Closing the Gap: Minimum Space Optimal Time Distance Labeling Scheme for Interval Graphs

Meng He
Faculty of Computer Science, Dalhousie University, Halifax, Canada

Kaiyu Wu
Faculty of Computer Science, Dalhousie University, Halifax, Canada

Abstract

We present a distance labeling scheme for an interval graph on n vertices that uses at most $3\lg n + \lg\lg n + O(1)$ bits per vertex to answer distance queries, which ask for the distance between two given vertices, in constant time. Our labeling scheme improves the distance labeling scheme of Gavoille and Paul for connected interval graphs which uses at most $5\lg n + O(1)$ bits per vertex to achieve constant query time. Our improved space cost matches a lower bound proven by Gavoille and Paul within additive lower order terms and is thus optimal. Based on this scheme, we further design a $6\lg n + 2\lg\lg n + O(1)$ bit distance labeling scheme for circular-arc graphs, with constant distance query time, which improves the $10\lg n + O(1)$ bit distance labeling scheme of Gavoille and Paul.

We give a $n/2 + O(\lg^2 n)$ bit labeling scheme for chordal graphs which answers distance queries in $O(1)$ time. The best known lower bound is $n/4 - o(n)$ bits.

2012 ACM Subject Classification Theory of computation → Data structures design and analysis; Theory of computation → Data compression

Keywords and phrases Distance Labeling, Interval Graph, Circular-Arc Graph, Chordal Graph

Digital Object Identifier 10.4230/LIPIcs.CPM.2024.17

Funding This work was supported by NSERC.

1 Introduction

A notion closely related to that of a centralized data structure for computing a query is the notion of a *labeling scheme*. Formally introduced by Peleg [29], a labeling scheme assigns a relatively short label to each vertex (using an encoder function), and to answer a query, uses only the labels of the vertices involved in the query (using a decoder function). Thus a labeling scheme can be viewed as a distributed form of a data structure, where we split the data structure among the vertices of the graph. The distributed nature of the data structure is highly applicable in distributed settings, where the computation only has access to the data stored at the node and not the overall topology of the network (nor the data at other nodes). By distributing the data structure, we avoid large centralized data structures, which are costly and often will not fit in faster levels of memory. Furthermore, the length of the labels are important as the labels will need to be transmitted between the nodes of a network. Thus the quality of a labeling scheme is measured as the worst case label length (i.e. we wish to split the data structure as evenly as possible) and the worst case time to decode the labels to answer the query. One important property to note is that a labeling scheme can be trivially converted to a more traditional data structure, by simply storing all the labels. Thus the total space of a labeling scheme will be at least as much as the space for an optimal data structure. However, it is often the case that labeling schemes will use more overall space than the optimal data structure due to the distributed nature of the model.

Many operations on graphs and trees have been considered in the labeling model. For instance, labeling schemes computing adjacency in general undirected graphs [8] or in trees [4] have been considered. In subclasses of graphs, hereditary graphs classes with at least $2^{\Omega(n^2)}$ members have adjacency labelings that use optimal space [10]. For trees, there are many operations that are considered, aside from adjacency. For example, labelings checking ancestry [15], computing the lowest/least common ancestor [21], and the ancestor at any given depth [16]. The distance operation, which returns the distance between two vertices of a graph, has been considered for many classes, such as general graphs [19, 5], planar graphs [22], interval graphs [17], and trees [16]. Labeling schemes for the distance operation is highly applicable to network routing [13, 14].

The distance operation, is a fundamental operation in graphs, and has been extensively studied outside of the labeling model. As distance queries are able to compute adjacency queries on unweighted graphs, the space needed is at least as much as for any data structure or labeling for the adjacency query. One solution to the distance query is to precompute and store the distances between all pairs of vertices, which incurs a quadratic space cost, but for general graphs, such a space cost is unavoidable. To achieve better space costs, work has focused on designing approximate solutions [28, 1]. For instance, Patrascu and Roditty [28] constructs a data structure occupying $O(n^{5/3})$ words of space which computes the approximate distance within a factor of 2. Other work has focused on subclasses of graphs which admits smaller space solutions, such as planar graphs [26, 25], interval graphs [17, 23] and chordal graphs [31, 27].

Here we consider labeling schemes for the distance query. The graph classes we consider are unweighted interval graphs, circular arc graphs and chordal graphs. These graphs are *intersection graphs*, where the edges can be encoded in the intersection structure of sets. That is for every vertex v, we can associate it with a set s_v so that two vertices u, v are adjacent exactly when $s_u \cap s_v \neq \emptyset$. We say that the collection of sets $\{s_v; v \in V\}$ is an *intersection model* of the graph. An *interval graph* is thus a graph where we can find an intersection model where the sets are intervals on the real line, or simply, the intersection graph of intervals on the real line. A *circular arc* graph is the intersection graph of arcs on a circle, and a *chordal graph* is the intersection graph of subtrees (a set of connected nodes, rather than an entire subtree rooted at some node) in a tree. These graphs have nice combinatorial structures where many otherwise NP-Hard problems (such as maximum independent set, clique etc...), can be solved on them in polynomial time. They also have applications in compiler design [30], operations research [9] and bioinformatics [34] among others where the specific objects they study can be modeled by these classes of graphs.

Particularly for interval graphs, there is a gap between the lower and upper bounds of a distance labeling scheme, where the lower bound is $3\lg n - O(\lg\lg n)$ bits while the upper bound is $5\lg n + O(1)$ bits [17]. And one of our aims is to close this gap and give tight results for this class of graphs.

1.1 Related Work

For the distance labeling model, many classes of graphs have been considered. For general graphs, a distance labeling scheme occupying $\frac{\lg 3}{2}n + o(n)$ bits (about $0.795n$) exists [5] with $O(1)$ decode (i.e time to compute the query) time along with a matching $\Omega(n)$ bit lower bound [19]. For planar graphs, A lower bound of $\Omega(n^{1/3})$ is shown [19]. The best labeling scheme uses $O(\sqrt{n})$ bits [22], but incurs a matching $O(\sqrt{n})$ decode time. With a bit more space, a labeling scheme using $O(\sqrt{n \lg n})$ bits can be decoded in $O(\lg^3 n)$ time [22]. For interval graphs, Gavoille and Paul [17] gave a $5\lg n + O(1)$ bit labeling scheme with $O(1)$

decode time along with a $3\lg n - o(\lg n)$ bit lower bound. For circular arc graphs, they gave a $10\lg n + O(1)$ bit labeling scheme with $O(1)$ decode time. For permutation graphs, Katz et al. [24] gave a $O(\lg^2 n)$ bit labeling scheme with $O(\lg n)$ decode time.

On trees, we also have a variety of queries based on the ancestor-descendant relationships, among others:

- Adjacency: determine if one vertex is the parent of another. A $\lg n + O(1)$ bit scheme with $O(1)$ decode time is given by Alstrup et al. [4], along with a matching lower bound.
- Ancestry: determine if one vertex is an ancestor of another. A $\lg n + O(\lg\lg n)$ bit scheme is given by Fraigniaud and Korman [15], and a matching lower bound is given by Alstrup et al. [3].
- Lowest common ancestor (LCA): determine the label of the lowest common ancestor of two vertices. A $2.318\lg n + o(\lg n)$ bit labeling scheme with $O(1)$ decode time is given by Gawrychowski [21], while the lower bound is $1.008\lg n$ bits [7]. If we also need to return a predetermined k bit label of the lower common ancestor, then Alstrup et al. [7] gives a labeling scheme of length $(3 + k)\lg n$ bits with $O(1)$ decode time. If $k = \Theta(\lg n)$, then a matching $\Theta(\lg^2 n)$ lower bound is shown by Peleg [29].
- Level ancestor: return the label of the ancestor of a vertex v at depth d. A $\frac{1}{2}\lg^2 n + O(\lg n)$ bit scheme is given by Alstrup et al. [6], which matches a lower bound of $\frac{1}{2}\lg^2 n - \lg n \lg\lg n$ of Freedman et al. [16].
- Distance: return the distance between between two vertices. A $\frac{1}{4}\lg^2 n + o(\lg^2 n)$ bit labeling scheme is given by Freedman et al. [16], with a matching lower bound given by Alstrup et al. [6].

We refer to the survey of Gavoille and Peleg [18], and references therein, for a survey of labeling schemes and their applications in distributed computing.

1.2 Our Results

Our main contribution is a $3\lg n + \lg\lg n + O(1)$ bit labeling scheme for interval graphs with $O(1)$ decode time, which improves the $5\lg n + O(1)$ labeling scheme given by Gavoille and Paul [17]. This matches the $3\lg n - o(\lg n)$ lower bound they proved up to lower order terms. We further note that Gavoille and Paul assumed that the interval graph is connected in their paper, and did not discuss what do if it were disconnected, while our solution works for general interval graphs. We also first consider connected interval graphs, and give a $3\lg n + O(1)$ bit distance labeling scheme with $O(1)$ decode time, before generalizing it.

To do this, we adapt the distance algorithm on interval graphs of He et al. [23] which uses level ancestor queries. The main advancement compared to similar structures such that of Chen et al. [12] is the fact that a tree encoding the distances can be constructed such that a level-order traversal of the tree gives exactly the vertices in a left to right scan of the (left endpoints of the) intervals, and thus comparisons of these endpoints can be done by comparing properties of the corresponding nodes of the tree. We note that Gavoille and Paul [17] stated that such an approach using level ancestor queries was impossible, as any labeling scheme for level ancestors queries would need $\Omega(\lg^2 n)$ bits [16], and this may be the reason for the gap between upper and lower bounds ($5\lg n$ vs $3\lg n$) to exist. The key insight for our approach is that, although we use level ancestor queries as our basis, we do not need to compute the exact ancestor node (and thus its label), but rather some properties of that ancestor. These properties are also not exact, but approximate (for example, rather than the exact index of a node visited in a post-order traversal, we only need to know whether this index is less than some integer i), which allows us to bypass the level ancestor query lower bound. This optimal scheme for interval graphs also immediately improves the distance labeling scheme of circular arc graphs from $10 \lg n + O(1)$ bits [17] to $6 \lg n + 2 \lg \lg n + O(1)$ bits, with $O(1)$ decode time.

We then apply the labeling scheme for interval graphs to chordal graphs to obtain the first distance labeling schemes for chordal graphs. We obtain a distance labeling scheme of length $n/2 + O(\lg^2 n)$ bits. We note that as the lower bound on chordal graphs data structures is $n^2/4 - o(n)$ bits via an enumeration argument [27, 33], any distance labeling scheme will require $n/4 - o(n)$ bit label lengths.

2 Preliminaries

2.1 Definitions and Notation

We will use the standard graph theoretic notations. Let $G = (V, E)$ be a graph. We set $n = |V|$ the number of vertices and $m = |E|$ the number of edges. As is standard, we will use the word-RAM model with $\omega = \Theta(\lg n)$ bit words.

We use the standard definitions of graph and tree operations. For graphs, the operations we use are

- `adjacent`(u, v) which tests if two vertices u and v are adjacent.
- `distance`(u, v) which returns the (unweighted) distance between two vertices u and v.

For trees the standard operations we use are

- `depth`(v) which returns the depth of node v.
- `lev_anc`(v, d) which returns the ancestor of a node v at a given depth d.
- `parent`(v) which returns the parent node of the given node v.
- `LCA`(u, v) which returns the lowest (i.e. largest depth) common ancestor of two given nodes u and v.
- `node_rank`$_X(v)$ which returns the index of v in the X traversal of the tree, where X is `PRE`, `POST` or `LEVEL` indicating a preorder, post-order or level-order (i.e. a breadth-first traversal where we visit the children from left to right) traversal of the tree.

A labeling scheme is a distributed data structure, where each vertex of the graph contains a piece of the data structure and queries must be computed using only those pieces available at the relevant vertices. Formally, a *distance labeling scheme* for a graph G with n vertices is a pair of functions (L, f) where $L(G, v)$, typically referred to as a encoder or marker algorithm, computes a label from a vertex v of G, and f, typically referred to as a decoder algorithm, computes the distance between two vertices given their labels. That is $f(L(G, u), L(G, v)) = \texttt{distance}_G(u, v)$. The size or length of the labeling scheme is the maximum length over all possible labels: $\max_{G,v} |L(G, v)|$. We note that there is a dichotomy between L and f, where L can be computed using information from the entire graph, f cannot and can use only the labels, without further information about the original graph that the labels come from.

2.2 Interval Graph

An interval graph is a graph where the intersection model is a set of closed interval on the real line, where we write the interval as $I_v = [l_v, r_v]$. By sorting the endpoints (and breaking ties such that left endpoints come before right endpoints to preserve the intersection, and arbitrarily otherwise) we may assume that the endpoints are distinct integers in the range $[1, 2n]$. We will name the vertices $1, \ldots, n$, in the order of their left endpoints, so that for two vertices $u < v$ we have $l_u < l_v$. We will now review some of the lemmas used in the computation of distances in interval graphs.

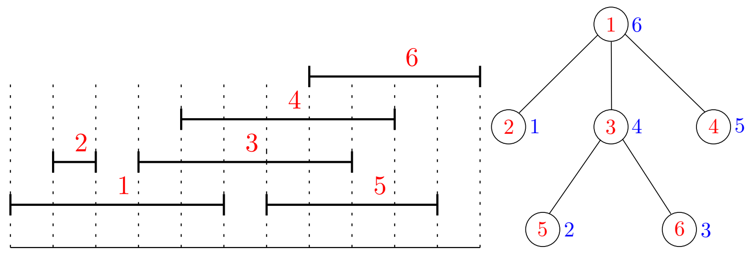


Figure 1 An interval graph on 6 vertices. The intersection model is shown on the left, while the distance tree (blue labels next to the nodes represent post-order numbers) is shown on the right.

For each vertex v, we define its parent, $\texttt{parent}(v)$, to be the minimum vertex u (i.e. the one with minimal left endpoint) adjacent to v. This can be expressed by the following formula: $\arg\min\{l_u \mid r_u \geq l_v\}$.

Using this parent-child relationship (where the root of the tree is the vertex v with $1 = v = \texttt{parent}(v)$), we may build a tree T which we will call the *distance tree*, where for each internal node, its children are in sorted order (by left endpoint).

Crucial to the data structure is the index of the nodes of the tree in some traversal of the tree denoted by $\texttt{node\_rank}_X(T, v)$, where X is $\texttt{PRE}, \texttt{POST}$ or $\texttt{LEVEL}$. We will omit the tree T when the tree being referred to is clear. The main property of this tree is that the vertex order sorted by left endpoint is exactly the vertex order obtained in a level-order traversal of the tree:

▶ **Lemma 1** (Lemma 7 of [23]). *Let G be an interval graph with distance tree $T(G)$ and vertices u, v. Then $\texttt{node\_rank}_{\texttt{LEVEL}}(u) < \texttt{node\_rank}_{\texttt{LEVEL}}(v)$ if and only if $l_u < l_v$.*

This is quite intuitive since we ordered the children of every node of the tree by their left endpoints, so that the property can propagate up the tree. Furthermore, by the property of these traversals, in the case that $\texttt{depth}(u) = \texttt{depth}(v)$, $\texttt{node\_rank}_X(u) < \texttt{node\_rank}_X(v)$ if and only if $l_u < l_v$ for $X = \texttt{PRE}, \texttt{POST}, \texttt{LEVEL}$, as on each level of the tree, the traversals visit the nodes from left to right; see Figure 1.

The shortest path algorithm used in previous works [12, 2, 27, 23] is the recursive algorithm given in Algorithm 1, for two vertices in the same connected component of the interval graph. The correctness can be summarized as the following lemma (though not explicitly stated as a lemma in some previous papers):

▶ **Lemma 2** (Lemma 8 [27], Lemma 4,6 [12]). *Let G be an interval graph with distance tree $T(G)$, and u, v be two vertices in the same connected component of G with $\texttt{node\_rank}_{\texttt{LEVEL}}(u) < \texttt{node\_rank}_{\texttt{LEVEL}}(v)$ (i.e. $u < v$). Let the node to root path of v be $v = v_{\texttt{depth}(v)}, \ldots, v_0 = r$, and i be the first (i.e. largest) index where $l_{v_i} \leq r_u$. Then a shortest path from u to v is $u = v_{\texttt{depth}(v)}, \ldots, v_i, u$, and furthermore, i is $\texttt{depth}(u) - 1$, $\texttt{depth}(u)$ or $\texttt{depth}(u) + 1$.*

Algorithm 1 Shortest Path computation between vertices u and v with $u < v$.

```
1: path = empty
2: while true do
3:   if adjacent $(u, v)$ then
4:     path = path, $v, u$
5:     return path
6:   path = path, $v$
7:   $v \leftarrow \texttt{parent}(v)$
```

The above lemma also allows us to compute the distance, as there are at most 3 candidates for v_i, which we may check individually using `lev_anc`.

3 Distance Labeling in Interval Graphs

In this section, we will consider distance labeling schemes for interval graphs. We will first assume that our graph is connected, and then generalize it to arbitrary interval graphs. As Gavoille and Paul [17] showed, any distance labeling scheme requires at least $3\lg n - O(\lg\lg n)$ bits. Thus, our goal is to give a labeling scheme that uses $3\lg n + \lg\lg n + O(1)$ bits matching their lower bound up to lower order terms. We will use the distance computation method outlined in Lemma 2 and level ancestor queries as the basis of our scheme.

3.1 Labeling Scheme for Connected Interval Graphs

Now we consider the distance computation method outlined in Lemma 2. Given two vertices u and v such that $u < v$ (we assume $u \neq v$ as that is trivial), we computed the path to the root in the distance tree from v as $v = v_{\texttt{depth}(v)}, \ldots, v_0$ and computed the first index i such that v_i is adjacent to u. In Lemma 2, we had 3 candidates for v_i. We will now narrow it down to 2 by examining the relative positions of u and v in the tree.

▶ **Lemma 3.** *Let G be an interval graph with distance tree T. Let u and v be two vertices of G such that $u < v$. Depending on the positioning of u and v we define the ancestor w (of v) as follows:*

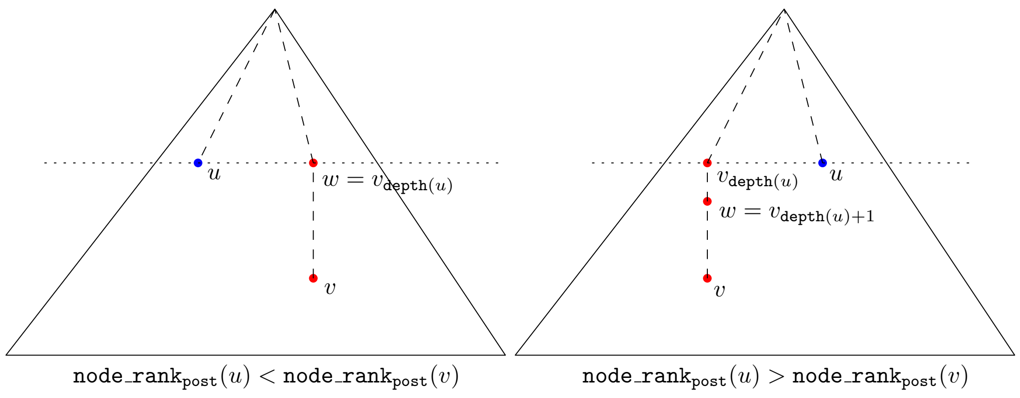
1. *Suppose that* $\texttt{node\_rank}_{\texttt{POST}}(u) < \texttt{node\_rank}_{\texttt{POST}}(v)$. *Let* $w = \texttt{lev\_anc}(v, \texttt{depth}(u))$.
2. *Suppose that* $\texttt{node\_rank}_{\texttt{POST}}(u) > \texttt{node\_rank}_{\texttt{POST}}(v)$. *Let* $w = \texttt{lev\_anc}(v, \texttt{depth}(u) + 1)$.

Then $w = v_i$ if u and w are adjacent. Otherwise, $w = v_{i-1}$.

Proof. See Figure 2 for an illustration of the two cases in the lemma statement. Define $d = \texttt{depth}(u)$ if $\texttt{node\_rank}_{\texttt{POST}}(u) < \texttt{node\_rank}_{\texttt{POST}}(v)$ and $d = \texttt{depth}(u) + 1$ if $\texttt{node\_rank}_{\texttt{POST}}(u) > \texttt{node\_rank}_{\texttt{POST}}(v)$. Then $w = v_d$ as defined in this lemma. By the definition of `parent`, $l_{v_j} > l_{v_{j-1}}$ for every j. Then $l_{v_j} > l_{v_d} > l_u$ for all $j > d$. Thus for any v_j with $j > d$, v_j cannot be adjacent to u, as otherwise $l_{v_j} \le r_u$ and thus u would be considered as a possible vertex in the definition of $\texttt{parent}(v_j)$. But as $v_{j-1} = \texttt{parent}(v_j)$, we must have $l_{v_d} \le l_{v_{j-1}} < l_u$, a contradiction.

In the case where u and w are adjacent, $w = v_i$ by definition as it is the first vertex adjacent to u on the path towards the root (Lemma 2). Otherwise, suppose that u and w are not adjacent. By the definition of a post-order traversal, we have $\texttt{node\_rank}_{\texttt{POST}}(w) \ge \texttt{node\_rank}_{\texttt{POST}}(v)$ since ancestors are visited later in the traversal. In the first case, since $\texttt{depth}(w) = \texttt{depth}(u)$ and $\texttt{node\_rank}_{\texttt{POST}}(u) < \texttt{node\_rank}_{\texttt{POST}}(v) \le \texttt{node\_rank}_{\texttt{POST}}(w)$, we have $l_u < l_w$. Furthermore, since $\texttt{depth}(\texttt{parent}(w)) = \texttt{depth}(u) - 1$, $l_{\texttt{parent}(w)} < l_u$ because it comes before u in a level-order traversal (it has a smaller depth). In the second case where $\texttt{node\_rank}_{\texttt{POST}}(u) > \texttt{node\_rank}_{\texttt{POST}}(v)$, because $\texttt{depth}(w) = \texttt{depth}(u) + 1$, we have $l_w > l_u$. This also implies that $\texttt{depth}(\texttt{parent}(w)) = \texttt{depth}(u)$. By assumption $\texttt{node\_rank}_{\texttt{POST}}(u) > \texttt{node\_rank}_{\texttt{POST}}(v)$ which implies that $\texttt{node\_rank}_{\texttt{POST}}(u) > \texttt{node\_rank}_{\texttt{POST}}(\texttt{parent}(w))$, so we have $l_{\texttt{parent}(w)} < l_u$ as it comes before u in a level-order traversal.

Therefore, in either situation, we have the inequalities $l_{\texttt{parent}(w)} < l_u < l_w$. By definition of `parent` we have $r_{\texttt{parent}(w)} > l_w$. Therefore, $l_{\texttt{parent}(w)} < l_u < r_{\texttt{parent}(w)}$ and thus u and $\texttt{parent}(w)$ are adjacent. Hence $\texttt{parent}(w)$ is the first vertex adjacent to u on the path towards the root from v so $w = v_{i-1}$ (Lemma 2). ◀



u
$w = v_{\mathtt{depth}(u)}$
v
$\mathtt{node\_rank}_{\mathtt{post}}(u) < \mathtt{node\_rank}_{\mathtt{post}}(v)$
$v_{\mathtt{depth}(u)}$
u
$w = v_{\mathtt{depth}(u)+1}$
v
$\mathtt{node\_rank}_{\mathtt{post}}(u) > \mathtt{node\_rank}_{\mathtt{post}}(v)$

Figure 2 The two cases based on the relative positioning of u and $v_{\mathtt{depth}(u)}$ on the level $\mathtt{depth}(u)$. In the first case, u is to the left of the ancestor of v, while in the second case, u to the right. The node w, which will later be called the *representative* of v with respect to u, is the smallest depth ancestor that is after u in a level-order traversal. In the first case, w would be on the same level as u, while in the second w is on the next level.

The node w, which is the smallest depth ancestor with $l_w > l_u$ (i.e. to the right of u in the intersection model and in a level-order traversal of the tree) will be denoted as the *representative* of v with respect to u. This is because to compute the distance between u and v, it suffices to determine whether w is adjacent to u or not, and to compute the distance between v and w. Since w is an ancestor of v, this distance is simply the difference in the depths of w and v. We will rewrite Algorithm 1 as Algorithm 2 to take advantage of this observation.

Algorithm 2 Distance computation between vertices u and v with $u < v$.

```
1: if $\mathtt{node\_rank}_{\mathtt{POST}}(u) < \mathtt{node\_rank}_{\mathtt{POST}}(v)$ then
2:     $w \leftarrow \mathtt{lev\_anc}(v, \mathtt{depth}(u))$
3: else
4:     $w \leftarrow \mathtt{lev\_anc}(v, \mathtt{depth}(u) + 1)$
5: distance = $\mathtt{depth}(v) - \mathtt{depth}(w)$
6: if $\mathtt{adjacent}(u, w)$ then
7:     distance = distance+1
8: else
9:     distance = distance+2
10: return distance
```

To convert Algorithm 2 into an algorithm that uses only labels, we need to do the following steps using only labels:

1. Test whether $u < v$
2. Compute $\mathtt{depth}(v)$ and $\mathtt{depth}(u)$
3. Compute $\mathtt{node\_rank}_{\mathtt{POST}}(v)$ and $\mathtt{node\_rank}_{\mathtt{POST}}(u)$
4. Compute (an approximation of) `lev_anc`
5. Compute `adjacent` using the approximation of `lev_anc`.

Our labeling scheme will consist of the following 3 integers for each vertex v, each using $\lceil \lg n \rceil$ bits:

1. $\mathtt{depth}(v)$
2. $\mathtt{node\_rank}_{\mathtt{POST}}(v)$
3. $\mathtt{node\_rank}_{\mathtt{POST}}(\mathtt{last}(v))$

Here, $\texttt{last}(v)$ is the rightmost neighbour of v (i.e. the neighbour with the largest left endpoint). In the case where v is the rightmost vertex, the rightmost neighbour of v is to its left, and we consider this to be an invalid case and set $\texttt{last}(v) = \text{nothing}$ (and we will not need it in our computation).

An important property of $\texttt{last}(v)$ is

▶ **Lemma 4.** *Let G be an interval graph with distance tree T. Let v be a vertex that is not the rightmost vertex. Then every vertex w such that $v < w \leq \texttt{last}(v)$ is adjacent to v.*

Proof. Let w be a vertex such that $v < w \leq \texttt{last}(v)$ (by our naming convention, this is also an inequality on the left endpoints of these vertices). Since $\texttt{last}(v)$ is adjacent to v, we have $l_v < l_{\texttt{last}(v)} < r_v$. Thus we have $l_v < l_w \leq l_{\texttt{last}(v)} < r_v$, so w is adjacent to v. ◀

Now we show how to compute the steps using our labels.

Step 1: Decide if $u < v$. By Lemma 1, we have $u < v$ exactly when $\texttt{node\_rank}_{\texttt{LEVEL}}(u) < \texttt{node\_rank}_{\texttt{LEVEL}}(v)$. If $\texttt{depth}(u) < \texttt{depth}(v)$, then the inequality of $\texttt{node\_rank}_{\texttt{LEVEL}}$ is implied. Otherwise, if $\texttt{depth}(u) = \texttt{depth}(v)$, then $\texttt{node\_rank}_{\texttt{LEVEL}}(u) < \texttt{node\_rank}_{\texttt{LEVEL}}(v)$ if and only if $\texttt{node\_rank}_{\texttt{POST}}(u) < \texttt{node\_rank}_{\texttt{POST}}(v)$. If neither is the case, then we have $v < u$, so we switch the two vertices in the algorithm.

Step 2,3: Compute $\texttt{depth}(u), \texttt{depth}(v)$, $\texttt{node\_rank}_{\texttt{POST}}(u), \texttt{node\_rank}_{\texttt{POST}}(v)$. This is immediate as we store them as part of the label.

Step 4,5: Compute $\texttt{adjacent}$ using the approximation of $\texttt{lev\_anc}$. We will use $\texttt{node\_rank}_{\texttt{POST}}(v)$ as our approximation of $\texttt{node\_rank}_{\texttt{POST}}(w)$ in our calculations. To compute $\texttt{adjacent}(u, w)$ using $\texttt{node\_rank}_{\texttt{POST}}(v)$, we will use the following lemma (see Figure 3):

▶ **Lemma 5.** *Let G be an interval graph and T be its distance tree. Let u and v be two vertices such that $u < v$. Let w be the representative of v with respect to u, as defined in Lemma 3. The following two cases mirror the two cases used to define w:*
Suppose that $\texttt{node\_rank}_{\texttt{POST}}(u) < \texttt{node\_rank}_{\texttt{POST}}(v)$. Then u and w are adjacent if and only if $\texttt{node\_rank}_{\texttt{POST}}(v) \leq \texttt{node\_rank}_{\texttt{POST}}(\texttt{last}(u))$ or $\texttt{node\_rank}_{\texttt{POST}}(\texttt{last}(u)) < \texttt{node\_rank}_{\texttt{POST}}(u)$.
Suppose that $\texttt{node\_rank}_{\texttt{POST}}(u) > \texttt{node\_rank}_{\texttt{POST}}(v)$. Then u and w are adjacent if and only if $\texttt{node\_rank}_{\texttt{POST}}(v) < \texttt{node\_rank}_{\texttt{POST}}(\texttt{last}(u)) < \texttt{node\_rank}_{\texttt{POST}}(u)$.

Proof. First we examine the relationship between the post-order ranks of u and $\texttt{last}(u)$ (i.e. $\texttt{node\_rank}_{\texttt{POST}}(u)$ and $\texttt{node\_rank}_{\texttt{POST}}(\texttt{last}(u))$). Since $\texttt{last}(u)$ is the rightmost neighbour of u, it comes after u in level-order, so $\texttt{depth}(\texttt{last}(u)) \geq \texttt{depth}(u)$. Since $\texttt{last}(u)$ is adjacent to u, $l_{\texttt{parent}(\texttt{last}(u))} \leq l_u$ by the definition of parent. Thus $\texttt{depth}(\texttt{last}(u)) = \texttt{depth}(\texttt{parent}(\texttt{last}(u))) + 1 \leq \texttt{depth}(u) + 1$. Thus $\texttt{last}(u)$ is either on the same level as u or the level below. If it is on the same level as u, then as it is to the right of u, we have $\texttt{node\_rank}_{\texttt{POST}}(u) \leq \texttt{node\_rank}_{\texttt{POST}}(\texttt{last}(u))$. If it is on the level below u, then since $\texttt{depth}(\texttt{parent}(\texttt{last}(u))) = \texttt{depth}(u)$, we have

$$\texttt{node\_rank}_{\texttt{POST}}(\texttt{last}(u)) < \texttt{node\_rank}_{\texttt{POST}}(\texttt{parent}(\texttt{last}(u))) \leq \texttt{node\_rank}_{\texttt{POST}}(u)$$

Thus by checking the relationship between their post-order ranks we may determine the depth of $\texttt{last}(u)$ (and vice versa).

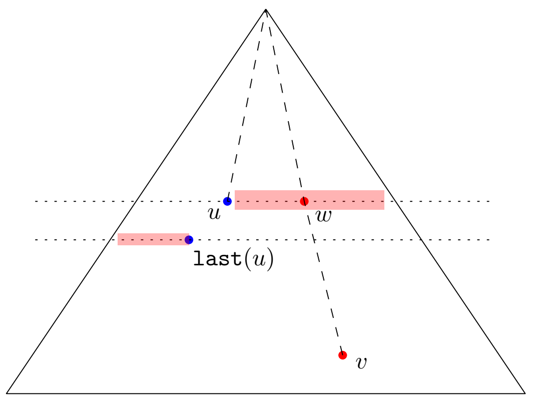


Figure 3 To determine if w is adjacent to u, we need to check if w lies in the shaded region between u and $\texttt{last}(u)$ in level-order. To do so, we will need to compare the relative positioning of w and $\texttt{last}(u)$ when they are on the same level. If w is on the level above $\texttt{last}(u)$ (so it must come before it in level-order), then w is in the shaded region. If w is on the level below $\texttt{last}(u)$ (so it must come after it in level-order), then w cannot be in the shaded region.

In the first case, suppose that $\texttt{node\_rank}_{\texttt{POST}}(u) < \texttt{node\_rank}_{\texttt{POST}}(v)$. By the definition of the representative of v with respect to u, $\texttt{depth}(w) = \texttt{depth}(u)$, and w is to the right of u on that level (so we have $\texttt{node\_rank}_{\texttt{POST}}(w) > \texttt{node\_rank}_{\texttt{POST}}(v) > \texttt{node\_rank}_{\texttt{POST}}(u)$). By Lemma 4, u and w are adjacent if and only if

$$\texttt{node\_rank}_{\texttt{LEVEL}}(w) \in (\texttt{node\_rank}_{\texttt{LEVEL}}(u), \texttt{node\_rank}_{\texttt{LEVEL}}(\texttt{last}(u))]$$

If $\texttt{depth}(\texttt{last}(u)) = \texttt{depth}(u)$ (equivalently, $\texttt{node\_rank}_{\texttt{POST}}(\texttt{last}(u)) > \texttt{node\_rank}_{\texttt{POST}}(u)$), then all three nodes are on the same level, so we may translate it them to post-order numbers as $\texttt{node\_rank}_{\texttt{POST}}(w) \in (\texttt{node\_rank}_{\texttt{POST}}(u), \texttt{node\_rank}_{\texttt{POST}}(\texttt{last}(u))]$. The first half is satisfied by assumption, so we are left with just $\texttt{node\_rank}_{\texttt{POST}}(w) \leq \texttt{node\_rank}_{\texttt{POST}}(\texttt{last}(u))$. If $\texttt{depth}(\texttt{last}(u)) = \texttt{depth}(u)+1$ (equivalently, $\texttt{node\_rank}_{\texttt{POST}}(\texttt{last}(u)) < \texttt{node\_rank}_{\texttt{POST}}(u)$), then the range $(\texttt{node\_rank}_{\texttt{LEVEL}}(u), \texttt{node\_rank}_{\texttt{LEVEL}}(\texttt{last}(u))]$, restricted to the level $\texttt{depth}(u)$ contains all the nodes on the level $\texttt{depth}(u)$ to the right of u, which w satisfies by definition (as it is a node on level $\texttt{depth}(u)$ to the right of u). Hence, in this case, u and w are adjacent if and only if $\texttt{node\_rank}_{\texttt{POST}}(v) \leq \texttt{node\_rank}_{\texttt{POST}}(\texttt{last}(u))$ or $\texttt{node\_rank}_{\texttt{POST}}(\texttt{last}(u)) < \texttt{node\_rank}_{\texttt{POST}}(u)$.

In the second case, we assume that $\texttt{node\_rank}_{\texttt{POST}}(u) > \texttt{node\_rank}_{\texttt{POST}}(v)$. Therefore, we have $\texttt{depth}(w) = \texttt{depth}(u)+1$, and $\texttt{node\_rank}_{\texttt{POST}}(w) < \texttt{node\_rank}_{\texttt{POST}}(u)$ (by the properties of a post-order traversal). Again u and w are adjacent if and only if $\texttt{node\_rank}_{\texttt{LEVEL}}(w) \in (\texttt{node\_rank}_{\texttt{LEVEL}}(u), \texttt{node\_rank}_{\texttt{LEVEL}}(\texttt{last}(u))]$. If $\texttt{depth}(\texttt{last}(u)) = \texttt{depth}(u)$ (equivalently, $\texttt{node\_rank}_{\texttt{POST}}(\texttt{last}(u)) > \texttt{node\_rank}_{\texttt{POST}}(u)$), then as $\texttt{depth}(w) = \texttt{depth}(\texttt{last}(u)) + 1$, w comes after $\texttt{last}(u)$ in level-order, so $\texttt{node\_rank}_{\texttt{LEVEL}}(\texttt{last}(u)) < \texttt{node\_rank}_{\texttt{LEVEL}}(w)$. Therefore, w cannot be adjacent to u. If $\texttt{depth}(\texttt{last}(u)) = \texttt{depth}(u) + 1 = \texttt{depth}(w)$ (equivalently, $\texttt{node\_rank}_{\texttt{POST}}(\texttt{last}(u)) < \texttt{node\_rank}_{\texttt{POST}}(u)$), then the restriction of the range $(\texttt{node\_rank}_{\texttt{LEVEL}}(u), \texttt{node\_rank}_{\texttt{LEVEL}}(\texttt{last}(u))]$ to level $\texttt{depth}(w) = \texttt{depth}(\texttt{last}(u))$ are exactly the nodes on level $\texttt{depth}(w)$ in the range $(-\infty, \texttt{node\_rank}_{\texttt{LEVEL}}(\texttt{last}(u))]$. Converted to a post-order traversal, this is the range $(-\infty, \texttt{node\_rank}_{\texttt{POST}}(\texttt{last}(u))]$ for those nodes on level $\texttt{depth}(w)$. w satisfies this exactly when $\texttt{node\_rank}_{\texttt{POST}}(w) \leq \texttt{node\_rank}_{\texttt{POST}}(\texttt{last}(u))$ (so that $\texttt{node\_rank}_{\texttt{POST}}(v) < \texttt{node\_rank}_{\texttt{POST}}(w) \leq \texttt{node\_rank}_{\texttt{POST}}(\texttt{last}(u))$). Hence, in this case, u and w are adjacent if and only if $\texttt{node\_rank}_{\texttt{POST}}(v) < \texttt{node\_rank}_{\texttt{POST}}(\texttt{last}(u)) < \texttt{node\_rank}_{\texttt{POST}}(u)$. ◀

Algorithm 3 Distance computation between vertices u, v by their labels $L(u), L(v)$.

```
1: if depth(v) < depth(u) or (depth(v) = depth(u) and node_rank_POST(v) <
   node_rank_POST(u)) then
2:   switch u and v {Now we have u < v}
3: if node_rank_POST(u) < node_rank_POST(v) then
4:   distance ← depth(v) − depth(u)
5:   if node_rank_POST(v) ≤ node_rank_POST(last(u)) or node_rank_POST(last(u)) <
     node_rank_POST(u) then
6:     distance = distance+1
7:   else
8:     distance = distance+2
9: else
10:  distance ← depth(v) − depth(u) − 1
11:  if node_rank_POST(v) < node_rank_POST(last(u)) < node_rank_POST(u) then
12:    distance = distance+1
13:  else
14:    distance = distance+2
15: return distance
```

To summarize the above, the decoding function f is given by algorithm 3. Note that line 6 and line 12 corresponds to the case in Lemma 5 where u and w are adjacent. Therefore, we add 1 to the distance from v to w to obtain the distance from v to u. In line 8 and 14, u and w are not adjacent, but u and parent(w) are by Lemma 3, and the distance between u and w is 2. It is clear that algorithm 3 runs in $O(1)$ time, using only the labels of u and v.

Regarding preprocessing, we claim that if the intervals' endpoints are given in sorted order, then all the labels can be constructed in $O(n)$ time. Otherwise, we can spend $O(n \log n)$ additional time sorting the endpoints. If the interval graph is given but not an intersection model, we may use a linear time ($O(n + m)$) interval graph recognition algorithm [11] to construct an intersection model of G in sorted order. Details for preprocessing are omitted due to space constraints. Thus we have the main theorem for this section:

▶ **Theorem 6.** *Let G be a connected interval graph. Then there exists a distance labeling scheme occupying at most $3\lceil \lg n \rceil = 3 \lg n + O(1)$ bits per vertex and can compute* distance *in $O(1)$ time. Furthermore, if the intervals are given in sorted order, then the labels can be constructed in $O(n)$ time.*

3.2 General Interval Graphs

Previously, we had assumed that our interval graphs were connected. For general interval graphs, it is possible that for two vertices u and v, their labels would be identical in two graphs, one where u and v belong to the same connected component G_j and one where they belong to different connected components G_j and $G_{j'}$. Thus the labels computed previously is insufficient to compute the distance as it cannot decide if the two vertices belong to the same connected component. As noted, Gavoille and Paul [17] assumed the graph were connected and did not discuss how to generalize it to the disconnected case. To generalize to general interval graphs, it suffices to be able to determine if two vertices are in the same component or not. One way to solve this is to add $\lg n$ bits to the label to state which component the vertex is in, but that would make the labels too costly. Instead, we will use the fact that the label size depends on the size of the component, and the number of components of a given size scales inversely with that size.

Define the ranges $[2^i, 2^{i+1})$ for $i = 0, \dots, \lfloor \lg n \rfloor$. For each component G_j, the size of the component n_j falls into one of these ranges. For range $[2^i, 2^{i+1})$, the number of components G_j falling into this range is at most $n/2^i$. Thus, to identify the components, we need to store i, the range that its size falls into, and c_i, an identifier for which component in that range. For a component of size n_j falling in the range $[2^i, 2^{i+1})$, these identifiers use at most $\lg(\lfloor \lg n \rfloor + 1)$ and $\lg(n/2^i)$ bits respectively. Also as $n_j < 2^{i+1}$, each of the three integers comprising of the labels of the vertices of G_j has size at most $\lceil \lg n_j \rceil \leq i + 1$ bits. Thus in total, for a component G_j of size $n_j \in [2^i, 2^{i+1})$, a label has size at most

$$\lg \lg n + 1 + \lg n - (i + 1) + 3(i + 1) = \lg n + 2i + \lg \lg n + O(1)$$

Since $i \leq \lfloor \lg n \rfloor$, this is at most $3 \lg n + \lg \lg n + O(1)$ bits. Thus our extension to general interval graphs is the following theorem.

▶ **Theorem 7.** *Let G be an interval graph. Then there exists a distance labeling scheme occupying at most $3 \lg n + \lg \lg n + O(1)$ bits per vertex and can compute* `distance` *in $O(1)$ time. Furthermore, if the intervals are given in sorted order, then the labels can be constructed in $O(n)$ time.*

Using this, we have an immediate application to circular arc graphs. Following the framework of Gavoille and Paul [17], we unroll the circular arc graph twice. That is, we start from the angle $\theta = 0$ and sweep the circle twice, writing down the endpoints of the arcs. In this fashion, each arc is recorded twice, once as its original $[\theta_1, \theta_2]$, and once on the second unroll, $[\theta_1 + 2\pi, \theta_2 + 2\pi]$ [1]. After unrolling, we obtain an interval graph, where each vertex of the original circular arc graph corresponds to two vertices in the unrolled interval graph. The distance can then be calculated using the following lemma:

▶ **Lemma 8** (Lemma 3.5 [17]). *For an circular arc graph G, unrolled twice into an interval graph $\tilde{G}$. For every $i < j$, $\texttt{distance}_G(x_i, x_j) = \min\{\texttt{distance}_{\tilde{G}}(x_i^1, x_j^1), \texttt{distance}_{\tilde{G}}(x_j^1, x_i^2)\}$, where x_i are sorted by their left endpoints in increasing i and x^1, x^2 are the two copies of the arc x in the interval graph.*

Thus it suffices to store two (interval) vertex labels for each vertex of the circular-arc graph, and so the length of the label is $6 \lg n + 2 \lg \lg n + O(1)$ bits.

▶ **Corollary 9.** *Let G be a circular arc graph. Then there exists a distance labeling scheme occupying at most $6 \lg n + 2 \lg \lg n + O(1)$ bits per vertex and can compute* `distance` *in $O(1)$ time. Furthermore, if G is connected, then $6 \lg n + O(1)$ bit labels suffices. If the arcs are given in sorted order, then the labels can be constructed in $O(n)$ time.*

4 Chordal Graph

4.1 Background

4.1.1 Chordal Graph Structure

One of the many equivalent definitions of chordal graph is that a chordal graph is the intersection graph of subtrees (i.e. connected sets of nodes) in a tree [20]. Thus, for a chordal graph G, there exists a tree T and a set of subtrees $\{T_v; v \in V\}$ of T such that two vertices

[1] Some more work need to be done for arcs which contain our origin angle.

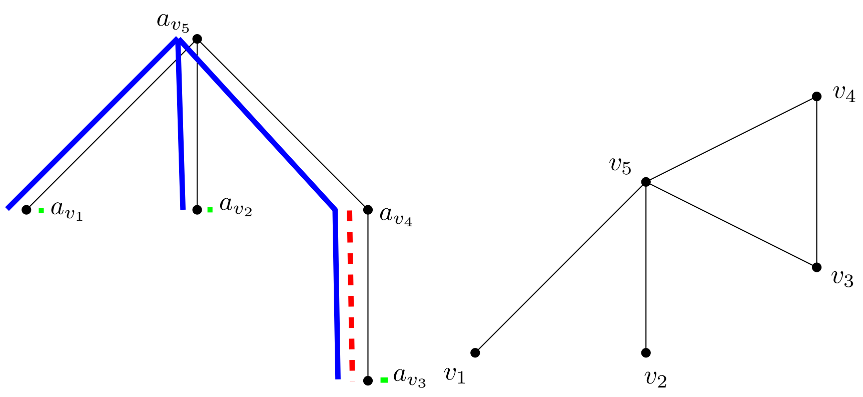


Figure 4 Left: The underlying tree for a chordal graph is depicted in black. Each of the leaves has a subtree containing only that leaf (the green dots), and these correspond to vertices v_1, v_2, v_3. T_{v_4} is depicted with a red dashed segment and T_{v_5} in blue bold segments. The apex of each path is labeled. Note that every subtree has a distinct apex. Right: The chordal graph generated by this set of subtrees.

u and v are adjacent if and only if the subtrees T_u and T_v intersect (at some node). If we further root the tree, then each subtree T_v corresponding to a vertex v has a unique smallest depth node a_v, which we will denote as the *apex* of the subtree (and of the vertex). To create a data structure, we would like to make some modifications to the tree T with additional exploitable properties.

Munro and Wu [27] showed that we may choose T such that the number of nodes is exactly n. Furthermore, this rooted tree T and the subtrees T_v corresponding to vertices has the property that for two vertices u and v, their apexes are distinct: $a_u \neq a_v$. See Figure 4 for an example. In this way, we have a bijection between vertices of the graph G and nodes of tree T. Thus it make sense to talk about the vertex v which corresponds to a node of T, and we will name the nodes of T as a_v for the vertex v whose apex is that node.

To characterize adjacency, we look at the relationship between the apexes of the two vertices u and v. If the subtrees rooted at a_u and at a_v are disjoint, then the subtrees T_u and T_v do not intersect and u and v are not adjacent. Otherwise, one of a_u and a_v is an ancestor of the other, say a_u is an ancestor of a_v. Then u and v are adjacent exactly when the subtree T_u corresponding to u contains the node a_v. Thus we may define a set of vertices S_v at each node a_v of T which is the set of ancestors of a_v, whose subtrees contain a_v (i.e. the set of ancestors which are adjacent to v).

Finally, we observe that in the degenerate case that T is a path, then all subtrees T_v become paths. By viewing T as a subset of the real number line, we see that all subtrees T_v are intervals, and thus the graph generated is an interval graph.

4.1.2 Chordal Graph Distance Algorithm

Munro and Wu [27] gave an algorithm computing the distance between two vertices u and v. In the case that a_u is an ancestor of a_v (or vice versa), the problem reduces to that of an interval graph by restricting the tree T to the path from a_v to the root. Otherwise, if a_u and a_v belong to different subtrees, we first compute the lowest common ancestor a_h of a_u and a_v. We then compute the distance from h to u and from h to v (with a caveat). Since a_h is an ancestor of a_u and a_v, we reduce to the interval graph case. Analogous to the distance tree in interval graphs, Munro and Wu construct a distance tree for chordal graphs.

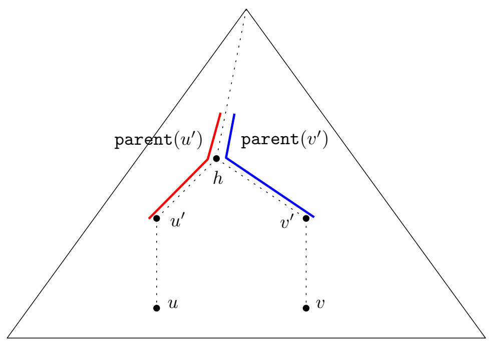


Figure 5 Left bolded (red) segment represents part of the subtree corresponding to $\texttt{parent}(u')$, which must contain the node $a_{u'}$, and the right (blue) segment for $\texttt{parent}(v')$. Here the apex $a_{u'}$ of the subtree corresponding to a vertex u' is labelled as u'. It can be seen that both $\texttt{parent}(u')$ and $\texttt{parent}(v')$ pass through a_h so they are adjacent.

We note that for a vertex v, when we restrict the tree T to the path from a_v to the root, the set of vertices S_v corresponds exactly to those vertices p such that $l_p < l_v$ and $r_p \geq l_v$ (when viewing this path as a subset of the real number line, with the coordinates being the depth of the node). Therefore, the formula $(\arg\min\{l_u \mid r_u \geq l_v\})$ used to define the distance tree in interval graphs can also be applied to chordal graphs. The parent $\texttt{parent}(v)$ of a vertex v in the distance tree T_D of a chordal graph is the vertex $\texttt{parent}(v) \in S_v$ with the smallest depth apex $a_{\texttt{parent}(v)}$. This is well-defined since the apexes are distinct.

Since this notion of parent matches the parent of an interval graph generated from T by restricting to any path towards the root, the distance tree T_D can be seen as the union of all distance trees of interval graphs generated by taking paths from leaves of T to the root. However, due to the multiple interval graph distance trees being unioned, we do not have the nice ordering property in Lemma 1.

When applying the interval graph distance algorithm between u and h (and between v and h), we reach their representatives u' and v', and compute $\texttt{distance}(u, h) = \texttt{distance}(u, u') + \texttt{distance}(u', h)$, where $\texttt{distance}(u', h) = 1$ if u' and h are adjacent, and $\texttt{distance}(u', h) = 2$ if not (a shortest path being u', $\texttt{parent}(u')$, h, similarly for v'). Finally these two paths must be joined. Naively, the path could be the shortest path from u to h followed by the shortest from h to v. So depending on whether u' and h, and v' and h are adjacent, the path from u' to v' has length either 2, 3, or 4. However, it is shown that $\texttt{parent}(u')$ and $\texttt{parent}(v')$ are adjacent (see Figure 5), so that in the case that both u' and v' are not adjacent to h, a shortest path is $u', \texttt{parent}(u'), \texttt{parent}(v'), v'$ which has length 3, rather than $u', \texttt{parent}(u'), h, \texttt{parent}(v'), v'$ with length 4. Thus the distance between u' and v' is either 2 or 3. Determining this distance between u' and v' is the bottleneck for the distance computation as it is shown that $\texttt{distance}(u, v) = \texttt{distance}(u, u') + \texttt{distance}(u', v') + \texttt{distance}(v', v)$.

Munro and Wu show that the distance between u' and v' is 2 exactly when there exists some vertex h' such that h' is adjacent to both u' and v' (i.e. the subtree $T_{h'}$ corresponding to h' contains both nodes $a_{u'}$ and $a_{v'}$). See Figure 6. If u' and v' are both adjacent to h, then the vertex h takes this role. However, even if u' and v' are not adjacent to h, such an h' can exist (which is independent of the adjacencies between u' and v' with h). Such a vertex h' would exist in both the sets $S_{u'}$ and $S_{v'}$, and so determining whether h' exists is equivalent to determining whether the intersection $S_{u'} \cap S_{v'} = \emptyset$.

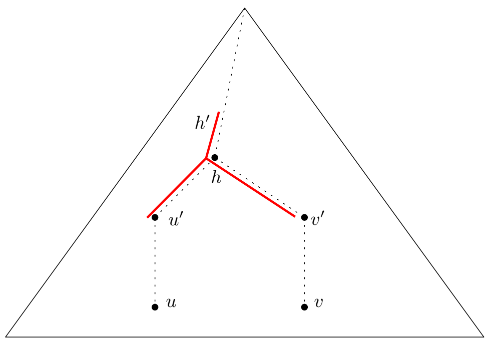


Figure 6 Part of the subtree corresponding to the vertex h' is depicted. The vertex h' is adjacent to both u' and v' so that the distance between them is 2.

This problem of preprocessing sets to answer queries of the form: determine whether the intersection of two of the given sets is empty, is the *set intersection oracle* problem. Thus the chordal graph distance problem can be reduced to it. Munro and Wu further argued that the set intersection oracle problem can also be reduced to the chordal graph distance problem so that the two are equivalent (up to a constant factor of the input sizes). The set intersection oracle problem is conjectured to be difficult (Conjecture 3 and some follow up discussions of Pătraşcu and Roditty[28] state that, to solve it using $O(1)$ time we must use $\Omega(n^2)$ space, even if the sets are small), so computing distances in chordal graphs quickly would also be difficult. Munro and Wu's [27] algorithm described above can be summarized in the following lemma.

▶ **Lemma 10.** *Let G be a chordal graph, T be the intersection model as described by Munro and Wu [27] with exactly n nodes, and T_D be the distance tree. Let u, v be two vertices, and let a_u, a_v be their apexes. We consider the general case where a_u and a_v do not have any ancestry relationship. Let h be the vertex such that $a_h = \texttt{LCA}(a_u, a_v)$. Let u' be the* representative *of u with respect to h, similarly for v'. Then the distance between u and v is $\texttt{distance}(u, v) = \texttt{distance}(u, u') + \texttt{distance}(v, v') + 2 + \mathbf{1}(C)$, where C is this condition: there does not exist any vertex h' such that $T_{h'}$ contains both the nodes $a_{u'}$ and $a_{v'}$.*

4.2 Labeling Scheme

Using the above `distance` algorithm, our labeling scheme must be able to check/compute the following steps.

- Given two vertices u and v, determine whether one is an ancestor of the other, and if not, locate $h = \texttt{LCA}(T, u, v)$. In the positive case, it reduces to an interval graph distance query.
- Locate u' and v', the representatives of u and v with respect to h in T_D.
- Compute $\texttt{distance}(u, u')$ and $\texttt{distance}(v, v')$.
- Decide the value of C for the exact distance.

Our labeling scheme will thus consist of the following:

- $\texttt{depth}(T_D, v), \texttt{node\_rank}_{\texttt{POST}}(T_D, v)$.
- A labeling scheme for `LCA` in T which can return $\texttt{depth}(T_D, h)$ and $\texttt{node\_rank}_{\texttt{POST}}(T_D, h)$ of the lowest common ancestor h of two vertices u and v.
- A bitvector of length $n/2$. Its content will be defined later.

The lowest common ancestor scheme we will use is the result of Alstrup et al. [7] which computes arbitrary labels of the lowest common ancestor of two nodes in $O(1)$ time.

▶ **Lemma 11** (Corollary 4.1 of [7]). *There exists an* `LCA`*-labeling scheme for Trees with predefined labels of fixed length k whose worst-case label size is at most $(3+k)\lfloor \log n \rfloor + 1$ bits, with $O(1)$ decode time.*

As our predefined labels that we must return are of size $2\lg n + O(1)$ bits, this `LCA`-labeling scheme uses uses $2\lg^2 n + O(\lg n)$ bits per label.

To perform step 1, we use the `LCA`-labeling scheme, and check if the returned node is u or v, by checking the label $\texttt{node\_rank}_{\texttt{POST}}(T_D, h)$ of the returned node against $\texttt{node\_rank}_{\texttt{POST}}(T_D, u)$ and $\texttt{node\_rank}_{\texttt{POST}}(T_D, v)$. If one is an ancestor of the other, we revert to the interval graph labeling scheme. However, since we do not have $\texttt{node\_rank}_{\texttt{POST}}(\texttt{last}(v))$, we cannot decide whether to add 1 or 2 in Algorithm 3 (lines 5-8, 11-14). To decide this we will store which term (+1 or +2) to add in the bitvector for deciding C as described below. Otherwise, we obtain the distance tree T_D labels of u, v and h. We again note that v' is the representative of v with respect to h is the node w in algorithm 2. Thus $\texttt{distance}(v, v')$ is simply $\texttt{depth}(v) - \texttt{depth}(v')$, and we do not need the value $\texttt{node\_rank}_{\texttt{POST}}(\texttt{last}(v))$ in the interval graph labeling scheme.

Finally, we need to decide the condition C. As discussed earlier, the condition is equivalent to the set intersection oracle problem. It is conjectured that to compute it in $O(1)$ time, it is necessary to store the result of the queries explicitly, rather than trying to compute it from the sets. Therefore, we will pre-compute the value of C for every pair of vertices. For a pair of vertices u and v where one is an ancestor of the other, we do not need to decide the value of C for this pair, as the computation of the distance between this pair of vertices reduces to the interval graph distance case. In this case, we must determine whether to add 1 or to add 2 in Algorithm 3. We use the bitvector C to store whether to add 1 (the bit 0) or to add 2 (the bit 1). Thus for every vertex v, for each other vertex u, either u is an ancestor or descendant of v, so we store in C a bit for the interval graph distance computation, or u is not an ancestor or descendant of v in which case we store a bit for the chordal graph distance computation of u and v. For a vertex u, the index into the bitvector is $\texttt{node\_rank}_{\texttt{POST}}(T_D, u)$. We may distribute this bitvector more evenly by storing the value for only those vertices u such that $(\texttt{node\_rank}_{\texttt{POST}}(T_D, u) - \texttt{node\_rank}_{\texttt{POST}}(T_D, v)) \mod n \leq n/2$, so that we store $n/2$ bits per vertex in the worst case. For any pair of vertices u and v, this bit for the distance computation is stored in the bitvector of one of the vertices.

The preprocessing time is dominated by the precomputation of the condition C for all pairs of vertices. We may compute all the results of the set intersection oracle problem via matrix multiplication. Details are omitted due to space constraints. Thus, we obtain the following theorem:

▶ **Theorem 12.** *Let G be a chordal graph. Then there exists a distance labeling scheme with maximum label size $n/2 + O(\lg^2 n)$ bits which can compute* `distance` *in $O(1)$ time. The labels can be constructed in $O(n^\omega)$ time where $\omega < 2.371552$ is the matrix multiplication exponent [32].*

References

1 Ittai Abraham and Cyril Gavoille. On approximate distance labels and routing schemes with affine stretch. In David Peleg, editor, *Distributed Computing - 25th International Symposium, DISC 2011, Rome, Italy, September 20-22, 2011. Proceedings*, volume 6950 of *Lecture Notes in Computer Science*, pages 404–415. Springer, 2011. `doi:10.1007/978-3-642-24100-0_39`.

2 Hüseyin Acan, Sankardeep Chakraborty, Seungbum Jo, and Srinivasa Rao Satti. Succinct data structures for families of interval graphs. In *Algorithms and Data Structures - 16th International Symposium, WADS 2019, Edmonton, AB, Canada, August 5-7, 2019, Proceedings*, pages 1–13, 2019. doi:10.1007/978-3-030-24766-9_1.

3 Stephen Alstrup, Philip Bille, and Theis Rauhe. Labeling schemes for small distances in trees. *SIAM Journal on Discrete Mathematics*, 19(2):448–462, 2005. doi:10.1137/S0895480103433409.

4 Stephen Alstrup, Søren Dahlgaard, and Mathias Bæk Tejs Knudsen. Optimal induced universal graphs and adjacency labeling for trees. In *2015 IEEE 56th Annual Symposium on Foundations of Computer Science*, pages 1311–1326, 2015. doi:10.1109/FOCS.2015.84.

5 Stephen Alstrup, Cyril Gavoille, Esben Bistrup Halvorsen, and Holger Petersen. Simpler, faster and shorter labels for distances in graphs. In *Proceedings of the Twenty-Seventh Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '16, pages 338–350, USA, 2016. Society for Industrial and Applied Mathematics.

6 Stephen Alstrup, Inge Li Gørtz, Esben Bistrup Halvorsen, and Ely Porat. Distance Labeling Schemes for Trees. In Ioannis Chatzigiannakis, Michael Mitzenmacher, Yuval Rabani, and Davide Sangiorgi, editors, *43rd International Colloquium on Automata, Languages, and Programming (ICALP 2016)*, volume 55 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 132:1–132:16, Dagstuhl, Germany, 2016. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.ICALP.2016.132.

7 Stephen Alstrup, Esben Bistrup Halvorsen, and Kasper Green Larsen. Near-optimal labeling schemes for nearest common ancestors. In *Proceedings of the 2014 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 972–982, 2014. doi:10.1137/1.9781611973402.72.

8 Stephen Alstrup, Haim Kaplan, Mikkel Thorup, and Uri Zwick. Adjacency labeling schemes and induced-universal graphs. *SIAM Journal on Discrete Mathematics*, 33(1):116–137, 2019. doi:10.1137/16M1105967.

9 Amotz Bar-Noy, Reuven Bar-Yehuda, Ari Freund, Joseph Naor, and Baruch Schieber. A unified approach to approximating resource allocation and scheduling. In F. Frances Yao and Eugene M. Luks, editors, *Proceedings of the Thirty-Second Annual ACM Symposium on Theory of Computing, May 21-23, 2000, Portland, OR, USA*, pages 735–744. ACM, 2000. doi:10.1145/335305.335410.

10 Marthe Bonamy, Louis Esperet, Carla Groenland, and Alex Scott. Optimal labelling schemes for adjacency, comparability, and reachability. In *Proceedings of the 53rd Annual ACM SIGACT Symposium on Theory of Computing*, STOC 2021, pages 1109–1117, New York, NY, USA, 2021. Association for Computing Machinery. doi:10.1145/3406325.3451102.

11 Kellogg S. Booth and George S. Lueker. Testing for the consecutive ones property, interval graphs, and graph planarity using pq-tree algorithms. *Journal of Computer and System Sciences*, 13(3):335–379, 1976. doi:10.1016/S0022-0000(76)80045-1.

12 Danny Z. Chen, D. T. Lee, R. Sridhar, and Chandra N. Sekharan. Solving the all-pair shortest path query problem on interval and circular-arc graphs. *Networks*, 31(4):249–258, 1998. URL: https://onlinelibrary.wiley.com/doi/abs/10.1002/%28SICI%291097-0037%28199807%2931%3A4%3C249%3A%3AAID-NET5%3E3.0.CO%3B2-D.

13 Lenore J Cowen. Compact routing with minimum stretch. *Journal of Algorithms*, 38(1):170–183, 2001. doi:10.1006/jagm.2000.1134.

14 Tamar Eilam, Cyril Gavoille, and David Peleg. Compact routing schemes with low stretch factor. *Journal of Algorithms*, 46(2):97–114, 2003. doi:10.1016/S0196-6774(03)00002-6.

15 Pierre Fraigniaud and Amos Korman. An optimal ancestry labeling scheme with applications to XML trees and universal posets. *J. ACM*, 63(1), February 2016. doi:10.1145/2794076.

16 Ofer Freedman, Paweł Gawrychowski, Patrick K. Nicholson, and Oren Weimann. Optimal distance labeling schemes for trees. In *Proceedings of the ACM Symposium on Principles of Distributed Computing*, PODC '17, pages 185–194, New York, NY, USA, 2017. Association for Computing Machinery. doi:10.1145/3087801.3087804.

17 Cyril Gavoille and Christophe Paul. Optimal distance labeling for interval graphs and related graph families. *SIAM J. Discret. Math.*, 22(3):1239–1258, July 2008. doi:10.1137/050635006.

18 Cyril Gavoille and David Peleg. Compact and localized distributed data structures. *Distrib. Comput.*, 16(2–3):111–120, September 2003. doi:10.1007/s00446-002-0073-5.

19 Cyril Gavoille, David Peleg, Stéphane Pérennes, and Ran Raz. Distance labeling in graphs. *Journal of Algorithms*, 53(1):85–112, 2004. doi:10.1016/j.jalgor.2004.05.002.

20 Fănică Gavril. The intersection graphs of subtrees in trees are exactly the chordal graphs. *Journal of Combinatorial Theory, Series B*, 16(1):47–56, 1974. doi:10.1016/0095-8956(74)90094-X.

21 Paweł Gawrychowski, Fabian Kuhn, Jakub Łopuszański, Konstantinos Panagiotou, and Pascal Su. Labeling schemes for nearest common ancestors through minor-universal trees. In *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '18, pages 2604–2619, USA, 2018. Society for Industrial and Applied Mathematics.

22 Pawel Gawrychowski and Przemyslaw Uznanski. Better distance labeling for unweighted planar graphs. *Algorithmica*, 85(6):1805–1823, 2023. doi:10.1007/S00453-023-01133-Z.

23 Meng He, J. Ian Munro, Yakov Nekrich, Sebastian Wild, and Kaiyu Wu. Distance oracles for interval graphs via breadth-first rank/select in succinct trees. In Yixin Cao, Siu-Wing Cheng, and Minming Li, editors, *31st International Symposium on Algorithms and Computation, ISAAC 2020, December 14-18, 2020, Hong Kong, China (Virtual Conference)*, volume 181 of *LIPIcs*, pages 25:1–25:18. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPIcs.ISAAC.2020.25.

24 Michal Katz, Nir A. Katz, and David Peleg. Distance labeling schemes for well-separated graph classes. *Discrete Applied Mathematics*, 145(3):384–402, 2005. doi:10.1016/j.dam.2004.03.005.

25 Hung Le and Christian Wulff-Nilsen. Optimal approximate distance oracle for planar graphs. In *62nd IEEE Annual Symposium on Foundations of Computer Science, FOCS 2021, Denver, CO, USA, February 7-10, 2022*, pages 363–374. IEEE, 2021. doi:10.1109/FOCS52979.2021.00044.

26 Yaowei Long and Seth Pettie. Planar distance oracles with better time-space tradeoffs. In Dániel Marx, editor, *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms, SODA 2021, Virtual Conference, January 10 - 13, 2021*, pages 2517–2537. SIAM, 2021. doi:10.1137/1.9781611976465.149.

27 J. Ian Munro and Kaiyu Wu. Succinct data structures for chordal graphs. In *29th International Symposium on Algorithms and Computation, ISAAC 2018, December 16-19, 2018, Jiaoxi, Yilan, Taiwan*, pages 67:1–67:12, 2018. doi:10.4230/LIPIcs.ISAAC.2018.67.

28 Mihai Patrascu and Liam Roditty. Distance oracles beyond the thorup-zwick bound. *SIAM J. Comput.*, 43(1):300–311, 2014. doi:10.1137/11084128X.

29 David Peleg. Informative labeling schemes for graphs. *Theoretical Computer Science*, 340(3):577–593, 2005. Mathematical Foundations of Computer Science 2000. doi:10.1016/j.tcs.2005.03.015.

30 Fernando Magno Quintão Pereira and Jens Palsberg. Register allocation via coloring of chordal graphs. In Kwangkeun Yi, editor, *Programming Languages and Systems*, pages 315–329, Berlin, Heidelberg, 2005. Springer Berlin Heidelberg.

31 Gaurav Singh, N. S. Narayanaswamy, and G. Ramakrishna. Approximate distance oracle in o(n 2) time and o(n) space for chordal graphs. In M. Sohel Rahman and Etsuji Tomita, editors, *WALCOM: Algorithms and Computation - 9th International Workshop, WALCOM 2015, Dhaka, Bangladesh, February 26-28, 2015. Proceedings*, volume 8973 of *Lecture Notes in Computer Science*, pages 89–100. Springer, 2015. doi:10.1007/978-3-319-15612-5_9.

32 Virginia Vassilevska Williams, Yinzhan Xu, Zixuan Xu, and Renfei Zhou. New bounds for matrix multiplication: from alpha to omega. In David P. Woodruff, editor, *Proceedings of the 2024 ACM-SIAM Symposium on Discrete Algorithms, SODA 2024, Alexandria, VA, USA, January 7-10, 2024*, pages 3792–3835. SIAM, 2024. doi:10.1137/1.9781611977912.134.

33 Nicholas C. Wormald. Counting labelled chordal graphs. *Graphs and Combinatorics*, 1(1):193–200, 1985. doi:10.1007/BF02582944.

34 Peisen Zhang, Eric A. Schon, Stuart G. Fischer, Eftihia Cayanis, Janie Weiss, Susan Kistler, and Philip E. Bourne. An algorithm based on graph theory for the assembly of contigs in physical mapping of DNA. *Computer Applications in the Biosciences*, 10(3):309–317, 1994. doi:10.1093/bioinformatics/10.3.309.