Noname manuscript No.
(will be inserted by the editor)

Succinct Encodings for Families of Interval Graphs

Hüseyin Acan* · **Sankardeep Chakraborty** · **Seungbum Jo**** · **Srinivasa Rao Satti**

Received: date / Accepted: date

Abstract We consider the problem of designing succinct data structures for *interval graphs* with n vertices while supporting degree, adjacency, neighborhood and shortest path queries in optimal time. Towards showing succinctness, we first show that at least $n \log_2 n - 2n \log_2 \log_2 n - O(n)$ bits are necessary to represent any unlabeled interval graph G with n vertices, answering an open problem of Yang and Pippenger [Proc. Amer. Math. Soc. 2017]. This is augmented by a data structure of size $n \log_2 n + O(n)$ bits while supporting not only the above queries optimally but also capable of executing various combinatorial algorithms (like proper coloring, maximum independent set etc.) on interval graphs efficiently. Finally, we extend our ideas to other variants of interval graphs, for example, *proper/unit interval graphs, k-improper interval graphs, and circular-arc graphs*, and design succinct data structures for these graph classes as well along with supporting queries on them efficiently.

* Research supported by a National Science Foundation Fellowship (Award No. 1502650).

** Work done while the author was at Université libre de Bruxelles. Research supported by Fonds de la Recherche Scientifique-FNRS under Grant no MISU F 6001 1

A preliminary version of these results have appeared in the 16th International Symposium on Algorithms and Data Structures (WADS, 2019) [1].

Hüseyin Acan
Drexel University
E-mail: huseyin.acan@drexel.edu

Sankardeep Chakraborty
National Institute of Informatics
E-mail: sankardeep.chakraborty@gmail.com

Seungbum Jo
Chungbuk National University
E-mail: sbjo@chungbuk.ac.kr

Srinivasa Rao Satti
Seoul National University
E-mail: ssrao@cse.snu.ac.kr

Keywords Space efficient data structures · Succinct encoding · Interval graphs · Proper interval graphs · Unit interval graphs · Unit interval graphs

1 Introduction

A simple undirected graph G is called an *interval graph* if its vertices can be assigned to intervals on the real line so that two vertices are adjacent in G if and only if their assigned intervals intersect. The set of intervals assigned to the vertices of G is called a *realization* of G. These graphs were first introduced by Hajós [33] who also asked for the characterization of them. The same problem was also asked, independently, by [6] while studying the structure of genes. Interval graphs naturally appear in a variety of contexts, for example, operations research and scheduling theory [4], biology especially in physical mapping of DNA [48], temporal reasoning [29] and many more. We refer the reader to [26, 27] for a thorough treatment of interval graphs and its applications. Eventually answering the question of Hajós [33], several researchers came up with different characterizations of interval graphs, including linear time algorithms for recognizing them; see, for example, [27, Chapter 8] for characterizations, and [8] and [32] for linear time algorithms. Moreover, by exploiting the special structure of interval graphs, many otherwise NP-hard problems in general graphs are also shown to have polynomial time algorithms for interval graphs [26]. These include computing maximum independent set, reporting a proper coloring, returning a maximum clique etc. In spite of having many applications in practically motivated problems, we are not aware of any study of interval graphs from the point of view of *succinct data structures*. The goal here is to store a set Z of objects using the information theoretic minimum $\log(|Z|) + o(\log(|Z|))$ bits of space[1] along with supporting relevant set of queries efficiently, which we focus on in this paper. We assume the usual model of computation, namely a $\Theta(\log n)$-bit word RAM model where n is the size of the input.

1.1 Related Work

Succinct data structures. There already exists a large body of work on representing various classes of graphs succinctly. This is partly motivated by theoretical curiosity and partly by the practical needs as these combinatorial structures do arise quite often in various applications. A partial list of such special graph classes would be trees [14, 39], planar graphs [2], chordal graphs [40], partial k-tree [20] among others, while succinct encoding for arbitrary graphs is also considered in [21]. Furthermore, such data structures for variety of other combinatorial objects are also well studied in the literature [5, 11, 38, 44]. We refer the interested reader to the recent book by Navarro [42] for a comprehensive treatment of these and many more related

[1] throughout the paper, we use log to denote the logarithm to the base 2

topics on succinct/compact data structures.

Algorithmic graph-theoretic results. For interval graphs, other than the algorithmic works mentioned earlier, there are plenty of attempts in exactly counting the number of unlabeled interval graphs [34, 36], and the state-of-the-art result is due to [47], which is what we improve in this work. For the variants of the interval graphs that we study in this paper, there exists also a fairly large number of algorithmic results on them as well as structural results. For example, combinatorial problems like 3-colourability [24], maximum clique and independent set [7, 28] can be solved in polynomial time for the circular-arc graph along with its recognition algorithm. See [26, 27] for more details regarding these combinatorial algorithms as well as various characterizations of these graph classes.

1.2 Our Results and Paper Organization

We list all the preliminary data structures and graph theoretic terminologies that will be used throughout this paper, in Section 2. Given an unlabeled interval graph G with n vertices, in Section 3 we first show that at least $n \log n - 2n \log \log n - O(n)$ bits are necessary to represent G, answering an open problem of Yang and Pippenger [47]. More specifically, Yang and Pippenger [47] showed a lower bound of $(n \log n)/3 + O(n)$-bit for representing any unlabeled interval graph and asked whether this lower bound can be further improved. As circular-arc graphs are generalizations of the interval graphs, note that, a same lower bound result holds true for circular-arc graphs as well. Next in Section 4, we improve the trivial $(2n \lceil \log n \rceil)$-bit representation of G (obtained by storing all the intervals correspond to the vertices in G explicitly), by proposing an $(n \log n + O(n))$-bit representation while being able to support the relevant queries optimally, where the queries are defined as follows. For any two vertices $u, v \in G$,

Table 1 Space lower/upper bounds of families of interval graphs.

Graph class	Space lower bound	Ref.	Space upper bound	Ref.
interval	$n \log n - 2n \log \log n - O(n)$	Thm. 1	$n \log n + (2 + \epsilon)n + o(n)$	Thm. 2
proper/unit	$2n - O(\log n)$	[22]	$2n + o(n)$	Thm. 4
k-(im)proper	open		$2n \log k + 6n + o(n \log k)$	Thm. 5
circular arc	$n \log n - 2n \log \log n - O(n)$	Thm. 1	$n \log n + o(n \log n)$	Thm. 6

Table 2 Query times of our data structures. In what follows, we denote the length of the shortest path between two vertices u and v, i.e., $|\mathsf{spath}(u, v)|$ by the parameter t, and s denotes the function $\log n / \log \log n$.

Graph class	$\mathsf{degree}(v)/\mathsf{adjacent}(u, v)$	$\mathsf{neighborhood}(v)$	$\mathsf{spath}(u, v)$	Ref.
interval	$O(1)$	$O(\mathsf{degree}(v))$	$O(t)$	Thm. 2
proper/unit	$O(1)$	$O(\mathsf{degree}(v))$	$O(t)$	Thm. 4
k-(im)proper	$O(\log \log k)$	$O(\log \log k \cdot \mathsf{degree}(v))$	$O(\log \log k \cdot t)$	Thm. 5
circular arc	$O(s)$	$O(\mathsf{degree}(v) \cdot s)$	$O(st)$	Thm. 6

- $\mathsf{degree}(v)$: returns the number of vertices that are adjacent to v in G,
- $\mathsf{adjacent}(u, v)$: returns true if u and v are adjacent in G, and false otherwise,
- $\mathsf{neighborhood}(v)$: returns all the vertices that are adjacent to v in G, and
- $\mathsf{spath}(u, v)$: returns the shortest path between u and v in G.

We show that all these queries can be supported optimally using our succinct data structure for interval graphs. More precisely, for any two vertices $v, u \in G$, we can answer $\mathsf{degree}(v)$ and $\mathsf{adjacent}(u, v)$ queries in $O(1)$ time, $\mathsf{neighborhood}(v)$ queries in $O(\mathsf{degree}(v))$ time, and $\mathsf{spath}(u, v)$ queries in $O(|\mathsf{spath}(u, v)|)$ time. Note that our results for interval graphs not only improve the previous best space bound of $2n \log n$ bits (given by Klavík et al. [35]), but we can also support navigational queries optimally, a feature not present in [35]. Furthermore, in Section 5, we show how one can implement various fundamental graph algorithms in interval graphs, for example depth-first search (DFS), breadth-first search (BFS), computing a maximum independent set, determining a maximum clique, both time and space efficiently using our succinct representation for interval graphs.

In Section 6, we extend our ideas to other variants of interval graphs, for example, *proper/unit interval graphs, k-proper and k-improper interval graphs, and circular-arc graphs*, and design succinct data structures for these graph classes as well along with supporting queries on them efficiently. For definitions of these graphs, see Section 6. Our succinct data structures (with efficient query support) for proper/unit, and k-(im)proper interval graphs improves the results of Klavík et al. [35] who designed encodings for these graph classes with no query support. More specifically, the asymptotic space consumption of their data structures is same as ours, yet we can support additionally the navigational queries efficiently. We summarize all of our results in Table 1 and Table 2 respectively where Table 1 captures the matching upper/lower bound on the space requirements of each of the graph classes and Table 2 lists all the query times of our data structures. Finally we conclude in Section 7 with some remarks on possible future directions for exploring.

2 Preliminaries

We will use the following data structures in the rest of this paper.

Rank and Select queries: Let $S = s_1, \ldots, s_n$ be a sequence of size n over an alphabet $\Sigma = \{0, 1, \ldots, \sigma - 1\}$. Then for $1 \leq i \leq n$, and $\alpha \in \Sigma$, one can define rank and select queries as follows.

- $\mathsf{rank}_\alpha(S, i)$ = the number of occurrences of α in $s_1 \ldots s_i$.
- $\mathsf{select}_\alpha(S, i)$ = the position j where s_j is the i-th α in S.

The following lemma shows that these operations can be supported efficiently using optimal space.

Lemma 1 ([17, 30]) *Given a sequence $S = s_1, \ldots, s_n$ of size n over an alphabet $\Sigma = \{0, 1, \ldots, \sigma - 1\}$ for any $\sigma > 1$, for any $\alpha \in \Sigma$, there exists an $n \log \sigma + o(n \log \sigma)$-bit data structure which answers rank_α queries on S and access any element of S in $O(\log(1 + \log(\sigma)))$ time, and select_α queries on S in $O(1)$ time.*

Range Maximum Queries: Given a sequence $S = s_1, \ldots, s_n$ of size n, for $1 \leq i, j \leq n$, the *range maximum query* on range $[i, j]$ (denoted by $\mathsf{RMax}_S(i, j)$) returns the position $i \leq k \leq j$ such that s_k is a maximum value in $s_i \ldots s_j$ (if there is a tie, we return the leftmost such position). One can define the *range minimum queries* on range $[i, j]$ ($\mathsf{RMin}_S(i, j)$) analogously. The following lemma shows that there exist data structures which can answer these queries efficiently using optimal space.

Lemma 2 ([10, 23]) *Given a sequence S of size n and for any $1 \leq c \leq n$,*

1. *there exists a data structure of size $O(n/c)$ bits, in addition to storing the sequence S, which supports RMax_S and RMin_S queries in $O(c)$ time while supporting access on S in $O(1)$ time.*
2. *there exists a data structure of size $2n + o(n)$ bits (that does not store the sequence S) which supports RMax_S or RMin_S queries in $O(1)$ time.*

Graph Terminology and Input Representation: We will assume the knowledge of basic graph theoretic terminology as given in [19] and basic graph algorithms as given in [18]. Throughout this paper, $G = (V, E)$ will denote a simple undirected graph with the vertex set V of cardinality n and the edge set E having cardinality m. We call G an *interval graph* if (a) with every vertex we can associate a closed interval on the real line, and (b) two vertices share an edge if and only if the corresponding intervals are not disjoint (see Figure 1 for an example). It is well known that given an interval graph with n vertices, one can assign intervals to vertices such that every end point is a distinct integer from 1 to $2n$ using $O(n \log n)$ time [34], and in the rest of this paper, we deal exclusively with such representations. Moreover, for vertex $v \in V$, we refer to I_v as the interval corresponding to v.

3 Counting the number of unlabeled interval graphs

This section deals with counting unlabeled interval graphs. Let $\mathcal{I}_n$ denote the number of unlabeled interval graphs on n vertices. This is the sequence with id A005975 in the On–Line Encyclopedia of Integer Sequences [45]. Initial values of this sequence are given by Hanlon [34] but he did not prove an asymptotic form for enumerating the sequence. Answering a question posed by Hanlon [34], Yang and Pippenger [47] proved that the generating function $\mathcal{I}(x) = \sum_{n \geq 1} \mathcal{I}_n x^n$ diverges for any $x \neq 0$ and they established the bounds

$$\frac{n \log n}{3} + O(n) \leq \log \mathcal{I}_n \leq n \log n + O(n). \tag{1}$$

The upper bound in (1) follows from $\mathcal{I}_n \leq (2n-1)!! = \prod_{j=1}^{n}(2j-1)$, where the right hand side is the number of matchings on $2n$ points on a line. For the lower bound, the authors showed $\mathcal{I}_{3k} \geq k!/3^{3k}$ by finding an injection from S_k, the set of permutations of length k, to three-colored interval graphs of size $3k$. Furthermore, they left it open whether the leading terms of the lower and upper bounds in (1) can be matched, which is what show in affirmative by improving the lower bound. In other words, we find the asymptotic value of $\log \mathcal{I}_n$. In what follows, for a set S, we denote by $\binom{S}{k}$ the set of k-subsets of S.

Theorem 1 *Let $\mathcal{I}_n$ be the number of unlabeled interval graphs with n vertices. As $n \to \infty$, we have*

$$\log \mathcal{I}_n \geq n \log n - 2n \log \log n - O(n). \tag{2}$$

Proof We consider certain interval graphs on n vertices with colored vertices. Let k be a positive integer smaller than $n/2$ such that $k^2 \geq n - 2k$, and ε a positive constant smaller than $1/2$. For $1 \leq j \leq k$, let B_j and R_j denote the intervals $[-j-\varepsilon, -j+\varepsilon]$ and $[j-\varepsilon, j+\varepsilon]$, respectively. These $2k$ pairwise-disjoint intervals will make up $2k$ vertices in the graphs we consider. Now let $\mathcal{W}$ denote the set of k^2 closed intervals with one endpoint in $\{-k, \ldots, -1\}$ and the other in $\{1, \ldots, k\}$. We color $B_1, \ldots, B_k$ with blue, $R_1, \ldots, R_k$ with red, and the k^2 intervals in $\mathcal{W}$ with white.

Together with $\mathcal{S} := \{B_1, \ldots, B_k, R_1, \ldots, R_k\}$, each $\{J_1, \ldots, J_{n-2k}\} \in \binom{\mathcal{W}}{n-2k}$ gives an n-vertex, three-colored interval graph. For a given $\mathcal{J} = \{J_1, \ldots, J_{n-2k}\}$, let $G_{\mathcal{J}}$ denote the colored interval graph whose vertices correspond to n intervals in $\mathcal{S} \cup \mathcal{J}$, and let $\mathcal{G}$ denote the set of all $G_{\mathcal{J}}$.

Now let $G \in \mathcal{G}$. For a white vertex $w \in G$, the pair $(d_B(w), d_R(w))$, which represents the numbers of blue and red neighbors of w, uniquely determine the interval corresponding to w; this is the interval $[-d_B(w), d_R(w)]$. In other words, $\mathcal{J}$ can be recovered from $G_{\mathcal{J}}$ uniquely. Thus $|\mathcal{G}| = \binom{k^2}{n-2k}$. Since there are at most 3^n ways to color the vertices of an interval graph with blue, red, and white, we have

$$\mathcal{I}_n \cdot 3^n \geq |\mathcal{G}| = \binom{k^2}{n-2k} \geq \left(\frac{k^2}{n-2k}\right)^{n-2k} \geq \left(\frac{k^2}{n}\right)^{n-2k}$$

for any $k < n/2$. Setting $k = \lfloor n/\log n \rfloor$ and taking the logarithms, we get

$$\log \mathcal{I}_n \geq (n-2k)\log(k^2/n) - O(n) = n \log n - 2n \log \log n - O(n).$$

□

Remark 1 Yang and Pippenger [47] also posed the question whether $\log \mathcal{I}_n = Cn \log n + O(n)$ for some C or not. According to Theorem 1, this boils down to getting rid of the $2n \log \log n$ term in (2). Such a result would imply that the exponential generating function $J(x) = \sum_{n \geq 1} I_n x^n/n!$ has a finite radius of convergence. (As noted in [47], the bound $\mathcal{I}_n \leq (2n-1)!!$ implies that the radius of convergence of $J(x)$ is at least $1/2$).

Remark 2 After the publication of the conference version of this article [1], it was pointed out to us by Cyrill Gavoille that an earlier paper from 2008 (much earlier than even Yang and Pipinger's sub-optimal lower bound result [47]) by Gavoille and Paul [25] already showed this lower bound for interval graphs, by obtaining a lower bound for labeled interval graphs.

Remark 3 A *circular-arc graph* G is defined as a graph whose vertices can be assigned to arcs on a circle so that two vertices are adjacent in G if and only if their assigned arcs intersect. It is easy to see that every interval graph is a circular-arc graph. Hence, from Theorem 1, we can also deduce that given a circular-arc graph G with n vertices, one needs at least $n \log n - 2n \log \log n - O(n)$ bits to represent G

4 Succinct representation of interval graphs

In this section, we introduce a succinct $n \log n + (2 + \epsilon)n + o(n)$-bit representation of unlabeled interval graph G on n vertices with constant $\epsilon > 0$, and show that the navigational queries (degree, adjacent, neighborhood, and spath queries) and some basic graph algorithms (BFS, DFS, PEO traversals, proper coloring, computing the size of a maximum clique and maximum independent set) on G can be answered/executed efficiently using our representation of G.

4.1 Succinct Representation of G

We first label the vertices of G using the integers from 1 to n, as described in the following. By the assumption in Section 2, the vertices in G can be represented by n intervals $I = \{I_1 = [l_1, r_1], I_2 = [l_2, r_2], \ldots, I_n = [l_n, r_n]\}$ where all the endpoints in I are distinct integers in the range $[1, 2n]$. Since there are $2n$ distinct endpoints for the n intervals in I, every integer in $[1, 2n]$ corresponds to a unique l_i or r_i for some $1 \leq i \leq n$. We assign the labels to the vertices in G based on the sorted order of left endpoints of their corresponding intervals, i.e., for any two vertices $a, b \in G$, $a < b$ if and only if $l_a < l_b$.

Now we describe the representation of G. Let $S = s_1 \ldots s_{2n}$ be the binary sequence of length $2n$ such that for $1 \leq i \leq 2n$, $s_i = 0$ if $i \in \{l_1, l_2, \ldots, l_n\}$ (i.e., if i corresponds to the left end point of an interval in I), and $s_i = 1$ otherwise. If $i = l_k$ or $i = r_k$, we say that s_i corresponds to the interval I_k. By storing the data structure of Lemma 1 on S, we can answer rank and select queries on S in $O(1)$ time, using $2n + o(n)$ bits. Next, we store the sequence $r = r_1 \ldots r_n$, and for some fixed constant $\epsilon > 0$, we also store an ϵn-bit data structure of Lemma 2(1) (with $c = 1/\epsilon$) to support RMax and RMin queries on r in $O(1)$ time. Using the representations of S and r, it is easy to show that for any vertex $v \in G$, we can return its corresponding interval $I_v = [l_v, r_v]$ in $O(1)$ time by computing $l_v = \mathsf{select}_0(S, v)$, and r_v can be accessed from the sequence r. Thus, the total space usage of our representation is $n \log n + (2 + \epsilon)n + o(n)$ bits. See Figure 1 for an example.

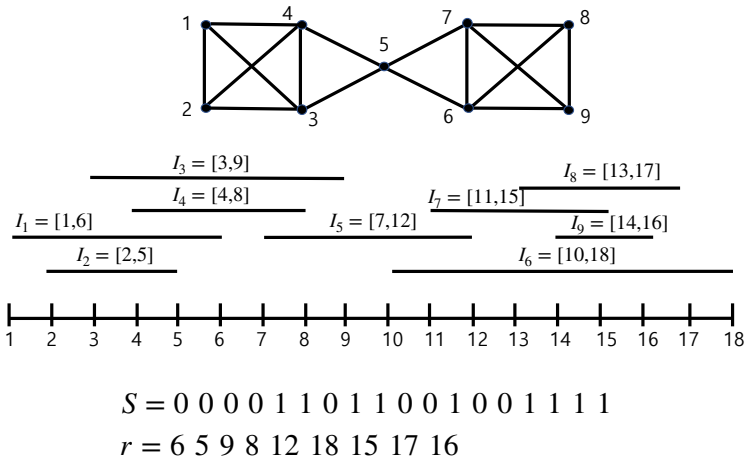


Fig. 1 Example of the interval graph and its representation.

4.2 Supporting Navigational Queries

In this section, we show that degree, adjacent, neighborhood, and spath queries on G can be answered in asymptotically optimal time using the representation described in the Section 4.1.

degree(v) query. We count the number of vertices in G which are not adjacent to v, which is a disjoint union of the two sets: (i) the set of intervals that end before the starting point l_v, and (ii) the set of intervals that start after the end point r_v. Using our representation the cardinalities of these two sets can be computed as follows. The number of intervals u with $r_u < l_v$ is given by $\mathsf{rank}_1(S, l_v)$. Similarly, the number of intervals u with $r_v < l_u$ is given by $n - \mathsf{rank}_0(S, r_v)$. Therefore, we can answer $\mathsf{degree}(v)$ query in $O(1)$ time by returning $n - \mathsf{rank}_1(S, l_v) - (n - \mathsf{rank}_0(S, r_v)) = \mathsf{rank}_0(S, r_v) - \mathsf{rank}_1(S, l_v)$.

adjacent(u, v) query. Since we can compute the intervals I_u and I_v in $O(1)$ time, $\mathsf{adjacent}(u, v)$ query can be answered in $O(1)$ by checking $r_u < l_v$ or $r_v < l_u$ (u and v are not adjacent if and only if one of these conditions is satisfied).

neighborhood(v) query. The set of all neighbors of a vertex v can be reported by considering all the intervals I_u whose left end points are within the range $[1, \ldots, r_v]$ and returning all such u's with $r_u > l_v$ (i.e., which start to the left of r_v and end after l_v). With our data structure, this query can be supported by returning the set $\{u \mid 1 \leq u \leq rank_0(S, r_v)$ and $r_u > l_v\}$. Using the RMax structure stored on r, this can be supported in $O(\mathsf{degree}(v))$ time. Note that given a threshold value l_v and a query range $[a, b]$ of the sequence r, the range max data structure can be used to report all the elements r_u within

the range $[a, b]$ such that $r_u > t$, in $O(1)$ time per element, using the following recursive procedure. Compute the position $c = \mathsf{RMax}_r(a, b)$. If $r_c > l_v$, then return r_c, and recurse on the subintervals $[a, c-1]$ and $[c+1, b]$; else stop.

spath(u, v) query. We first define the SUCC query as described in [16]. For an interval I_u, $\mathsf{SUCC}(I_u)$ returns the interval $I_{u'}$ such that $I_u \cap I_{u'} \neq \emptyset$ and there is no $I_{u''}$ with $I_u \cap I_{u''} \neq \emptyset$ and $r_{u'} < r_{u''}$. (For example in Figure 1, $\mathsf{SUCC}(I_2) = I_3$ and $\mathsf{SUCC}(I_5) = I_6$.) To answer the $\mathsf{spath}(u, v)$ query, let P_{uv} be the shortest path from u to v initialized with $\emptyset$ (without loss of generality, we assume that $u \leq v$). If u and v are identical, we simply add u to P_{uv} and return P_{uv}. If not, we first add u to P_{uv} and consider two cases as follows [16].

- If u is adjacent to v, add v to P_{uv} and return P_{uv}.
- If I_u is not adjacent to I_v, we perform $\mathsf{spath}(\mathsf{SUCC}(u), v)$ query recursively.

Since we can answer adjacent queries in $O(1)$ time, it is enough to show how to answer the SUCC queries in $O(1)$ time. Let k be the number of vertices v which satisfies $l_v < r_u$, which can be answered in $O(1)$ time by $k = rank_0(S, r_u))$. Then by the definition of SUCC query, I_i with $i = \mathsf{RMax}_r(1, k)$ gives an answer of $\mathsf{SUCC}(I_u)$ if $r_i > l_u$ (if not, there is no vertex in G adjacent to u). Therefore we can answer the SUCC query in $O(1)$ time, which implies $\mathsf{spath}(u, v)$ query can be answered in $O(|spath(u, v)|)$ time. Thus, we obtain a following theorem.

Theorem 2 *Given an interval graph G with n vertices, there exists an $n \log n + (2 + \epsilon)n + o(n)$-bit representation of G which answers $\mathsf{degree}(v)$ and $\mathsf{adjacent}(u, v)$ queries in $O(1)$ time, $\mathsf{neighborhood}(v)$ queries in $O(\mathsf{degree}(v))$ time, and $\mathsf{spath}(u, v)$ queries in $O(|\mathsf{spath}(u, v)|)$ time, for any vertices $u, v \in G$.*

5 Some graph algorithms on the succinct representation of interval graphs

The above set of operations along with the representation essentially captures the entire role of the adjacency list/array representation of the underlying interval graph. Once we have such a representation of I, we can talk about executing various algorithms on I. Here we are interested in the following set of algorithms.

Depth-first search (DFS) and Breath-first search (BFS). DFS and BFS are the two most widely known and popular graph search methods because of their versatile usage as the backbone of so many other powerful and important graph algorithms. In what follows, we show that essentially the vertices sorted by its ascending order of the labels i.e., $1, \ldots, n$ gives both DFS and BFS vertex ordering of the graph G. Note that there may be more than one valid DFS or BFS ordering on G, but here we are interested in any of those valid and correct orderings. Moreover along the lines of recent papers [3, 12, 13, 15], here we are interested only in the ordering of the vertices

in DFS and BFS traversals i.e., the order in which the vertices are visited for the first time during the DFS/BFS traversal of the input graph G, not in actually reporting the final DFS/BFS tree. Towards this, we show the following,

Theorem 3 *Given an interval graph G with n vertices, suppose we label the vertices of G with $\{1, \ldots, n\}$ suvh that for any $a, b \in G$, we have $a < b$ if and only if $l_a < l_b$. Then the ascending order from 1 to n gives a valid* DFS *and* BFS *ordering of G.*

Proof We only consider the DFS traversal in the proof (the case of BFS traversal can be proved using a similar argument). We prove by induction on the number of visited vertices. Since we can start from an arbitrary vertex in G, the theorem statement holds with starting the traversal with the vertex 1. Next, suppose that we already visited the vertices $1 \ldots i$ with $i < n - 1$ (the case $i = n - 1$ is trivial) and for every valid DFS traversal, there exists a vertex $i' > i + 1$ which is visited prior to $i + 1$. This implies that there exists at least one vertex $v \in \{1, \ldots, i\}$ such that v is adjacent to i' but not $i + 1$, contradicting the fact that $l_v < l_{i+1} < l_{i'}$. Therefore there exists a valid DFS traversal which visits the vertex $i + 1$ after visiting the vertex i. □

Perfect Elimination Ordering (PEO). PEO of a graph G, if it exists, is defined as an ordering of the vertices of G such that, for each vertex v, v and the neighbors of v that occur before v in the order form a clique [27]. If we order the vertices corresponding to the intervals by sorting based on their left endpoints, then the resulting vertex order is a PEO, as the predecessor set of every vertex forms a clique. Thus, from our representation it is trivial to generate a PEO of the given interval graph.

Maximum Independent Set (MIS) and Minimum Vertex Cover (MVC). To compute an MIS, we simulate the greedy algorithm of [31] which works as follows. Initialize the sets E and M to $\emptyset$. We first find the vertex i such that r_i is the leftmost among all the right endpoints of the intervals in $I - E$. If such an i exists, we add i to M and add $E = E \cup I'$ where $I' \subseteq I$ is the set of all intervals whose corresponding vertices are adjacent to i. We repeat this procedure until no such vertex i exists, and return M. Also MVC can be computed from MIS by returning the complement of MIS, in $O(n)$ time. (For the graph in Figure 1, MIS $= \{2, 5, 9\}$ and MVC $= \{1, 3, 4, 6, 7, 8\}$.)

Now we show how the algorithm can be implemented in time linear in the size of the input, with our representation of G. We first initialize the set M to $\emptyset$ and compute $i = \mathsf{RMin}(1, n)$ (which returns the interval with the smallest right end point among all the intervals), and add vertex i to M. Then the greedy algorithm picks the next interval with the smallest right end point in the range $[\mathsf{rank}_0(S, r_i) + 1, n]$ of the sequence r. In general, suppose $M = \{m_1, m_2 \ldots m_k\}$ and m_k is the last vertex added to M. Then we compute $m_{k+1} = \mathsf{RMin}(\mathsf{rank}_0(S, r_{m_k}) + 1, n)$, and add m_{k+1} to M, if it

exists. Thus, we can compute MIS in time linear in the size of MIS.

Computing a Maximum Clique. In order to find a maximum clique in G, we define a sequence $D = d_1, \ldots, d_{2n}$ of length $2n$ where (i) $d_1 = 1$, and (ii) for $1 < i \leq 2n$, $d_i = d_{i-1} + 1$ if $s_i = 0$ and $d_i = d_{i-1} - 1$ otherwise (for example, for the interval graph of Figure 1, $D = 1\ 2\ 3\ 4\ 3\ 2\ 3\ 2\ 1\ 2\ 3\ 2\ 3\ 4\ 3\ 2\ 1\ 0$). From the definition of d_i, if $s_i = 0$, there are exactly d_i vertices in G such that all corresponding intervals of these vertices have left endpoints at most i and right endpoints larger then i. Thus all such d_i vertices form a clique. This gives an algorithm for computing a maximum clique in G as follows. While constructing the sequence D in $O(n)$ time, we maintain the index k such that d_k is a largest value in D. We then scan all the intervals and return those intervals whose left end point is at most k and right end point is larger than k. Therefore we can compute the maximum clique in G in $O(n)$ time in total.

Computing a Proper Coloring. It is well-known that the greedy algorithm on G yields the optimal proper coloring if we process the vertices of G in the order of their corresponding intervals' left endpoints [27]. Thus, we simply implement this greedy coloring on G from the vertex 1 to n as follows. We first maintain n values $c_1, \ldots, c_n$ such that for $1 \leq i \leq n$, $c_i \leq \mathsf{degree}(i)$ stores the color of vertex i. Since each c_i can be stored using $O(\log(\mathsf{degree}(i)))$ bits, we can maintain all c_i's using $\sum_{i=1}^{n} O(\log(\mathsf{degree}(i))) = O(n \log(m/n))$ bits in total by storing it as a sequence of variable length codes (m denotes the number of edges in G). To access any element of this sequence in $O(1)$ time, we store an another bit vector of size $O(n \log(m/n))$ bits, which stores a 1 at the starting position of each code (i.e., each vertex's color), and 0 in all other positions; and store auxiliary data structure to support select queries on it. Now, initialize all $c_1, \ldots, c_n$ to 0 and scan the vertices from 1 to n. While we visit the vertex i, we perform the $\mathsf{neighborhood}(i)$ query and choose the minimum color in $\{1 \ldots \mathsf{degree}(i)\} - \{c_v | v \in \mathsf{neighborhood}(i)\}$. Since we use $O(\mathsf{degree}(i))$ time for each $\mathsf{neighborhood}(i)$ query to assign the color of i, we can assign the color of all vertices in G in $O(n + m)$ time, using $O(n \log(m/n))$ extra bits of space.

Another alternative way to implement the greedy coloring on G is to use a priority queue. In this case, we first compute $\chi(G)$, which is a chromatic number of G. Since G is an interval graph, we can compute $\chi(G)$ in $O(n)$ time on our representation by computing the size of the maximum clique of G. Now we initialize $c_1, \ldots, c_n$ to 0 and insert $1, \ldots, \chi(G)$ to the priority queue PQ, and scanning S from left to right. Suppose we currently access s_i which corresponds to I_j (we can compute the index j in $O(1)$ time). If $s_i = 0$, we assign the minimum element of PQ to c_j, and delete c_j from PQ. Otherwise, we insert c_j to PQ. Note that we exactly perform $2n$ insert operations and n delete operations on PQ. Therefore we can compute a proper coloring of G in $O(n \log\log \chi(G))$ time using $O(n \log n)$ bits of space, using the integer priority queue structure of [46]. Note that these two solutions use $\Omega(n)$ bits of space, With $O(n)$ bits, we cannot store the colors of all the vertices simultaneously

(unless the graph is sparse), and this poses a challenge for the greedy algorithm. We leave open the problem to find a proper coloring of interval graphs using extra $O(n)$ bits.

6 Representation of some related families of interval graphs

In this section, we propose space-efficient representations for proper interval graphs, k-proper and k-improper interval graphs, and circular arc graphs. Since these graphs are restrictions or extensions (i.e., sub/super-classes) of interval graphs, we can represent them by modifying the representation in Section 4.1 (to make the representation asymptotically optimal in terms of space). We also show that navigation queries on these graph classes can be answered efficiently with the modified representation.

6.1 Proper interval graphs

An interval graph G is *proper* if there exists an interval representation of G such that for any two vertices $u, v \in G$, $I_u \not\subset I_v$ and $I_v \not\subset I_u$ (let such interval representation of G be *proper representation* of G). Also it is known that proper interval graphs are equivalent to the *unit interval graphs*, which have an interval representation such that every interval has the same length [43].

Now we consider how to represent a proper interval graph G with n vertices while supporting navigational queries efficiently on G. We first obtain an interval representation of the graph G where the intervals satisfy the property of proper interval graph. We then assign labels to vertices of G based on the sorted order left end points of their corresponding intervals, as described in Section 4.1. Let S be the bit sequence obtained from this representation, as defined in Section 4.1. Then by the definition of G, there are no two vertices $u, v \in G$ with $l_u < l_v$ and $r_u > r_v$ (if so, $I_v \subset I_u$). Thus by the Lemma 1, for any vertex $i \in G$ we can compute l_i and r_i in $O(1)$ time by $\mathsf{select}_0(S, i)$ and $\mathsf{select}_1(S, i)$ respectively using $2n + o(n)$ bits. Also note that r is strictly increasing sequence when G is a proper interval graph, and hence one can support the RMax queries on $r = r_1 \dots r_n$ in $O(1)$ time without maintaining any data structure, by simply returning the rightmost position of the query range. Thus, we obtain the following theorem.

Theorem 4 *Given a proper interval graph or unit interval graph G with n vertices, there exists a $2n + o(n)$-bit representation of G which answers $\mathsf{degree}(v)$ and $\mathsf{adjacent}(u, v)$ queries in $O(1)$ time, $\mathsf{neighborhood}(v)$ queries in $O(\mathsf{degree}(v))$ time, and $\mathsf{spath}(u, v)$ queries in $O(|\mathsf{spath}(u, v)|)$ time, for any vertices $u, v \in G$.*

It is known that there are asymptotically $\frac{1}{8\kappa\sqrt{\pi}} n^{-3/2} 4^n$ non-isomorphic unlabeled unit interval graphs with n vertices, for some constant $\kappa > 0$ [22], and

hence $2n - O(\log n)$ bits is an information-theoretic lower bound on representing an arbitrary proper interval graph. Thus our representation in Theorem 4 gives a succinct representation for proper interval graphs.

6.2 k-proper and k-improper interval graphs

One can generalize the proper interval graph to the following two sub-classes of interval graphs. An interval graph G with n vertices is a *k-proper interval graph* (resp. *k-improper interval graph*) if there exists an interval representation of G such that for any vertex $v \in G$, I_v *is* contained by (resp., *c*ontains) at most $k \leq n$ intervals in G other than I_v. We call such an interval representation of G as the *k-proper representation* (resp. *k-improper representation*) of G. Note that every proper interval graph is both a 0-proper and a 0-improper graph. The graph in Figure 1 is a 2-proper, and a 3-improper graph.

Now we consider how to represent a k-proper interval graph G with n vertices and support navigation queries efficiently on G. We first represent G k-properly into n intervals, and assign the labels to vertices of G based on the sorted order of their left end points, as described in Section 4.1. Same as the representation in Section 4.1, we first maintain the data structure for supporting rank and select queries on S in $O(1)$ time, using $2n + o(n)$ bits in total. Also we maintain the $2n + o(n)$-bit data structure of Lemma 2 on $r = r_1, \ldots, r_n$ for supporting RMax queries on r in $O(1)$ time. Next, to access r without using $n \log n$ bits, we define the sequence $T = t_1 \ldots t_{2n}$ of size $2n$ over the alphabet $\{0, \ldots, 2k+1\}$ such that $t_i = 2k'$ (resp. $t_i = 2k' + 1$) if $s_i = 0$ (resp. $s_i = 1$) and its corresponding interval is contained by $k' \leq k$ intervals in $I = \{I_1 \ldots I_n\}$. Now for any $0 \leq i \leq k$, let $R_i \subset I$ be the set of all intervals such that for any $[a, b] \in R_i$, $t_a = 2i$ and $t_b = 2i + 1$. It is easy to show that each R_i corresponds to a proper interval graph. For example the graph in Figure 1 is a 2-proper interval graph, and $T = 0\,2\,0\,2\,3\,1\,0\,3\,1\,0\,2\,1\,2\,4\,3\,5\,3\,1$, $R_0 = \{I_1, I_3, I_5, I_6\}$, $R_1 = \{I_2, I_4, I_7, I_8\}$, and $R_2 = \{I_9\}$. Using $2n \log(2k+2) + o(n \log k) = 2n \log k + 2n + o(n \log k)$ bits, we can maintain the data structure of Lemma 1 on T to support rank queries in $O(\log \log k)$ time, and select queries in $O(1)$ time. Then for any vertex $v \in G$, we can answer its corresponding interval $I_v = [l_v, r_v]$ in $O(\log \log k)$ time by $l_v = \mathsf{select}_0(S, v)$ and $r_v = \mathsf{select}_{(t_{l_v}+1)}(T, \mathsf{rank}_{t_{l_v}}(T, l_v))$.

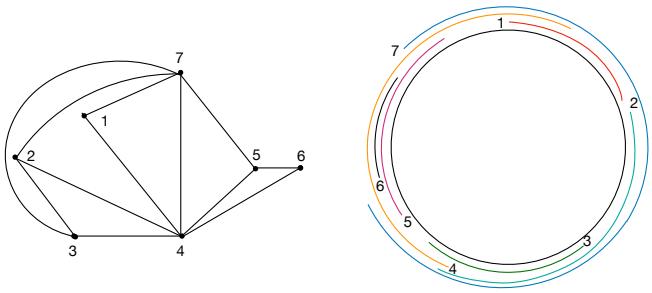
Note that we can represent k-improper interval graphs in same space with same query time as we did for k-proper interval graphs by changing the definition of T to be $t_i = 2k'$ (resp. $t_i = 2k' + 1$) if $s_i = 0$ (resp. $s_i = 1$) and its corresponding interval contains $k' \leq k$ intervals in $\{I_1 \ldots I_n\}$. Thus, we obtain the following theorem.

Theorem 5 *Given a k-(im)proper interval graph G with n vertices, there exists a $(2n \log k + 6n + o(n \log k))$-bit representation of G which answers $\mathsf{degree}(v)$ and $\mathsf{adjacent}(u, v)$ queries in $O(\log \log k)$ time, $\mathsf{neighborhood}(v)$ queries in $O(\log \log k \cdot \mathsf{degree}(v))$ time, and $\mathsf{spath}(u, v)$ queries in $O(\log \log k \cdot |\mathsf{spath}(u, v)|)$ time, for any vertices $u, v \in G$.*

6.3 Circular-arc graphs

In this section, we propose a succinct representation for circular-arc graphs, and show how to support navigation queries efficiently on the representation. Note that, a *circular-arc graph* G is a graph whose vertices can be assigned to arcs on a circle so that two vertices are adjacent in G if and only if their assigned arcs intersect. Now suppose that G is represented by the circle C together with n arcs of C. For an arc, we define its start point to be the unique point on it such that the arc continues from that point in the clockwise direction but stops in the anti-clockwise direction; and similarly define its end point to be the unique point on it such that the arc stops in the clockwise direction but continues in the anti-clockwise direction. As in the case of interval graphs, we assume, without loss of generality, that all the start and end points of all the arcs are distinct. We label the vertices of G with the integers form 1 to n as described below. We first select an arbitrary arc, and label the vertex (and the arc) corresponding to this arc by 1. We then traverse the circle from the starting point of that arc in the clockwise direction, and label the remaining vertices and arcs in the order in which their starting points are encountered during the traversal, and finish the traversal when we return to the starting point of the first arc. We also map all the start and end points of all arcs, in the order in which they are encountered in the above traversal, into the range $[1, \ldots, 2n]$ (since the start and end points of all the n arcs are distinct). Note that all these steps can be performed in $O(n+m)$ time, where m is the number of edges in G [37]. With the above defined labeling of the arcs, and the numbering of their start and end points, let l_i and r_i start and end points of the arc labeled i, for $1 \leq i \leq n$. Now the arcs can be thought of as two types of intervals in the range $[1, \ldots, 2n]$; we call an interval (and its corresponding vertex) i as *normal* if $l_i < r_i$ (i.e., we traverse l_i prior to r_i), and *reversed* otherwise. A normal vertex i corresponds to the interval $[l_i, r_i]$, while a reversed vertex i actually corresponds to the union of the two intervals $[1, \ldots, r_i]$ and $[l_i, \ldots, 2n]$. See Figure 2 for an example; vertex 4 and 7 are reversed, while the others are normal. Our representation of G consists of the following substructures.

1. Define a binary sequence $S = s_1, \ldots, s_{2n}$ of length $2n$ such that for $1 \leq i \leq 2n$, $s_i = 0$ (resp. $s_i = 1$) if i-th end point encountered during the traversal of C is in $\{l_1, \ldots, l_n\}$ (resp. $\{r_1, \ldots, r_n\}$). Now, construct a sequence $S' = s'_1, \ldots, s'_{2n}$ of size $2n$ over an alphabet $\{0, 1, 2, 3\}$ such that for all $1 \leq i \leq 2n$, $s'_i = s_i + 2$ if the position s_i corresponds to the start or end point of a reversed interval, and $s'_i = s_i$ otherwise (i.e., if s_i corresponds to a normal interval). We represent S' using the structure of Lemma 1, using $4n + o(n)$ bits, so that we can answer rank and select queries on S' in $O(1)$ time. In addition, we also store auxiliary structures (of $o(n)$ bits) on top of S' to support rank and select queries on S (without explicitly storing S – note that one can efficiently reconstruct any subsequence of S from S').
2. To store the interval end points efficiently, we introduce two 2-dimensional grids of points, R_1 and R_2, defined as follows. Suppose there are $q \leq$



$S = 0\ 1\ 1\ 0\ 0\ 0\ 1\ 1\ 0\ 1\ 0\ 1\ 0\ 1$

$S' = 0\ 3\ 1\ 0\ 0\ 2\ 1\ 1\ 0\ 3\ 0\ 1\ 2\ 1$

$R_1 = \{(1,1), (2,2), (3,3), (4,5), (5,4)\}$

$R_2 = \{(1,1), (2,2)\}$

Fig. 2 Example of the circular graph and its representation.

n normal vertices in G and $n - q$ reversed vertices. Then let R_1 be a set of q points on the 2-dimensional grid $[1, q] \times [1, q]$ which consist of $(\mathsf{rank}_0(S', l_i), \mathsf{rank}_1(S', r_i))$, for all $1 \leq i \leq n$ with $l_i < r_i$. Similarly let R_2 be a set of $n-q$ points on the 2-dimensional grid $[1, n-q] \times [1, n-q]$ which consist of $(\mathsf{rank}_2(S', l_i), \mathsf{rank}_3(S', r_i))$, for all $1 \leq i \leq n$ with $r_i < l_i$. Given a set of points R on 2-dimensional grid, we define the following queries (for any rectangular range A):
- $Y(R, x)$: returns y with $(x, y) \in R$.
- *count*(R, A): returns the number of points in R within the range A.

We represent R_1 and R_2 using $n \log n + o(n \log n)$ bits in total, such that Y and *count* queries can be supported in $O(\log n / \log \log n)$ time [9]. Thus when the vertex i is given, we can compute l_i and r_i in $O(\log n / \log \log n)$ time, using the equations $l_i = \mathsf{select}_0(S, i)$, and $r_i = \mathsf{select}_1(S', Y(R_1, \mathsf{rank}_0(S', l_i)))$ if $S'_{l_i} = 0$ (i.e., if l_i is the left end point of a normal interval), and $r'_i = \mathsf{select}_3(S', Y(R_2, \mathsf{rank}_2(S', l'_i)))$ otherwise (i.e., if l_i is the left end point of a reversed interval).

3. Finally, let $r' = r'_1, \ldots, r'_q$ be a sequence such that for $1 \leq i \leq q$, $r'_i = r_{j_i}$ with $j_i = \mathsf{select}_0(S', i)$. Similarly, let $r'' = r''_1, \ldots, r''_{n-q}$ be a sequence such that for $1 \leq i \leq n - q$, $r''_i = r_{j_i}$ with $j_i = \mathsf{select}_2(S', i)$. For example, for the circular-arc graph of Figure 2, $r = 3\ 7\ 8\ 2\ 14\ 12\ 10$, $r' = 3\ 7\ 8\ 14\ 12$, and $r'' = 2\ 10$. Then we maintain the data structure of Lemma 2 on r' and r'', using a total of $2n + o(n)$ bits, to support RMax queries on each of them. Thus, the overall representation takes $n \log n + o(n \log n)$ bits in total.

One can show that this representation supports degree, adjacent, neighborhood and spath queries efficiently, to prove the following theorem.

Theorem 6 *Given a circular arc graph G with n vertices, there exists an $(n \log n + o(n \log n))$-bit representation of G which supports $\mathsf{degree}(v)$ and $\mathsf{adjacent}(u, v)$ queries in $O(\log n / \log \log n)$ time, $\mathsf{neighborhood}(v)$ queries in $O(\mathsf{degree}(v) \cdot \log n / \log \log n)$ time, and $\mathsf{spath}(u, v)$ queries in $O(|\mathsf{spath}(u, v)| \log n / \log \log n)$ time for any two vertices $u, v \in G$.*

Proof Suppose we have the $n \log n + o(n \log n)$-bit representation described in Section 6.3. Now we consider the following queries, and show how to support them efficiently, by extending the proof in Section 4.2.

degree(v) query. To answer $\mathsf{degree}(v)$ query, we count the number of vertices u which adjacent to v by considering u into two cases as (i) u is normal and (ii) u is reversed, and return the sum of them. Now we consider the two cases by the type of v as follows.

- **v is normal:** We can count the number of vertices in (i) in $O(\log n / \log \log n)$ time by returning $\mathsf{rank}_0(S, r_v) - \mathsf{rank}_1(S, l_v)$, same as answering degree queries on interval graphs (see Section 4.2). Next, we count the vertices u in (ii) by considering three cases as 1) $l_u < l_v$, 2) $r_v < r_u$, and 3) $l_v < l_u < r_v$ or $l_v < r_u < r_v$ separately and return the sum of them. First, number of vertices in case 1) and 2) can be easily answered in $O(\log n / \log \log n)$ time by returning $\mathsf{rank}_2(S', l'_v)$ and $\mathsf{rank}_3(S', r'_v)$ respectively. To count the number of vertices in case 3), we first count the number of start and end points of reversed intervals between l_v and r_v by returning $(\mathsf{rank}_2(S', r_v) - \mathsf{rank}_2(S', l_v)) + (\mathsf{rank}_3(S', r_v) - \mathsf{rank}_3(S', l_v))$. After that we subtract the number of vertices whose both start and end points exist between l_v and r_v, which is $count(R_2, R)$ where $R = [\mathsf{rank}_2(S', l_v), \mathsf{rank}_2(S', r_v)] \times [\mathsf{rank}_3(S', l_v), \mathsf{rank}_3(S', r_v)]$. Thus we can count the number of vertices in this case in $O(\log n / \log \log n)$ time.
- **v is reversed:** We count the number of vertices u in case (i) by considering three cases as 1) $r_u < r_v$, 2) $l_v < l_u$, and 3) $r_v < l_u < l_v$ or $r_v < r_u < l_v$ separately and return the sum of them. This can be answered in $O(\log n / \log \log n)$ time by the same argument as above. For counting the vertices in (ii), we simply return $\mathsf{rank}_2(S', 2n) - 1$ since all the vertices corresponds to the reverse interval cross l_1 in C, i.e., all such vertices form a clique in G.

adjacent(u, v) query. This can be answered in $O(\log n / \log \log n)$ time by checking l_u, r_u, l_v, and r_v.

neighborhood(v) query. We only describe how to answer the vertices u adjacent to v when v is normal. The case when v is reversed can be handled similarly. First we can return the all normal vertices u adjacent to v in $O(\log n / \log \log n \cdot \mathsf{degree}(v))$ time using the same argument in the $\mathsf{neighborhood}$ query on interval graphs (see Section 4.2). Next, the set of reversed vertices u adjacent to v, is a disjoint union of the following two sets: 1) the set S_1 of all vertices u with $l_u < r_v$, and 2) the set S_2 of all vertices u

with $l_v < r_u$. We can answer all the vertices in S_1 in $O(\mathsf{degree}(v))$ time by returning $\mathsf{rank}_0(S, \mathsf{select}_2(S', 1)), \ldots, \mathsf{rank}_0(S, \mathsf{select}_2(S', \mathsf{rank}_2(S', r_v)))$, which takes $O(1)$ time per each element. Finally vertices in $S_2 - S_1$ is equivalent to the the vertices u in $\{\mathsf{rank}_0(S, r_v) + 1, \ldots, n\}$ with $l_v < r_u$. Using the data structure RMax on r'' with a query range $[\mathsf{rank}_2(S, r_v) + 1, \ldots, n - q]$ on r'', these vertices can be answered in $O(\log n / \log\log n)$ time per element by the same procedure to answer the $\mathsf{neighborhood}$ queries on interval graphs. Thus, we can answer $\mathsf{neighborhood}(v)$ query in $O(\log n / \log\log n \cdot \mathsf{degree}(v))$ time in total.

$\mathsf{spath}(u, v)$ query. We simulate the algorithm of [16] with our representation of G. We first define $SUCC$ query on circular-arc graph G and show how to answer the $SUCC(u)$ query in $O(\log n / \log\log n)$ time. For a set of vertices V of G, let V_1 and V_2 be the set of normal and reversed vertices in V respectively. Then for vertex $u \in V$, we can define $SUCC(u)$ as follows.

- If there exists a vertex in $V_2 \cap V_u$ where $V_u = \{u' | l_{u'} < r_u\}$, $SUCC(u)$ returns a vertex $u' \in V_2 \cap V_u$ with the arc u and u' are intersect, and there is no vertex $u'' \in V_2 \cap V_u$ with the arc u and u'' are intersect and $r_{u'} < r_{u''} < r_u$. Let this vertex be u_1.
- Otherwise, $SUCC(u)$ returns a vertex $u' \in V_1$ with the arc u and u' are intersect, and there is no vertex $u'' \in V_1$ with the arc u and u'' are intersect and $r_u < r_{u'} < r_{u''}$. Let this vertex be u_2.

To answer u_1, we consider two cases as follows. If $u \in V_1$, We can find u_1 by returning $\mathsf{rank}_0(S, \mathsf{select}_2(S', v'))$ where $v' = \mathsf{RMax}_{r''}(1, \mathsf{rank}_2(S', r_u))$, which can be answered in $O(\log n / \log\log n)$ time. Similarly if $u \in V_2$, we can find u_1 in $O(\log n / \log\log n)$ time by returning $\mathsf{rank}_0(S, \mathsf{select}_2(S', v'))$ where $v' = \mathsf{RMax}_{r''}(1, \mathsf{rank}_2(S', l_u))$. Also we can find u_2 in $O(\log n / \log\log n)$ time by the same argument for answering $SUCC$ queries on interval graphs.

To answer the $\mathsf{spath}(u, v)$ query, we do a same procedure for answering $\mathsf{spath}(u, v)$ and $\mathsf{spath}(v, u)$ queries on interval graphs (with the $SUCC$ function defined on circular arc graphs) in parallel, and return one of them which completes the procedure earlier. Since we can answer $SUCC$ query in $\log n / \log\log n$ time, we can answer $\mathsf{spath}(u, v)$ query in $O(\log n / \log\log n \cdot |\mathsf{spath}(u, v)|)$ time. □

It is easy to prove that we can answer Y and *count* queries on R_1 and R_2 in $O(\log n)$ time with $n \log n + o(n \log n)$ bits of space by maintaining the wavelet tree [41] on r' and r'', instead of maintaining the data structure of [9] on R_1 and R_2. This gives a simple succinct representation of G while using the same space and support degree and $\mathsf{adjacent}$ queries in $O(\log n)$ time, $\mathsf{neighborhood}$ queries in $O(\log n \cdot \mathsf{degree}(v))$ time, and $\mathsf{spath}(u, v)$ queries in $O(|\mathsf{spath}(u, v)| \log n)$ time.

Also the difference in query time on interval graphs and circular-arc graphs comes from the fact that (i) when $\mathcal{A}_v$ is given, we need to know the number of arcs which are fully contained in $\mathcal{A}_v$ on circular-arc graph to answer $\mathsf{degree}(v)$

query, and (ii) since we use the data structure of [9], we need $O(\log n/\log\log n)$ time to access any element in r' and r''. Thus, the query time can be improved by maintaining an $n\log n + O(n/c)$-bit data structure on r' and r'' to support RMax on them, instead of maintaining the data structure of [9] on R_1 and R_2. In this case, since we can access any elements in r' and r'' in $O(1)$ time, we can support $\mathsf{adjacent}$, $\mathsf{neighborhood}$, and spath queries in same time as interval graphs, using $n\log n + o(n\log n)$ bits of space in total. In addition, for every vertex $v \in G$, by storing $\mathsf{degree}(v)$ explicitly using $n\lceil\log n\rceil$ bits, we can support degree, $\mathsf{adjacent}$, $\mathsf{neighborhood}$, and spath queries in same time as interval graphs, using $2n\log n + o(n\log n)$ bits of space in total. This gives us the following:

Corollary 1 *Given a circular arc graph G with n vertices, there exists an $(n\log n + o(n\log n))$-bit representation of G which supports $\mathsf{adjacent}(u,v)$ queries in $O(1)$ time, $\mathsf{neighborhood}(v)$ queries in $O(\mathsf{degree}(v))$ time, and $\mathsf{spath}(u,v)$ queries in $O(|\mathsf{spath}(u,v)|)$ time for any two vertices $u, v \in G$. Also, using an additional $n\lceil\log n\rceil$ bits, we can support $\mathsf{degree}(v)$ queries in $O(1)$ time.*

7 Conclusion and Final Remarks

We considered the problem of succinctly encoding an unlabeled interval graph with n vertices so as to support adjacency, degree, neighborhood and shortest path queries. To this end, we designed a succinct data structure that can support these queries optimally. We also showed how one can implement various combinatorial algorithms in interval graphs using our succinct data structure in both time and space efficient manner. Extending these ideas, finally, we also showed succinct/compact data structures for multiple other variants of interval graphs. One interesting open problem is to find a lower bound on space for representing k-proper or improper graphs, to show succinctness or improve the space of our data structure of Theorem 5. Also for k-(im)proper and circular-arc graphs, the query times of our data structures are super constant while using succinct space, hence (probably) non-optimal and we leave them as open problems whether we can design succinct data structures for supporting these queries in constant time.

References

1. Acan, H., Chakraborty, S., Jo, S., Satti, S.R.: Succinct data structures for families of interval graphs. In: WADS, pp. 1–13 (2019)
2. Aleardi, L.C., Devillers, O., Schaeffer, G.: Succinct representations of planar maps. Theor. Comput. Sci. **408**(2-3), 174–187 (2008)
3. Banerjee, N., Chakraborty, S., Raman, V., Satti, S.R.: Space efficient linear time algorithms for BFS, DFS and applications. Theory Comput. Syst. **62**(8), 1736–1762 (2018)
4. Bar-Noy, A., Bar-Yehuda, R., Freund, A., Naor, J., Schieber, B.: A unified approach to approximating resource allocation and scheduling. J. ACM **48**(5), 1069–1090 (2001)

5. Barbay, J., He, M., Munro, J.I., Satti, S.R.: Succinct indexes for strings, binary relations and multilabeled trees. ACM Trans. Algorithms **7**(4), 52:1–52:27 (2011)
6. Benser, S.: On the topology of the genetic fine structure. Proc. Nat. Acad. Sci. **45**, 1607–1620
7. Bhattacharya, B.K., Kaller, D.: An o(m + n log n) algorithm for the maximum-clique problem in circular-arc graphs. J. Algorithms **25**(2), 336–358 (1997)
8. Booth, K.S., Lueker, G.S.: Testing for the consecutive ones property, interval graphs, and graph planarity using pq-tree algorithms. J. Comput. Syst. Sci. **13**(3), 335–379 (1976)
9. Bose, P., He, M., Maheshwari, A., Morin, P.: Succinct orthogonal range search structures on a grid with applications to text indexing. In: WADS, pp. 98–109 (2009). DOI 10.1007/978-3-642-03367-4_9
10. Brodal, G.S., Davoodi, P., Rao, S.S.: On space efficient two dimensional range minimum data structures. Algorithmica **63**(4), 815–830 (2012)
11. Chakraborty, S., Grossi, R., Sadakane, K., Satti, S.R.: Succinct representation for (non)deterministic finite automata. CoRR **abs/1907.09271** (2019)
12. Chakraborty, S., Mukherjee, A., Raman, V., Satti, S.R.: A framework for in-place graph algorithms. In: ESA, pp. 13:1–13:16 (2018). DOI 10.4230/LIPIcs.ESA.2018.13
13. Chakraborty, S., Raman, V., Satti, S.R.: Biconnectivity, st-numbering and other applications of DFS using O(n) bits. J. Comput. Syst. Sci. **90**, 63–79 (2017)
14. Chakraborty, S., Sadakane, K.: Indexing graph search trees and applications. In: 44th MFCS, pp. 67:1–67:14 (2019)
15. Chakraborty, S., Satti, S.R.: Space-efficient algorithms for maximum cardinality search, its applications, and variants of BFS. J. Comb. Optim. **37**(2), 465–481 (2019)
16. Chen, D.Z., Lee, D.T., Sridhar, R., Sekharan, C.N.: Solving the all-pair shortest path query problem on interval and circular-arc graphs. Networks **31**(4), 249–258 (1998)
17. Clark, D.R., Munro, J.I.: Efficient suffix trees on secondary storage. SODA, pp. 383–391 (1996)
18. Cormen, T.H., Leiserson, C.E., Rivest, R.L., Stein, C.: Introduction to Algorithms (3. ed.). MIT Press (2009)
19. Diestel, R.: Graph Theory, 4th Edition, *Graduate texts in mathematics*, vol. 173. Springer (2012)
20. Farzan, A., Kamali, S.: Compact navigation and distance oracles for graphs with small treewidth. Algorithmica **69**(1), 92–116 (2014)
21. Farzan, A., Munro, J.I.: Succinct encoding of arbitrary graphs. Theor. Comput. Sci. **513**, 38–52 (2013)
22. Finch, S.R.: Mathematical Constants (2003)
23. Fischer, J., Heun, V.: Space-efficient preprocessing schemes for range minimum queries on static arrays. SIAM J. Comput. **40**(2), 465–492 (2011)
24. Garey, M.R., Johnson, D.S., Miller, G.L., Papadimitriou, C.H.: The complexity of coloring circular arcs and chords. SIAM J. Matrix Analysis Applications **1**(2), 216–227 (1980)
25. Gavoille, C., Paul, C.: Optimal distance labeling for interval graphs and related graph families. SIAM J. Discret. Math. **22**(3), 1239–1258 (2008)
26. Golumbic, M.C.: Interval graphs and related topics. Discrete Mathematics **55**(2), 113–121 (1985)
27. Golumbic, M.C.: Algorithmic Graph Theory and Perfect Graphs (2004)
28. Golumbic, M.C., Hammer, P.L.: Stability in circular arc graphs. J. Algorithms **9**(3), 314–320 (1988)
29. Golumbic, M.C., Shamir, R.: Complexity and algorithms for reasoning about time: A graph-theoretic approach. J. ACM **40**(5), 1108–1133 (1993)
30. Golynski, A., Munro, J.I., Rao, S.S.: Rank/select operations on large alphabets: a tool for text indexing. In: SODA, pp. 368–373 (2006)
31. Gupta, U.I., Lee, D.T., Leung, J.Y.: Efficient algorithms for interval graphs and circular-arc graphs. Networks **12**(4), 459–467 (1982)
32. Habib, M., McConnell, R.M., Paul, C., Viennot, L.: Lex-bfs and partition refinement, with applications to transitive orientation, interval graph recognition and consecutive ones testing. Theor. Comput. Sci. **234**(1-2), 59–84 (2000)

33. Hajós, G.: Über eine art von graphen. Int. Math. Nachr. **11**, 1607–1620
34. Hanlon, P.: Counting interval graphs. Transactions of the American Mathematical Society **272**(2), 383–426 (1982)
35. Klavík, P., Otachi, Y., Sejnoha, J.: On the classes of interval graphs of limited nesting and count of lengths. In: 27th ISAAC, pp. 45:1–45:13 (2016)
36. Klavzar, S., Petkovsek, M.: Intersection graphs of halflines and halfplanes. Discrete Mathematics **66**(1-2), 133–137 (1987)
37. McConnell, R.M.: Linear-time recognition of circular-arc graphs. Algorithmica **37**(2), 93–147 (2003)
38. Munro, J.I., Raman, R., Raman, V., Rao, S.S.: Succinct representations of permutations and functions. Theor. Comput. Sci. **438**, 74–88 (2012)
39. Munro, J.I., Raman, V.: Succinct representation of balanced parentheses and static trees. SIAM J. Comput. **31**(3), 762–776 (2001)
40. Munro, J.I., Wu, K.: Succinct data structures for chordal graphs. In: ISAAC, pp. 67:1–67:12 (2018). DOI 10.4230/LIPIcs.ISAAC.2018.67
41. Navarro, G.: Wavelet trees for all. J. Discrete Algorithms **25**, 2–20 (2014)
42. Navarro, G.: Compact Data Structures - A Practical Approach. Cambridge University Press (2016)
43. Roberts, F.S.: "Indifference graphs", in Harary, Frank, Proof Techniques in Graph Theory (1969)
44. Sadakane, K.: Succinct data structures for flexible text retrieval systems. J. Discrete Algorithms **5**(1), 12–22 (2007)
45. Sloane, N.J.E.: The on–line encyclopedia of integer sequences. http://oeis.org
46. Thorup, M.: Integer priority queues with decrease key in constant time and the single source shortest paths problem. J. Comput. Syst. Sci. **69**(3), 330–353 (2004)
47. Yang, J.C., Pippenger, N.: On the enumeration of interval graphs. Proc. Amer. Math. Soc. Ser. B **4**(1), 1–3 (2017)
48. Zhang, P., Schon, E.A., Cayanis, E., Fischer, S.G., Bourne, P.E., Weiss, J., Kistelr, S.: An algorithm based on graph theory for the assembly of contigs in physical mapping of DNA. Bioinformatics **10**(3), 309–317 (1994)