Exact Distance Labelings Yield Additive-Stretch Compact Routing Schemes

Arthur Brady and Lenore Cowen

Department of Computer Science, Tufts University, Medford, MA, USA
{abrady,cowen}@cs.tufts.edu

Abstract. Distance labelings and compact routing schemes have both been active areas of recent research. It was already known that graphs with constant-sized recursive separators, such as trees, outerplanar graphs, series-parallel graphs and graphs of bounded treewidth, support both exact distance labelings and optimal (additive stretch 0, multiplicative stretch 1) compact routing schemes, but there are many classes of graphs known to admit exact distance labelings that do not have constant-sized separators. Our main result is to demonstrate that *every* unweighted, undirected n-vertex graph which supports an exact distance labeling with $l(n)$-sized labels also supports a compact routing scheme with $O(l(n) + \log^2 n/ \log\log n)$-sized headers, $O(\sqrt{n}(l(n)+\log^2 n/ \log\log n))$-sized routing tables, and an additive stretch of 6.
We then investigate two classes of graphs which support exact distance labelings (but do not guarantee constant-sized separators), where we can improve substantially on our general result. In the case of interval graphs, we present a compact routing scheme with $O(\log n)$-sized headers, $O(\log n)$-sized routing tables and additive stretch 1, improving headers and table sizes from a result of [1], which uses $O(\log^3 n/ \log\log n)$-bit headers and tables. We also present a compact routing scheme for the related family of circular arc graphs which guarantees $O(\log^2 n)$-sized headers, $O(\log n)$-sized routing tables and an additive stretch of 1.

1 Introduction

According to the usual representation of a graph, vertices are assigned labels (say 1 to n for an n-vertex graph) in an arbitrary way, in the sense that the names are just $\log n$ bit placeholders for the rows and columns of the adjacency matrix (or alternatively, the list of edges), used to encode the structure of the graph. Breuer and Folkman [2] introduced the problem of determining which classes of graphs support the assignment of short vertex labels (i.e. $O(\log n)$ or $O(\log^c n)$ bits, c constant) so that, given only the labels of vertices i and j, it can be inferred whether or not i and j are adjacent. David Peleg [3] introduced the notion of distance labelings by generalizing this question: that is, he asked if can we assign short labels to the vertices of a graph so that the *distance* between vertices i and j can be computed just from the labels of i and j.

Definition 1.1. *Given an undirected graph $G = (V, E)$, a* ***distance labeling on*** *G (over some string alphabet Σ) is a function $DL : V \longmapsto \Sigma^*$ which assigns a string to each vertex $v \in V$ in such a way that for any two vertices $v,\ w \in V$, a function dist can be computed so that $dist(DL(v),\ DL(w))$ is the distance between v and w in G. We refer to the string $DL(v)$ assigned to a vertex v as v's* ***distance label****, and we refer to the computation of $dist(DL(v),\ DL(w))$ as a* ***distance query****.*

Notice that our definition bounds neither the size of the distance labels nor the time required to compute the distance function (once the labels have been generated). For these structures to be of any practical use, we would prefer to find a distance labeling which guarantees small labels (say $O(\log^c n)$ bits for some small constant c) and fast computation times. Unfortunately, not all classes of graphs enjoy distance labelings which have these properties. For example, it's known [4] that any distance labeling on general graphs must use $\Theta(n)$-bit labels, and any distance labeling on planar graphs must use $\Omega(n^{1/3})$-bit labels. On the other hand, distance labelings meeting these criteria have been discovered for several well-studied classes of graphs. The distance labeling on trees given in [3] uses $O(\log^2 n)$-bit labels and answers distance queries in polylogarithmic time. In fact, it was shown in [4] that the size of the labels in this result matches the lower bound for any distance labeling on trees.

Much of the existing work on distance labelings for particular graph families has relied on the existence of *separators* for these families. It was shown in [4] that any n-node graph with a recursive separator of size $f(n)$ supports a distance labeling which uses labels of size $O(f(n) \cdot \log^2 n)$. Trees, series-parallel graphs, and graphs of bounded treewidth all have constant-sized separators, yielding distance labelings on these graph families which use $O(\log^2 n)$-bit labels. This idea was extended in [5] to construct a distance labeling using $O(g(n) \cdot \log n)$-bit labels for any graph with an $f(n)$-sized separator. In this case, $f(n)$ may be large, but $g(n)$ – which is derived from certain structural properties of the separator – may be quite small (see [5] for full discussion and details). The authors went on to demonstrate that the families of *interval graphs* and *permutation graphs* exhibit separators for which $g(n) = O(\log n)$, yielding distance labelings for these graphs with $O(\log^2 n)$-bit labels. The result for interval graphs was improved in [6], which gives a labeling scheme which guarantees $O(\log n)$-bit labels (for both interval and circular arc graphs), and the scheme of [7] improves upon the previous result for permutation graphs, also using $O(\log n)$-bit labels.

1.1 Compact Routing

Consider a communications network modeled as a connected undirected graph $G = (V, E)$, $|V| = n$, with network nodes represented as vertices and direct communications links represented as edges.

A *routing scheme* R is a distributed algorithm defined on G which guarantees that any vertex v can send a message M to any other vertex w (along some (v, w)-path P in G), using metadata stored in M along with information stored locally at each vertex along P.

We refer to the metadata stored in a message M as M's *header,* and to the local information stored at a vertex v as v's *routing table.* Given an input graph $G = (V, E)$, a routing scheme R must specify the construction of the routing table at each vertex $v \in V$, the construction of the header of any message M originating at a given source v and intended for a given destination w, and a deterministic forwarding function $F(table(x),\ header(M))$. This function, when given the information in the routing table of a vertex x and the information in M's header, possibly rewrites the header, then selects an edge adjacent to x along which to forward M.

F is known as R's *routing function.* Given a source vertex v, a destination vertex w and a message M, the sequence of vertices $\langle v = v_0,\ v_1,\ \ldots,\ v_k\rangle$ defined by successive applications of $F(table(v_i),\ header(M))$ must be such that k is finite and $v_k = w$. We refer to this (v, w)-path as the *route* P_{vw} from v to w with respect to R. If F is allowed to alter the header of a message at intermediate vertices between its source and destination (while maintaining bounds on the size of the header), we refer to R as supporting *writeable packet headers.* Our general result requires writeable packet headers; our schemes for interval graphs and circular arc graphs do not. All our results are in the *name-dependent* model of compact routing (see [8] for discussion), and hold for unweighted, undirected graphs.

Definition 1.2. *The* ***multiplicative stretch*** *of a routing scheme R with respect to a graph G is $\max\limits_{v,w \in V(G)} \frac{P_{vw}}{d(v,w)}$, where P_{vw} is defined as above and $d(v, w)$ is the length of a shortest (v, w)-path in G. The* ***additive stretch*** *of a routing scheme R with respect to a graph G is $\max\limits_{v,w \in V(G)} |P_{vw} - d(v, w)|$.*

Additive stretch has been studied in the context of graph spanners (cf. [9–11]), and is referred to as "deviation" in the work of [1, 12].

We assume that each vertex in G is arbitrarily assigned a unique $\log n$-bit **ID** $\in \{1 \ldots n\}$, and that these IDs are provided along with the input graph, although message headers will contain additional information, such as distance labels, so we are considering compact routing in the *name-dependent* model (cf. [8]). We consider the *fixed-port* routing model, in which each vertex v has locally assigned an arbitrary $\log n$-bit **port name** $port_v(w) \in \{1 \ldots d(v)\}$ to each of its adjacent edges (v, w), and that the retrieval of this port name is the only mechanism by which v can forward a message along edge (v, w).

The field of *compact routing* is concerned with creating routing schemes which minimize worst-case routing table size, header size and stretch. In particular, we would like header sizes to be polylogarithmic in n, table sizes to be sublinear, and stretch to be bounded by a constant.

Most work in compact routing to date has dealt only with *multiplicative* stretch. Our main result is the following theorem:

Theorem 1.3. *Let $G = (V, E)$ with $|V| = n$ have an exact distance labeling with $O(l(n))$-sized distance labels. Then there exists a compact routing scheme for G*

which uses $O(l(n)+\log^2 n/\log\log n)$-sized headers, $O(\sqrt{n}\cdot(l(n)+\log^2 n/\log\log n))$-sized routing tables, and has an ***additive*** *stretch of 6.*

We prove this theorem in Sect. 2. This immediately implies compact routing schemes for interval graphs and the related family of circular arc graphs [6], as well as for permutation graphs [7], with $O(\log^2 n/\log\log n)$-sized headers, $O(\sqrt{n}\cdot\log^2 n/\log\log n)$-sized routing tables and an additive stretch of 6, by plugging the known exact distance labeling schemes for these graphs into Theorem 1.3. We note that for interval graphs and permutation graphs, better schemes were already known: Dragan, Yan and Corneil [12] present compact routing schemes for these graphs with $O(\log n)$-bit routing tables and message headers, and an additive stretch of 2, while Dragan and Lomonosov [1] present a compact routing scheme for interval graphs with $O(\log^3 n/\log\log n)$-bit headers and tables, and an additive stretch of 1. In Sects. 3 and 4, we show that for interval graphs and circular arc graphs, we can exploit the structural properties of these graphs to do better. In Sect. 3, we introduce a compact routing scheme for interval graphs which guarantees an additive stretch of 1 and uses $O(\log n)$-bit headers and routing tables. In Sect. 4, we give a compact routing scheme for circular arc graphs which also guarantees an additive stretch of 1, using $O(\log^2 n)$-bit headers and $O(\log n)$-bit routing tables.

2 Proof of Theorem 1.3

Definition 2.1. *Given any graph $G = (V, E)$, the* ***square*** *$G^2 = (V, E')$ of G is the graph which places an edge between u and v if $d(u, v) \leq 2$ in G.*

Fact 2.2. [13, 14] *There exists an exact (i.e. additive stretch 0) compact routing scheme for n-vertex* ***trees*** *which uses $O(\log^2 n/\log\log n)$-bit headers and $O(\log^2 n/\log\log n)$-bit routing tables.*

2.1 Algorithm

Given a connected, unweighted, undirected n-vertex graph $G = (V, E)$, with vertices assigned arbitrary IDs $\in \{1 \dots n\}$, and a distance labeling $\langle DL,\ dist\rangle$ on G, which guarantees labels of size at most $l(n)$, greedily construct a set I by iteratively choosing vertices in G of degree $\geq \lceil\sqrt{n}\rceil$, in such a way that each chosen vertex v satisfies $d(v, i) \geq 3$ for each i already in I, until no more vertices can be chosen. Note that $0 \leq |I| \leq \sqrt{n}$, and that I is an independent set in G^2. Identify a single-source shortest-path tree T_i, spanning G, rooted at each vertex $i \in I$. Process each tree T_i according to the scheme referenced in Fact 2.2. We will refer to the header assigned to v by the scheme of Fact 2.2 in tree T_i as $treeHeader_i(v)$, and to the table assigned to v by this scheme as $treeTable_i(v)$.

For each vertex v and each tree T_i, store $treeTable_i(v)$ in v's routing table. We refer to the entire collection of these tree-routing tables at v as v's *local tree data.* We will refer to the fixed $\log n$-bit ID assigned to each vertex v in the input graph as $ID(v)$. All vertices store their own IDs and distance labels in their tables; in addition, for each vertex $v \in V$:

Case 1: $v \in I$. Distribute the storage of all pairs $\langle ID(w), treeHeader_v(w)\rangle$, for all vertices $v \neq w \in V$, across v's neighbors as follows: first, sort these n pairs by $ID(w)$ and store the result in a sequence S_v. Next, choose an arbitrary sequence N_v of $\lceil\sqrt{n}\rceil$ of v's neighbors. Partition the ordered list S_v into $\lceil\sqrt{n}\rceil$ blocks, each of size $\lceil\sqrt{n}\rceil$ (except possibly the last block, which can be smaller if $\sqrt{n}$ is not an integer), and store the k^{th} such block at v's neighbor $N_v[k]$. Also at $N_v[k]$, store $port_{N_v[k]}(v)$. We refer to this collection of data stored at $N_v[k]$ as $N_v[k]$'s *dictionary data.* At v, store a triple $\langle port_v(N_v[k]), ID(w_{start}), ID(w_{end})\rangle$ for each of v's $\lceil\sqrt{n}\rceil$ neighbors in N_v, where w_{start} and w_{end} are the first and last entries in $N_v[k]$'s dictionary data, respectively. We refer to this collection of triples stored at v as v's *dictionary index.*

Case 2: $v \notin I$ and $deg(v) \geq \lceil\sqrt{n}\rceil$. In this case, select some $i \in I$ for which $d(v, i) \leq 2$. (Such an i always exists because I was constructed to be maximal in G^2.) At v, store at most two port names, containing exact shortest-path routing information from v to i. Mark v "SUBORDINATE TO i." We refer to this collection of data as v's *local link data.*

Case 3: $deg(v) < \lceil\sqrt{n}\rceil$. At v, store $\langle ID(x), DL(x), port_v(x)\rangle$, for each neighbor x of v. We refer to this collection as v's *local neighbor data.*

The header for a message destined for a vertex w will contain four elements: $ID(w)$, $DL(w)$, an integer IN_TREE $\in \{0 \ \dots \ n\}$, and $treeHeader_v(w)$ for some tree T_v. Headers will initially contain only $ID(w)$ and $DL(w)$, with the other elements empty.

Routing from the current vertex v toward a destination vertex w proceeds iteratively according to several cases. If $deg(v) < \lceil\sqrt{n}\rceil$, we search v's local neighbor data for a neighbor v' of v such that $d(v, v') + d(v', w) = d(v, w)$, using the distance labels to compute this value, then route to v'. If $deg(v) \geq \lceil\sqrt{n}\rceil$ but $v \notin I$, we route (along at most two edges) to v's closest neighbor i in I using v's local link data. Lastly, if $v \in I$, we use v's dictionary index to determine which of v's neighbors has stored information for w in its block, route to that neighbor, retrieve $treeHeader_v(w)$, write it into w's message header, and route to w along T_v according to the scheme referenced in Fact 2.2.

2.2 Analysis

At any point in the iteration of the routing procedure, the header on a message destined for a vertex w contains at most $ID(w)$, $DL(w)$, one $\log n$-bit integer, and $treeHeader_v(w)$ for some $v \in I$, for a total header size of $O(l(n) + \log^2 n/ \log\log n)$.

If $deg(v) < \lceil\sqrt{n}\rceil$, then v's local neighbor data contains $O(\sqrt{n})$ triples, each containing a $\log n$-bit ID, a $\log n$-bit port name, and an $O(l(n))$-bit distance label; v's local tree data contains $O(\sqrt{n})$ tree tables assigned to v by the scheme of Fact 2.2, each of size $O(\log^2 n/ \log\log n)$; and lastly v also stores its own $\log n$-bit ID and its $O(l(n))$-bit distance label. If $deg(v) \geq \lceil\sqrt{n}\rceil$, then v may store a dictionary index containing $O(\sqrt{n})$ triples, each of size $O(\log n)$; local tree data of total size $O(\sqrt{n}\log^2 n/\log\log n)$; and v's own ID and distance label. Lastly, v (irrespective of $deg(v)$) may have been selected to store at most one block of dictionary data for some vertex in I, of size $O(\sqrt{n}\log^2 n/\log\log n)$. Taken together, this gives a worst-case routing table size of $O(\sqrt{n}\cdot(l(n)+\log^2 n/\log\log n))$.

Lemma 2.3. ***(Stretch)*** *The algorithm routes messages with maximum stretch OPT+6.*

Proof. When routing from v to w, if all intermediate vertices encountered are of degree $< \lceil\sqrt{n}\rceil$ (except possibly w), then each step routes along a shortest path from the current vertex to w, yielding an optimal route overall. If an intermediate vertex h of degree $\geq \lceil\sqrt{n}\rceil$ is encountered, we consider two cases:

Case 1: The first such vertex h is in I. In this case, we look up w's header in T_h, using at most two edges to route to some neighbor of h which stores dictionary data for w and back again, then continue along a shortest path from h to w in T_h. Since h was the first vertex of degree $\geq \lceil\sqrt{n}\rceil$ to be encountered and since at each previous step we used distance labels to route optimally toward w, we have $d(v,w) = d(v,h) + d(h,w)$. Since in this case our algorithm uses a route of length at most $d(v,h) + 2 + d(h,w)$, this case yields an additive stretch of at most 2.

Case 2: The first such vertex h is not in I. In this case, we route along at most two edges to h's closest neighbor $i \in I$, route to one of i's neighbors j to obtain tree-routing information for w, and route back along at most three edges through h (possibly fewer, if j or i is closer to w), continuing along some shortest path in T_i from i to w. Since $d(v,w) = d(v,h) + d(h,w)$, since $d(i,w) \leq d(h,w) + 2$, and since T_i is a shortest-path spanning tree rooted at i, we have $P_{vw} \leq d(v,h) + d(h,i) + d(i,j) + d_{T_i}(j,w) \leq d(v,h) + 2 + 1 + [d(i,w) + 1] = d(v,h) + d(i,w) + 4 \leq d(v,h) + [d(h,w) + 2] + 4 = d(v,h) + d(h,w) + 6 = d(v,w) + 6$, for a total of at most 6 more edges than an optimal route. □

3 Compact Routing on Interval Graphs

Our general result implies the existence of a compact routing scheme for interval graphs with $O(\log^2 n/\log\log n)$-sized headers, $O(\sqrt{n}\log^2 n/\log\log n)$-sized routing tables, and an additive stretch of 6. We now show that we can do substantially better for this family of graphs, proving the following:

Theorem 3.1. *Any (unweighted, undirected) interval graph supports a compact routing scheme with $O(\log n)$-bit headers, $O(\log n)$-bit tables and an additive stretch of 1.*

Definition 3.2. *Given a finite set H of closed intervals $v = [L(v), R(v)]$ on the real line, $|H| = n$, the **interval graph** $G = (V,E)$ of H is defined by assigning one vertex to each interval v, with an edge between every pair of vertices whose corresponding intervals intersect. For a vertex $v \in V$, we define **L(v) (R(v))** to* *be the left (right) endpoint of the interval in H which corresponds to v, which we will refer to as v's* ***reference interval****. For a set $J \subseteq V$, we define* ***L(J)*** *to be $\min_{v \in J} L(v)$ and* ***R(J)*** *to be $\max_{v \in J} R(v)$. The* ***reference set*** *of a set $J \subseteq V$ consists of the union on the real line of all the reference intervals of vertices in J, with overlapping intervals combined: that is, two elements in J's reference set (which correspond to two subsets of J) are distinct only if there is no edge between those two subsets in G. We say a vertex v (respectively, a set of vertices J)* ***spans*** *a given interval i if the reference interval of v (resp., the reference set of J) contains i.*

We will use the same variable to describe both a vertex in an interval graph and its reference interval on the real line, and we will similarly overload a single variable to represent both a vertex set in G and its reference set. We also only consider the case where there is no single vertex i whose reference interval spans the reference set for the whole graph, since compact routing with additive stretch 1 is otherwise simple: always route through i, since it is adjacent to every other vertex; the routing table at v contains only $port_v(i)$, and the header for w contains only $port_i(w)$.

Definition 3.3. *Given an interval graph $G = (V, E)$, choose a vertex $a \in V$ so that $L(a) = L(V)$ and $R(a)$ is maximum, and choose a vertex $b \in V$ so that $R(b) = R(V)$ and $L(b)$ is minimum. (Note that $a \neq b$ since we are assuming that no single vertex spans the reference set of V.) A* ***spine*** *of G is a sequence of vertices $S = \langle a = s_1, \ldots, s_k = b \rangle$ chosen so that $R(s_i)$ is maximum over all neighbors of s_{i-1}, for all $1 < i \leq k$, choosing $s_k = b$ as soon as it becomes available. Given an interval graph $G = (V, E)$, a spine S of G and a vertex $v \notin S$, we refer to the set of vertices in S adjacent to v as v's* ***landmarks****.*

Note that every $v \notin S$ has at least one and at most three landmarks; this follows from the fact that the reference set of S is equal to that of V, combined with the fact that S is a shortest path from a to b (cf. Lemma 3.5).

Given an interval graph $G = (V, E)$ with a spine S, consider a pair of vertices a and b where $R(a) < L(b)$, i.e. a is to the left of b on the real line. Consider a sequence $P = \langle a = p_0, p_1, \ldots, p_l = b \rangle$ describing a shortest path from a to b in G. Since $R(a) < L(b)$, there must be some vertex $p \in P$ for which $R(p) > R(a)$. Notice that p_1 is the first such vertex (if it weren't, we could delete vertices from P to form a shorter path). The following lemma is then immediate by induction:

Lemma 3.4. *For all $p_i \in P$, $0 \leq i < l$, $R(p_{i+1}) > R(p_i)$.* □

Lemma 3.5. *Let $T = \langle a = t_0, t_1, \ldots, t_k = b \rangle$ be a sequence constructed iteratively by setting t_0 to a, setting an index i to 0, and while t_i is not adjacent to b, setting t_{i+1} to be a neighbor of t_i for which $R(t_i)$ is maximum, incrementing i at each step. Then T is a sequence describing a shortest path from a to b.*

Proof. Let $P = \langle a = p_0, p_1, \ldots, p_l = b \rangle$ describe a shortest path from a to b. If $P = T$, there is nothing to prove. So assume $P \neq T$, and let p_i be the first vertex of P where $p_i \neq t_i$. Observe that $L(t_i) \leq R(p_{i-1})$, since t_i is adjacent to p_{i-1} and $R(p_{i-1}) \leq R(t_i)$ by construction. Since P describes a shortest path, p_{i-1} cannot be adjacent to p_{i+1}, so $R(p_{i-1}) < L(p_{i+1})$. Since (by Lemma 3.4) $R(p_i) < R(p_{i+1})$, and since p_i is adjacent to p_{i+1}, we have $L(p_{i+1}) \leq R(p_i)$. Since t_i was chosen from among the neighbors of p_{i-1} so that $R(t_i)$ was maximum, we have $R(p_i) \leq R(t_i)$. Taken together, this gives $L(t_i) \leq R(p_{i-1}) < L(p_{i+1}) \leq R(p_i) \leq R(t_i)$, meaning t_i is adjacent to p_{i+1}. We can thus substitute t_i for p_i in P to form a new sequence P' which also describes a shortest path from a to b; we can then perform such a substitution for every intermediate vertex $a \neq p_i \neq b$ of P, and since p_{l-1} is adjacent to b, we have that $|T| = k = l = |P|$. $\square$

Lemma 3.6. *If $v, w \notin S$ do not share a landmark, they are not adjacent.* $\square$

3.1 Algorithm

The header for each vertex $w \in S$ contains $L(w)$ and $R(w)$. The header for each vertex $w \notin S$ contains $L(w), R(w)$, and for each landmark x of w, $L(x), R(x)$, and $port_x(w)$. The routing table for any vertex v is constructed as follows: if $v \in S$, let a and b be its neighbors in S to the left and right respectively (if they both exist). Store $L(v), R(v), L(a), R(a), port_v(a), L(b), R(b)$, and $port_v(b)$. If $v \notin S$, store $L(v)$ and $R(v)$, and for each landmark x of v, store $L(x), R(x)$, and $port_v(x)$. Both v's table and header contain the (fixed) ID assigned in the input graph, plus an extra bit, signaling whether or not v is a member of S.
To route from v to w:

Case 1: $v, w \in S$. If v and w are adjacent, then use the information in v's routing table to route along $port_v(w)$ to w. Otherwise, compare the information in w's header (containing w's endpoints) with the information in v's table (containing v's endpoints) to determine whether w lies to the left or to the right of v on the real line, and route along S in the appropriate direction, using the adjacency information in v's routing table and in the tables of all intermediate vertices in S.

Case 2: $v, w \notin S$. If v and w share a landmark a, retrieve $port_v(a)$ from v's table and $port_a(w)$ from w's header, and route to w along those two edges. If v and w do not share a landmark, then they are not adjacent by Lemma 3.6. Say v is to the left of w on the real line. Let x be v's rightmost landmark, and let y be w's leftmost landmark. Retrieve $port_v(x)$ from v's routing table and route along that edge to x, then from x to y along S, and finally retrieve $port_y(w)$ from w's header, and route along that edge from y to w. If v is to the right of w on the real line, route analogously, reversing all directions.

Case 3: $v \in S, w \notin S$. If v is one of w's landmarks, retrieve $port_v(w)$ from w's header and route along that edge. If v is not one of w's landmarks, but is adjacent to one of w's landmarks x, route to x using the information in v's table, then to w using the information in w's header. Finally, if v is adjacent neither to w nor to any of w's landmarks, say that v lies to the left of w's landmarks' reference set. Route from v to w's leftmost landmark a, along S, using the routing tables of v and all intermediate vertices in S, then retrieve $port_a(w)$ from w's header and route along that edge. If v lies to the right of w's landmarks' reference set, route analogously, reversing all directions.

Case 4: $v \notin S, w \in S$. If w is one of v's landmarks, retrieve $port_v(w)$ from v's table and route along that edge. If w is not one of v's landmarks, but is adjacent to one of v's landmarks x, route to x using the information in v's table, then to w using the adjacency information in x's table. Finally, if w is adjacent neither to v nor to any of v's landmarks, say that w lies to the right of v's landmarks' reference set. Let x be v's rightmost landmark; route to x using the information in v's table, then along S to w using the routing tables of x and all intermediate vertices in S. If w lies to the left of v's landmarks' reference set, route analogously, reversing all directions.

3.2 Analysis

Despite the fact that we have defined interval graphs for sets of intervals on the real line, we assume for the sake of convenience that the endpoints of each reference interval are $O(\log n)$-bit integers; since we're dealing with finite sets, a straightforward scaling process can ensure that this is the case. Each header contains at most 12 $O(\log n)$-bit integers plus one bit, for a total asymptotic header size of $O(\log n)$. Each routing table also contains at most 12 $O(\log n)$-bit integers plus one bit, for a total asymptotic table size of $O(\log n)$.

Lemma 3.7. *Given an interval graph $G = (V, E)$, let I be the connected interval on the real line which is the reference set of V. Given any interval C which is a subset of I, let A be the set of vertices in V whose reference intervals contain $L(C)$, and let B be the set of vertices in V whose reference intervals contain $R(C)$. Let l denote the length of a shortest path between any vertex in A and any vertex in B. Then given the subsequence of S defined by $\langle s_0, s_1, \dots, s_k\rangle$, where s_0 is the vertex of S containing $L(C)$ for which $R(s_0)$ is maximum, and s_k is the vertex of S containing $R(C)$ for which $R(s_k)$ is minimum, $k \le l + 1$.*

Proof. Let I be defined as above, and let C be any closed interval contained in I, with $a = L(C)$ and $b = R(C)$. Let $P = \langle p_0, p_1, \dots, p_l\rangle$ be a path from some vertex p_0 containing a to some vertex p_l containing b such that $|P| = l$ is minimized. Let s_0 be the vertex of S containing a for which $R(s_0)$ is maximum, and let s_k be the leftmost vertex of S containing b, where $k = d(s_0, s_k)$.

Case 1: s_0 is adjacent to p_1, and p_{l-1} is adjacent to s_k. In this case, the result is immediate: the sequence $\langle s_0, p_1, , \dots, p_{l-1}, s_k\rangle$ represents a path from s_0 to s_k of length l, and the subsequence $\langle s_0, \dots, s_k\rangle$ of S represents a shortest path from s_0 to s_k, so $k \le l$.

Case 2: s_0 is not adjacent to p_1, but p_{l-1} is adjacent to s_k. In this case, notice that s_0 is adjacent to p_0 since they both contain a. The path represented by the sequence $\langle s_0, p_0, p_1, \dots, p_{l-1}, s_k\rangle$ is of length $l+1$, and since $d(s_0, s_k) = k$, we have that $k \le l + 1$.

Case 3: s_0 is adjacent to p_1, but p_{l-1} is not adjacent to s_k. Using an argument analogous to that of Case 2, we again have $k \le l + 1$.

Case 4: s_0 is not adjacent to p_1, and p_{l-1} is not adjacent to s_k. Since s_1 was chosen from among the neighbors of s_0 such that $R(s_1)$ was maximum, we know that since p_0 is a neighbor of s_0, $R(s_1) \geq R(p_0)$, hence s_1 is adjacent to p_1. Similarly, s_i is adjacent to p_i for all $0 \leq i \leq \min\{k, l\}$, and $\min\{k, l\} = l$, since l was chosen to be minimum. Consider s_l, which is adjacent to p_l. If $l = k$, there is nothing to prove. Otherwise, since s_{l+1} was chosen from among the neighbors of s_l so that $R(s_{l+1})$ was maximum, and since p_l is a neighbor of s_l, we have that $R(s_{l+1}) \geq R(p_l)$, so s_{l+1} must contain b, hence $k \leq l + 1$. □

Lemma 3.8. ***(Stretch)*** *Given any connected interval graph $G = (V, E)$, our algorithm routes messages between any two vertices $v, w \in V$ along a path P_{vw} of length $\leq d(v, w) + 1$, where $d(v, w)$ is the distance between v and w in G.*

Proof. **Case 1:** $v, w \in S$. Since we are routing along a shortest path between v and w, $|P_{vw}| = d(v, w)$.

Case 2: $v, w \notin S$. If v and w share a landmark, then $|P_{vw}| = 2$, and since $d(v, w) \geq 1$, $|P_{vw}| \leq d(v, w) + 1$. If v and w do not share a landmark, then assume without loss of generality that v is to the left of w on the real line. Let x be v's rightmost landmark, and let y be w's leftmost landmark. Since v is not adjacent to x's right-hand neighbor in S, $R(x) > R(v)$, and similarly $L(y) < L(w)$. Consider the interval $C = [R(x) + \epsilon, L(y) - \epsilon]$ on the real line, where ϵ is chosen small enough so that $R(x) + \epsilon$ lies between $R(x)$ and the next (greater) endpoint of any interval in S to the right of $R(x)$, and $L(y) - \epsilon$ lies between $L(y)$ and the next (lesser) endpoint of any interval in S to the left of $L(y)$. Let $I \subseteq V$ be the set of vertices of G which intersect C, and notice that I is a vertex cut of G separating v and w, so that $d(v, w) \geq d(v, I) + D_I + d(I, w)$, where D_I is the minimum length of a path from a vertex containing $L(C)$ to a vertex containing $R(C)$. Let $\langle s_0, s_1, \ldots, s_k \rangle$ be the sequence of vertices in S traversed by our algorithm between x and y, not including x and y. Since s_0 is the only vertex in S containing $R(x) + \epsilon$, s_0 must be the vertex in S containing $L(C)$ where $R(s_0)$ is maximum, and similarly s_k is the vertex in S containing $R(C)$ where $L(s_k)$ is minimum. By Lemma 3.7, then, $k \leq D_I + 1$. Since v is not adjacent to any vertex containing $R(x) + \epsilon$, $d(v, I) \geq 2$, and since w is not adjacent to any vertex containing $L(y) - \epsilon$, $d(I, w) \geq 2$. So we have $|P_{vw}| = 2 + k + 2 \leq d(v, I) + (D_I + 1) + d(I, w) \leq d(v, w) + 1$.

Case 3: $v \in S, w \notin S$. If v is one of w's landmarks, then $|P_{vw}| = d(v, w)$. If v is not one of w's landmarks, but is adjacent to one of w's landmarks, then $d(v, w) = 2$, and $|P_{vw}| = 2 = d(v, w)$. If neither of the above is the case, then we have $|P_{vw}| \leq d(v, w) + 1$, by an analogous argument to that of Case 2, with $C = [R(v) + \epsilon, L(y) - \epsilon]$, where w's leftmost landmark y is assumed without loss of generality to be to the right of v on the real line.

Case 4: $v \notin S, w \in S$. This case is proved by an identical argument to that of Case 3, with the interval C of the final part defined to be $[R(x) + \epsilon, L(w) - \epsilon]$, where v's rightmost landmark x is assumed without loss of generality to be to the left of w on the real line. □

4 Compact Routing on Circular Arc Graphs

Definition 4.1. *Given a circle C and a set A of n closed arcs which are subsets of C, the* ***circular arc graph*** *corresponding to A is the graph $G = (V, E)$ constructed by creating one vertex $v_a \in V$ for each arc a in A, with an edge $(v_a, v_b) \in E$ if the arcs a and b corresponding to v_a and v_b intersect. We refer to C as the* ***reference circle*** *of G, to A as the* ***arc set*** *of G, and to a as the* ***reference arc*** *of v_a. For any arc $a \in A$ which is a strict subset of C, we define* $\boldsymbol{L(a) = L(v_a)}$ *to be the counterclockwise endpoint of a, and similarly we define* $\boldsymbol{R(a) = R(v_a)}$ *to be the clockwise endpoint of a. For any set J of vertices of G, the* ***reference set*** *of J consists of the union (on the reference circle) of all the reference arcs of vertices in J, with overlapping arcs combined: that is, two elements in J's reference set (which correspond to two subsets of J) are distinct only if there is no edge between those two subsets in G. Note in particular that the reference set of V is not necessarily equal to the arc set of G. If the reference set of some subset $X \subseteq V$ of vertices of G is a single connected arc which is not equal to the entire reference circle C, we define* $\boldsymbol{L(X)}$ *to be the counterclockwise endpoint of the reference set of X, and* $\boldsymbol{R(X)}$ *to be its clockwise endpoint. We say a vertex v (respectively, a set of vertices J)* ***spans*** *a given closed arc a on C if the reference arc of v (resp., the reference set of J) contains a.*

We will use the same variable to describe both a vertex in a circular arc graph G and its reference arc on G's reference circle C, and we will similarly overload a single variable to represent both a vertex set in G and its reference set on C. We can consider only the case where there is no single vertex a whose reference arc spans the entire reference circle, since compact routing with additive stretch 1 in the other case is straightforward: always route through a, since it is adjacent to every other vertex; the routing table at v contains only $port_v(a)$, and the header for w contains only $port_a(w)$. Since each component of a circular arc graph whose vertex set does not span its entire reference circle is also an interval graph, and we already have a compact routing scheme for interval graphs, we will only consider circular arc graphs whose vertex sets completely span their reference circles.

Definition 4.2. *Given a circular arc graph G with reference circle C and arc set A, and given any point x on C, the* ***plug of G corresponding to x*** *is the subset of vertices of G whose reference arcs contain x.*

Lemma 4.3. *Given a circular arc graph G with reference circle C, if there exists a plug P of G (corresponding to a point x on C) such that the reference set of P is C itself, then there exists a set of exactly two vertices in P whose reference arcs also span all of C.*

Proof. Since we have ruled out the case in which one arc may span the entire reference circle, and the reference set of P is the entire reference circle C, there must be a set of *at least* two arcs in P which spans C. To show that there is such a set containing *exactly* two arcs, first note that every arc in P must contain x

by definition. Let a be the arc in P which covers the largest portion of C in the direction clockwise from x, and let b be the arc in P which covers the largest portion of C in the direction counterclockwise from x. If the reference set of $\{a, b\}$ is not equal to C, then P cannot span C, since both a and b were chosen as maximal. Hence a and b must together span C. $\square$

Note that since all vertices in a given plug intersect at x, a plug is also a clique. According to Lemma 4.3, if the reference set of some plug P is the entire reference circle C, then there exists a pair of vertices $a, b \in P$ which together span all of C, and so each vertex in G must be adjacent to a, b, or both. Finally, we use the fact that there exists an exact distance labeling scheme for interval graphs which uses $O(\log n)$-bit labels [6].

4.1 Algorithm

Let $G = (V, E)$ be an n-vertex circular arc graph, with reference circle C and arc set A.

Case 1: There exists some plug P of G for which the reference set of P equals all of C. In this case, let a and b be a pair of arcs of P spanning C whose existence is guaranteed by Lemma 4.3. The routing table of every vertex $v \in V$ will contain at most five elements: the ID assigned to v in the input graph, one bit indicating whether or not v is adjacent to a (and, if so, $port_v(a)$), and one bit indicating whether or not v is adjacent to b (and, if so, $port_v(b)$). The header of every vertex $w \in V$ will similarly contain at most five elements: the ID assigned to w in the input graph, a bit indicating whether or not w is adjacent to a (and, if so, $port_a(w)$), and a bit indicating whether or not w is adjacent to b (and, if so, $port_b(w)$).

Case 2: There is no plug of G whose reference set equals all of C. In this case, we select an arbitrary point x on C, and let P be the plug of G corresponding to x. Since the reference set of P does not equal C, and since P is connected in G, the reference set of P is a single closed arc which is a proper subset of C. Because of this, $G[V \setminus P]$ is an interval graph. Let a and b be chosen from P such that $L(a) = L(P)$ and $R(b) = R(P)$. (If one vertex satisfies both requirements, we choose that vertex; for clarity, our language throughout the following will assume that a and b are distinct.) Compute distance labels for all vertices in $G[V \setminus P]$ using the scheme given in [6]. The header and routing table for each vertex v in $V \setminus P$ will contain the distance label assigned to v by this scheme; the header and routing table for each vertex p in P will contain a bit marking p as a member of P. Next, identify two shortest-path trees T_a and T_b spanning $G[V \setminus P]$, rooted at a and b respectively. Preprocess these trees according to the compact tree-routing scheme referenced in Fact 2.2, and append the headers and routing tables generated by this scheme, for both trees, to the headers and routing tables, respectively, of every vertex in each tree. For each vertex $v \in V \setminus P$, add $d(v, a)$ and $d(v, b)$ to both the header and the routing table of v. Also for each vertex $v \in V \setminus P$, add the tree-routing headers of a and b, in T_a and T_b respectively, to v's local routing table. Preprocess $G[V \setminus P]$ according to the compact routing

scheme for interval graphs given in Sect. 3, and append the headers and tables generated by that scheme to the headers and tables, respectively, for all vertices in $V \setminus P$. For each vertex p in P, record $port_p(a)$, $port_p(b)$, $port_a(p)$ and $port_b(p)$ in p's header and routing table. Finally, both the header and routing table of every vertex v will contain the ID assigned to it in the input graph.
Routing now proceeds as follows:

Case 1: To route from v to w, first extract adjacency information for a and b from the routing table of v and the header of w. If both v and w are adjacent to one of these (say a), route from v to a using $port_v(a)$ from v's routing table, then from a to w using $port_a(w)$ in w's header. If this is not the case (say v is adjacent only to a, and w is adjacent only to b), route from v to a using $port_v(a)$ in v's routing table, then from a to b using $port_a(b)$ in a's routing table, then from b to w using $port_b(w)$ from w's header.

Case 2: If neither v nor w is in the plug P, compute d_1, the distance between v and w in $G[V \setminus P]$, using the distance labels in v's routing table and w's header. Next extract $d(v,a)$ and $d(v,b)$ from v's routing table, extract $d(w,a)$ and $d(w,b)$ from w's header, and compute $d_2 = \min\{d(v,a), d(v,b)\} + \min\{d(w,a), d(w,b)\} + 1$. If $d_1 < d_2$, route from v to w through $G[V \setminus P]$ using the scheme of Sect. 3. Otherwise, say v is closest to a and w is closest to b. We route from v to a along T_a, using the tree-routing header for a stored in v's routing table (and the tree-routing tables of all intermediate vertices), then from a to b using $port_a(b)$ in a's routing table, and finally from b to w along T_b using the information in w's tree-routing header for T_b (and the tree-routing tables at all intermediate vertices). (If the closer of a and b to v and w is the same (say a), then route as above from v to a and from a to w.) If only v is in P, use the information in w's header to compute the closer of a or b to w (say a), route to a using $port_v(a)$ in v's routing table, then to w along T_a using the information in w's tree-routing header (and the tree-routing tables at all intermediate vertices). If only w is in P, use the information in v's routing table to compute the closer of a or b to v (say a), route to a along T_a using the tree-routing header for a in v's routing table (and the tree-routing table at each intermediate vertex), then route to w using $port_a(w)$, stored in w's header. If both v and w are in P, use $port_v(a)$ in v's routing table to route from v to a, then use $port_a(w)$ from w's header to route from a to w.

4.2 Analysis

If Case 1 above holds, the header for each vertex contains at most two bits and two port names, for a header size of $O(\log n)$. If Case 2 above holds, then in the worst case, the header for a vertex v contains a $\log n$-bit ID, an $O(\log n)$-bit distance label in $G[V \setminus P]$, two $O(\log^2 n/ \log\log n)$-bit tree-routing headers for T_a and T_b, two $O(\log n)$-bit integers representing $d(v,a)$ and $d(v,b)$, and one $O(\log n)$-bit header from the interval graph scheme of Sect. 3, yielding a total header size of $O(\log^2 n/ \log\log n)$.

If Case 1 above holds, then the routing table for each vertex contains at most one $\log n$-bit ID, two bits and two edge labels, for a table size of $O(\log n)$. If Case

2 above holds, then in the worst case, the routing table for a vertex v contains a $\log n$-bit ID, an $O(\log n)$-bit distance label in $G[V \setminus P]$, two $O(\log^2 n/\log\log n)$-bit tree-routing tables for T_a and T_b, two $O(\log n)$-bit integers representing $d(v, a)$ and $d(v, b)$, and one $O(\log n)$-bit table from the interval graph scheme of Sect. 3, yielding a total routing table size of $O(\log^2 n/\log\log n)$.

Lemma 4.4. *If Case 2 above holds and a vertex v is not in the plug P, then $d(v, P) = \min\{d(v, a), d(v, b)\}$.*

Proof. $L(a) = L(P)$ and $R(b) = R(P)$ by definition, and $v \notin P$. Observe that the reference set of any shortest path from v to any vertex in P must be an arc containing $L(P)$ or $R(P)$, say $L(P)$. If p is the vertex of P on this path containing $L(P)$ and $p \neq a$, clearly we can substitute a for p to obtain a path of equal length. The case for the crossing of $R(P)$ is analogous. □

Lemma 4.5. ***(Stretch)*** *Our algorithm routes messages from any source v to any destination w with maximum stretch OPT+1.*

Proof. If Case 1 above holds and if v and w are both adjacent to one of a or b, we route along a path of length 2, and since $d(v, w) \geq 1$, this is $\leq OPT + 1$. If Case 1 above holds and v and w are not both adjacent to a or to b, then say for example that v is adjacent to a and w to b. Then since the reference arcs of a and b span the entire reference circle C, observe that the reference arcs for v and w can't intersect, and so v and w are not adjacent in G, so $d(v, w) \geq 2$. Since we then route along a path of length 3, we have a route of length $\leq OPT + 1$. If Case 2 above holds and v and w are both in the plug P, then $d(v, w) = 1$, and we route along a path of length $\leq 2 = OPT + 1$. If Case 2 above holds and if one of v and w is in the plug P (say w), then $d(v, w) \geq d(v, P)$. Since we route from v to the closer of a or b and then along at most one edge to w, by Lemma 4.4, we route along a path of length at most $d(v, P) + 1 \leq OPT + 1$. Finally, if Case 2 above holds and if neither v nor w is in P, we consider two sub-cases:

Case 2A: Every shortest path from v to w crosses P. In this case, $d(v, w) \geq d(v, P) + d(w, P)$. Since $d_2 = d(v, P) + d(w, P) + 1$ by Lemma 4.4, we have that $d_2 \leq d(v, w) + 1$. Since every shortest path between v and w crosses P, we also know that $d_1 > d(v, w)$, since it represents the length of a path which does not contain any vertices in P. If $d_1 < d_2$, then we have $d(v, w) < d_1 < d_2$, implying (since the quantities involved are integers) that $d_2 \geq d(v, w) + 2$, a contradiction. So it must be the case that $d_2 \leq d_1$, and according to our routing algorithm, we therefore route along a path of length at most $d_2 \leq OPT + 1$.

Case 2B: No shortest path between v and w crosses P. Then $d_1 = d(v, w)$, so d_2 can't be less than d_1, since d_1 is the length of a path between v and w and it's minimum. Also, $d_2 \neq d_1$, since d_1 is minimum, d_2 is the length of a path through P, and no shortest path crosses P by assumption. So $d_1 < d_2$, and we route through $G[V \setminus P]$ using the scheme given in Sect. 3, which uses a route of length at most $OPT + 1$. □

We have now proved the following: any circular arc graph supports a compact routing scheme with $O(\log^2 n/\log\log n)$-bit headers, $O(\log^2 n/\log\log n)$-bit tables and an additive stretch of 1.

We note that substituting alternate tree-routing schemes for the one mentioned in Fact 2.2 will produce different tradeoffs in table and header size. Substituting the alternate tree-routing scheme given in [14], which guarantees $O(\log^2 n)$-bit headers and $O(\log n)$-bit routing tables, achieves the bounds stated in the abstract and Sect. 1. Substituting the tree-routing scheme of [15], which guarantees $O(\log n)$-bit headers and $O(\min\{deg(v), \sqrt{n}\} \cdot \log n)$-bit routing tables at each vertex v, yields a compact routing scheme for circular arc graphs with a header size of $O(\log n)$ and a (much larger) table size of $O(\min\{deg(v), \sqrt{n}\} \cdot \log n)$ at each vertex v.

The authors gratefully acknowledge support from NSF grant CCR-0208629.

References

1. Dragan, F.F., Lomonosov, I.: On compact and efficient routing in certain graph classes. In: Proc. 15th Annual Symp. on Algorithms and Computation (ISAAC). (2004) 240–414
2. Breuer, M.A., Folkman, J.: An unexpected result on coding the vertices of a graph. J. Mathematical Analysis and Applications **20**(3) (1967) 583–600
3. Peleg, D.: Proximity-preserving labeling schemes and their applications. In: Proc. 25th Intl. Wkshp. on Graph-Theoretic Concepts in Computer Science (WG). (1999) 30–41
4. Gavoille, C., Peleg, D., Pérennes, S., Raz, R.: Distance labeling in graphs. J. Algorithms **53**(1) (2004) 85–112
5. Katz, M., Katz, N.A., Peleg, D.: Distance labeling schemes for well-separated graph classes. Discrete Applied Mathematics **145**(3) (2005) 384–402
6. Gavoille, C., Paul, C.: Optimal distance labeling for interval and circular-arc graphs. In: Proc. 11th Annual European Symp. on Algorithms. (2003) 254–265
7. Bazzaro, F., Gavoille, C.: Localized and compact data-structure for comparability graphs. In: Proc. 16th Annual Symp. on Algorithms and Computation (ISAAC). (2005) 1122–1131
8. Arias, M., Cowen, L., Laing, K., Rajaraman, R., Taka, O.: Compact routing with name independence. In: Proc. 15th Annual ACM Symp. on Parallel Algorithms and Architectures (SPAA). (2003) 184–192
9. Dor, D., Halperin, S., Zwick, U.: All pairs almost shortest paths. In: Proc. 37th Annual Symp. on Foundations of Computer Science (FOCS). (1996) 452–461
10. Elkin, M., Peleg, D.: $(1+\epsilon, \beta)$-spanner constructions for general graphs. SIAM J. Comput. **33**(3) (2004) 608–631
11. Baswana, S., Kavitha, T., Mehlhorn, K., Pettie, S.: New constructions of (α, β)-spanners and purely additive spanners. In: Proc. 16th Annual ACM-SIAM Symp. on Discrete Algorithms (SODA). (2005) 672–681
12. Dragan, F.F., Yan, C., Corneil, D.G.: Collective tree spanners and routing in at-free related graphs. In: Proc. 30th Intl. Wkshp. on Graph-Theoretic Concepts in Computer Science (WG). (2004) 68–80
13. Fraigniaud, P., Gavoille, C.: Routing in trees. In: Proc. 28th Intl. Colloq. on Automata, Languages and Programming (ICALP). (2001) 757–772
14. Thorup, M., Zwick, U.: Compact routing schemes. In: Proc. 13th Annual ACM Symp. on Parallel Algorithms and Architectures, ACM (2001) 1–10
15. Cowen, L.: Compact routing with minimum stretch. J. Algorithms **38**(1) (2001) 170–183