

An Optimal Algorithm to Solve the All-Pair Shortest Path Problem on Interval Graphs

R. Ravi,* Madhav V. Marathe,† and C. Pandu Rangan

Department of Computer Science, I.I.T.
Madras 600036, India

We present an $O(n^2)$ time-optimal algorithm for solving the unweighted all-pair shortest path problem on interval graphs, an important subclass of perfect graphs. An interesting structure called the *neighborhood tree* is studied and used in the algorithm. This tree is formed by identifying the successive neighborhoods of the vertex labeled last in the graph according to the IG-ordering.

1. INTRODUCTION

Interval graphs are an important subclass of perfect graphs, and areas as diverse as archeology, molecular biology, genetics, and traffic planning use this class of graphs to model problems frequently. Interval graphs are discussed extensively in [4].

An interval graph is a graph whose vertices can be mapped to unique intervals on the real line such that two vertices in the graph are adjacent iff their corresponding intervals intersect. Given a *connected* interval graph G , we wish to find the shortest path between any pair of its vertices. We discuss here only a method to compute and store the length of such paths. Also, observe that this is the best that can be done in $O(n^2)$ time, where n is the number of vertices in G , for there are interval graphs for which an *enumeration* of all the shortest paths can require $\Theta(n^3)$ time. However, an easy extension of our algorithm can be tailored to output the shortest paths between any pair of vertices in G .

For any general graph with n vertices, the well-known solution to the all-pair shortest path problem in [1] takes $O(n^3)$ time. A lot of work has been done in improving the running time using randomization and probabilistic methods and also in the case of special kinds of graphs. For instance, see [6] for a recent

*Current Address: Box 1910, Department of Computer Science, Brown University, Providence, RI 02912.

†Current address: Dept. of Computer Science, SUNY, Albany, NY 12222.

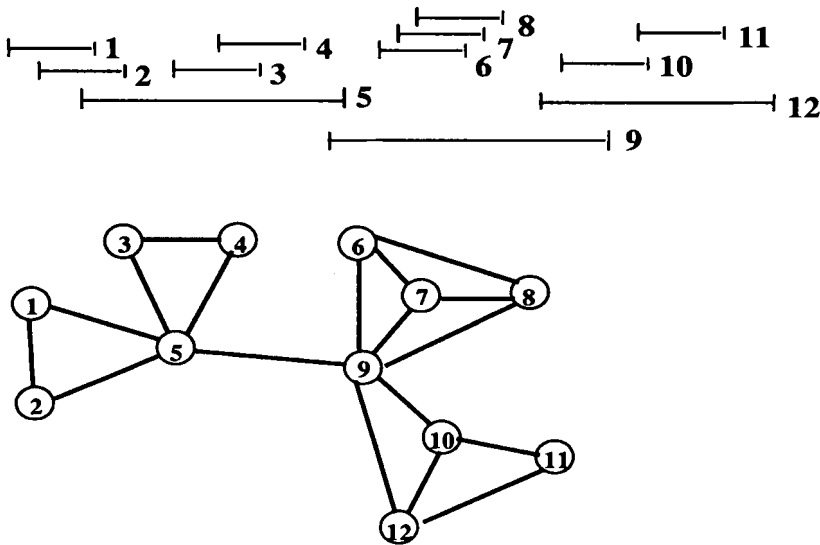


FIG. 1. An interval graph and its interval representation.

algorithm for the all-pair shortest path problem on any graph with $O(n^2 \log n)$ average case time complexity.

The algorithm presented below solves the all-pair shortest path problem on interval graphs where all edges are of unit distance in $O(n^2)$ time. This matches the obvious lower bound to solve this problem, and, hence, the algorithm presented is time-optimal. The algorithm uses the concept of a neighborhood tree formed by identifying the successive neighborhoods of the vertex labeled last in the graph according to the IG-ordering of the vertices. The IG-ordering of the vertices of an interval graph was introduced in [7] and has been useful in solving a variety of problems on interval graphs efficiently [2, 5, 8].

2. DEFINITIONS

Let $G = (V, E)$, $|V| = n$, $|E| = m$ be an interval graph. We assume that $V = \{1, \dots, n\}$ and that the vertices of G are labeled according to the IG ordering [7]. This is simply the order in which we consider the intervals in the intersection model of G in the nondecreasing order of their right endpoints. Figure 1 shows an interval graph with its interval representation and its vertices labeled in the IG ordering. An important property of this ordering is given below.

Property 2.1. *If vertices, $x, y, z \in V$ are such that $x < y < z$ in the ordering, if x is adjacent to z , then y is also adjacent to z [7].*

If vertices x and y are adjacent, then we write $x \sim y$ or $y \sim x$. For any vertex v in V , we define

$$LO(v) = \begin{cases} w & \text{such that } w < v \text{ and } w \text{ is the smallest vertex adjacent to } v \\ v & \text{otherwise, i.e., if no neighbor of } v \text{ has smaller IG numbering than } v. \end{cases}$$

Similarly, we define

$$HI(v) = \begin{cases} w & \text{such that } w > v \text{ and } w \text{ is the highest numbered vertex adjacent to } v \\ v & \text{otherwise, i.e., if no neighbor of } v \text{ has a higher IG numbering than } v. \end{cases}$$

$LO(v)$ for each vertex v can be computed in linear time in a single scan of the vertices of G in the IG-ordering [7]. A variation of this method can be used to compute $HI(v)$ for each v in linear time. Let $c(H)$ denote the number of connected components of the graph H . The distance between a pair of vertices u and v , denoted as $d(u, v)$, is the length of the shortest path between them in G . The i^{th} neighborhood in G of a vertex v is denoted as $N_i(v)$ and is defined as follows:

$$N_i(v) = \{u : d(v, u) = i\}.$$

Let k be the length of the longest chordless path from vertex n to any other vertex in G . It is easy to see that $N_k(n)$ is nonempty while $N_{k+1}(n)$ is empty. In this paper, we shall be interested only in the neighborhoods of the vertex numbered n in the IG-ordering and shall, henceforth, refer to $N_i(n)$ as N_i without ambiguity.

3. THE NEIGHBORHOOD TREE OF AN INTERVAL GRAPH

Define G_i^j to be the subgraph of G induced on the set of vertices N_i and let $c(G_i^j) = p_i$. Let CC_i denote the set of the p_i connected components of G_i^j . We define CC_0 to be the singleton set $\{n\}$. We now define an auxiliary graph $NT(G)$ built from the graph G as follows:

$$NT(G) = (V', E'), \text{ where}$$

$$V' = \{\cup_{i \geq 0}^k CC_i\} \text{ and}$$

$$E' = \{(X, Y) : X \in CC_i, Y \in CC_j, j \neq i, \text{ and there exist } x \in X \text{ and } y \in Y \text{ such that } x \sim y\}.$$

Since we have two different graphs under consideration, G and $NT(G)$, we shall refer to a vertex of G simply as a vertex and a vertex of $NT(G)$ as a node to

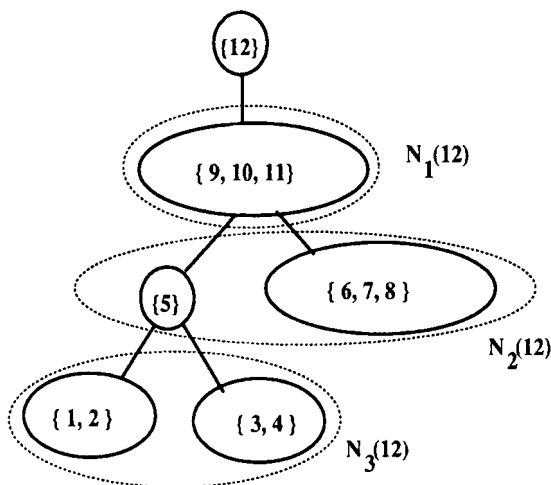


FIG. 2. The neighborhood tree $NT(G)$ for the interval graph in Figure 1. In this figure, $p(1) = 1$ and $p(2) = p(3) = 2$.

avoid confusion. The auxiliary graph $NT(G)$ of the interval graph in Figure 1 is shown in Figure 2. Here, $n = 12$, $p_0 = p_1 = 1$, $p_2 = p_3 = 2$; the nodes of $NT(G)$ are $C_1 = \{9, 10, 11\}$, $C_2 = \{5\}$, $C_3 = \{6, 7, 8\}$, $C_4 = \{1, 2\}$, and $C_5 = \{3, 4\}$. As noted above, two nodes of $NT(G)$ are joined by an edge if there exists an edge between a vertex in one node to a vertex in another. Thus, for instance, the node C_3 is joined to the node C_1 since a vertex, say 7, in the former is connected to a vertex, 9, in the latter.

Lemma 3.1. *The auxiliary graph $NT(G)$ built from a connected interval graph G is a tree.*

Proof. We can see directly that $NT(G)$ is connected iff G is. We now show that $NT(G)$ constructed from a connected interval graph is indeed acyclic. We shall define a node C to be at a distance i from the node denoting vertex n if $d(x, n) = i$ for any $x \in C$. Note that the notion of distance given above is well defined since C is the connected component of the subgraph induced on the i^{th} neighborhood of the vertex n .

To prove that $NT(G)$ is acyclic, first observe that no edges can exist in $NT(G)$ between nodes that are at the same distance from $\{n\}$. Also, there are no edges in $NT(G)$ connecting nodes C_i and C_j whose distances from n differ by more than one. This is by virtue of the definition of $NT(G)$. Thus, to show that $NT(G)$ is a tree, we show that each node C_x at a distance i from $\{n\}$ is adjacent to exactly one node that is at a distance $(i - 1)$ from $\{n\}$.

We prove this by contradiction. Assume that some node C_x of $NT(G)$ at distance i from $\{n\}$ is adjacent to two nodes C_p and C_q of $NT(G)$ at a distance $(i - 1)$ from $\{n\}$ (see Fig. 3). We can identify vertices u_1, u_2, u_3 , and u_4 as in

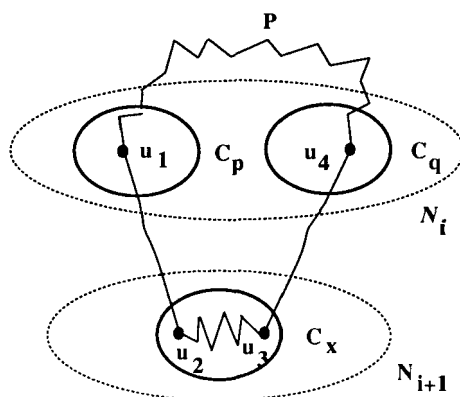


FIG. 3. The contradiction that the graph is an interval graph in the proof of acyclicity of $NT(G)$ for an interval graph G : Lemma 3.1. A chordless cycle of length at least four is identified above whenever a node C_x at distance $i + 1$ is adjacent to two distinct nodes C_p and C_q in level i .

Figure 3 and thus an induced chordless cycle P of length at least 4 in the graph G . This contradicts our assumption that G is an interval graph. Hence, every node C_x at a distance i from $\{n\}$ is adjacent exactly to a single node at a distance $(i - 1)$ from $\{n\}$. Thus, we see that $NT(G)$ is a tree whenever G is a connected interval graph. ■

We shall refer to $NT(G)$ as the *neighborhood tree* of G . Also, we shall consider this tree to be rooted at $\{n\}$. We remark here that $NT(G)$ is a tree even if G is a chordal graph with its vertices labeled according to a perfect elimination ordering. This is because the proof of Lemma 3.1 uses only the triangulated property of interval graphs that chordal graphs also have. For each node C of $NT(G)$, define $Level(C)$ to be the distance of C from $\{n\}$ in $NT(G)$ and for any $v \in V$, let $Level(v) = Level(C)$ such that $v \in C$.

Note that in Figure 2 the vertices in level 1 are 9, 10, and 11 and, among these, the vertex with the minimum value of LO is 9 with $LO(9) = 5$. Also, note that level 2 is made of 5, 6, 7, and 8. More formally, if level i is made up of the vertex set $\{x_1, \dots, x_r\}$ in ascending order and x_g is the vertex in this set with the smallest value of LO , then we see that the vertices in level $(i + 1)$ of $NT(G)$ are exactly those in the range $\{LO(x_g), LO(x_g) + 1, \dots, x_1 - 1\}$. This observation is proved in the lemma below, but before that, we need the following notation. Recall that

$$\begin{aligned} N_i &= \{\cup_{C \in \mathcal{C}_i} V(C)\} \\ &= \{x_1, \dots, x_r\}, \text{ where } x_1, \dots, x_r \text{ is sorted in ascending order.} \end{aligned}$$

For ease of reference and to avoid ambiguity, let us denote this sorted set representation of N_i by $L(i)$.

Lemma 3.2. *If $L(i) = \{x_1, \dots, x_r\}$ and $MINLO_i = \min \{LO(x_j) : x_j \in L(i)\}$, then*

- (1) x_1, \dots, x_r are consecutive integers, and
- (2) $L(i + 1) = \{MINLO_i, MINLO_i + 1, \dots, x_1 - 1\}$.

Proof. We prove the lemma by induction on i . Observe that $L(0) = \{n\}$ and $L(1) = \{LO(n), LO(n) + 1, \dots, n - 1\}$ and the basis is true.

Assume that $L(i) = \{x_1, \dots, x_r\}$, where x_1, \dots, x_r are consecutive integers. Now, by definition, $L(i + 1)$ consists of the neighbors of all the vertices in $L(i)$ and $MINLO_i$ is the least numbered vertex in G to which any vertex in $L(i)$ is connected. Property 2.1 of the IG-ordering leads us to conclude that $MINLO_i, MINLO_{i+1}, \dots, x_1 - 1$ are all vertices in $L(i + 1)$. Hence, the theorem is true by induction. ■

From the lemma above, we see that if $x > y$ in the IG-ordering, i.e., if x lies to the right of y in the ordering, then $Level(x) \leq Level(y)$. Also, if $x > y > z$ and $Level(x) = Level(z) = t$, then $Level(y) = t$. The next lemma establishes one of the most important properties of the neighborhood tree of an interval graph.

Lemma 3.3. *Let G be a connected interval graph. All nodes of $NT(G)$ that are in level i are adjacent to a unique node of $NT(G)$ at level $(i - 1)$.*

Proof. To see this, consider level i to be made up of nodes C_1, \dots, C_u with vertices $L(i) = \{x_1, \dots, x_r\}$. Let the node C_p contain the vertex x_1 . Note that $x_1 \sim y$ for some y in level $(i - 1)$ of $NT(G)$. Also, $y > x_1$ from the observation following Lemma 3.2 above. Let C_r be the node at level $(i - 1)$ containing y . It is clear that C_r and C_p are adjacent in $NT(G)$. We will show that C_i is adjacent to C_r for all i such that $1 \leq i \leq u$. This follows from the observation that each such node C_i contains some x_i such that $x_1 < x_i < y$. The IG-ordering Property 2.1 leads us to conclude that each C_i must be connected to C_r . Finally, from the proof of Lemma 3.1, we know that each node in level i is connected to *exactly* one node at level $(i - 1)$. Thus, all nodes in level i are connected to a unique node in level $(i - 1)$. ■

We define a node in level i of $NT(G)$ to be a *special* node if every node in level $(i + 1)$ is adjacent to it. From Lemma 3.3, we conclude that there is exactly one special node at each level of $NT(G)$ whenever G is an interval graph. The *backbone* of $NT(G)$ is defined as the path $\{n\}, X_1, \dots, X_{k-1}$, where X_i is the special node of $NT(G)$ at level i . Figure 4 shows the backbone of the neighborhood tree of the interval graph in Figure 1. We shall number the components in CC_i from 1 through p_i in such a way that the component corresponding to the special node gets the number one and the other components are numbered arbitrarily (see Fig. 4). Using this convention, the components of CC_i will be denoted as $C_i^1, \dots, C_i^{p_i}$. C_i^1 denotes the component corresponding to the special node, and, hence, it forms part of the backbone. We call the

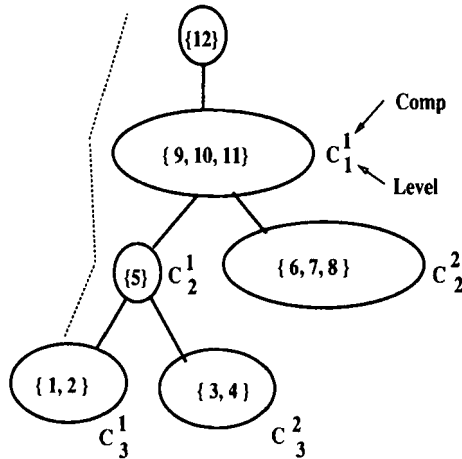


FIG. 4. The neighborhood tree in Figure 2 depicting the backbone in dotted line. The nodes of the tree are labeled as C_j^i , where i is the component number and j is the level number. The special nodes at each level have component number 1.

nodes $C_j^i, j > 1$ the *suspended* nodes of $NT(G)$. If node $x \in V$ is in C_j^i , then we see that $Level(x) = i$ and we define $Comp(x) = j$.

4. CONSTRUCTION OF THE NEIGHBORHOOD TREE

We shall not explicitly construct the neighborhood tree of the input graph G . Instead, note, that computing $Level(x)$ and $Comp(x)$ for each vertex $x \in V$ determines the structure of $NT(G)$ completely. We use the convention described above to allow us to determine if a vertex is in a special node or in a suspended node of $NT(G)$.

We describe below procedures that would allot to each vertex x the values of $Level(x)$ and $Comp(x)$. The input to these procedures is the adjacency matrix of the graph G and the values of $LO(x)$ for every vertex $x \in V$. We do this in a single left-to-right sweep of the interval representation. In the sweep, we can identify the level of each vertex using Lemma 3.2 by computing $MINLO_i$ for each level i to determine the vertices in level $(i + 1)$. Before we explain how we can determine the values of $Comp(x)$ for each vertex x , we state the following useful lemma:

Lemma 4.1. *At any level i of $NT(G)$, the special node at this level contains the lowest numbered vertex in that level.*

Proof. This can be easily proved by contradiction. Let the smallest numbered vertex x_i at level i be in the node $C_j^i, (j \neq 1)$, a suspended node. Note that every node in level $(i + 1)$ will be adjacent to C_j^i , by definition of the special

node. This means that there is a vertex y in level $(i + 1)$ that is adjacent to some vertex x' in C_j^i such that $y < x' < x'$. Property 2.1 implies that x_i is also adjacent to x' , which implies that $j = 1$ since x_i is in the same connected component as x' (a contradiction). Hence, the lemma follows. ■

Since we know by Lemma 3.2 that all vertices in a given level form a contiguous block in the IG-ordering, it is easy to identify the components in each level by a procedure we call *Lo-riding*. Consider the level i with $L(i) = \{x_l, \dots, x_r\}$. Lo-riding starts with the last vertex x_r and finds its *LO*. Now, we find the *LO* of this vertex and continue in this fashion until we either come to a vertex smaller than x_l or we have reached the end of a component. If so, we have come to a vertex x_e whose *LO* is itself. In the former case, we have crossed the current level i , and so the component that we are working with is the special node of this level. We identify this component and number it with one. In the latter case, we assign the block of vertices that we have as a component and give it a number other than one. We then continue lo-riding from the vertex $x_e - 1$. This process will eventually end when we exhaust lo-riding through all the components in this level. Lo-riding is illustrated in Figure 5. The procedures that allot the *Level* and *Comp* entries for each vertex are described below.

Procedure Allot

1. While there are no more levels to be processed,
 - [1.1] Assign level numbers to the vertices in the current level.
 - [1.2] Find the range of vertices in the next level.
 - [1.3] Assign component numbers to each vertex in the current level.

Now, we go on to describe the procedures themselves.

Procedure Allot

For each vertex $x \in V$, we assume that we have the value of $LO(x)$ in an array LO .

1. Initialize $Level(n) = 0$, $Comp(n) = 1$, $p = n - 1$, $lvlcnt = 1$, and $q = LO(p)$.
2. Repeat until $p = 1$
 - [1.1] For each vertex x in the range $\{q, \dots, p - 1\}$, do
 - [1.1.1] $Level(x) = lvlcnt$. Here, we allot level numbers for the vertices in this level.
 - [1.2] $lvlcnt = lvlcnt + 1$.
 - [1.3] *Find - comp*(q, p). We find the components in this level and allot the component entries for vertices in this level.
 - [1.4] Reset the values p and q appropriately. $temp = \min\{LO(x) : x \in \{q, \dots, p - 1\}\}$. This is the MINLO of this level and marks off the start of the next level. $p = q, q = temp$.

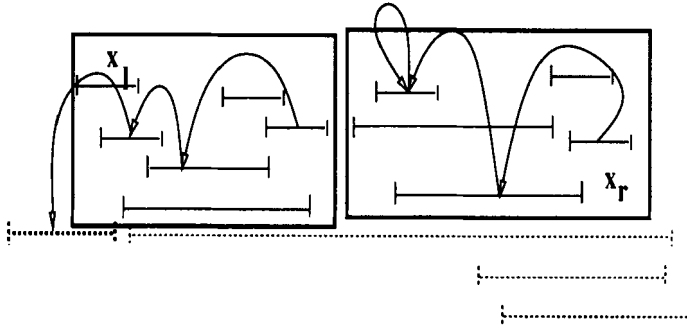


FIG. 5. Lo-riding: At each level, we start at the highest vertex x_r . An arrow from a vertex represents the operation of computing the LO of the vertex. When a vertex whose LO is itself is reached without overshooting the sentinel x_l , we mark off a component. At the next run of lo-ride, we start from the vertex numbered one less than this and continue until we overshoot the sentinel. At this point, we have identified all the connected components of the induced subgraph of this level. The dotted intervals represent vertices not in the level for which the lo-ride is performed. The dark rectangles enclose the components identified for this level.

Procedure Find-comp(x_l, x_r)

This procedure determines the component entries for vertices in the range $\{x_l, \dots, x_r\}$ by successively finding the components by Lo-riding and labeling them.

1. Initialize $last = x_r$ to begin the right-to-left sweep, $mainlo = x_l$, a sentinel for ending lo-riding and $count = 2$ for the component number. The component number one is reserved for the special node. Also, initialize $ret - val = x_r$ for the current vertex we are at.
2. Repeat until $ret - val = mainlo$
 - [2.1] $ret - val = Lo - ride(last, mainlo)$. Here, $ret - val$ identifies the beginning of the component that the vertex $last$ ends. Thus, we have identified the set $\{ret - val, \dots, last\}$ as a component.
 - [2.2] If $ret - val \neq mainlo$,
 - [2.2.1] then we have still not exhausted the vertices in this level. So,
 - [2.2.1.1] For every vertex $x \in \{ret - val, \dots, last\}$, set $Comp(x) = count$.
 - [2.2.1.2] $count = count + 1$. Reset count for the next component number.
 - [2.2.1.3] Lo-ride again now setting $last$ to $ret - val - 1$. Thus, the new $ret - val = Lo - ride(last, ret - val - 1)$.
 - [2.2.2] else, we have finished the sweep of this level and come to the special node. So,
 - [2.2.2.1] For every vertex $x \in \{ret - val, \dots, last\}$, set $Comp(x) = 1$.

Procedure Lo-ride(*last,mainlo*)

This procedure is used within a level to identify the connected components of this level. It takes in as arguments *last*, the vertex to begin the lo-riding from, and also a sentinel *mainlo*, overshooting which means that the current level is exhausted. The procedure returns a vertex $z \leq y$ such that the set $\{z, z + 1, \dots, y\}$ forms a single component in the level i .

1. Initialize $p = y$.
2. While $((LO(p) > mainlo) \text{ AND } (LO(p) \neq p))$,
 [2.1] $p = LO(p)$.
3. If $(LO(p) < mainlo)$
 [3.1] then, we have crossed the sentinel into the next level, so we stop and return $z = mainlo$ signaling this.
 [3.2] else, we have just encountered the end of a component in this level and so we return the last vertex in this component as $z = p$.

The complexity of computing $Level(x)$ and $Comp(x)$ for each vertex x in V is stated below.

Theorem 4.1. *The complexity of computing $Level(x)$ and $Comp(x)$ for each vertex x in V is $O(n + m)$.*

Proof. $Level(x)$ for each vertex x in V is computed in the "Allot" procedure above and takes only $O(n)$ time. This construction follows the result in Lemma 3.2 and involves a single scan of the LO values of all the vertices at a given level to determine the next level.

Determination of the component numbers at each level also takes only $O(m + n)$ time since it involves finding the connected components at each level of $NT(G)$. To do this for level i , it takes time $O(n_i + m_i)$ time where n_i and m_i are, respectively, the number of vertices and edges in the induced subgraph on N_i . Summing over all the levels, we see that the overall time complexity is only $O(m + n)$. Thus, the theorem follows. ■

5. THE ALGORITHM AND THE PROOF OF CORRECTNESS

To compute the distance between every pair of vertices in the interval graph G , we assign to each vertex the information about whether it is a special node or one of the suspended nodes and its level in the tree. The component number determines whether the vertex is in a special node or a suspended node.

We make a top-down pass of the neighborhood tree of the graph and compute at each level of the tree the distance between each vertex in this level and all the vertices in the same level and in the levels above it. This computation can be viewed as a right-to-left scan of the vertices of the interval graph from vertex n to vertex 1 in the IG-ordering. In the scan, at each vertex x , we determine $d(x, y)$ for all $y > x$. The distance calculations at each stage can be viewed to be of

essentially three types. Typically at level i , we wish to enumerate $d(x, y)$, where $Level(x) = i$ and one of the following cases hold:

- (i) $Level(y) = i$;
- (ii) $Level(y) = i - 1$;
- (iii) $Level(y) = j$ where $j < (i - 1)$.

Further, we perform the distance calculations differently depending on whether x and/or y belong to a suspended node (i.e., have their $Comp$ entry greater than 1) or a backbone node (i.e., with the $Comp$ entry equal to 1) in $NT(G)$.

We present below recurrence relations for computing $d(x, y)$, where $y > x$ for the different possible cases of x and y . We also present a concise reason below each of the expressions for the correctness of the recurrences without providing a formal proof. In all these cases, we assume that $Level(x) = i$ and that $z = HI(x)$. Notice that if x is in level i then z must belong to level $i - 1$.

Case 1. $Level(y) = i$.

[Case 1a.] $Comp(x) = Comp(y) = 1$. In this case, both x and y are in the backbone. We then have

$$d(x, y) = 1 \text{ if } x \sim y \\ = 2 \text{ otherwise.}$$

This is obvious, since if there is an edge between x and y , the distance is 1. Otherwise, there exists an obvious path of length two through some vertex w in level $(i - 1)$. This case is shown in Figure 6.

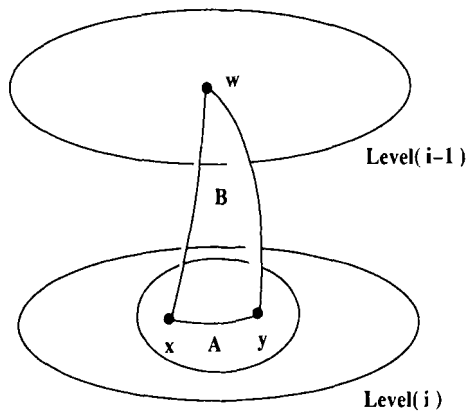


FIG. 6. Case 1a of the recurrences for $d(x, y)$. If the edge labeled A exists between x and y , then $d(x, y) = 1$. Otherwise, there is a path between x and y of length two through some vertex w in level $(i - 1)$.

[Case 1b.] $Comp(x) = 1$ and $Comp(y) > 1$. Here we have

$$d(x, y) = 1 + d(y, z).$$

The recurrence above stems from the observation that if there exists a shortest path P from x to y via some vertex $z' < z$ in level $(i - 1)$, then there is a path P' from x to y via z itself of the same length. This case is depicted in Figure 7. To see this, observe that if z'' is the next vertex after z' in this shortest path P , then $z'' > z'$ and $z' \sim z''$ implies that $z \sim z''$ since we know that $z' < z < z''$. This follows from Property 2.1 of the IG-ordering. Using this edge (z, z'') and the rest of the shortest path in P , we can construct a new shortest path P' via z from x to y .

[Case 1c.] $Comp(x) > 1$ and $Comp(y) > 1$. In this case also,

$$d(x, y) = 1 + d(y, z).$$

The reasoning is identical to that for Case 1b.

Case 2. $Level(y) = i - 1$.

[Case 2a.] $Comp(x) = 1$ and $Comp(x) \geq 1$. In this case, we have

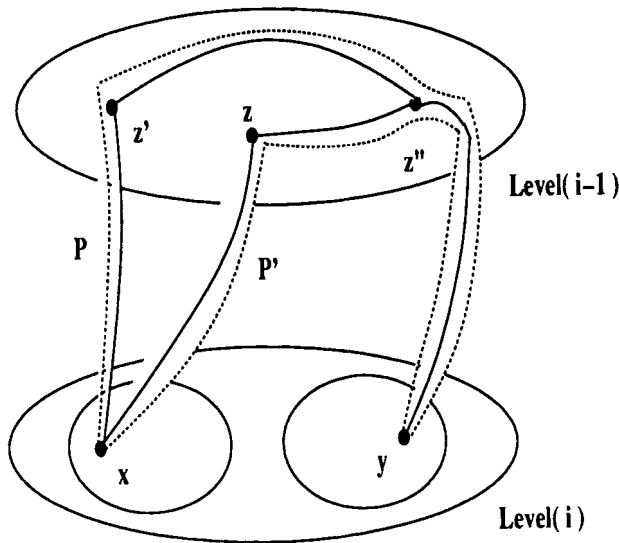


FIG. 7. Cases 1b, 1c, 2, and 3 of the recurrences for $d(x, y)$ showing that there is a shortest path from x to y via $z = HI(x)$. Let P be a shortest path that goes via a vertex $z' \neq z$, and let z'' denote the vertex following z' in P from x to y . Since $z' < z \leq z''$, by the interval graph Property 2.1, another shortest path P' between x and y can be identified via z .

$$\begin{aligned}
 d(x, y) &= 1 \text{ if } x \sim y \\
 &= 2 \text{ if } x \text{ is not adjacent to } y \text{ and } z > y \\
 &= 1 + d(z, y) \text{ otherwise.}
 \end{aligned}$$

These recurrences are drawn again based on arguments similar to those in Case 1, except for the case when $d(x, y) = 2$. This is also easy since if x is not adjacent to y , then $x \sim z$ where $x < y < z$. By the interval graph Property 2.1, we then have that $y \sim z$, and so there is a path of length two from x to y via z . This case is shown in Figure 8.

- [Case 2b.] $Comp(x) = 1$ and $Comp(y) > 1$.
- [Case 2c.] $Comp(x) > 1$ and $Comp(y) > 1$.

In both the cases above, by an argument similar to that offered for Case 1b, we can conclude that

$$d(x, y) = 1 + d(z, y).$$

Case 3. $Level(y) = j$, where $j < (i - 1)$. In all the subcases of this case, too, we can see that a shortest path between x and y must pass through the vertex z and arrive at

$$d(x, y) = 1 + d(z, y).$$

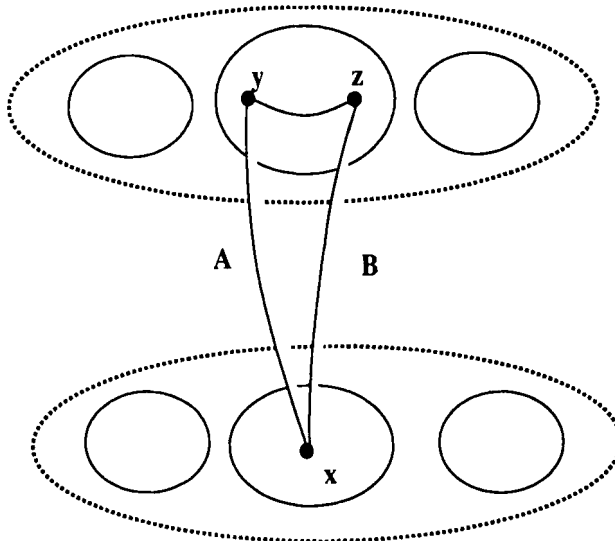


FIG. 8. Case 2a of the recurrences for $d(x, y)$ where $y < z$. If the edge labeled A exists between x and y , then $d(x, y)$ is obviously 1. Otherwise, there is a path between x and y of length two using the edge labeled B from x to $z = HI(x)$ and then to y . The existence of the edge from z to y is by the interval graph Property 2.1 given that $x < y < z$.

The distances between every pair of vertices of G can thus be calculated by making a top-down evaluation of $NT(G)$. The only point that must be clarified is the order in which we calculate the distances associated with the vertices at a given level i . At each level, we make the following four passes of computation. Note that whenever $d(x, y)$ is computed, both the values $d(x, y)$ as well as $d(y, x)$ are updated to facilitate easy computation of the recurrences later.

- Pass 1.** In the first pass, we compute the distances between all pairs of vertices in level i that lie within the same component (and, hence, have the same component number). This corresponds to the application of Case 1a and a few of the cases in Case 1c of the recurrences above.
- Pass 2.** In the second pass, we compute all distance $d(x, y)$ such that $Level(x) = i$ and $Level(y) = i - 1$. This is case 2 of the recurrences.
- Pass 3.** This third pass corresponds to computations in case 3 of the recurrences above and determines all the distances $d(x, y)$ such that $Level(x) = i$ and $Level(y) = j$ where $j < (i - 1)$.
- Pass 4.** In this last pass, we compute the distance between all pairs of vertices in level i such that they lie in different components (and, hence, have different component numbers). These are the rest of the cases in case 1 of the recurrences.

It is easy to verify that this order of computing the distances at each level ensures that all values needed for substitution in a recurrence are already computed at an earlier pass and are available. This makes each of the recurrence computations a constant time table look-up.

6. COMPLEXITY AND CONCLUSIONS

The complexity of computing $LO(x)$ for each vertex x in V is $O(n + m)$ [7]. $HI(x)$ can be computed similarly in linear time. In Theorem 4.1, we have shown that $Level(x)$ and $Comp(x)$ for each vertex X can be computed in linear time. Thus, all the preprocessing takes time $O(m + n)$.

Since the determination of $d(x, y)$ for any given x and y in the order proposed above takes only constant time and there are $O(n^2)$ entries to fill, the whole algorithm has an overall time complexity of $O(n^2)$. Since there are $O(n^2)$ entries to be calculated to solve the all-pair shortest path problem, it is clear that our bound is optimal.

It would be interesting to investigate if the weighted version of the all-pair shortest path problem can be handled in a similar fashion.

REFERENCES

- [1] A. Aho, J. Hopcroft, and J. Ullman, *The Design and Analysis of Computer Algorithms*, Addison-Wesley, Reading, MA (1974).

- [2] S. Aravind, R. Mahesh, and C. Pandu Rangan, Bandwidth optimization for interval graphs revisited. *Inform. and Computation*, in press.
- [3] D. G. Corneil and J. Fonlupt, The complexity of generalized clique covering. *Discrete Appl. Math.* **22**, (1988–89), 109–118.
- [4] M. C. Golumbic, *Algorithmic Graph Theory and Perfect Graphs*. Academic Press, New York (1980).
- [5] J. M. Keil, Finding Hamiltonian circuits in interval graphs. *Infor. Process. Lett.* **20** (1985), 201–206.
- [6] A. Moffat and T. Takoka, An all-pair shortest path algorithm with expected running time $O(n^2 \log n)$. *Proceedings of the IEEE Symposium on Foundations of Computer Science* (1985) 101–105.
- [7] G. Ramalingam and C. Pandu Rangan, A unified approach to domination problems in interval graphs'', *Infor. Process. Lett.* **27** (1988) 271–274.
- [8] A. Srinivas Rao and C. Pandu Rangan, A linear algorithm for domatic number of interval graphs. *Infor. Process. Lett.* **33** (1989–90), 29–33.

Received December 1989

Accepted May 1991