Succinct Data Structure for Chordal Graphs with Bounded Vertex Leafage

Girish Balakrishnan
Indian Institute of Technology Madras, Chennai, India

Sankardeep Chakraborty
University of Tokyo, Japan

N. S. Narayanaswamy
Indian Institute of Technology Madras, Chennai, India

Kunihiko Sadakane
University of Tokyo, Japan

Abstract

Chordal graphs is a well-studied large graph class that is also a strict super-class of path graphs. Munro and Wu (ISAAC 2018) have given an $(n^2/4 + o(n^2))-$bit succinct representation for $n-$vertex unlabeled chordal graphs. A chordal graph $G = (V, E)$ is the intersection graph of sub-trees of a tree T. Based on this characterization, the two parameters of chordal graphs which we consider in this work are *leafage*, introduced by Lin, McKee and West (Discussiones Mathematicae Graph Theory 1998) and *vertex leafage*, introduced by Chaplick and Stacho (Discret. Appl. Math. 2014). Leafage is the minimum number of leaves in any possible tree T characterizing G. Let $L(u)$ denote the number of leaves of the sub-tree in T corresponding to $u \in V$ and $k = \max_{u \in V} L(u)$. The smallest k for which there exists a tree T for G is called its vertex leafage.

In this work, we improve the worst-case information theoretic lower bound of Munro and Wu (ISAAC 2018) for $n-$vertex unlabeled chordal graphs when vertex leafage is bounded and leafage is unbounded. The class of unlabeled $k-$vertex leafage chordal graphs that consists of all chordal graphs with vertex leafage at most k and unbounded leafage, denoted $\mathcal{G}_k$, is introduced for the first time. For $k > 0$ in $o(n^c), c > 0$, we obtain a lower bound of $((k-1)n \log n - kn \log k - O(\log n))-$bits on the size of any data structure that encodes a graph in $\mathcal{G}_k$. Further, for every $k-$vertex leafage chordal graph G and $k > 1$ in $o(n^c), c > 0$, we present a $((k-1)n \log n + o(kn \log n))-$bit succinct data structure, constructed using the succinct data structure for path graphs with $(k-1)n$ vertices. Our data structure supports adjacency query in $O(k \log n)$ time and using additional $2n \log n$ bits, an $O(k^2 d_v \log n + \log^2 n)$ time neighbourhood query where d_v is degree of $v \in V$.

2012 ACM Subject Classification Information systems → Data structures

Keywords and phrases succinct data structure, chordal graphs, leafage, vertex leafage, path graphs

Digital Object Identifier 10.4230/LIPIcs.SWAT.2024.4

Related Version *Full Version*: `https://arxiv.org/abs/2402.03748`

Funding *Sankardeep Chakraborty*: supported by MEXT Quantum Leap Flagship Program (MEXT Q-LEAP) Grant Number JPMXS0120319794.

1 Introduction

A data structure for a graph class $\mathcal{G}$ of graphs with $n-$vertices is succinct if it takes $(\log |\mathcal{G}| + o(\log |\mathcal{G}|))-$bits of space; here $|\mathcal{G}|$ denotes the number of $n-$vertex graphs in $\mathcal{G}$. A succinct representation for graph $G = (V, E)$ in $\mathcal{G}$ is expected to support the following queries for each pair of vertices $u, v \in V$:

- `adjacency`(u, v): returns "YES" if and only if vertices u and v are adjacent in G.
- `neighborhood`(u): returns all the vertices in V that are adjacent to vertex u.
- `degree`(u): returns the number of vertices adjacent to vertex u.

The earliest work on space-efficient data structures for graph classes was by Itai and Rodeh [16] for vertex labeled planar graphs and by Jacobson [17] for class of static unlabeled trees and planar graphs. In recent years, succinct data structures for intersection graphs is getting lot of attention. A few of them are, Acan et al. [2] for interval graphs, Acan et al. [1] for families of intersection graphs on circle, Munro and Wu [22] for chordal graphs and Balakrishnan et al. [4] for path graphs. There are also other results on representation of chordal graphs, for instance, the space-efficient representation for chordal graphs by Markenzon et al. [19]. This shows that representation of chordal graphs are of interest to the research community.

A recent paper by Chakraborty and Jo [6] improve the information theoretic lower bound for interval graphs as given by Acan et al. [2] by bounding maximum degree. For bounded maximum degree Δ in $O(n^\epsilon)$ where $1 < \epsilon < 1$, they give a lower bound of $(\frac{1}{6}n\log\Delta - O(n))-$bits and a $(n\log\Delta + O(n))-$bit space-efficient data structure. They also give a $((\chi-1)n + o(\chi n))-$bit space-efficient data structure for interval graphs with bounded chromatic number χ for $\chi = o(\log n)$. In Balakrishnan et al. [3], such a parameterization has been applied to a larger graph class, namely, the class of graphs with $d-$dimensional $t-$representation where parameters d and t are bounded. A $((2td-1)n\log n + o(tdn\log n))-$bit succinct data structure for graphs with $d-$dimensional $t-$representation is presented in [3]. Special cases of this graph class are graphs with bounded boxicity with $t = 1$ and bounded interval number with $d = 1$. Chordal graphs is also a large graph class that is a generalization of interval graphs. A graph is a chordal graph if it contains only cycles of length at most three. Lin et al. [18] introduced the parameter leafage and Chaplick and Stacho [8] the parameter vertex leafage for chordal graphs. In this work, we define the $k-$vertex leafage chordal graphs that contains chordal graphs with bounded vertex leafage and unbounded leafage. When vertex leafage and leafage are equal to two we get interval graphs and when vertex leafage is two and leafage is unbounded we get path graphs. We present a data structure for $k-$vertex leafage chordal graphs using succinct data structure for path graphs and prove that it is succinct.

Convention. Throughout the rest of this paper, set of vertices and edges of a graph G will be denoted by $V(G)$ and $E(G)$, respectively.

Our Results. We present the following two theorems. The first theorem proves a lower bound on the class of $k-$vertex leafage graphs, denoted $\mathcal{G}_k$.

▶ **Theorem 1.** *For $k > 0$ in $o(n^c), c > 0, \log|\mathcal{G}_k| \geq (k-1)n\log n - kn\log k - O(\log n)$.*

The next theorem proves the existence of a matching data structure that supports adjacency query efficiently. For neighbour query we use additional $2n\log n$ bits.

▶ **Theorem 2.** *For $k > 1$ and in $o(n^c), c > 0$, a graph $G \in \mathcal{G}_k$ has a $(k-1)n\log n + o(kn\log n)$-bit succinct data structure that supports adjacency query in $O(k\log n)$ time and using additional $2n\log n$ bits the neighbourhood query for vertex v in $O(k^2 d_v \log n + \log^2 n)$ time where d_v is the degree of $v \in V(G)$.*

Our Main Ideas. The main objectives of this work are two fold.

1. Prove a worst-case information theoretic lower bound on the cardinality of chordal graphs with bounded vertex leafage by using a simple and constructive counting technique motivated by the method of partial coloring as used by Acan et al. [1].

2. Present a data structure for chordal graphs with bounded vertex leafage that uses the succinct data structure for path graphs given in Balakrishnan et al. [4] as a black box. Also, using the lower bound show that this data structure is succinct. In order to design the data structure, we use the characterisation of chordal graphs as intersection graph of sub-trees of a tree. The sub-trees of this tree are carefully decomposed into paths and the resulting paths along with the tree are stored using the succinct data structure for path graphs.

Organisation of the Paper. Section 2 gives details of the compact data structures we have used in the construction of the succinct data structure of $k-$vertex leafage chordal graphs along with the relevant characterisations of chordal and path graphs. It also formalizes and explains the method of partial coloring, first used in Acan et al. [1] to obtain the lower bound for $\mathcal{G}_k$. Section 3 defines the $k-$vertex leafage chordal graphs and also gives a lower bound on the cardinality of the class. The succinct data structure design is given in Section 4 and it contains in Section 4.1, the method to convert a $k-$vertex leafage chordal graph with n vertices to a path graph with $(k-1)n$ vertices. Section 5 concludes by giving motivation to extend this work by generalizing the parameters vertex leafage and leafage to general graphs.

2 Preliminaries

From Gavril [12] we have, the following characterisation of chordal graphs; see Golumbic [14] for more details on chordal graphs or otherwise called triangulated graphs.

▶ **Theorem 3.** *The graph G is a chordal graph if and only if there exists a clique tree T, such that for every $v \in V(G)$, the set of maximal cliques containing v form a sub-tree of T such that two vertices are adjacent if the corresponding sub-trees intersect.*

Also, from Gavril [13] we have the following characterisation of path graphs; see Monma and Wu [20] for more details on path graphs.

▶ **Theorem 4.** *The graph G is a path graph if and only if there exists a clique tree T, such that for every $v \in V(G)$, the set of maximal cliques containing v is a path in T such that two vertices are adjacent if the corresponding paths intersect.*

Succinct Data Structure for Ordinal Trees. Tree T is called an *ordinal tree* if for $z > 0$ and any $u \in V(T)$ with children $\{u_1, \ldots, u_z\}$, for $1 \leq i < j \leq z$, u_i is to the left of u_j [24]. By considering ordinal trees as balanced parenthesis Navarro and Sadakane [24] has given a $2n + o(n)$ bit succinct data structure.

▶ **Lemma 5.** *For any ordinal tree T with n nodes, there exists a $2n + o(n)$ bit Balanced Parentheses (BP) based data structure that supports the following functions among others in constant time :*

1. $\texttt{lca}(i, j)$*, returns the lowest common ancestor of two nodes i, j in T, and*
2. $\texttt{child}(i, v)$*, returns the $q-$th child of v in T.*

Rank-Select Data Structure. Bit-vectors are extensively used in the succinct representation given in Section 4. The following data structure due to Golynski et al. [15] and the functions supported by it are useful.

▶ **Lemma 6.** *Let B be an $n-$bit vector and $b \in \{0,1\}$. There exists an $n + o(n)$ bit data structure that supports the following functions in constant time:*

1. $\texttt{rank}(B, b, i)$*: Returns the number of b's up to and including position i in the bit vector B from the left.*
2. $\texttt{select}(B, b, i)$*: Returns the position of the i-th b in the bit vector B from left. For $i \notin [n]$ it returns 0.*

Non-Decreasing Integer Sequence Data Structure. Given a set of positive integers in the non-decreasing order we can store them efficiently using the differential encoding scheme for increasing numbers; see Section 2.8 of [23]. Let S be the data structure that supports differential encoding for increasing numbers then the function $\texttt{accessNS}(S, i)$ returns the $i-$th number in the sequence.

▶ **Lemma 7.** *Let S be a sequence of n non-decreasing positive integers $a_1, \ldots, a_n, 1 \leq a_i \leq n$. There exists a $2n + o(n)$ bit data structure that supports $\texttt{accessNS}(S, i)$ in constant time.*

Proof. We will prove the lemma by giving a construction of such a data structure. a_1 will be represented by a sequence of a_1 1's followed by a 0. Subsequently, a_i's are represented by storing $a_i - a_{i-1}$ many 1's followed by a 0. It will take at most $2n$ bits since there are n 0's and at most n 1's. Let this bit string be stored using the data structure of Lemma 6 and be denoted as B. B takes $2n + o(n)$ bits. $\texttt{accessNS}(S, i)$ can be implemented using $\texttt{rank}(B, 1, \texttt{select}(B, 0, i))$. ◀

Succinct Data Structure for Path Graphs. From [4] we have the following succinct data structure for path graphs that supports adjacency and neighbourhood queries with slight modifications in input. In the following lemma the endpoints of paths are input to the queries whereas in [4] the path indices are given as input.

▶ **Lemma 8.** *A path graph G has an $n \log n + o(n \log n)$-bit succinct representation. For a $u \in V(G)$, let $P_u = (s_u, t_u)$ be the path corresponding u in clique tree T of G. The succinct representation constructed from the clique tree T representation supports for $u \in V(G)$ the following queries:*

1. $\texttt{adjacencyPG}(s_u, t_u, s_v, t_v)$*: Returns true if $P_u = (s_u, t_u)$ intersects $P_v = (s_v, t_v)$ in $O(\log n)$ time else false,*
2. $\texttt{pathep}(u)$*: Returns the endpoints of path P_u, corresponding to u, in T in $O(\log n)$ time, and*
3. $\texttt{neighbourhoodPG}(s_u, t_u)$*: Returns the list of paths intersecting $P_u = (s_u, t_u)$ in $O(d_u \log n)$ time where d_u is the degree of vertex u.*
4. $\texttt{getHPStartNode}(v)$*: Returns the start node of heavy path π that contains $v \in V(T)$ in constant time. If v is not the first child, that is, it is adjacent to its parent by a light edge, then v itself is returned.*

Permutations. The following data structure by Munro et al. [21], gives a succinct representation for storing permutation of $[n]$.

▶ **Lemma 9** ([21])**.** *Given a permutation of $[n]$ there exists an $(n \log n + o(n \log n))$-bit data structure that supports the following queries.*

- $\pi(i)$*: Returns the $i-$th value in the permutation in $O(1)$ time.*
- $\pi^{-1}(j)$*: Returns the position of the $j-$th value in the permutation in $O(f(n))$ time for any increasing function $f(n) = o(\log n)$.*

Method of Partial Coloring. Let $\mathcal{G}$ be a graph class. A *partial coloring* of $G \in \mathcal{G}$ is the triple $\langle G, U, g\rangle$ where,

- $U \subseteq V(G)$ such that for $s > 0, |U| = s$, and
- $g : U \to \{1, \ldots, s\}$ is a bijection.

Every vertex $u \in U$ is said to have color $g(u)$ and vertices in $V(G)\backslash U$ are said to be uncolored. Two partially colored graphs $\langle H_1, U_1, g_1\rangle$ and $\langle H_2, U_2, g_2\rangle$ are said to be different when either:

1. $E(H_1) \neq E(H_2)$, or
2. $E(H_1) = E(H_2) = E(H)$ for some $H \in \mathcal{G}$ and
   a. $U_1 \neq U_2$ and there does not exist a bijection $f : V(H_1) \to V(H_2)$ such that for every $u \in V(H_1)\backslash U_1$ there exists a $f(u) \in V(H_2)\backslash U_2$ with same set of colored neighbours, or
   b. for $U_1 = U_2 = U$ there exists $u \in V(H)\backslash U$ such that its colored neighbourhood in $\langle H_1, U_1, g_1\rangle$ and $\langle H_2, U_2, g_2\rangle$ are different.

Else, they are same. The method of counting by partial coloring as given in Theorem 1 of Acan et al. [1] can be defined using the following proposition.

▶ **Proposition 10.** *Let $\mathcal{G}'$ be the class of partially colored graphs obtained from class of graphs $\mathcal{G}$ by selecting m vertices out of n and coloring them using m distinct colors. Then, $|\mathcal{G}'| \leq \binom{n}{m} m! |\mathcal{G}|$. If there exists a graph class $\mathcal{G}^c \subset \mathcal{G}'$ then $|\mathcal{G}'| \geq |\mathcal{G}^c|$ and $|\mathcal{G}| \geq \frac{|\mathcal{G}^c|}{\binom{n}{m} m!}$.*

▶ Remark. While computing $|\mathcal{G}'|$, indistinguishable partially colored graphs can also be counted since we only require an upper bound, however, this is not the case while computing $|\mathcal{G}^c|$.

3 Class of $k-$Vertex Leafage Chordal Graphs and its Lower Bound

In this section, we define the class of $k-$vertex leafage chordal graphs, denoted $\mathcal{G}_k$, followed by the lower bound for $\log |\mathcal{G}_k|$. According to Theorem 3, a graph G is a chordal graph if there exists a tree T such that corresponding to every vertex $v \in V(G)$ there exists a sub-tree T_v of T and $\{u, v\} \in E(G)$ if and only if $V(T_u) \cap V(T_v) \neq \phi$ where T_u is the sub-tree corresponding to u. We call T the tree model of G. For every chordal graph there exists a tree T called the *clique tree* such that $V(T)$ is the set of maximal cliques of G. The *leafage* of a chordal graph G, denoted $l(G)$, is the minimum number of leaves of a tree T out of all possible trees characterising G. Leafage was studied by Lin et al. in [18]. Later, Chaplick and Stacho in [8] studied *vertex leafage*, denoted $vl(G)$. Let $L(u)$ denote the number of leaves of the sub-tree in T corresponding to $u \in V$ and $k = \max_{u \in V} L(u)$. The smallest k for which there exists a tree T for G is called its vertex leafage.

Class of k-Vertex Leafage Chordal Graphs. The class of chordal graphs can be considered as the generalisation of path graphs using vertex leafage as the parameter. Theorem 4 implies that path graphs are chordal graphs with vertex leafage equal to two and unbounded leafage. Generalizing this, $\mathcal{G}_k$ is the set of all chordal graphs with vertex leafage at most k and unbounded leafage. Succinct data structure for path graphs is given by Balakrishnan et al. in [4]. In this paper, we present a succinct data structure for chordal graphs with vertex leafage at most k for $k \in o(n^c), c > 0$ using succinct data structure for path graphs as black-box.

Lower Bound. Counting chordal graphs involves heavy mathematical machinery as can be seen from Wormald [10]. Munro and Wu [22] uses the result from [10] to obtain lower bound for the cardinality of unlabeled chordal graphs. Here we give a much simpler technique for

counting unlabeled chordal graphs with bounded vertex leafage. In order to derive the lower bound for $\log|\mathcal{G}_k|$ by implementing Proposition 10, we define two new graph classes, $\mathcal{G}'_k$ and $\mathcal{G}^c_k$, where $\mathcal{G}'_k$ corresponds to $\mathcal{G}'$ and $\mathcal{G}^c_k$ to $\mathcal{G}^c$ of Proposition 10.

Graph Class $\mathcal{G}'_k$. We consider the class of all partially colored $k-$vertex leafage chordal graphs, denoted $\mathcal{G}'_k$, that has for fixed $1 \leq m \leq n$, m out of n vertices colored using colors $\{1,\ldots,m\}$. Graphs in $\mathcal{G}'_k$ are obtained from graphs in $\mathcal{G}_k$ as follows. The input to the procedure is $G \in \mathcal{G}_k$ and a set $\{v_1,\ldots,v_m\}$ of m vertices of G. For each $G \in \mathcal{G}_k$, we get a set of $\binom{n}{m}m!$ graphs of $\mathcal{G}'_k$ where each graph G' is obtained by coloring the selected m vertices of G by a permutation of $\{1,\ldots,m\}$. A partially colored graph in $\mathcal{G}'_k$ is denoted $\langle H,U,g\rangle$ where $U \subset V(H), |U| = m$, and $g : U \to \{1,\ldots,m\}$.

▶ **Lemma 11.** *For each $k > 1$, $\log|\mathcal{G}'_k| \leq \log|\mathcal{G}_k| + n\log n - (n-m)\log(n-m) - 2n + O(\log n)$.*

Graph Class $\mathcal{G}^c_k$. As per Proposition 10, we construct the class $\mathcal{G}^c_k \subset \mathcal{G}'_k$ for which we can obtain exact count or a lower bound. We give a construction mechanism for graphs in $\mathcal{G}^c_k$ such that all graphs in it have the following properties. For $G \in \mathcal{G}^c_k$,

- $U \subseteq V(G)$, with $|U| = m$, is fixed and have a fixed coloring using colors $\{1,\ldots,m\}$,
- let T be the tree model corresponding to G then the sub-trees in the tree model corresponding to vertices in U consist of only one node, and
- sub-trees of T corresponding to vertices in $V(G)\backslash U$ have at most k leaves with at least $\frac{m+1}{2(k-1)}$ sub-trees with exactly k leaves.

In other words, for all the partially colored graphs in $\mathcal{G}^c_k$, U and g are fixed. Based on the tree model, the vertices of a graph $H \in \mathcal{G}^c_k$ with tree model T are of two types:

1. **basis vertices U:** all m vertices in U are represented by single-node sub-trees of T, and
2. **dependent vertices $V(H)\backslash U$:** rest of the $(n-m)$ vertices are represented by sub-trees in T with number of leaves at most k with at least $\frac{m+1}{2(k-1)}$ sub-trees with exactly k leaves. Let the dependent vertices corresponding to these $\frac{m+1}{2(k-1)}$ sub-trees be denoted U'.

We have the following useful proposition:

▶ **Proposition 12.** *For a rooted tree T, the set $V' \subseteq V(T)$ uniquely defines a sub-tree T' of T such that if $u, v \in V'$ then the path connecting it is in T'.*

The sub-trees corresponding to the basis and dependent vertices are called *basis sub-trees* and *dependent sub-trees*, respectively. Let $F = \frac{m+1}{2(k-1)}$. The input to the procedure that constructs H are:

1. n, m, k, and
2. $\mathcal{J} = \{J_1,\ldots,J_{n-m-F}\}$ where for $1 \leq i \leq n-m-F, 1 \leq t \leq k$, $J_i = \{a^i_1,\ldots,a^i_k\}$ and $1 \leq a^i_t \leq m$.

The construction mechanism that constructs H is as follows.

1. Consider a rooted complete binary tree T with $m > 0$ nodes and let them be colored from 1 to m.
2. **Construction of basis sub-trees.** For $1 \leq j \leq m$, let each node $b_j \in V(T)$ be a single node sub-tree, T_j. These sub-trees are the *basis sub-trees*. The basis sub-trees correspond to the m basis vertices of H, denoted by U. Let the basis vertices be colored by the color assigned to the nodes to which they are assigned.
3. **Construction of dependent sub-trees.** First we define the fixed sub-trees with k leaves. Since T is a complete binary tree it has $\frac{m+1}{2}$ leaves. Let the set of leaves of T be denoted by L. For $1 \leq t \leq F$, partition L into blocks $L = \bigcup_t L_t$ such that $|L_t| = k-1$

and for $1 \leq z < z' \leq t$, there does not exist a node in L_z with color greater the smallest color of nodes in $L_{z'}$. For every $u_t \in U'$, construct a sub-tree T_{u_t} in T by connecting paths from the $k-1$ leaves in L_t to the root of T. This ensures that T_{u_t} has k leaves. So, $|U'| = F$. For $1 \leq i \leq n-m-F$, construct sub-tree T_i from $J_i = \{a_1^i, \ldots, a_k^i\}$, where for $1 \leq t \leq k, a_t^i \in V(T)$, such that J_i is the set of k nodes of T_i; as per Proposition 12, J_i uniquely defines a sub-tree. These sub-trees are called *dependent sub-trees* and correspond to the dependent vertices in $V(H) \backslash U$. The sub-trees corresponding to nodes of U' ensure that the chordal graph is connected and there are at least F sub-trees with k leaves there by making H a $k-$vertex leafage chordal graph. Also, apart from U and g, U' is also fixed for any partially colored $k-$vertex leafage chordal graph in $\mathcal{G}_k^c$.

Convention. A node $a_j \in V(T)$ will also be used to denote the sub-tree T_j corresponding to the basis vertex of H.

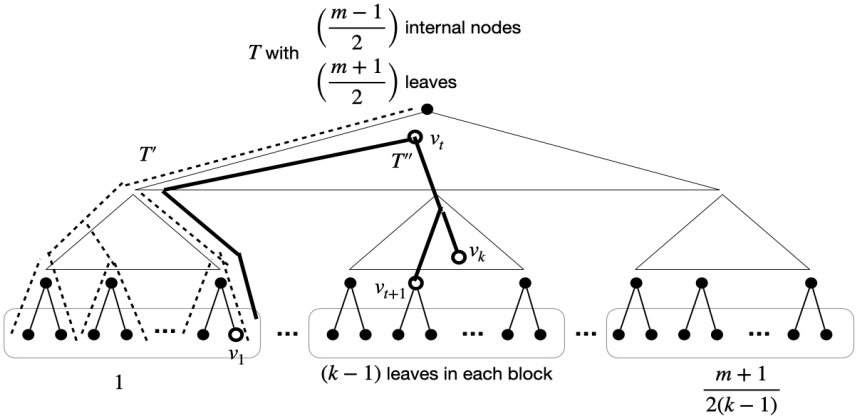


Figure 1 The complete rooted binary tree T with m nodes constructed to produce a graph in $\mathcal{G}_k^c$. T' is one of the sub-trees of T with exactly k leaves corresponding to a dependent vertex in U'. T'' is the sub-tree corresponding to a dependent vertex in $V(H) \backslash (U \cup U')$ constructed from $\{v_1, \ldots, v_t, \ldots, v_k\}$ which are the k selected nodes of T.

Thus, H is defined by $\mathcal{J} = \{J_1, \ldots, J_{n-m-F}\}$ where each J_i defines a sub-tree T_i. For an example construction refer Figure 1. From the construction above we have the following lemma.

▶ **Lemma 13.** $\mathcal{G}_k^c \subseteq \mathcal{G}_k'$.

Computing $|\mathcal{G}_k^c|$. In order to compute $|\mathcal{G}_k^c|$ and use Proposition 10, we first note the following consequence of Lemma 13.

▶ **Proposition 14.** $\log |\mathcal{G}_k'| \geq \log |\mathcal{G}_k^c|$.

Let K denote the set of all possible $\mathcal{J}$'s. We have the following useful proposition and lemma.

▶ **Proposition 15.** *For $1 \leq i, i' \leq n-m-F, 1 \leq j \leq m$, if $J_i \neq J_{i'}$ then wlog there exists $a_j \in U$ such that $a_j \in J_i$ and $a_j \notin J_{i'}$.*

▶ **Lemma 16.** *Let $\mathcal{J}, \mathcal{J}' \in K$ where $\mathcal{J} = \{J_1, \ldots, J_{n-m-F}\}$ and $\mathcal{J}' = \{J_1', \ldots, J_{n-m-F}'\}$ such that for $1 \leq s \leq n-m-F, J_s \neq J_s'$. Also, let H_1 and H_2 be the graphs generated from $\mathcal{J}$ and $\mathcal{J}'$, respectively, and u be the vertex corresponding to s. Then, $N_{H_1}(u) \cap U \neq N_{H_2}(u) \cap U$ where $N_{H_1}(u)$ and $N_{H_2}(u)$ are the neighbours of u in H_1 and H_2, respectively.*

The following is the central lemma used to obtain the lower bound. For $1 \leq s \leq n-m-F$, let $\mathcal{J}(H)$ denote the $\mathcal{J} \in K$ that produces the graph $H \in \mathcal{G}_k^c$.

▶ **Lemma 17.** *Let $\langle H_1, U, g\rangle, \langle H_2, U, g\rangle$ be constructed from $\mathcal{J}, \mathcal{J}' \in K$. Then $\langle H_1, U, g\rangle$ and $\langle H_2, U, g\rangle$ are same if and only if $\mathcal{J} = \mathcal{J}'$.*

In order to obtain $|\mathcal{G}_{t,d}^c|$ we prove the following lemma first.

▶ **Lemma 18.** $|\mathcal{G}_k^c| = |K|$.

The following is an important lemma.

▶ **Lemma 19.** $\log |\mathcal{G}_k^c| \geq kn \log n - kn \log k$.

The following theorem gives the lower bound.

▶ **Theorem 1.** *For $k > 0$ in $o(n^c), c > 0, \log |\mathcal{G}_k| \geq (k-1)n \log n - kn \log k - O(\log n)$.*

We have the following proposition.

▶ **Proposition 20.** *For $k > 0$ in $o(n^c), c > 0$ and sufficiently large n, $\log |\mathcal{G}_k| \geq (k-1)n \log n$.*

4 Succinct Data Structure

The high-level procedure used to obtain the succinct data structure for $k-$vertex leafage chordal graph G given as $(T, \{T_1, \ldots, T_n\})$ is as follows:

1. From the tree T, obtain path graph H with $(k-1)n$ vertices by decomposing each sub-tree $T_i, 1 \leq i \leq n$, into at most $(k-1)$ paths such that each path created corresponds to a vertex of H. The input $(T, \{T_1, \ldots, T_n\})$ is decomposed into paths and we get $(T, \{\mathcal{P}_1, \ldots, \mathcal{P}_n\})$ where $\mathcal{P}_i = \mathcal{P}_i' \cup \mathcal{P}_i'', 1 \leq i \leq n$, where $\mathcal{P}_i' = \{P_1^i, \ldots, P_{k/2}^i\}$ and $\mathcal{P}_i'' = \{P_{k/2+1}^i, \ldots, P_{k-1}^i\}$ such that if we have T and $\mathcal{P}_i'$ we can compute $\mathcal{P}_i''$.
2. Out of the $(k-1)n$ paths, we store $kn/2$ paths of H and tree T using the data structure for path graphs given in [4]. The rest of the $(k/2-1)n$ paths are computed from the tree T and the stored $kn/2$ paths of H. In other words, for each $T_i, \mathcal{P}_i'$ is stored and $\mathcal{P}_i''$ is computed from $\mathcal{P}_i'$ and T.

The succinct representation for G consists of the following contents:

1. $(T, \mathcal{P}_1' \cup \mathcal{P}_2' \cup \ldots \mathcal{P}_n')$ is stored using the method for storing path graphs given in [4]. This data structure stores the tree T using the data structure of Lemma 5.
2. for $1 \leq j \leq k/2$, the mapping from indices of $P_j^i \in \mathcal{P}_i'$ to the indices of paths in the data structure of [4] that stores $(T, \mathcal{P}_1' \cup \mathcal{P}_2' \cup \ldots \cup \mathcal{P}_n')$ as mentioned in the above step.

The following lemma is important before getting into the details of the data structure.

▶ **Lemma 21.** *Consider clique tree T of $k-$vertex leafage chordal graph G such that k is odd. Then there exists a tree model, denoted T', for G with at most $3n$ nodes such that for $1 \leq i \leq n, T_i'$ is the sub-tree in T' corresponding to $v_i \in V(G)$ and number of leaves of T_i is even.*

In this paper, we only consider even $k \geq 2$ as any T_i with odd number of leaves can be converted to even number of leaves as per Lemma 21. Note that the total increase in number of nodes of T is only constant times n. We first present the method to transform a $k-$vertex leafage chordal graph G to path graph H with $(k-1)n$ vertices followed by the construction of the data structure.

4.1 Transforming $(T, \{T_1, \ldots, T_n\})$ to $(T, \mathcal{P}'_1 \cup \ldots \cup \mathcal{P}'_n)$

The transformation from $(T, \{T_1, \ldots, T_n\})$ to $(T, \mathcal{P}'_1 \cup \ldots \cup \mathcal{P}'_n)$ happens in two steps as below:

a. pre-process the tree T into an ordinal tree, and
b. decompose each $T_i, 1 \leq i \leq n$, into $\mathcal{P}'_i \cup \mathcal{P}''_i$.

Pre-processing is done using the method as explained in Section 3 of [4]. We describe it here for ease of reading.

Pre-processing the Tree. Fix a root node for T and perform heavy path decomposition on it. For $v \in V(T)$ order its children $(w_1, \ldots, w_c)$ such that $\{v, w_1\}$ is a heavy edge. Let the children adjacent to v by light edges $(w_2, \ldots, w_c)$, be ordered arbitrarily. This ordering of children of a node of the tree makes it an ordinal tree. Label the nodes of this ordinal tree based on the pre-order traversal; see Section 12.1 of [9] for more details of the pre-order traversal of trees. Labels assigned to nodes in this manner are called the *pre-order labels of the nodes*. Throughout the rest of our paper, this ordinal rooted tree labeled with pre-order will be referred to as the tree model; see Section 3 of [4] for more details. After pre-processing, T is an ordinal tree on which heavy-path decomposition is performed such that all heavy edges are left aligned and nodes are numbered based on the order in which they are visited in the pre-order traversal of T. Since G is $k-$vertex leafage chordal graph, the number of leaves of T_i is at most k. We obtain the tree representation of the path graph $H \in \mathcal{G}_k(2, (k-1)n)$, denoted $(T, \mathcal{P}'_1 \cup \mathcal{P}''_1 \cup \ldots \cup \mathcal{P}'_n \cup \mathcal{P}''_n)$, by carefully selecting the $k-1$ paths from sub-tree $T_i, 1 \leq i \leq n$. The function `getPaths` that does this is explained next.

Function `getPaths`. Given the sub-tree T_i of T with number of leaves at most k, the function returns the set of paths $\mathcal{P}'_i$ such that:

1. $|\mathcal{P}'_i| \leq k/2$,
2. $\mathcal{P}'_i$ along with T can uniquely determine $\mathcal{P}''_i$, and
3. $T_i = \mathcal{P}'_i \cup \mathcal{P}''_i$.

The function can be implemented as follows. Let the leaves of T_i labeled based on the order in which nodes are visited in the pre-order traversal of the tree, be $\{l_1, \ldots, l_{k_i}\}, k_i \leq k$. For $1 \leq j \leq k_i/2$, pair the smallest leaf l_j with the largest leaf $l_{k_i - j + 1}$ to get path P_j. Let the set of paths obtained from T_i be denoted by $\mathcal{P}'_i$.

Ordering Paths in $\mathcal{P}'_i$. Let $\mathcal{P}'_i = \{P^i_1, \ldots, P^i_{k_i}\}, 1 \leq k_i \leq k/2$. For $1 \leq j < j' \leq k_i/2$, $P^i_j = (a_j, b_j)$ and $P^i_{j'} = (a_{j'}, b_{j'})$, $P^i_j \prec_{\mathcal{P}} P^i_{j'}$ if $a_j \leq a_{j'} \leq b_{j'} \leq b_j$. $\prec_{\mathcal{P}}$ is a total order on $\mathcal{P}_i$, since for any two paths $P^i_j, P^i_{j'} \in \mathcal{P}_i$, either $a_j \leq a_{j'} \leq b_{j'} \leq b_j$ or $a_{j'} \leq a_j \leq b_j \leq b_{j'}$.

Relation Between $P^i_j, P^i_{j+1} \in \mathcal{P}'_i$. For $1 \leq j \leq k/2 - 1$, the paths connecting P^i_j and P^i_{j+1} are the paths in $\mathcal{P}''_i$. They are computed using the function `connector`. Let $P^i_j = (a_j, b_j)$ and $P^i_{j+1} = (a_{j+1}, b_{j+1})$. We define `connector` based on the following two conditions:

1. P^i_j and P^i_{j+1} are not intersecting: Since $P^i_j \prec_{\mathcal{P}} P^i_{j+1}$ and P^i_{j+1} is contained inside a sub-tree rooted at $\texttt{lca}(a_j, b_j)$, $\texttt{connector}(i, j, j+1) = (\texttt{lca}(a_j, b_j), \texttt{lca}(a_{j+1}, b_{j+1}))$ connects P^i_j and P^i_{j+1}.
2. P^i_j and P^i_{j+1} are intersecting: $\texttt{connector}(i, j, j+1) = \texttt{NULL}$ in this case.

We have the following useful lemmas.

▶ **Lemma 22.** *For $1 \leq i \leq n$, the following holds:*

1. $|\mathcal{P}'_i| \leq k/2$ *and* $|\mathcal{P}''_i| \leq k/2 - 1$,
2. $T_i = \mathcal{P}'_i \cup \mathcal{P}''_i$ *where* $\mathcal{P}''_i = \texttt{connector}(i,1,2) \cup \texttt{connector}(i,2,3) \cup \ldots \cup \texttt{connector}(i, k_i/2 - 1, k_i/2)$, *and*
3. $T = \bigcup_{i=1}^{n} F_i$ *where* $F_i = \mathcal{P}'_i \cup \mathcal{P}''_i$.

▶ **Lemma 23.** *For* $1 \leq j \leq k_i/2$, *let* $P^i_j \in \mathcal{P}'_i$ *such that* $P^i_j = (a^i_j, b^i_j)$, $e = \texttt{lca}(a^i_j, b^i_j)$ *and* $Q_1 = (e, b^i_j)$. *Also, let* $\texttt{connector}(i, j, j+1) \neq \texttt{NULL}$. *Then,*

1. $(e, c) \in E(Q)$ *is a light edge, and*
2. $(e, c') \in E(\texttt{connector}(i, j, j+1))$ *is a light edge.*

Finally, we have the following proposition and lemma regarding adjacency and neighbourhood of $v \in V(G)$.

▶ **Proposition 24.** $\{u, v\} \in E(G)$ *if and only if there exists* $P \in \mathcal{P}'_u \cup \mathcal{P}''_u, Q \in \mathcal{P}'_v \cup \mathcal{P}''_v$, *such that* $P \cap Q \neq \phi$.

For $1 \leq j \leq k_v, 1 \leq j' \leq k_v - 1$, let $\beta(P^v_j)$ and $\beta(\texttt{connector}(v, j', j'+1))$ represent the paths in $\mathcal{P}'_i \cup \mathcal{P}''_i \cup \ldots \cup \mathcal{P}'_n \cup \mathcal{P}''_n$ intersecting P^v_j and $\texttt{connector}(v, j', j'+1)$, respectively.

▶ **Lemma 25.** *For* $1 \leq i \leq n$, *let* $\mathcal{P}_i = \mathcal{P}'_i \cup \mathcal{P}''_i$. *For* $P \in \mathcal{P}_i$ *the following holds:*

1. $|\beta(P)| \leq (k-1)d_i$, *and*
2. $\sum_{P \in \mathcal{P}_i} |\beta(P)| \leq (k-1)^2 d_i$ *where* d_i *is degree of* $v_i \in V(G)$.

4.2 Construction

Index of Paths in G and H. For $P^i_1, \ldots, P^i_{k_i} \in \mathcal{P}'_i$, we index paths of G and H in the following ways:

1. in G paths are ordered $\{P^1_1, \ldots, P^1_{k_1}, \ldots, P^n_1, \ldots, P^n_{k_n}\}$ where $P^i_1, 1 \leq i \leq n$ is in the increasing order of the starting nodes of P^i_1 and for $1 \leq j \leq k_i$, P^i_j are ordered based on $\prec_{\mathcal{P}}$, and
2. in H paths are ordered based on their starting nodes; this is as per the storage scheme followed in [4] for path graphs.

We will order the vertices of G based on the order of the starting nodes of the first paths $P^i_1 \in \mathcal{P}_i$ of each T_i. The data structure for $k-$vertex leafage chordal graphs consists of the following components. We distinguish the case when k is odd or even only when the difference alters the higher order term in the space complexity, else we assume k is even. We will show later that k being odd or even does not impact the space complexity of our data structure.

Path Graph H. The path graph H that we store is $(T, \bigcup_{i=1}^{n} \mathcal{P}'_i)$ where $|\bigcup_i \mathcal{P}'_i| \leq nk/2$ if k is even and $|\bigcup_i \mathcal{P}'_i| \leq \left(\frac{k-1}{2} + \frac{1}{2}\right)n$ if k is odd. For $1 \leq i \leq n$, we do not store $\texttt{connector}(i,1,2), \texttt{connector}(i,2,3), \ldots, \texttt{connector}(i, k_i - 1, k_i)$ but compute it when required. In other words, T_i can be computed from T and $\mathcal{P}'_i$. Thus, path graph H can be stored using $\frac{nk}{2} \log n + o(nk \log n)$ bits as per Lemma 8 if k is even and $\left(\frac{k-1}{2} + \frac{1}{2}\right) n \log n + o(nk \log n)$ bits if k is odd. The data structures used in the succinct representation of path graphs as given in Lemma 8 does not require T to be a clique tree of H as long as the number of nodes in T is constant times $|V(H)|$, so $(T, \bigcup_{i=1}^{n} \mathcal{P}_i)$ is a valid input that represents H.

Array K. K is a one dimensional array of length n. For $1 \leq i \leq n, K[i]$ stores $|\mathcal{P}'_i|$, that is, the number of paths that the sub-tree T_i gets decomposed into. K takes at most $n \log k$ bits of space. The following function is supported.

- `getSize`(i) : Given $1 \leq i \leq n$, the function returns $K[i]$ else 0.

Bit-vector F. F is a bit vector of length at most $kn/2$. Let the index of the first path of each vertex in the order $(P^1_1, \ldots, P^1_{k_1}, \ldots, P^n_1, \ldots, P^n_{k_n})$ be $\{i_1, \ldots, i_n\}$ where $P^i_j \in \mathcal{P}'_i, 1 \leq i \leq n, 1 \leq j \leq k_i$. Since the vertices are numbered based on the start node of their first paths, $\{i_1, \ldots, i_n\}$ is an increasing sequence of numbers with maximum value of $kn/2$ and can be stored using the differential encoding scheme of Lemma 7. This data structure is also denoted by F and takes at most $kn + o(kn)$ bits of space. The following function is supported.

- `getIndex`(i) : Given $1 \leq i \leq n$, the function returns `accessNS`(i, F) else 0. `accessNS`(i, F) returns the value at the $i-$th position in the sequence $\{i_1, \ldots, i_n\}$.

Bit-vector D. Consider the order of paths $O = (P^1_1, \ldots, P^1_{k_1}, \ldots, P^n_1, \ldots, P^n_{k_n})$. For $1 \leq j \leq kn/2, D[j] = i$ if $O[j]$ is a path corresponding to $u_i \in V(G)$. Since D is an increasing sequence it can be stored using the data structure of Lemma 7 taking $O(kn)$ bits. D contains at most n 1's and $kn/2$ 0's.

Permutation π. π store the mapping of indices of paths in H to paths in G by storing the indices of paths $(P^1_2, \ldots, P^1_{k_1}, \ldots, P^n_2, \ldots, P^n_{k_n})$ in G in that order. π is stored using the data structure of Lemma 9 and takes $\left(\frac{k}{2} - 1\right) n \log n + o(kn \log n)$ bits of space if k is even and $\left(\frac{k-1}{2} - \frac{1}{2}\right) n \log n + o(kn \log n)$ bits if k is odd.

Function `alpha`. The function `alpha` takes $1 \leq i \leq n$, as input and returns the indices of paths in $\mathcal{P}'_i$ corresponding to $u_i \in V(G)$ in the tree representation of H. Function returns $\{\pi(p), \ldots, \pi(p+s)\}$ where $p \leftarrow$ `getIndex`$(i) - i + 1$ and $s \leftarrow$ `getSize`(i). `getIndex`$(i) - i + 1$ is the index of P^i_2 in $(P^1_2, \ldots, P^1_{k_1}, \ldots, P^n_2, \ldots, P^n_{k_n})$.

▶ **Lemma 26.** *Given the index $1 \leq i \leq n$, of $u_i \in V(G)$,* `alpha`(i) *returns the indices in H of paths in $\mathcal{P}'_i$ corresponding to u_i in $O(k_i)$ time.*

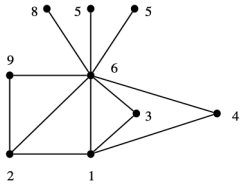
Bit-vector C. C is an array of n bit-vectors each of length $(k/2 - 1)$. For $1 \leq i \leq n, 1 \leq j \leq k/2 - 1$, $C[i][j] = 1$ if `connector`$(i, j, j+1) \neq$ `NULL` else $C[i][j] = 0$. C takes a total space of $n(k/2 - 1)$ bits. The following function is supported.

- `isConnector`(i, j) : Given $1 \leq i \leq n, 1 \leq j \leq k/2 - 1$, the function returns true if $C[i][j] = 1$ else false.

Function `getCPath`. For $1 \leq i \leq n, 1 \leq j \leq k_i/2 - 1$ and $P^i_j = (a_j, b_j), P^i_{j+1} = (a_{j'}, b_{j'})$, the function takes paths P^i_j and P^i_{j+1} as input and returns `connector`$(i, j, j+1)$ if `isConnector`(i, j)) is true else `NULL`. Note that H stored as per Lemma 8 contains the tree T and this allows us to perform the `lca` operation.

▶ **Lemma 27.** *For $1 \leq i \leq n, 1 \leq j \leq k_i/2 - 1$, given paths P^i_j, P^i_{j+1} as input, the function* `getCPath`(P^i_j, P^i_{j+1}) *returns* `connector`$(i, j, j+1)$ *in constant time.*

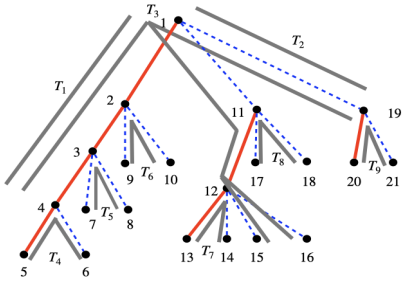
▶ **Lemma 28.** *For $k > 1$ and in $o(n^c), c > 0$, there exists a $(k-1)n \log n + o(kn \log n)$-bit succinct data structure for the class of $k-$vertex leafage chordal graphs.*



(a) Graph G with 9 vertices.

T_1	T_2	T_3		T_4	T_5	T_6	T_7	T_8	T_9
P_1^1	P_1^2	P_1^3	P_2^3	P_1^4	P_1^5	P_1^6	P_1^7	P_1^8	P_1^{10}
1	1	4	15	5	7	9	13	17	20
3	19	19	16	6	8	10	14	18	21
1	2	3	4	5	6	7	8	9	10

(c) Index of paths after decomposition of sub-trees, $T_1, \ldots, T_n$



(b) Tree model T and sub-trees $T_i, 1 \leq i \leq n$

Y

1	$(T_1,1)(T_2,1)(T_3,12)$
2	$(T_6,2)$
3	$(T_5,3)$
4	$(T_4,4)$
	...
19	$(T_9,19)$
20	NULL
21	NULL

π	1	2	3	8	4	5	6	7	9	10
K	1	1	2	1	1	1	1	1	1	1
F	1	2	3	5	6	7	8	9	10	
D	1	2	3	3	4	5	6	7	8	9

(d) Succinct data structure for $\mathcal{G}_k$. Note that getNhb(parent(11),11,18) returns 3.

Figure 2 (a) An example 4-vertex leafage chordal graph G, (b) tree representation of G after pre-processing, (c) the index of the paths generated from the sub-trees along with their start node (second row), end node (third row), and index (forth row), (d) the components of the succinct data structure for G.

4.3 Adjacency and Neighbourhood Queries

The following lemma is useful.

▶ **Lemma 29.** *Let T_i and $T_{i'}$ intersect and $P_1^i = (a_1^i, b_1^i), P_1^{i'} = (a_1^{i'}, b_1^{i'})$. Wlog,*

1. *if $a_1^i \leq a_1^{i'} \leq b_1^{i'} \leq b_1^i$, then*
   a. *there exists $P_j^i, 1 \leq j \leq k_i/2$ such that $P_j^i \cap P_1^{i'} \neq \phi$, or*
   b. *there exists $\texttt{connector}(i, j, j+1) \cap P_j^{i'} \neq \phi, 1 \leq j \leq k_i/2 - 1$.*
2. *Else, $a_1^i \leq a_1^{i'} \leq b_1^i \leq b_1^{i'}$ or $a_1^{i'} \leq a_1^i \leq b_1^{i'} \leq b_1^i$.*

Adjacency Query. Given indices of two vertices $u_i, u_j \in V(G)$ and the succinct representation for the $k-$vertex leafage chordal graphs, the adjacency query returns true if $\{u_i, u_j\} \in E(G)$ else false. The characterisation of $\{u_i, u_j\} \in E(G)$ in terms of path intersections in $(T, \mathcal{P}_1 \cup \ldots \cup \mathcal{P}_n)$ where $\mathcal{P}_i = \mathcal{P}_i' \cup \mathcal{P}_i'', 1 \leq i \leq n$, is given by Proposition 24. Algorithm 1 gives an implementation of the adjacency query.

▶ **Lemma 30.** *For $k \in o(n^c), c > 0$, the class of $k-$vertex leafage chordal graphs have a $(k-1)n \log n + o(kn \log n)$ bit succinct data structure that supports adjacency query in $O(k \log n)$ time.*

Neighbourhood Query. Given $v \in V(G)$ and the succinct representation of $k-$vertex leafage chordal graph G, the neighbourhood query returns the neighbours of v. We use the following additional data structure.

Algorithm 1 Given indices i, j as input, the function returns true if $\{u_i, u_j\} \in E(G)$ else false.

```
1  Function adjacency(i, j):
2      P'_i ← alpha(i), P'_j ← alpha(j), s ← NULL
3      Let P^i_1 = (a^i_1, b^i_1) and P^j_1 = (a^j_1, b^j_1) obtained using
         pathep(getIndex(i)) and pathep(getIndex(j)), respectively.
4      Check if P^i_1 and P^j_1 intersect as follows:
5      if adjacencyPG(H, P^i_1, P^j_1) is true then
6          return true
7      Check if endpoints of first path of one falls within the range of the end points of
         the first path of the other as follows:
8      if a^i_1 ≤ a^j_1 ≤ b^j_1 ≤ b^i_1 then
9          s ← i
10
11     if a^j_1 ≤ a^i_1 ≤ b^i_1 ≤ b^j_1 then
12         s ← j
13
14     if s ≠ NULL then
15         if s = i then
16             foreach 1 ≤ t ≤ getSize(i) − 1 do
17                 if adjacencyPG(H, pathep(P'_i[t]), pathep(P'_j[1])) is true then
18                     return true
19             foreach 2 ≤ t ≤ getSize(i) − 1 do
20                 if t = 2 and
                     adjacencyPG(H, getCPath(getIndex(i), P'_i[t]), pathep(P'_j[1])) is
                     true then
21                     return true
22                 else
23                     if adjacencyPG(H, getCPath(P'_i[t − 1], P'_i[t]), pathep(P'_j[1])) is
                         true then
24                         return true
25         if s = j then
26             Perform the same steps as done in Line 15 for the case s = i with i and j
                 interchanged.
27     return false
```

Array Y. Y is a one dimensional array of length at most n that stores an array at each of its locations. For $1 \leq i \leq n$, $Y[i]$ stores the array of records where each record is of the form (r, s) such that $1 \leq r \leq n$ is the index of the sub-tree that has node $u_i \in V(T)$ as the lca of endpoints of P^r_1 and for $1 \leq s \leq n$, node u_s as the lca of endpoints of $P^r_{k_r}$. The records at $Y[i]$ are stored in the increasing order of s. The total space taken by Y is $2n \log n$ bits as each tree takes $2 \log n$ bits and there are n trees. The following function is supported.

- `getNhb`(l, l', a): Given $1 \leq l, l' \leq n$, the function returns the list of trees with index $1 \leq j \leq n$, such that lca of endpoints of P_1^j is equal to l and lca of endpoints of $P_{k_j}^j$ greater than or equal to l' and less than or equal to a in $O(\log n + d)$ time where d is the number of trees returned. The function can be implemented as follows:
  1. Performing binary search on the array A of records at $Y[l]$ to first obtain the range of s values greater than or equal to l'. Let this range be $[p_1, p_2]$.
  2. Performing second binary search on $A[p_1, p_2]$ to obtain the range of s values that are less than or equal to a in $A[p_1, p_2]$. Let this range be $[p'_1, p'_2]$.
  3. Return the indices of trees, that is, the r values stored in records $A[p'_1, p'_2]$.

  The binary search takes $O(\log n)$ time and the tree indices are returned in d time where $d = p'_2 - p'_1 + 1$. Thus, `getNhb` takes a total time of $O(\log n + d)$.

Algorithm 2 gives an implementation of the neighbourhood query.

▶ **Lemma 31.** *For $k \in o(n^c), c > 0$, the class of $k-$vertex leafage chordal graphs have a $(k-1)n \log n + o(kn \log n)$ bit succinct data structure that supports neighbourhood query for vertex v_i in $O(k^2 d_i \log n + \log^2 n)$ time using additional $2n \log n$ bits where d_i is the degree of v_i.*

Thus, we have the following theorem.

▶ **Theorem 2.** *For $k > 1$ and in $o(n^c), c > 0$, a graph $G \in \mathcal{G}_k$ has a $(k-1)n \log n + o(kn \log n)$-bit succinct data structure that supports adjacency query in $O(k \log n)$ time and using additional $2n \log n$ bits the neighbourhood query for vertex v in $O(k^2 d_v \log n + \log^2 n)$ time where d_v is the degree of $v \in V(G)$.*

Proof. From Lemma 28, we know that for $k \in o(n/\log n)$, the $(k-1)n \log n + o(kn \log n)$-bit data structure is succinct. From Lemma 30, we know that the data structure supports adjacency query in time $O(k \log n)$ and from Lemma 31, we know that the neighbourhood query is supported in time $O(k^2 d_i \log n + \log^2 n)$ using additional $2n \log n$ bits. ◀

5 Conclusion

The parameters, leafage and vertex leafage, are defined for chordal graphs which is a special case of intersection graphs. In comparison, boxicity and interval number allow us to model general graphs as intersection graphs. However, they do not give a tree model like in the case of chordal graphs. We ask the following question: "*Can leafage and vertex leafage be generalized for any graph?*" The answer is positive if we consider the nice tree-decomposition. For any graph, the nice tree-decomposition allows us to establish a correspondence between vertices of the graph and sub-trees of the tree obtained. As one can see clearly, the parameters leafage and vertex leafage become applicable to general graphs now. For chordal graphs we know that the clique tree has nodes that correspond to the maximal cliques of the graph. However, we lose such nice properties in the case of nice tree-decomposition. Despite these limitations we think it is worthwhile to generalize these parameters and design a succinct data structure for the more general class thus formed. It is also interesting to note that a unit increase in the leafage parameter increases the number of graphs in the class by $n \log n$ as compared to $2n \log n$ in the case of boxicity or interval number. From Balakrishnan et al. [3] we know that for the succinct data structure designed for graphs with bounded boxicity $d > 0$ using the succinct data structure for interval graphs an efficient but easy implementation for neighbourhood query is not possible. For bounded leafage parameter where the intersection model is a tree, it will be interesting to see if there exists a simple and efficient implementation of the neighbourhood query. Another challenging direction is to consider whether space-efficient graph algorithms can be designed for these specialized graph classes [5, 7, 11].

Algorithm 2 Given index i as input, the function returns the set of vertices adjacent to $u_i \in V(G)$.

```
1  Function neighbourhood(i):
2      P'_i ← alpha(i), P ← φ
3      Add pathep(getIndex(i)) to P
4      foreach j ∈ P'_i do
5          Add pathep(j) to P
6      Add getCPath(getIndex(i), P'_i[1]) to P if it is not NULL
7      foreach 1 ≤ j ≤ getSize(i) − 1 do
8          Add getCPath(P'_i[j], P'_i[j + 1]) to P if it is not NULL
9      N ← φ
10     Let t be a bit-vector of length n initialised to 0
11     foreach (a, b) ∈ P do
12         N' ← neighbourhoodPG(a, b)
13     foreach j' ∈ N' do
14         p ← π^{-1}(j')
15         if t[p] ≠ 1 then
16             Add D[p] to N
17             t[p] ← 1
18     P_1^i ← pathep(getIndex(i))
19     P_2^i ← pathep(P_i[1])
20     Let P_1^i = (a_1, b_1), P_2^i = (a_2, b_2), l ← lca(a_1, b_1), l' ← l
21     if lmost_child(parent(l)) = l then
22         v ← parent(getHPStartNode(l))
23     else
24         v ← parent(l)
25     while v ≠ NULL do
26         Add getNhb(v, l', b_1) to N
27         l ← v
28         if child(1, parent(l)) = l then
29             v ← parent(getHPStartNode(l))
30         else
31             v ← parent(l)
```

References

1. H. Acan, S. Chakraborty, S. Jo, K. Nakashima, K. Sadakane, and S.R. Satti. Succinct navigational oracles for families of intersection graphs on a circle. *Theor. Comput. Sci.*, 928(C):151–166, September 2022. doi:10.1016/j.tcs.2022.06.022.
2. H. Acan, S. Chakraborty, S. Jo, and S. R. Satti. Succinct encodings for families of interval graphs. *Algorithmica*, 83(3):776–794, 2021.
3. G. Balakrishnan, S. Chakraborty, S. Jo, N. S. Narayanaswamy, and K. Sadakane. Succinct data structure for graphs with d-dimensional t-representation, 2023. arXiv:2311.02427.
4. G. Balakrishnan, S. Chakraborty, N.S. Narayanaswamy, and K. Sadakane. Succinct data structure for path graphs. *Information and Computation*, 296:105124, 2024. doi:10.1016/j.ic.2023.105124.

5 N. Banerjee, S. Chakraborty, V. Raman, and S. R. Satti. Space efficient linear time algorithms for BFS, DFS and applications. *Theory Comput. Syst.*, 62(8):1736–1762, 2018. doi:10.1007/S00224-017-9841-2.

6 S. Chakraborty and S. Jo. Compact representation of interval graphs and circular-arc graphs of bounded degree and chromatic number. *Theor. Comput. Sci.*, 941:156–166, 2023.

7 S. Chakraborty, S. Jo, and S. R. Satti. Improved space-efficient linear time algorithms for some classical graph problems. *CoRR*, abs/1712.03349, 2017. arXiv:1712.03349.

8 S. Chaplick and J. Stacho. The vertex leafage of chordal graphs. *Discrete Applied Mathematics*, 168:14–25, 2014. Fifth Workshop on Graph Classes, Optimization, and Width Parameters, Daejeon, Korea, October 2011. doi:10.1016/j.dam.2012.12.006.

9 T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms, Third Edition*. The MIT Press, 3rd edition, 2009.

10 N. C.Wormald. Counting labelled chordal graphs. *Graph. Comb.*, 1(1):193–200, December 1985. doi:10.1007/BF02582944.

11 A. Elmasry, T. Hagerup, and F. Kammer. Space-efficient basic graph algorithms. In Ernst W. Mayr and Nicolas Ollinger, editors, *32nd STACS*, volume 30 of *LIPIcs*, pages 288–301. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2015. doi:10.4230/LIPICS.STACS.2015.288.

12 F. Gavril. The intersection graphs of subtrees in trees are exactly the chordal graphs, 1973.

13 F. Gavril. A recognition algorithm for the intersection graphs of paths in trees, 1978.

14 M. C. Golumbic. *Algorithmic Graph Theory and Perfect Graphs*. North-Holland Publishing Co., NLD, 2004.

15 A. Golynski, J. I. Munro, and S. S. Rao. Rank/select operations on large alphabets: A tool for text indexing. In *SODA*, SODA '06, pages 368–373, USA, 2006. Society for Industrial and Applied Mathematics.

16 A. Itai and M. Rodeh. Representation of graphs. *Acta Inf.*, 17(2):215–219, June 1982. doi:10.1007/BF00288971.

17 G. J. Jacobson. Space-efficient static trees and graphs. *30th Annual Symposium on Foundations of Computer Science*, pages 549–554, 1989.

18 I.J. Lin, T.A. McKee, and D.B. West. The leafage of a chordal graph. *Discussiones Mathematicae Graph Theory*, 18(1):23–48, 1998. URL: http://eudml.org/doc/270535.

19 L. Markenzon, C F. E. M. Waga, P. R. C Pereira, C. V. P. Friedmann, and A. R. G. Lozano. An efficient representation of chordal graphs. *Operations Research Letters*, 41(4):331–335, 2013. doi:10.1016/j.orl.2013.03.008.

20 C. L. Monma and V. K.-W. Wei. Intersection graphs of paths in a tree. *J. Comb. Theory, Ser. B*, 41(2):141–181, 1986.

21 J. I. Munro, R. Raman, V. Raman, and S. S. Rao. Succinct representations of permutations and functions. *Theor. Comput. Sci.*, 438:74–88, 2012.

22 J. I. Munro and K. Wu. Succinct data structures for chordal graphs. In *ISAAC*, volume 123 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 67:1–67:12, 2018.

23 G. Navarro. *Compact Data Structures - A Practical Approach*. Cambridge University Press, 2016.

24 G. Navarro and K. Sadakane. Fully functional static and dynamic succinct trees. *ACM Trans. Algorithms*, 10(3), May 2014. doi:10.1145/2601073.