Proceedings of the 14th IASTED International Conference
PARALLEL AND DISTRIBUTED COMPUTING AND SYSTEMS
November 4-6, 2002, Cambridge, USA

New Routing Schemes for Interval, Circular-Arc, and Permutation Graphs

Feodor F. Dragan
Department of Computer Science
Kent State University
Kent, OH 44240, USA
email: dragan@cs.kent.edu

Irina Lomonosov
Department of Computer Science
Kent State University
Kent, OH 44240, USA
email: ilomonos@cs.kent.edu

ABSTRACT

In this note we present new routing schemes for three classes of intersection graphs: interval graphs, circular-arc graphs and permutation graphs. The routing schemes are compact, efficient and optimal for interval and circular-arc graphs and almost optimal for permutation graphs. For interval graphs, our routing scheme outperforms existing ones in efficiency. To the best of our knowledge, these are first non-trivial routing schemes that are presented for circular-arc graphs and permutation graphs.

KEY WORDS

Communication algorithms, message routing, intersection graphs.

1 Introduction

Routing is one of the basic tasks that a distributed network of processors must be able to perform. A *routing scheme* is a mechanism that can deliver packets of information from any node of the network to any other node of the network. More specifically, a routing scheme is a distributed algorithm. Each processor in the network has a routing daemon (known also as a *message passing algorithm* or a *forwarding protocol*) running on it. This daemon receives packets of information and has to decide whether these packets have already reached their destination, and if not, how to forward them towards their destination.

A network can be viewed as a graph, with the nodes representing processors and the edges representing direct connections between processors. It is naturally desirable to route messages along paths of this graph that are as short as possible. A straightforward approach to achieve this goal is to store a *complete routing table* in each node x of the network, specifying for each destination y the first edge (or identifier of that edge, indicating the output port) along some shortest path from x to y. While this approach guarantees optimal (shortest path) routing, it is too expensive for large systems since it requires $O(n\ log\ d)$ memory bits for a node of degree d in an n-node network. Thus, for large scale communication networks, it is important to design such routing schemes which produce short enough routes and have relatively low *memory requirements*.

Most routing schemes are *labeling schemes* that assign two kind of labels to every node of a graph. The first label is the *address* of the node, the second is a data structure called *local routing table*. These labels are assigned in such a way that at every source node x its routing daemon can quickly decide, based on two labels stored locally in x and the address of any destination node y, whether the packet has reached its destination, and if not, to which neighbor of x to forward the packet.

Unfortunately, for every shortest path routing strategy and for all d, there is a graph of degree bounded by d for which $\Omega(n\ log\ d)$ bit routing tables are required simultaneously on $\Theta(n)$ nodes [1]. This matches the memory requirements of complete routing tables. To obtain routing schemes for general graphs that use $o(n)$ of memory at each node, one has to abandon the requirement that packets are always delivered via shortest paths, and settle instead for the requirement that packets are routed on paths that are relatively close to shortest. The *multiplicative* (or *additive*) *stretch* of a routing scheme is the maximum ratio (or surplus) between the length of a route produced by the scheme for some pair of nodes, and their distance. But again, any multiplicative t stretched routing scheme must use $\Omega(\sqrt{n})$ bits for some nodes in some graphs for $t < 5$ [2], $\Omega(n)$ bits for $t < 3$ [3], and $\Omega(n\ log\ n)$ bits for $t < 1.4$ [1].

These lower bounds suggest to look for specific routing strategies with compact routing tables on special (better structured) families of graphs. The goal is, for a family of graphs, to find a routing scheme with short enough labels (a *compact* routing scheme) and fast massage passing mechanism (an *efficient* routing scheme), which forwards messages along shortest paths (an *optimal* routing scheme).

There are many results on optimal routing schemes for particular graph classes, including complete graphs, grids (alias meshes), hypercubes, complete bipartite graphs, unit interval and interval graphs, trees and 2-trees, rings, tori, unit circular-arc graphs, and outerplanar graphs (see [4, 5, 6, 7, 8]). All those graph families admit optimal routing schemes with $O(\Delta\ log\ n)$ bit labels and $O(\log \Delta)$ forwarding protocol, where Δ is the maximum degree of a node of the graph. Hence, it takes $O(dist(x, y) \log \Delta)$ total time to deliver a message from source x to destination y. These results follow from the existence of special so called *interval routing schemes* for those graphs. We

will not discuss details of this scheme here; for precise definitions and an overview of this technique as well as other routing strategies, we refer the reader to [9, 10] and [11]. Note only that recently authors of [12] described a new shortest path routing scheme for trees of arbitrary degree and diameter that assigns each node of an n-node tree a $(1+o(1))\ log\ n$-bit label. Given the label of a source node and the label of a destination it is possible to compute, in constant time, the neighbor of the source that heads in the direction of the destination. Also, in [13] an additive 2 stretched routing scheme with addresses and routing tables of $O(log^3\ n/log\ log\ n)$ bits per node and $O(1)$ forwarding protocol is designed for chordal graphs.

In this note, we design simple routing schemes for interval graphs, circular-arc graphs, and permutation graphs. For both interval graphs and circular-arc graphs, we present optimal routing schemes with $O(\Delta\ log\ n)$ bit labels and $O(1)$ forwarding protocol. For permutation graphs, our routing scheme is additive 1 stretched and also uses $O(\Delta\ log\ n)$ bit labels and $O(1)$ forwarding protocol. To the best of our knowledge these are first routing schemes that are presented for circular-arc graphs and permutation graphs. Previous optimal routing scheme for interval graphs was based on interval routing strategy and its construction is quite involved (see [7]). As for circular-arc graphs, optimal interval routing schemes were known only for unit circular-arc graphs, a proper subclass of circular-arc graphs (see [4]). And, it was pointed out in [4] that no interval routing scheme with one interval per edge exists for all circular-arc graphs. As a byproduct, for proper circular-arc graphs (subclass of circular-arc graphs and superclass of unit circular-arc graphs), we obtain an optimal routing scheme with $O(log\ n)$ bit labels and $O(1)$ forwarding protocol. Assuming that *routing speed* is most critical consideration, our schemes improve efficiency of previous available routing schemes for interval graphs and unit circular-arc graphs. In designing our routing schemes we essentially used the intersection models of graphs under consideration.

Proofs of all lemmas are omitted in this version. They will be presented in the full version of the paper.

2 Definitions

The intersection graph of a family of n sets is the graph where the nodes are the sets, and the edges are the pairs of sets that intersect. Every graph is the intersection graph of some family of sets. A graph $G=(V,E)$ is an *interval graph* if it is the intersection graph of a finite set of intervals (line segments) on a line. A graph G is a *circular-arc graph* if it is the intersection graph of a finite set of arcs on a circle. An interval graph is a special case of a circular-arc graph; it is a circular-arc graph that can be represented with arcs that do not cover the entire circle. The set of intervals (circular arcs) is called the interval (circular-arc) model of G. We may assume without loss of generality that no endpoints of intervals (arcs) coincide. It is known that interval graphs as well as circular-arc graphs can be recognized in linear $O(|V|+|E|)$ time (see e.g., [14, 15]). Moreover, if G is an interval graph (circular-arc graph) then interval (circular-arc) model of G can be constructed within the same time bound. Therefore, in what follows we will assume that interval (circular-arc) model of G is available.

Let now $P=(P(1),\ldots,P(n))$ be a permutation on the set $V_n=\{1,\ldots,n\}$. Then graph $G(P)=(V,E)$ is the graph with node set $V=V_n$ such that node i is adjacent to j in G if and only if $(i-j)(P^{-1}(i)-P^{-1}(j))<0$. (Read $P^{-1}(i)$ as the position of the number i in the sequence $(P(1),\ldots,P(n))$.) A graph G is called a *permutation graph* if there exists a permutation P such that G is isomorphic to $G(P)$. A permutation graph can be viewed as an intersection graph in the following way. Consider two parallel horizontal lines with n evenly spaced points on each of the lines. The points on the top line are numbered $1,\ldots,n$ from left to right and those on the bottom line, $(P(1),\ldots,P(n))$. The points on the two lines with the same number are connected by a straight line segment. We refer to each such line segment by the same number i. Note that segment i intersects segment j if and only if i and j appear in the reversed order in P. This is the same as the criterion for the nodes i and j of the permutation graph to be adjacent. Thus, a permutation graph is exactly the same as the intersection graph of the set of line segments with ends on two parallel lines. This set of segments is called a *permutation model* (*permutation diagram*) of G. In [16], a linear time recognition algorithm is given for permutation graphs which builds a permutation model of a given permutation graph G in linear time as well. Hence, again in what follows we will assume that permutation model of G is available.

The *distance* $dist(v,u)$ between nodes v and u of a connected graph $G=(V,E)$ is the smallest number of edges in a path connecting v and u. Denote by $N(v)=\{u\in V: uv\in E\}$ the *neighborhood* of a node v of G and by $N[v]=N(v)\cup\{v\}$ the *closed neighborhood* of v. Let also $d(v)=|N(v)|$ be the degree of v and Δ the maximum degree of a node of G.

3 Interval graphs

Let $G=(V,E)$ be a connected n-node interval graph given with an interval representation (model). We find it convenient to let v stand both for a node in G and for the corresponding line segment in the interval model.

We can consequently assign numbers from 1 to $2n$ to endpoints of the line segments from left to right. Each node v of the graph is represented by a distinct pair of integers $l(v), r(v)$, where $l(v)$ and $r(v)$ are the numbers of the left and right endpoints of the segment representing node v. We will use $r(v)$ as an *identifier* (ID) for a node v. We can sort all nodes $u\in N(v)$ according to their right endpoints $r(u)$. Obviously, $u\in N(v)$ if and only if at least one of the following holds 1) $l(u)<l(v)<r(u)$, 2) $l(u)<r(v)<r(u)$, 3) $l(v)<l(u)<r(v)$, 4) $l(v)<r(u)<r(v)$. Hence, the adjacency of two nodes can be established in

$O(1)$ time.

For each segment v, we consider a set of segments that contain the right endpoint of v. Among these neighbors of node v we choose that one with maximum $r(\cdot)$-value, denote it by $x(v)$ and call it the *right maximum neighbor* of v. In a similar way we define the *left maximum neighbor* $y(v)$ of v by considering all segments containing the left endpoint of v and taking that one with minimum $l(\cdot)$-value.

We say that a node v is a *good* node if the interval corresponding to it is not a subinterval of any other interval u in the model. In other words v is good if there is no $u \neq v$ such that $l(u) < l(v)$ and $r(v) < r(u)$. If a node v is not good then it is a *bad* node. We can determine for any node v of $G = (V, E)$ if it is good or bad in $O(d(v))$ time. Hence, in $O(|E|)$ time we can determine all good and bad nodes in G. Obviously, if a node v is right or left maximum neighbor of some node w then v is a good node.

For each node we construct a *list of gates* (to the neighbors) in the following way. Note that if $u \in N(v)$ and v is a good node then either $r(u) \in [l(v), r(v)]$ or $l(u) \in [l(v), r(v)]$ or both. To each endpoint in the interval $(l(v), r(v))$ we assign a gate number, which is the index in ordered list $N(v)$ of the node, segment of which has this endpoint. The size of the gate list is $r(v) - l(v) - 1 = O(d(v))$. Then, the gate number for $u \in N(v)$ can be found in constant time as $r(u) - l(v)$-th entry in the gate list of v if $r(u) \in [l(v), r(v)]$ or as $l(u) - l(v)$-th entry if $l(u) \in [l(v), r(v)]$. For a bad node v, there is a neighbor u which does not have any endpoint in $[l(v), r(v)]$ (i.e., interval v is a subinterval of u).

For each bad node v of a graph G, we construct also an *extended list of gates* in the following way. Note that if $u \in N(v)$ and interval v is a subinterval of u then $r(u) \in (r(v), r(x(v))]$. To each endpoint in the interval $(r(v), r(x(v))]$ we assign a gate number if this is the right endpoint of a segment corresponding to a node from $N(v)$ or mark it invalid otherwise. Again, the gate number of an endpoint is the index in ordered list $N(v)$ of the node, segment of which has that endpoint. The number of entries in this list is $r(x(v)) - r(v) = O(d(x(v)) = O(\Delta)$. The gate number can be found in constant time as $r(u) - r(v)$-th entry in the extended gate list of v if $u \in N(v)$ but $l(u) < l(v) < r(v) < r(u)$.

Lemma 1 *For each node v of an interval graph G and any node $u \notin N(v)$ the following holds. If $r(u) > r(v)$ then there exits a shortest path between v and u which goes through $x(v)$. Otherwise, there exits a shortest path between v and u which goes through $y(v)$.*

The right maximum neighbors ($x(v)$'s) and the left maximum neighbors ($y(v)$'s) of all nodes of G can be found in $O(n)$ time, if endpoints of intervals are already sorted.

Now we will describe our routing scheme for interval graphs. A pair of integers $\{l(u), r(u)\}$ is considered to be an address of a node u. Routing a message from a source node to a destination node u can be done as follows. *Message Passing Algorithm* described in Fig. 1 is executed at each node v after receiving a message (v is an intermediate node on a path between a source and a destination). The same algorithm is executed at a node v when it initiates a message (v is a source) and v is a good node.

If a node v initiates a message (v is a source node) and v is a bad node then one of the three *Message Initiating Algorithms* given in Fig. 2 can be used. The following information is kept at each node v of the graph G:

- $\{l(v), r(v)\}$, the endpoints of the segment v.
- $r(x(v))$ and $r(y(v))$, the identifiers of the segments $x(v)$ and $y(v)$.
- $Gate_v = \{g_1, g_2, \ldots, g_m\}$, a gate list for v, where $m = r(v) - l(v) - 1$.

If v is a bad node then additionally we need to keep $N(v)$, a sorted list of neighbors if *Message Initiation Algorithm 1* is used, or an extended list of gates if *Message Initiation Algorithm 2* is used. No extra information is needed for *Message Initiation Algorithm 3*.

Input: A current node v and a destination node u
Output: A neighbor to send the message to

---

if $(r(v) < l(u))$ then send the message to $x(v)$ and stop;
if $(r(u) < l(v))$ then send the message to $y(v)$ and stop;
if $(l(v) < r(u) < r(v))$ then send the message through $Gate_v(r(u) - l(v))$ and stop;
if $(l(v) < l(u) < r(v))$ then send the message through $Gate_v(l(u) - l(v))$ and stop;

Figure 1. *Message Passing Algorithm* for interval graphs

Input: A source node v and a destination node u
Output: A neighbor to send the message to

---

execute *Message Passing Algorithm*;
do case
  case Message Initiating Algorithm 1
    search a sorted list of neighbors $N(v)$ for a gate and send the message to u;
  case Message Initiating Algorithm 2
    send the message through $ExGate_v(r(u) - r(v))$;
  case Message Initiating Algorithm 3
    send the message to $x(v)$;
endcase

Figure 2. *Three Message Initiating Algorithms.*

Theorem 1 *(i) There exists an optimal routing scheme for interval graphs that uses addresses of size $O(\log n)$ bits, routing tables of size $O(d(v)\ \log n)$ bits per node v, and it takes $O(dist(v,u) + \log \Delta)$ total time to route a message from v to u. Moreover, the time complexity to setup all the data structures is bounded by $O(n \log n + |E|)$.*
(ii) There exists an optimal routing scheme for interval graphs that uses addresses of size $O(\log n)$ bits, routing tables of size $O(\Delta \log n)$ bits per node and forwarding protocols of complexity $O(1)$. Moreover, the time complexity to setup all the data structures is bounded by $O(n \log n + \Delta\, n)$.
(iii) There exists an additive 1 stretched routing scheme for interval graphs that uses addresses of size $O(\log n)$ bits, routing tables of size $O(d(v)\ \log n)$ bits per node v and forwarding protocols of complexity $O(1)$. Moreover, the time complexity to setup all the data structures is bounded by $O(n \log n + |E|)$.

Interval graphs without bad nodes are known as *proper interval graphs* (alias, *unit interval graphs*). Since for every proper interval graph G, one can assign gate numbers in such a way that $Gate_v(i) = i$ will hold for any node v of G, we do not need to keep gate lists in local tables. Therefore, the following corollary is true.

Corollary 1 *There is an optimal routing scheme for proper interval graphs that uses addresses and routing tables of size $O(\log n)$ bits per node and forwarding protocols of complexity $O(1)$. Moreover, the time complexity to setup all the data structures is bounded by $O(n \log n + |E|)$.*

4 Circular-Arc Graphs

Let $G = (V, E)$ be a connected n-node circular-arc graph given with an circular-arc model. If G is an interval graph then the routing scheme for an interval graph can be used. So, in what follows we will assume that G is a circular-arc graph but not an interval graph. Again, we let v stand both for a node of G and for the corresponding arc in the circular-arc presentation.

The endpoint of an arc encountered first in counterclockwise order we will call the *left* endpoint of the arc, and the other endpoint we will call the *right* endpoint of the arc. In a similar to interval graphs way we can consequently assign numbers from 1 to $2n$ to endpoints of the arcs in counterclockwise order. Each node v of the graph is represented by a distinct pair of integers $l(v), r(v)$, where $l(v)$ and $r(v)$ are the numbers of the left and right endpoints of the arc representing node v.

We will consider circular intervals in the form $[a, b]$, which include all endpoints of the arcs starting from the point a and going counterclockwise until the point b. Obviously, $u \in N(v)$ if and only if at least one of the following holds 1) $l(v) \in [l(u), r(u)]$, 2) $r(v) \in [l(u), r(u)]$, 3) $l(u) \in [l(v), r(v)]$, 4) $r(u) \in [l(v), r(v)]$. Hence, the adjacency of two nodes can be established in $O(1)$ time.

For each arc v we consider a set of arcs that contain the right endpoint of v. Among these neighbors of node v we choose that one whose arc stretches farthest counterclockwise, we denote it by $rmax(v)$ and call it the *right maximum neighbor* of v. In a similar way we define the *left maximum neighbor* $lmax(v)$ of v by considering all arcs containing the left endpoint of v and taking the one that stretches farthest clockwise.

For each node v we define two sequences of nodes in the following way:

$$x_1(v) = rmax(v);\ \ x_{i+1}(v) = rmax(x_i(v)),\ i > 0;$$
$$y_1(v) = lmax(v);\ \ y_{i+1}(v) = lmax(y_i(v)),\ i > 0.$$

As for interval graphs, here we also distinguish between good and bad nodes and construct a *list of gates* for every node and an *extended list of gates* for each bad node.

Let $N_k(u) = N(u) \cap L_k(v)$, where $L_k(v)$ is the k-th level of a BFS tree rooted at v.

Lemma 2 *For each node v of a circular-arc graph G there exist a node $\varepsilon(v)$ such that for any arc $u \notin N(v)$ the following holds. If $r(u) \in [r(v), r(\varepsilon(v))]$ then there exits a shortest path between v and u which goes through $x_1(v)$. Otherwise, there exits a shortest path between v and u which goes through $y_1(v)$.*

Lemma 3 *If $r(x_1(v)) \in [l(y_1(v)), r(y_1(v))]$ then $\varepsilon(v) = x_1(v)$. Otherwise, if $k > 1$ is the minimal number such that $x_k(v) \in N[y_k(v)]$, then the following holds.*

(i) If $r(x_k(v)) \notin [l(y_{k-1}(v)), r(y_{k-1}(v))]$ then $\varepsilon(v) = x_k(v)$.

(ii) If $r(x_k(v)) \in [l(y_{k-1}(v)), r(y_{k-1}(v))]$ and there are nodes in $L_k(v)$ which are adjacent to $x_{k-1}(v)$ but not to $y_{k-1}(v)$, then $\varepsilon(v)$ is such a node w that stretches farthest counterclockwise.

(iii) Otherwise, $\varepsilon(v) = x_{k-1}(v)$.

We say that $\varepsilon(v)$ is *antipodal* to v. Antipodal nodes for all nodes of G can be found in $O(n^2)$ time as described in Lemma 3. The right maximum neighbors ($x_1(v)$'s) and the left maximum neighbors ($y_1(v)$'s) of all nodes of G can be found in $O(n)$ time, if endpoints of arcs are already sorted counterclockwise.

Similar to routing in interval graphs, a pair of integers $\{l(u), r(u)\}$ is considered to be an address of a node u. Routing a message from a source node to a destination node u can be done as follows. *Message Passing Algorithm* described in Fig. 3 is executed at each node v after receiving a message (v is an intermediate node on a path between a source and a destination). The same algorithm is executed at a node v when it initiates a message (v is a source) and v is a good node. If a node v initiates a message (v is a source node) and v is a bad node then one of the Message Initiating Algorithms described for interval graphs can be used.

The following information is kept at each node v of the graph G:

- $\{l(v), r(v)\}$, the endpoints of the arc v.

Input: A current node v and a destination node u
Output: A neighbor to send the message to

---

if $(r(u) \in [l(v), r(v)])$ then send the message through $Gate_v(r(u) - l(v))$ and stop;

if $(l(u) \in [l(v), r(v)])$ then send the message through $Gate_v(l(u) - l(v))$ and stop;

if $(l(v) \notin [l(u), r(u)])$ then

if $(r(u) \in [r(v), r(\varepsilon(v))])$ then send the message to $x_1(v)$ and stop;

if $(r(u) \in [r(\varepsilon(v)), l(v)])$ then send the message to $y_1(v)$ and stop;

Figure 3. *Message Passing Algorithm* for circular-arc graphs.

- $r(x_1(v))$ and $r(y_1(v))$, the identifiers of the arcs $x_1(v)$ and $y_1(v)$.
- $r(\varepsilon(v))$, the identifier of antipodal node.
- $Gate_v = \{g_1, g_2, \ldots, g_m\}$, a gate list for v, where $m = r(v) - l(v) - 1$.

Again, if v is a bad node then additionally we need to keep $N(v)$, a sorted list of neighbors if *Message Initiation Algorithm 1* is used, or an extended list of gates if *Message Initiation Algorithm 2* is used. No extra information needed for *Message Initiation Algorithm 3*.

Summarizing we have the following theorem. Recall that a circular-arc graph without bad arcs is called a *proper circular-arc graph*.

Theorem 2 *Results formulated in Theorem 1 and Corollary 1 hold for circular-arc graphs and proper circular-arc graphs, respectively. The only difference is that the time complexity to setup all the data structures is now bounded by* $O(n^2)$.

5 Permutation graphs

Let $G = (V, E)$ be a connected n-node permutation graph given with a permutation diagram and let $V = \{1, 2, \ldots, n\}$. As before, integer v ($v \in \{1, 2, \ldots, n\}$) will stand both for a node of G and for the corresponding segment in the permutation diagram.

Let $N^+(v) = \{u \in N(v) : u > v\}$ and $N^-(v) = N(v) \setminus N^+(v)$. For each node v, we define an *upper right maximum neighbor* $x_\uparrow(v)$, which is the largest integer in $N^+(v)$, and a *lower right maximum neighbor* $x_\downarrow(v)$, which is a node from $N^-(v)$ such that $P^{-1}(x_\downarrow(v)) \geq P^{-1}(u)$ for any $u \in N^-(v)$. Similarly we define an *upper left maximum neighbor* $y_\uparrow(v)$, which is the smallest integer in $N^-(v)$, and a *lower left maximum neighbor* $y_\downarrow(v)$, which is a node from $N^+(v)$ such that $P^{-1}(y_\downarrow(v)) \leq P^{-1}(u)$ for any $u \in N^+(v)$. Note that neighbors $x_\uparrow(v)$, $y_\downarrow(v)$ or $y_\uparrow(v)$, $x_\downarrow(v)$ can be undefined for a node v if either $N^+(v)$ or $N^-(v)$ is empty.

Lemma 4 *Let $v < u$ and $u \notin N(v)$. Then, there exists a shortest path $v, a_1, a_2, \ldots, a_l, u$ between v and u with $a_1 \in \{x_\uparrow(v), x_\downarrow(v)\}$ such that for any i ($2 \leq i \leq l$) the following holds. If $a_1 = x_\uparrow(v)$, then $a_i = x_\uparrow(a_{i-1})$, $N^+(a_i) = \emptyset$, when i is odd, and $a_i = x_\downarrow(a_{i-1})$, $N^-(a_i) = \emptyset$, when i is even. If $a_1 = x_\downarrow(v)$, then $a_i = x_\downarrow(a_{i-1})$, $N^-(a_i) = \emptyset$, when i is odd, and $a_i = x_\uparrow(a_{i-1})$, $N^+(a_i) = \emptyset$, when i is even.*

Similar result can be stated for nodes v, u such that $v > u$ and $u \notin N(v)$. We will need only to replace $x_\uparrow, x_\downarrow$ with $y_\uparrow, y_\downarrow$, respectively.

We call that shortest $(v-u)$-path described in Lemma 4 a *maximum neighbor shortest path*. Clearly, it is solely determined by its first edge. Given that edge va_1, each node a_i ($1 \leq i \leq l-1$) of the path has only one left maximum neighbor (either $x_\uparrow(a_i)$ or $x_\downarrow(a_i)$) in G and the path has to continue via that node, until it reaches node a_l from $N(u)$.

Let now v be a node that has both upper right and lower right maximum neighbors. For v and any node u of G such that $v < u$ and $u \notin N(v)$ (case $v > u$ is similar), we can define two $(v-u)$-paths: $Q = v, q_1, q_2, \ldots, q_k, u$ and $R = v, r_1, r_2, \ldots, r_l, u$, where $q_1 = x_\downarrow(v)$, $r_1 = x_\uparrow(v)$, $q_i = x_\downarrow(q_{i-1})$ when $3 \leq i \leq k$ is odd, $q_i = x_\uparrow(q_{i-1})$ when $2 \leq i \leq k$ is even, $r_i = x_\uparrow(r_{i-1})$ when $3 \leq i \leq l$ is odd, $r_i = x_\downarrow(r_{i-1})$ when $2 \leq i \leq l$ is even, and k and l are smallest integers such that $q_k, r_l \in N(u)$.

Lemma 4 says that one of these paths (maybe both) is a shortest $(v-u)$-path. The following lemma describes how far these paths are from each other in length and inside the graph.

Lemma 5 *Let Q and R be the $(v-u)$-paths defined as above. Then, for any $1 \leq i \leq \min\{l, k\}$, $q_i \in N(r_i)$ holds. Moreover, $|l - k| \leq 1$.*

The upper right, low right, upper left and lower left maximum neighbors for all nodes can be found in linear time.

Now we describe our routing scheme for permutation graphs. A pair of integers $\{u, P^{-1}(u)\}$ is considered to be an address of a node. Routing a message from a source node to a destination node u is done as follows. *Routing* algorithm given in Fig. 4 is executed at each node v after receiving a message (v is an intermediate node on a path between a source and a destination). The same algorithm is executed at a node v when it initiates a message (v is a source).

If current node v is adjacent to destination node u, i.e., $(v-u)(P^{-1}(v) - P^{-1}(v)) < 0$, then we can do the following. We either can keep at each node v a sorted list of its neighbors and search that list for a gate in $O(log\ d(v))$ time, or keep at each node v an extended list of gates of size $O(\Delta\ log\ n)$ bits and have a direct access to the gate number of u. Here an extended list of gates will consist of

Input: A current node v and a destination node u
Output: A neighbor to send the message to

---

if $(v < u)$ then

if $(P^{-1}(v) > P^{-1}(u))$ then send the message to u and stop;

if $x_{\uparrow}(v)$ exists then send the message to $x_{\uparrow}(v)$

else send the message to $x_{\downarrow}(v)$;

if $(v > u)$ then

if $(P^{-1}(v) < P^{-1}(u))$ then send the message to u and stop;

if $y_{\uparrow}(v)$ exists then send the message to $y_{\uparrow}(v)$

else send the message to $y_{\downarrow}(v)$;

Figure 4. *Routing* algorithm.

two sublists $ExGate_v^+$ and $ExGate_v^-$. Sublist $ExGate_v^+$ is of size $x_{\uparrow}(v) - v = O(deg(v) + deg(x_{\uparrow}(v))) = O(\Delta)$ and, for each integer i $(v < i \leq x_{\uparrow}(v))$, will contain a gate number if $i \in N^+(v)$ or will be marked as invalid otherwise. The gate number of a neighbor u of v with $v < u$ can be found in constant time as $(u - v)$-th entry in $ExGate_v^+$. Similarly, sublist $ExGate_v^-$ is of size $v - y_{\uparrow}(v) = O(deg(v) + deg(y_{\uparrow}(v))) = O(\Delta)$ and, for each integer i $(y_{\uparrow}(v) \leq i < v)$, will contain a gate number if $i \in N^-(v)$ or will be marked as invalid otherwise. The gate number of a neighbor u of v with $v > u$ can be found in constant time as $(u - y_{\uparrow}(v) + 1)$-th entry in $ExGate_v^-$.

Thus, the following information will be kept at each node v of the permutation graph G:

- $\{v, P^{-1}(v)\}$
- $x_{\uparrow}(v), x_{\downarrow}(v), y_{\uparrow}(v), y_{\downarrow}(v)$.
- either $N(v)$, a sorted list of neighbors of v, or $ExGate_v^+$ and $ExGate_v^-$.

Summarizing, we have the following theorem for permutation graphs.

Theorem 3 *(i) There exists an additive 1 stretched routing scheme for permutation graphs that uses addresses of size $O(\log n)$ bits, routing tables of size $O(d(v) \log n)$ bits per node v, and it takes $O(dist(v,u) + \log \Delta)$ total time to route a message from v to u. Moreover, the time complexity to setup all the data structures is bounded by $O(n + |E|)$. (ii) There exists an additive 1 stretched routing scheme for permutation graphs that uses addresses of size $O(\log n)$ bits, routing tables of size $O(\Delta \log n)$ bits per node and forwarding protocols of complexity $O(1)$. Moreover, the time complexity to setup all the data structures is bounded by $O(\Delta\, n)$.*

6 Conclusion

In this note we presented new routing schemes for interval graphs, circular-arc graphs and permutation graphs. The routing schemes are compact, efficient and optimal for interval and circular-arc graphs and almost optimal for permutation graphs. As a byproduct, for proper interval graphs and proper circular-arc graphs, we obtained best possible routing schemes. In designing our routing schemes we essentially used the intersection models of those families of graphs.

Few questions remain open. Among them:
(1) Do those three families of graphs admit optimal routing schemes with $O(log\ n)$ bit labels and $O(1)$ forwarding protocols? Or size of local routing tables has always to depend on degrees of vertices(?).
(2) Can presented routing scheme for permutation graphs be improved to route messages via shortest paths only?
(3) Which other families of intersection graphs admit efficient routing schemes? One can consider trapezoid graphs, circle graphs, intersection graphs of discs on the plane, and others.

References

[1] C. Gavoille and S. Pérennès, Memory requirements for routing in distributed networks, *Proc. 15th Annual ACM Symposium on Principles of Distributed Computing*, Philadelphia, 1996, 125–133.

[2] M. Thorup and U. Zwick, Approximate distance oracles, *Proc. 33^{rd} Ann. ACM Symp. on Theory of Computing*, 2001, 183–192.

[3] C. Gavoille and M. Gengler, Spaceefficiency of routing schemes of stretch factor three, *Journal of Parallel and Distributed Computing*, 61 (2001), 679–687.

[4] P. Fraigniaud and C. Gavoille, Interval routing schemes, *Algorithmica*, 21 (1998), 155–182.

[5] G.N. Frederickson and R. Janardan, Designing networks with compact routing tables, *Algorithmica*, 3 (1988), 171–190.

[6] J. van Leeuwen and R.B. Tan, Interval routing, *The Computer Journal*, 30 (1987), 298–307.

[7] L. Narayanan and S. Shende, Partial characterizations of networks supporting shortest path interval labeling schemes, *Networks*, 32 (1998), 103-113.

[8] N. Santoro and R. Khatib, Labeling and implicit routing in networks, *The Computer Journal*, 28 (1985), 5–8.

[9] C. Gavoille, A survey on interval routing schemes, *Theoretical Computer Science*, 245 (1999), 217–253.

[10] C. Gavoille, Routing in distributed networks: Overview and open problems, *ACM SIGACT News - Distributed Computing Column*, 32 (2001), to appear.

[11] D. Peleg, Distributed computing – A locality-sensitive approach, *Society for Industrial and Applied Mathematics (SIAM)*, Philadelphia, PA, 2000.

[12] M. Thorup and U. Zwick, Compact routing schemes, *Proc. 13^{th} Ann. ACM Symp. on Par. Alg. and Arch.*, 2001, 1–10.

[13] Y. Dourisboure and C. Gavoille, Improved Compact routing Scheme for Chordal Graphs, *Manuscript*, 2002.

[14] D.G. Corneil, S. Olariu, and L. Stewart, The ultimate interval graph recognition algorithm? (extended abstract). *Proc. 9^th Annual ACM-SIAM Symposium on Discrete Algorithms*, San Francisco, 1998, 175–180.

[15] R.M. McConnell, Linear-time recognition of circular-arc graphs, *Proc. 42^n d Annual IEEE Symposium on Foundations of Computer Science*, Las Vegas, 2001, 386–394.

[16] R.M. McConnell and J.P. Spinrad, Linear-time transitive orientation, *Proc. 8^th Annual ACM-SIAM Symposium on Discrete Algorithms*, New Orleans, 1997, 19–25.