn)$.*

3 Treedepth representation

In this section, we consider compact representation of graphs of size n and treedepth $k \in o(n)$. We start with the following lower bound:

Theorem 2. *At least $k \log n - O(k \log k)$ bits are necessary to encode any graph of size n and treedepth $k \in O(\log n)$. When $k \in \omega(\log n)$, at least $(k-1)n - k^2 - o(kn)$ bits are required.*

Proof. Any bipartite graph G with k vertices on its left and $n - k$ vertices on its right has treedepth of at most k: consider a tree T formed by a path $x_1, x_2, \ldots, x_k$, where x_1 is the roote of T, such that the $n - k$ nodes $y_1, \ldots, y_{n-k}$ are connected to x_k. Mapping the k nodes on the left to x_i's and the other $n - k$ nodes to y_i's gives a valid tree mapping of depth k. So, in order to find a lower bound for the number of graphs of bounded treedepth, we just count the number of bipartite graphs. Assuming $k < n - k$, the number of (unlabeled) bipartite graphs of size $k \times (n - k)$ is at least X and at most $2X$, where $X = \binom{n-k+2^k-1}{n-k}/k!$ [24]. We have $X \geq \binom{n}{k}/k! \geq \frac{n^k}{k^k \cdot k!}$, which implies that $\log(X) \geq k \log n - O(k \log k)$. This lower bound is useful when $k \in O(\log n)$. When $k \in \omega(\log n)$, we have $X \geq \binom{2^k}{n-k} \geq \frac{(2^k)^{n-k}}{(n-k)^{n-k}}$. This gives $\log(X) \geq k(n-k) - (n-k)\log n = (k-1)n - k^2 - o(kn)$. □

Navigation oracle. Consider an ordered tree of size n, which is equivalent to a balanced parenthesis sequence of size $2n$. In such tree, each node can be represented by its index in the depth-first order traversal of τ. Now, *is-ancestor*(i, j) query indicates

whether node i is an ancestor of node j, *depth(i)* indicates depth of node i, *depth-ancestor(i, d)* indicates the ancestor at depth d of node i, *lmost-leaf(i)*, *rmost-leaf(i)* respectively indicate the left-most and right-most descendants of node i. Sadankane and Navarro [25] present a fully functional representation of ordered trees which is also easy-to-implement.

Lemma 5. *[25] An ordered tree of size n, can be represented in $2n + o(n)$ bits, with the support of following queries, all in constant time:* is-ancestor(i, j), depth(i), depth-ancestor(i, d), level-ancestor(i, d), lmost-leaf(i), rmost-leaf(i).

We introduce a simple navigation oracle for a graph G of size n and treedepth k. Assume a mapping from vertices of G to nodes of a tree T is provided that certifies a treedepth of k for G. Our oracle has the following components: (I) Tree T is represented as an ordered tree, in $2n+o(n)$ bits, using the structure of Lemma 5. We refer to each vertex by its index in the depth-first traversal of T. (II) A binary matrix M of size $k \times n$, stored in $kn + o(kn)$ bits, using the structure of Lemma 4. The i-th column of the table is associated with the i-th vertex. Let v_i be the i-th node in T. The entry $M[q, i]$ indicates whether there is an edge between v_i and its ancestor in T at depth q. (III) a simple array of size $n \log k$ in which there is an entry of size $\log k$ for each vertex u, indicating the degree of u. Note that the degree of each vertex is at most k. Provided with this array, *degree* query becomes trivial. Next, we explain how other queries are supported:

Adjacency: let v_i, v_j be a pair of vertices with indices i and j, respectively. In order to answer the adjacency query, we first check whether the two vertices have an ancestor-descendant relationship in T. This can be checked in $O(1)$ time using *is-ancestor(i,j)* and *is-ancestor(j,i)* queries. If no vertex is the ancestor of the other, then there is no edge between them. W.l.o.g. assume v_i is an ancestor of v_j. Let q be the depth of v_i in T; this value can be found using *depth(i)* in $O(1)$. There is an edge between v_i and v_j if $M[q, i] = 1$; this can be done in $O(1)$ by calling an *access* operation on M.

Neighborhood: Assume we want to report neighbors of a vertex v_i. First, we report neighbors of v_i that are its ancestors in T. For that, we successively apply *rank-select* operation on the i'th column of M to find the '1'-entries. Let q be the row-index associated with the next '1'. That means there is an edge between v_i and its ancestor at depth q; we can find such ancestor with using *depth-ancestor*(i, q) query in $O(1)$. Next, we describe how to report neighbors of v_i that are its descendant in T. Let q' be the depth of v_i in T; this value can be found in using *depth* query in $O(1)$. The columns in M associated with the potential descendant-neighbors of v_i have indices in the range $[L, R]$, where L and R are the indices of the left-most and right-most descendants of v_i in T, respectively. The values of L and R can be found in $O(1)$ using *lmost-leaf(i)* and *rmost-leaf(i)* queries in T. Provided with values of L and R, reporting descendant-neighbors of v can be done by successive application of *column-select* in the row q' of M to find the '1'-entries in the range $[L, R]$. Let $j \in [L, R]$ be one such '1'-entry. Since v_j is a descendant of v_i, and v_i has depth q', there is an edge between v_j and v_i. So, v_j is reported.

Theorem 3. *Given a graph of size n and treedepth k, an oracle is constructed to answer degree, adjacency, and neighborhood queries in constant time. The storage requirement of the oracle is $(k + \log k + 2)n + o(kn)$.*

References

[1] Daniel K. Blandford, Guy E. Blelloch, and Ian A. Kash, "Compact representations of separable graphs," 2003, pp. 679–688.

[2] Neil Robertson and Paul D. Seymour, "Graph minors. III. planar tree-width," *J. Comb. Theory, Ser. B*, vol. 36, no. 1, pp. 49–64, 1984.

[3] L. H. Harper, "Optimal assignments of numbers to vertices," *Journal of the Society for Industrial and Applied Mathematics*, vol. 12, no. 1, pp. 131–135, 1964.

[4] Dominique Barth, François Pellegrini, André Raspaud, and Jean Roman, "On bandwidth, cutwidth, and quotient graphs," *ITA*, vol. 29, no. 6, pp. 487–508, 1995.

[5] B. Monien and H. Sudborough, *Embedding one Interconnection Network in Another*, pp. 257–282, 1990.

[6] F. Fiorenzi, P. Ochem, P. O. de Mendez, and X. Zhu, "Thue choosability of trees," *Discrete Applied Mathematics*, vol. 159, no. 17, pp. 2045–2049, 2011.

[7] Jaroslav Nesetril and Patrice Ossona de Mendez, "On low tree-depth decompositions," *Graphs and Combinatorics*, vol. 31, no. 6, pp. 1941–1963, 2015.

[8] Daniel K. Blandford, Guy E. Blelloch, and Ian A. Kash, "Compact representations of separable graphs," in *Proc. SODA*, 2003, pp. 679–688.

[9] Guy E. Blelloch and Arash Farzan, "Succinct representations of separable graphs," in *Proc. CPM*, 2010, pp. 138–150.

[10] Kenneth Keeler and Jeffery Westbrook, "Short encodings of planar graphs and maps," *Discrete Appl. Math.*, vol. 58, pp. 239–252, 1995.

[11] J. I. Munro and V. Raman, "Succinct representation of balanced parentheses, static trees and planar graphs," in *Proc. FOCS*, 1997, pp. 118–126.

[12] Hüseyin Acan, Sankardeep Chakraborty, Seungbum Jo, and Srinivasa Rao Satti, "Succinct data structures for families of interval graphs," in *WADS'19*, 2019, pp. 1–13.

[13] Arash Farzan and Shahin Kamali, "Compact navigation and distance oracles for graphs with small treewidth," *Algorithmica*, vol. 69, no. 1, pp. 92–116, 2014.

[14] Shahin Kamali, "Compact representation of graphs of small clique-width," *Algorithmica*, vol. 80, no. 7, pp. 2106–2131, 2018.

[15] Christos H. Papadimitriou, "The NP-completeness of the bandwidth minimization problem," *Computing*, vol. 16, no. 3, pp. 263–270, 1976.

[16] E. M. Gurari and I. H. Sudborough, "Improved dynamic programming algorithms for bandwidth minimization," *J. Algorithms*, vol. 5, no. 4, pp. 531–546, 1984.

[17] A. S. Shrestha, S. Tayu, and S. Ueno, "Bandwidth of convex bipartite graphs and related graphs," *Inf. Process. Lett.*, vol. 112, no. 11, pp. 411–417, 2012.

[18] E. Cuthill and J. McKee, "Reducing the bandwidth of sparse symmetric matrices," in *Proceedings of the 1969 24th National Conference*, 1969, ACM '69, pp. 157–172.

[19] Alex Pothen, "The complexity of optimal elimination trees," *Technical Report*, 1988.

[20] H. L. Bodlaender, J. S. Deogun, K. Jansen, T. Kloks, D. Kratsch, H. Müller, and Z. Tuza, "Rankings of graphs," *J. Disc. Math.*, vol. 11, no. 1, pp. 168–181, 1998.

[21] Alejandro A. Schäffer, "Optimal node ranking of trees in linear time," *Inf. Process. Lett.*, vol. 33, no. 2, pp. 91–96, 1989.

[22] B. Aspvall and P. Heggernes, "Finding minimum height elimination trees for interval graphs in polynomial time," *BIT Num. Math.*, vol. 34, pp. 484–509, 01 1994.

[23] Arash Farzan and J. Ian Munro, "Succinct encoding of arbitrary graphs," *Theor. Comput. Sci.*, vol. 513, pp. 38–52, 2013.

[24] A. Atmaca and A. Y. Oruc, "On the number of unlabeled bipartite graphs," 2017.

[25] Gonzalo Navarro and Kunihiko Sadakane, "Fully functional static and dynamic succinct trees," *ACM Transactions on Algorithms*, vol. 10, no. 3, pp. 16:1–16:39, 2014.