Distributed computing of efficient routing schemes in generalized chordal graphs *

Nicolas Nisse[1], Ivan Rapaport[2] and Karol Suchan[3,4]

[1]MASCOTTE, INRIA, I3S, CNRS, UNS, Sophia Antipolis, France.
[2]DIM, CMM (UMI 2807 CNRS), Universidad de Chile, Santiago, Chile.
[3]Facultad de Ingeniería y Ciencias, Universidad Adolfo Ibáñez, Santiago, Chile.
[4]Faculty of Applied Mathematics, AGH - University of Science and Technology, Cracow, Poland.

Abstract

Efficient algorithms for computing routing tables should take advantage of the particular properties arising in large scale networks. There are in fact at least two properties that any routing scheme must consider: low (logarithmic) diameter and high clustering coefficient.

High clustering coefficient implies the existence of few large induced cycles. Therefore, we propose a routing scheme that computes short routes in the class of k-chordal graphs, i.e., graphs with no chordless cycles of length more than k. We study the tradeoff between the length of routes and the time complexity for computing them. In the class of k-chordal graphs, our routing scheme achieves an additive stretch of at most $k-1$, i.e., for all pairs of nodes, the length of the route never exceeds their distance plus $k-1$.

In order to compute the routing tables of any n-node graph with diameter D we propose a distributed algorithm which uses messages of size $O(\log n)$ and takes $O(D)$ time. We then propose a slightly modified version of the algorithm for computing routing tables in time $O(\min\{\Delta D, n\})$, where Δ is the the maximum degree of the graph. Using these tables, our routing scheme achieves a better additive stretch of 1 in chordal graphs (notice that chordal graphs are 3-chordal graphs). The routing scheme uses addresses of size $\log n$ bits and local memory of size $2(d-1)\log n$ bits per node with degree d.

Keywords: Routing scheme, stretch, chordal graph, distributed algorithm.

1 Introduction

In any distributed communication network it is important to deliver messages between pairs of processors. Routing schemes are employed for this purpose. A routing scheme is a distributed algorithm that directs traffic in a network. More precisely, any source node must be able to route messages to any destination node, given the destination's network identifier. When investigating routing schemes, several complexity measures arise. On one hand, it is desirable to use as short paths as possible for routing messages. The efficiency of a routing scheme is measured in terms of its *multiplicative stretch factor* (resp., *additive stretch factor*), i.e., the maximum ratio (resp., difference) between the length of a route computed by the scheme and that of a shortest path connecting the same pair of nodes. On the other hand, as the amount of storage at each processor is limited, the routing information stored in the processors' local memory, the *routing tables*, must not require too much space with respect to the size of the network. Last but not

*Partially supported by programs Fondap and Basal-CMM (I.R. and K.S.), Fondecyt 1090156 (I.R.) and the European project IST FET AEOLUS (N.N.).

least, because of the dynamic character of networks, it is important to be able to compute the routing information in an efficient distributed way. While many works propose good tradeoffs between the stretch and the size of routing tables, the algorithms that compute those tables are often impracticable because they are centralized algorithms or because of their time-complexity. Indeed, in the context of large scale networks like social networks or Internet, even polynomial time algorithms are inefficient. In this paper, we focus on the tradeoff between the length of the computed routes and the time complexity of the computation of routing tables.

One way to design efficient algorithms in large scale networks consists in taking advantage of their specific properties. In particular, they are known to have low (logarithmic) diameter and to have high clustering coefficient. Therefore, their chordality (the length of the longest induced cycle) is somehow limited (e.g., see [Fra05]). That is why, in this paper, we focus on the class of k-chordal graphs. A graph G is called *k-chordal* if it does not contain induced cycles longer than k. A 3-chordal graph is simply called *chordal*. This class of graphs received particular interest in the context of compact routing. Dourisboure and Gavoille proposed routing tables of at most $\log^3 n/\log\log n$ bits per node, computable in time $O(m + n\log^2 n)$, that give a routing scheme with additive stretch $2\lfloor k/2\rfloor$ in the class of k-chordal graphs [DG02]. Also, Dourisboure proposed routing tables computable in polynomial time, of at most $\log^2(n)$ bits, but that give an additive stretch $k+1$ [Dou05]. Using a Lexicographic Breadth-First Search (Lex-BFS) ordering (resp., BFS ordering) of the vertices, Dragan designed a $O(n^2)$-time algorithm to approximate the distance up to an additive constant of 1 (resp., up to $k-1$) between all pairs of nodes of any n-node chordal graph (resp., k-chordal graph) [Dra05]. All these time results consider the centralized model of computation.

In this paper we propose a simpler routing scheme which, in particular, can be quickly computed in a distributed way and achieves good additive stretch for k-chordal graphs. However, the simplicity comes at a price of $O(\log n)$ bits per port needed to store the routing tables.

Distributed Model. An interconnection network is modeled by a simple undirected connected n-node graph $G = (V, E)$. In the following, D denotes the diameter of G and Δ denotes its maximum degree. The processors (nodes) are autonomous computing entities with distinct identifiers of size $\log n$ bits. We consider an all-port, full-duplex, $O(\log n)$ bounded message size, synchronous communication model. That is, any processor is able to send (resp., receive) different messages of size $O(\log n)$ to (resp., from) each of its neighbors in one communication step; links (edges) are bidirectional.

Our results. We present a simple routing scheme using a relabeling of the vertices based on a particular BFS-tree. Using a Strong BFS-tree, our algorithm achieves an additive stretch $k-1$ in the class of k-chordal graphs, and using a Maximum Neighborhood BFS-tree (Max-BFS-tree), it achieves an additive stretch 1 in the class of chordal graphs. It uses addresses of size $\log n$ bits and local memory of size $2(d-1)\log n$ bits per node of degree d. More precisely, each node must store an interval ($2\log n$ bits) per port, except for one port.

The stretches we achieve equal the best ones obtained in previous works. But our algorithm is a (simple) distributed one. It uses messages of size $O(\log n)$ bits. It computes a relabeling of the vertices and the routing tables in time $O(D)$ when a Strong BFS-tree is used, and in time $O(\min\{\Delta D, n\})$ when a Max-BFS-tree is used.

In the class of chordal graphs, our results simplify those of Dragan since a Lex-BFS ordering is more constrained than a Max-BFS ordering. In particular, the design of a distributed algorithm that computes a Lex-BFS ordering of the vertices of any n-node graph G in time $o(n)$ is an open problem even if G has small diameter and maximum degree.

Related work. Two kinds of routing schemes have been studied. In the *name-independent* model, the designer of the routing scheme has no control over the node names (see, e.g., [PU89, GP96, GG01]). Here we focus on *labeled routing*, where the designer of the routing scheme is free to name the nodes with labels containing some information about the topology of the network, the location of the nodes in the network, etc. In this context, a routing scheme with multiplicative stretch $4k-5$, $k \geq 2$, and using $\tilde{O}(n^{1/k})$ bits per node[1] in arbitrary graphs is designed in [TZ01]. In the case of trees, optimal labeled routing schemes using $\tilde{O}(1)$ bits per node have been proposed in [FG01, TZ01]. In [FG01], it is shown that any optimal routing scheme using addresses of $\log n$ bits requires $\Omega(\sqrt{n})$ bits of local memory-space. Several network classes have been studied, like planar graphs [Tho04], graphs with bounded doubling dimension [AGGM06], graphs excluding a minor [AG06], etc.

A particular labeled routing scheme is *interval routing*. Defined in [SK85], interval routing has received particular interest [Gav00]. In such a scheme, the nodes of the network are labeled using integers, and outgoing arcs in a node are labeled with a set of intervals. The set of all the intervals associated to all the outgoing edges of a node forms a partition of the name range. The routing scheme consists in sending the message through the unique outgoing arc labeled by an interval containing the destination's label. The complexity measure is the maximum number of intervals used in the label of an outgoing arc. An asymptotically tight complexity of $n/4$ intervals per arc in an n-node network is given in [GP99]. Moreover, almost all networks support an optimal interval routing scheme using at most 3 intervals per outgoing link [GP01]. Specific graph classes have been studied in this context (e.g., bounded treewidth graphs [NN98]).

2 Generalities on BFS-orderings and BFS-trees

In the following, $G = (V, E)$ denotes a connected n-node graph. Let $H = (V(H), E(H))$ be a subgraph of G, i.e., $V(H) \subseteq V$ and $E(H) \subseteq \{\{u, v\} \in E \mid u, v \in V(H)\}$. $d_H(x, y)$ denotes the distance in H between $x, y \in V(H)$. $N_H(x)$ denotes the neighborhood of $x \in V(H)$ in H. The length $|P|$ of a path P is its number of edges. A vertex $v \in V$ is *simplicial* if its neighborhood induces a clique. An ordering $\{v_1, \cdots, v_n\}$ on the vertices of G is called a *perfect elimination ordering* (PEO) if, for any $1 \leq i \leq n$, v_i is simplicial in G_i, where G_i is the graph induced by $\{v_i, \cdots, v_n\}$. In the context of a vertex ordering, we denote $w < v$ if w has a smaller index in this ordering. Note that in a PEO, if $z < w < v$, $\{z, w\} \in E$ and $\{z, v\} \in E$, then $\{w, v\} \in E$.

Theorem 1 *[FG65] A graph is chordal iff it admits a PEO.*

Let $r \in V$. A *Breadth-First Search* (BFS) ordering of G rooted at r is an ordering of its vertices such that r is the greatest vertex and, for any $u, v \in V(G) \setminus \{r\}$, $v < u$ implies that the greatest neighbor of u is greater than or equal to the greatest neighbor of v. A *Maximal Neighborhood Breadth-First Search* (MaxBFS) ordering of G rooted at r is a BFS ordering of its vertices with the following additional constraint: for any $u, v \in V(G) \setminus \{r\}$ with the same greatest neighbor, $v < u$ implies that the number of neighbors of u greater than u is at least the number of neighbors of v greater than u. The following theorem will be widely used.

Theorem 2 *[BKS05, CK] A graph G is chordal if and only if any MaxBFS ordering is a PEO.*

Given an ordering $\mathcal{O}$ of the vertices of G, the spanning tree *defined* by $\mathcal{O}$ is the spanning tree obtained by choosing for each vertex, but the root, its greatest neighbor as the parent. Such a tree defined by a BFS ordering (resp., by a MaxBFS ordering) will be called a Strong BFS-tree[2] (resp., MaxBFS-tree). Such a tree is rooted at the greatest vertex in the ordering.

[1] The notation $\tilde{O}()$ indicates complexity similar to $O()$ up to polylogarithmic factors.

[2] The name BFS-tree is often used in distributed computing literature to denote any shortest paths tree. To emphasize the particular properties of BFS-trees that are used in this work, the authors decided to add "Strong" in the name, even though the BFS-trees found in many textbooks are Strong BFS-trees in this sense.

3 Routing Scheme using Strong BFS and MaxBFS

This section is devoted to presenting a simple routing scheme based on Strong BFS-trees. We prove that this scheme achieves a good additive stretch in k-chordal graphs, and an improvement of this routing scheme is provided for chordal graphs.

First, let us present some notation. Let T be a spanning tree of a graph G. Given $x, y \in V$, $T_{x \to y}$ denotes the path in T between x and y. When T is defined by some BFS ordering, for any $v, w \in V$, $v > w$ denotes that v has a bigger index than w in this ordering. When T is rooted at $r \in V(T)$, its vertices are partitioned into *layers*: the layer $\ell(v)$ of a vertex v corresponds to $d_G(v, r)$. Note that $\{u, v\} \in E \Rightarrow |\ell(u) - \ell(v)| \leq 1$. In this paper we consider rooted trees, so we may say that two vertices are in the same *branch* if their least common ancestor is equal to one of them. Finally, given a routing scheme $\mathcal{R}$, $Str(\mathcal{R}(G), xy)$ denotes the difference between the length of the path computed by $\mathcal{R}$ and the real distance in G between x and y. The (additive) stretch $Str(\mathcal{R}(G))$ of $\mathcal{R}$ in G corresponds to $\max_{x,y \in V} Str(\mathcal{R}(G), xy)$.

3.1 General Routing Scheme

Let G be a graph and T be any Strong BFS-tree of G. Roughly, the routing scheme we propose proceeds as follows to send a message from any source $x \in V(G)$ to any destination $y \in V(G)$. The message follows the path from x to y in T, but if at some step the message can go through an edge $e \in E(G) \setminus E(T)$ that leads to the branch of T containing y, then it will use this *shortcut*. More formally, our routing scheme $\mathcal{R}(G, T)$ is defined as follows.

```
If $x = y$, stop.
If there exists $w \in N_G(x)$ ancestor of $y$ in $T$, choose such a vertex $w$ minimizing $d_T(w, y)$;
Otherwise, choose the parent of $x$ in $T$.
```

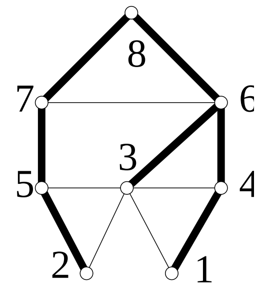
For instance, Figure 1 represents 3 graphs where the spanning trees are depicted with bold edges. In Figure 1(a), a message from 1 to 2 will follow the path $\{1, 4, 6, 7, 5, 2\}$. In Figure 1(c), the same message will follow $\{1, 4, 5, 2\}$. Let us make some simple remarks.

1. The routing scheme $\mathcal{R}(G, T)$ is well defined. Indeed, the message will eventually reach its destination since its distance to y in T is strictly decreasing at each step. Note that even if T is an arbitrary rooted spanning tree of G, the distance from y in T may increase at most once, if the message is passed from a descendant of y to an ancestor at a larger distance from y.

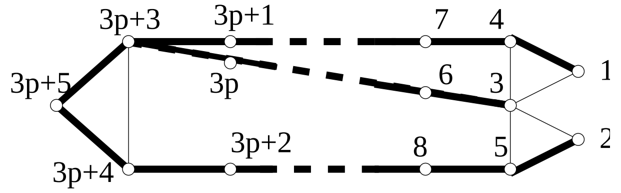
2. Once a spanning tree T rooted at an $r \in V(G)$ has been defined, this scheme can be efficiently implemented. It is sufficient to label the vertices such that any rooted subtree of T corresponds to a single interval. For any $u \in V(G)$ and any neighbor v of u but its parent, u stores the interval corresponding to the subtree of T rooted at v. Then, the routing function chooses the port corresponding to the inclusion-minimal interval containing the destination's address, and it chooses the parent of the current location if no such interval exists. Note that this is not an interval routing scheme because some intervals may be contained in others.

3. Since we assume that T is a BFS-tree, it is easy to see that the route computed by the routing scheme $\mathcal{R}(G, T)$ between two arbitrary nodes contains at most one edge that is not an edge of T. Indeed, after having taken such a *shortcut*, the message reaches y by following the path in T, which is a shortest path in G since T is a BFS-tree.

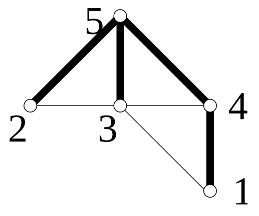
This section is devoted to proving the following theorem.



(a) 4-chordal graph and Strong BFS-tree.



(b) $2p+2$-chordal graph and MaxBFS-tree.



(c) Chordal graph and MaxBFS-tree.

Figure 1: Lemmata 3 and 4 give optimal bounds.

Theorem 3 *Let $k \geq 3$ and let G be a k-chordal graph.*

- *Let T be* any *Strong BFS-tree of G. Then $Str(\mathcal{R}(G,T)) \leq k-1$.*
- *Let $k=3$ and let T be* any *MaxBFS-tree of G. Then $Str(\mathcal{R}(G,T)) \leq 1$.*
- *Both bounds are tight.*

3.2 Stretch in k-Chordal Graphs

Let $k \geq 3$ and let G be a k-chordal graph and T be a (rooted) Strong BFS-tree of G. Let $x, y \in V$ be an arbitrary source and destination, respectively. The proof is a case by case analysis to bound $Str(\mathcal{R}(G,T), xy)$. Let R_{xy} be the route from x to y computed by $\mathcal{R}(G,T)$. In the following, we compare the length of R_{xy} with the length of some shortest path between x and y in G. Several parts of the following discussion are depicted in Figure 2, where bold lines represent edges, thin lines represent paths belonging to T and dotted lines represent paths with edges not necessarily belonging to T.

3.2.1 Restriction w.l.o.g.

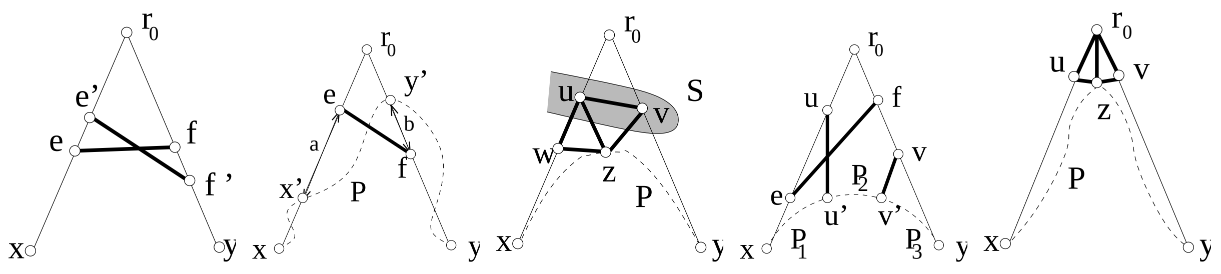
In this subsection, we prove that it is sufficient to consider x and y with a smallest common ancestor r_0 such that there is a shortest path P between x and y in G with no internal vertices of P in $V(T_{r_0 \to y}) \cup V(T_{x \to r_0})$.

If x is an ancestor or adescendant of y, R_{xy} is the path between x and y in T. Since T is a BFS-tree, this is a shortest path. From now on, we assume that x and y have a least common ancestor $r_0 \in V(G)$ distinct from x and y. By definition of $\mathcal{R}(G,T)$, R_{xy} either passes through r_0, or it uses an edge $\{e,f\} \in E(G) \setminus E(T)$ with $e \in V(T_{x \to r_0}) \setminus \{r_0\}$ and $f \in V(T_{r_0 \to y}) \setminus \{r_0\}$. I.e., the route R_{xy} from x to y is either $T_{x \to e} \cup \{e,f\} \cup T_{f \to y}$, or $T_{x \to r_0} \cup T_{r_0 \to y}$.

First, we need a technical lemma that shows that the roles of x and y are somehow symmetric.

Lemma 1 *$Str(\mathcal{R}(G,T), xy) = Str(\mathcal{R}(G,T), yx)$ and the computed routes are almost identical.*

Proof. Let R_{yx} be the route computed by $\mathcal{R}(G,T)$ from y to x. If R_{xy} passes through r_0, then R_{yx} does so, and $R_{xy} = R_{yx}$. If $R_{xy} \neq R_{yx}$, then R_{yx} must contain an edge $\{e', f'\} \in E(G) \setminus E(T)$ not $\{e,f\}$ (see Fig. 2(a)), and $e' \in V(T_{e \to r_0}) \setminus \{e\}$ and $f' \in V(T_{r_0 \to f}) \setminus \{f\}$. Because T is a Strong BFS-tree, e' is the parent of e and f is the parent of f'. Indeed, $d_G(r_0, f) < d_G(r_0, f') \leq d_G(r_0, e') + 1 \leq d_G(r_0, e) \leq d_G(r_0, f) + 1$. To conclude, if $R_{xy} \neq R_{yx}$, $R_{xy} = T_{x \to e} \cup \{e,f\} \cup \{f,f'\} \cup T_{f' \to y}$, and $R_{yx} = T_{y \to f'} \cup \{f', e'\} \cup \{e', e\} \cup T_{e \to x}$. ∎



(a) Lemma 1 (b) Lemma 2 (c) $r_0 \notin N(P)$ (d) Lem.3 $r_0 \notin N(P)$ (e) Lem.3 $r_0 \in N(P)$

Figure 2: Illustrations of the bounds of $Str(\mathcal{R}(G,T),xy)$.

Let P_0 be any shortest path in G between x and y. Let y' be the first vertex of P_0 in $V(T_{r_0 \to y})$, and x' be the last vertex of P_0, before y', in $V(T_{x \to r_0})$. Let P' be the subpath of P_0 between x' and y'. Because T is a Strong BFS-tree, $P = T_{x \to x'} \cup P' \cup T_{y' \to y}$ is a shortest path between x and y in G. The following technical lemma restricts our investigation to the case when P has no internal vertices in $V(T_{r_0 \to y}) \cup V(T_{x \to r_0})$.

Lemma 2 *Either* $Str(\mathcal{R}(G,T),xy) = 0$, *or* $Str(\mathcal{R}(G,T),xy) = Str(\mathcal{R}(G,T),x'y')$.

Proof. If $x' = y' = r_0$, then it is easy to see that $P = R_{xy}$. By definition, x' and y' must be both equal to or different from r_0. Therefore, let us assume both are different from r_0. Recall that $R_{xy} = T_{x \to e} \cup \{e,f\} \cup T_{f \to y}$, or $R_{xy} = T_{x \to r_0} \cup T_{r_0 \to y}$. In the second case, we set $e = f = r_0$. The proof is a case by case analysis according to the relative positions of x' and e in $T_{x \to r_0}$, and of y' and f in $T_{r_0 \to y}$.

If $T_{x \to e} \subseteq T_{x \to x'}$ and $T_{f \to y} \subseteq T_{y' \to y}$, then $Str(\mathcal{R}(G,T),xy) = 0$ because $|P'| \geq 1$.

Let us assume that $T_{x \to x'} \subset T_{x \to e}$ and $T_{f \to y} \subset T_{y' \to y}$. This case is illustrated in Figure 2(b). Note that in this case $e \neq f$, therefore $\{e,f\} \in E(G)$. Let $a = |T_{x \to e}| - |T_{x \to x'}| > 0$, and let $b = |T_{y' \to y}| - |T_{f \to y}| > 0$. We study the layers of x, x', y and y' to prove that $Str(\mathcal{R}(G,T),xy) = 0$. Let $L = \ell(e)$ be the layer of e. Then, $\ell(x') = L + a$. Because $\{e,f\} \in E(G)$ and T is a Strong BFS-tree, $L - 1 \leq \ell(f) \leq L + 1$. Therefore, $L - 1 - b \leq \ell(y') \leq L + 1 - b$. However, because T is a Strong BFS-tree and P' is a shortest path between x' and y', we must have $\ell(x') \leq \ell(y') + |P'|$. Thus, $\ell(x') \leq L + 1 - b + |P'|$. Finally, we get that $a + b - 1 \leq |P'|$. Since $b > 0$, then $a \leq |P'|$. To conclude, let us observe that $|Q| = |T_{x \to x'}| + a + 1 + |T_{y' \to y}| - b = |P| - |P'| + a + 1 - b \leq |P|$. Hence, $Str(\mathcal{R}(G,T),xy) = 0$.

If $T_{y \to y'} \subset T_{f \to y}$ and $T_{e \to x} \subset T_{x' \to x}$, we prove in a similar way that $Str(\mathcal{R}(G,T),yx) = 0$. By Lemma 1, we get that $Str(\mathcal{R}(G,T),xy) = 0$.

Finally, if $T_{x \to x'} \subseteq T_{x \to e}$ and $T_{y' \to y} \subseteq T_{f \to y}$, the route computed by $\mathcal{R}(G,T)$ from x' to y' is clearly $T_{x' \to e} \cup \{e,f\} \cup T_{f \to y'}$. Moreover, $Str(\mathcal{R}(G,T),xy) = |T_{x' \to e} \cup \{e,f\} \cup T_{f \to y'}| - |P'| = Str(\mathcal{R}(G,T),x'y')$. ■

3.2.2 When the computed route and the shortest path are vertex disjoint

It remains to consider the case when x and y have a smallest common ancestor r_0 such that there is an xy-shortest path P in G with no internal vertices of P in $V(T_{r_0 \to y}) \cup V(T_{x \to r_0})$ (cf. Figures 2(c), 2(d) 2(e)). Basically, the proof proceeds by considering the distances in the cycle $T_{x \to r_0} \cup T_{y \to r_0} \cup P$ and finding convenient chords in it.

Claim 1 *If* $r_0 \notin N_G(P)$, *there exist* u *in* $T_{x \to r_0}$, *and* v *in* $T_{y \to r_0}$, *such that* $u, v \in N_G(P)$. *Moreover, if* G *is chordal,* u *and* v *may be chosen adjacent:* $\{u,v\} \in E(G) \setminus E(T)$.

Proof. Since $r_0 \notin N_G(P)$, let C be the connected component of $G \setminus N_G(P)$ that contains r_0. Let $N = N_G(C)$. Clearly, $N \subseteq N_G(P)$ and there exists an inclusion-minimal separator $S \subseteq N$ separating r_0 from x and y. Let u (resp., v) be a vertex of S in the path between x (resp., y) and r_0 in T (see Fig. 2(c)). If G is chordal, S induces a clique since S is a minimal separator [Gol04], therefore $\{u, v\} \in E(G)$. Finally, $\{u, v\} \notin E(T)$ because the opposite would imply that u or v is the smallest common ancestor of x and y, i.e., $r_0 \in \{u, v\}$, a contradiction since $u, v \in S$. ■

Lemma 3 *Let $k \geq 3$. Let G be a k-chordal graph and let T be a spanning tree defined by any BFS ordering. Then, for any $x, y \in V(G)$, $Str(\mathcal{R}(G,T), xy) \leq k-1$.*

Proof. By the previous subsection, it remains to prove the following case: r_0 is the smallest common ancestor of x and y, and some shortest path P between x and y has no internal vertex in $T_{x\to r_0} \cup T_{r_0 \to y}$. Recall that, if the route R_{xy} computed by $\mathcal{R}(G,T)$ takes a shortcut, this edge is denoted $\{e, f\}$. There are two cases to be considered.

We first assume that r_0 is not in the neighborhood of P, $N_G(P)$ (cf., Figure 2(d)).

Let us choose u and v as defined in Claim 1 and such that $d_G(r_0, u) + d_G(v, r_0)$ is minimum. Let $u' \in N_G(u) \cap P$ and $v' \in N_G(v) \cap P$ such that $d_G(u', v')$ is minimum. Let $P = P_1 \cup P_2 \cup P_3$, where P_1 is the subpath of P between x and u', P_2 is the subpath of P between u' and v', and P_3 is the subpath of P between v' and y. In the following we assume that u' is between v' and x in P. Otherwise, the proof is similar by setting P_1 is the subpath of P between x and v', P_2 is the subpath of P between v' and u', and P_3 is the subpath of P between u' and y.

Because T is a Strong BFS-tree, $|T_{x\to u}| \leq 1 + |P_1|$ and $|T_{v \to y}| \leq 1 + |P_3|$.

- First, let us assume that $e \in T_{r_0 \to u}$ and $f \in T_{v \to r_0}$ (possibly $e = f = r_0$).

  In particular, this means that there are no edges between a vertex in $T_{e\to u}$ and $T_{v\to f}$ but $\{e, f\}$. Therefore, by the choice of u, u', v, v', the cycle $C = \{u, u'\} \cup P_2 \cup \{v', v\} \cup T_{v\to f} \cup \{e, f\} \cup T_{e \to u}$ has no chord. Thus, $|C| = 3 + |P_2| + |T_{v\to f}| + |T_{e\to u}| \leq k$.

  It follows that $|R_{xy}| = |T_{x\to u}| + |T_{u\to e}| + 1 + |T_{f\to v}| + |T_{v\to y}| \leq k - |P_2| + |P_1| + |P_3|$.

  If $|P_2| > 0$, $|R_{xy}| - |P| = Str(\mathcal{R}(G,T), xy) \leq k-1$.

  Therefore, let us consider the case when $|P_2| = 0$, i.e., $u' = v'$. We first consider the case when $u > v$ (in the BFS ordering defining T). We aim at proving that $|T_{v\to y}| \leq |P_3|$. For purpose of contradiction, let us assume that $|T_{v\to y}| = |P_3| + 1$. Let w_1 be the child of v in $T_{v\to y}$. For any $1 \leq i \leq |P_3| + 1$ and let u_i be the i^{th} vertex on the path P_3 ($u' = u_1$), and let w_i be the i^{th} vertex on the path $T_{w_1 \to y}$. Note that, because $|T_{w_1\to y}| = |P_3|$, $u_{|P_3|+1} = w_{|P_3|+1} = y$. Because u' is adjacent to u and $u > v$, it follows that $u' > w_1$. By a trivial induction on $i \leq |P_3|$, we get that, for any $1 \leq i \leq |P_3|$, $u_i > w_i$. Moreover, $w_{|P_3|}$ and $u_{|P_3|}$ are in the same layer, they are both adjacent to y and $u_{|P_3|} > w_{|P_3|}$. Hence, $\{w_{|P_3|}, y\}$ cannot belong to T, a contradiction.

  Therefore, $|T_{v\to y}| \leq |P_3|$. Hence, $|R_{xy}| = |T_{x\to u}| + |T_{u\to e}| + 1 + |T_{f\to v}| + |T_{v\to y}| \leq k - |P_2| + |P_1| + |P_3| - 1 = k + |P_1| + |P_3| - 1 \leq |P| + k - 1$.

  Now, let us consider the case when $u < v$. We prove that R_{yx} (the computed route from y to x) has length at most $|P| + k - 1$. By Lemma 1, it proves that $|R_{xy}| \leq |P| + k - 1$. If $R_{yx} = R_{xy}$, the proof is similar to the previous one (by symmetry). Therefore, let us assume that $R_{yx} \neq R_{xy}$. By Lemma 1, R_{yx} uses a shortcut $\{e', f'\} \in E(G) \setminus E(T)$ such that e' is the parent of e and f' is a child of f. If $e' \in T_{r_0\to u}$ and $f' \in T_{v \to r_0}$, again, the proof is similar to the previous one by symmetry. Hence, the only remaining case is $e' \in T_{r_0\to u} \setminus \{u\}$, f' is the child of $v = f$ (because of the relative positions of e, e', f, f', u, v). This case is similar (by symmetry) to the case treated in the third item of this proof.

- Second, let us assume that $e \in T_{x \to u} \setminus \{u\}$ and $f \in T_{v \to y}$ In this case, $|R_{xy}| = |T_{x \to e}| + 1 + |T_{f \to y}| \le |T_{x \to u}| + |T_{v \to y}| \le 2 + |P_1| + |P_3| \le |P| + k - 1$ (because $k \ge 3$).

  The case $e \in T_{x \to u}$ and $f \in T_{v \to y} \setminus \{v\}$ is symmetric. Indeed, in this case, consider $\{e', f'\}$ the shortcut used by R_{yx}. By Lemma 1, $f' \in T_{y \to v} \setminus \{v\}$ and $e \in T_{x \to u}$. Hence, applying the same proof to R_{yx}, $|R_{xy}| = |R_{yx}| \le |P| + k - 1$.

- Finally, let us assume that $e \in T_{x \to u} \setminus \{u\}$ and $f \in T_{v \to r_0} \setminus \{v\}$ (cf., Figure 2(d)). In this case, let us study the layers of e, f and y.

  First, $\ell(e) = \ell(u) + |T_{u \to x}| - |T_{e \to x}|$. Because $\{e, f\} \in E(G)$ and T is a Strong BFS-tree, $\ell(e) - 1 \le \ell(f) \le \ell(e) + 1$. Besides, $\ell(y) = \ell(f) + |T_{f \to v}| + |T_{v \to y}|$, and, because $P_2 \cup P_3$ is a shortest path, $\ell(y) \le \ell(u') + |P_2| + |P_3| \le \ell(u) + 1 + |P_2| + |P_3|$.

  Therefore, $\ell(u) + 1 + |P_2| + |P_3| \ge \ell(u) + |T_{u \to x}| - |T_{e \to x}| - 1 + |T_{f \to v}| + |T_{v \to y}|$. Hence, $2 + |P_2| + |P_3| \ge |T_{u \to x}| - |T_{e \to x}| + |T_{f \to v}| + |T_{v \to y}|$.

  Now, $|R_{xy}| = |T_{e \to x}| + 1 + |T_{f \to v}| + |T_{v \to y}| \le |P_2| + |P_3| - |T_{u \to x}| + 2|T_{e \to x}| + 3 = |P_2| + |P_3| + |T_{u \to x}| - 2(|T_{u \to x}| - |T_{e \to x}|) + 3 \le |P_1| + |P_2| + |P_3| + 2 \le |P| + k - 1$.

  Again, the case $e \in T_{u \to r_0} \setminus \{u\}$ and $f \in T_{v \to y} \setminus \{v\}$ is symmetric.

To conclude, let us assume that $r_0 \in N_G(P)$ (cf., Figure 2(e)). Let $z \in N_P(r_0)$. Let $P = P_1 \cup P_2$ where P_1 is the subpath of P between x and z, and P_2 is the subpath of P between z and y. Because T is a Strong BFS-tree, $|T_{x \to r_0}| \le 1 + |P_1|$ and $|T_{r_0 \to y}| \le 1 + |P_2|$. Therefore, $|R_{xy}| \le |T_{x \to r_0}| + |T_{r_0 \to y}| \le |P_1| + |P_2| + 2 \le |P| + k - 1$ (because $k \ge 3$). ■

It is important to note that the previous result is valid whatever be the Strong BFS-tree used. However, it is easy to observe that the inequality given by Lemma 3 is optimal. Indeed, Figure 1(b) represents a k-chordal graph with $k = 2p + 2$ ($p \ge 1$) and a Strong BFS-tree T (that actually is a MaxBFS-tree) such that $Str(\mathcal{R}(G, T))) = 2p + 1 = k - 1$: a message from 1 to 2 will pass through the edge $\{3p + 3, 3p + 4\}$.

Lemma 3 gives that, for any chordal graph G and for any Strong BFS-tree T, $Str(\mathcal{R}(G, T)) \le 2$. The following lemma proves that we can improve a bit the stretch in case of a chordal graph by using a "better" BFS-tree, i.e., a MaxBFS-tree.

Lemma 4 *Let G be a chordal graph and let T be a spanning tree defined by any MaxBFS ordering. Then, for any $x, y \in V(G)$, $Str(\mathcal{R}(G, T), xy) \le 1$.*

Sketch of the Proof. Due to lack of space, the proof is sketched and the full proof can be found in [NSR08]. Again, it only remains to consider the case when r_0 is the smallest common ancestor of x and y, and some x-y-shortest path P in G has no internal vertex in $T_{x \to r_0} \cup T_{r_0 \to y}$.

If $r_0 \notin N_G(P)$ (cf., Figure 2(c)), let u and v be defined as in Claim 1. $\{u, v\} \in E(G)$ because G is chordal. Hence, we prove that u and v have a common neighbor z in P. If $z \in \{x, y\}$, we prove that either $|T_{v \to y}| \le |P|$ or x is adjacent to the child of v in $T_{v \to y}$, and in both cases $Str(\mathcal{R}(G, T), xy) \le 1$. Otherwise, let P_1 be the subpath of P between x and z, and let P_2 be the subpath of P between z and y. By symmetry, w.l.o.g., assume $u > v$ (otherwise, the same proof holds for R_{yx}). Because $u > v$, we have $|T_{v \to y}| \le |P_2|$. Finally, we prove that either $|T_{x \to u}| \le |P_1|$, or $|T_{x \to u}| = |P_1| + 1$ and v is adjacent to the child w of u in $T_{u \to x}$. Indeed, if $|T_{x \to u}| = |P_1| + 1$, we prove that $\{w, z\} \in E(G)$ (by chordality) and $u > v > w > z$. $\{v, w\} \in E(G)$ follows because we consider a MaxBFS ordering and by Theorem 2. It is then easy to conclude because $|R_{xy}| \le |T_{x \to u}| + 1 + |T_{v \to y}|$.

If $r_0 \in N_G(P)$, the proof shows that either $|T_{x \to r_0}| < |P_1| + 1$ or $|T_{r_0 \to y}| < |P_2| + 1$. ■

It is easy to observe that the inequality given by Lemma 4 is optimal. Indeed, consider Figure 1(c) and the route between 1 and 2. The above discussion and lemmata prove Theorem 3.

4 Distributed algorithm

In this section, we present a simple distributed algorithm that computes the routing tables sufficient for the execution of the routing scheme described in the previous section. For space restrictions, let us give just an informal description. The algorithm consist of three phases. The first two of them aim at building a Strong BFS-tree T. Then, during the third phase each vertex x is assigned an integral label $P(x)$ that corresponds to its position in a DFS postorder traversal of T. It is easy to check that it gives a Strong BFS-ordering of G. Moreover, x learns $I(x)$, the interval corresponding to the labels of its descendants in T (including x). $P(x)$ is used as the identifier of x in the routing scheme. At each vertex y, for every neighbor x of y (except for the parent of y), the edge yx is labeled with $I(x)$. Let us describe the algorithm in detail.

1^{st} **Phase.** The first phase chooses an arbitrary vertex $r \in V(G)$ as the root and gives to each vertex its *layer*, i.e. its distance from r. Moreover, each vertex informs its neighbors of its own layer. This trivially takes at most $D + 1$ steps by broadcasting a counter initially set to 1 by the root. Now, if each vertex chooses an arbitrary neighbor in the lower layer as the parent, the obtained graph is a BFS-tree. However, as soon as Strong BFS-trees or MaxBFS-tree are concerned, not any neighbor in the lower layer can be chosen as the parent.

2^{nd} **Phase.** The second phase aims at determining an appropriate parent for each vertex. For this purpose, we assign an ordering on the vertices based on the following labeling: the root receives an empty label and any vertex $v \in V(G)$ in the layer $i \geq 1$ will eventually have a full label $label(v)$, where $label(v)$ is a sequence of i integers that consists of the full label of its parent u concatenated with the integer p that indicates that v is the p^{th} child of u. The labels will be constructed gradually, in a way that each vertex will be aware of the current (partial) labels of its neighbors. Notice that the lexicographic ordering of full labels gives the inverse of the Strong BFS-ordering (or MaxBFS-ordering) under construction. Transforming it into integer numbers ranging from n down to 1 can be easily computed once we have fixed T and ordered the children of each node (see the third phase).

To see how the algorithm assigns full labels, let us assume that, at some step, each vertex in layers up to $i - 1$ has received his full label as defined previously. Moreover, assume that each vertex in layer i knows the full labels of its neighbors in layer $i - 1$ and the partial labels of its neighbors in layer i. In particular, each vertex v in layer i knows its neighbor in layer $i - 1$ with the smallest label in the lexicographical ordering. So v can choose this node as its parent and inform all its neighbors of its choice. Once each vertex in layer i has chosen a parent, the vertices in layer $i - 1$ establish an ordering on their children: any vertex u in layer $i - 1$ sends an integer p_v to its child v so that v knows it is the p_v^{th} child of u (see below). This implies that the induction condition holds for layers i and $i + 1$. Notice that at layers 0 and 1 the condition trivially holds, since layer 0 contains a single vertex, the root r, with an empty label.

Spreading of labels. Let us describe how to spread the labels of the vertices efficiently. For any $i \geq 1$, each vertex v in the layer i maintains a subset $PP(v)$ (for potential parent) of its neighbors in layer $i - 1$ that initially contains all these neighbors. Once a vertex v in layer i has received a label with integer p_v (that corresponds to its position among its siblings), it transmits the pair (i, p_v) to all its neighbors. Once a vertex v in the layer $j > i$ has received such a message (i, p) from all of its neighbors u in $PP(v)$, it keeps in $PP(v)$ the vertices that have the smallest p. Then, v transmits the corresponding pair to all its neighbors. Moreover, receiving such a message from any neighbor u', v adds p to the locally stored (partial) label of u'. Proceeding in this way, once any vertex v in layer i has received a label, any vertex in layer $i + 1$ knows its potential parents, i.e., its neighbors in $PP(v)$, and the corresponding label.

Ordering of children in Strong BFS-tree. Once each vertex in layer i has chosen a parent, the vertices in layer $i-1$ establish an ordering on their children. If we want to obtain a Strong BFS-tree without additional properties, any ordering is valid. Therefore, each vertex in the layer $i-1$ arbitrarily orders its children and sends them their position in this ordering. Then, each vertex in layer i has a full label. This takes one step per layer, i.e., this takes time $O(D)$ in total.

Ordering of children in MaxBFS-tree. In this case, each vertex in layer $i-1$ will order its children according to the number of neighbors with smaller labels they have. In other words, each vertex in layer $i-1$ orders its children according to the number of their neighbors that will have larger numbers in the final ordering. Notice that as soon as the vertices of layer i have chosen their parent and broadcasted them to their neighbors, a vertex v in layer i only needs to learn its position in the ordering relatively to its siblings in T. Therefore, children of different vertices from layer $i-1$ can be ordered in parallel.

A vertex u in layer $i-1$ orders its children as follows. Let us assume u has already ordered its first p neighbors ($p \geq 0$). These neighbors of u have full label while remaining neighbors of u only have partial label. u chooses its $p+1^{th}$ child v as the one with the greatest number of neighbors that either have a parent greater than u or that are siblings of v with a full label. v receives $p+1$ from u and completes its label (that becomes full). Then, v informs its siblings that it has received a full label, and each child of u updates the number of its neighbors that already have full label. In this way, u orders its $d(u)$ children in $O(d(u))$ time. So, in total, this step is executed in $O(\Delta)$ time per layer. Therefore, in at most $O(\min\{\Delta D, n\})$ steps, any vertex has chosen a unique vertex as its parent and the tree T rooted in r is well defined.

3^{rd} **Phase.** The third phase consists in assigning to each vertex v his position in the ordering and the interval of positions of vertices that belong to T_v, the subtree of T rooted in v. It is easy to do so by two stages, the first one consisting in propagation of messages from the leaves toward the root and the second one from the root toward the leaves. During the first stage, the leaves of T send 1 to their parents, and any vertex u with children $v_1, \cdots, v_r$ receives from v_i ($1 \leq i \leq r$) the number ℓ_i of vertices belonging to the subtree of T rooted in v_i and sends to its parent $1 + \sum_{i \leq r} \ell_i$. During the second stage, the root is assigned the position n and the interval $[1, \ldots, |V(G)|]$; each vertex v takes the last position in the interval it has received and partitions the rest into subintervals corresponding to each of its children. It is easy to check that the resulting ordering corresponds to a DFS postorder traversal of T. This phase takes at most $2D$ steps. The discussion of this section can be summarized with the following theorem.

Theorem 4 *In any n-node network G with diameter D and maximum degree Δ, the distributed protocol described above computes routing tables of $O(\Delta \log n)$ bits per node, for the execution of the routing scheme $\mathcal{R}(G,T)$. Our protocol is executed in time $O(D)$ if the desired tree T is an arbitrary Strong BFS-tree, and in time $O(\min\{\Delta D, n\})$ if T is a MaxBFS-tree.*

5 Open Problems

Many questions remain open in this study. In particular, is it possible to design a routing scheme achieving same stretch and time-complexity but using smaller routing tables? Which stretch can we achieve when few large cycles are allowed? Routing schemes in dynamic networks, i.e., when nodes are free to leave or to arrive in the network at any time, are needed. Fault-tolerant and self stabilizing algoritms to compute routing tables would be appreciated.

References

[AG06] I. Abraham and C. Gavoille. Object location using path separators. In *Proceedings of the 25th Annual ACM Symposium on Principles of Distributed Computing (PODC)*, pages 188–197, 2006.

[AGGM06] I. Abraham, C. Gavoille, A. V. Goldberg, and D. Malkhi. Routing in networks with low doubling dimension. In *Proceedings of the 26th IEEE International Conference on Distributed Computing Systems (ICDCS)*, page 75, 2006.

[BKS05] A. Berry, R. Krueger, and G. Simonet. Ultimate generalizations of lexbfs and lex m. In *Proceedings of the 31st International Workshop on Graph-Theoretic Concepts in Computer Science (WG)*, pages 199–213, 2005.

[CK] D. G. Corneil and R. Krueger. A unified view of graph searching. to ap. in SIAM J. Comput.

[DG02] Y. Dourisboure and C. Gavoille. Improved compact routing scheme for chordal graphs. In *Proceedings of the 16th International Conference on Distributed Computing (DISC)*, pages 252–264, 2002.

[Dou05] Y. Dourisboure. Compact routing schemes for generalised chordal graphs. *Journal of Graph Algorithms and Applications*, 9(2):277–297, 2005.

[Dra05] F. F. Dragan. Estimating all pairs shortest paths in restricted graph families: a unified approach. *J. Algorithms*, 57(1):1–21, 2005.

[FG65] D. R. Fulkerson and O. A. Gross. Incidence matrices and interval graphs. *Pacific J. Math*, 15:835–855, 1965.

[FG01] P. Fraigniaud and C. Gavoille. Routing in trees. In *Proceedings of the 28th Int. Colloquium on Automata, Languages and Programming (ICALP)*, pages 757–772, 2001.

[Fra05] P. Fraigniaud. Greedy routing in tree-decomposed graphs. In *Proceedings of the 13th Annual European Symposium on Algorithms (ESA)*, pages 791–802, 2005.

[Gav00] C. Gavoille. A survey on interval routing. *Theor. Comput. Sci.*, 245(2):217–253, 2000.

[GG01] C. Gavoille and M. Gengler. Space-efficiency for routing schemes of stretch factor three. *J. Parallel Distrib. Comput.*, 61(5):679–687, 2001.

[Gol04] M. C. Golumbic. *Algorithmic Graph Theory and Perfect Graphs.* 2004.

[GP96] C. Gavoille and S. Perennes. Memory requirements for routing in distributed networks (extended abstract). In *PODC*, pages 125–133, 1996.

[GP99] C. Gavoille and D. Peleg. The compactness of interval routing. *SIAM J. Discrete Math.*, 12(4):459–473, 1999.

[GP01] C. Gavoille and D. Peleg. The compactness of interval routing for almost all graphs. *SIAM J. Comput.*, 31(3):706–721, 2001.

[NN98] L. Narayanan and N. Nishimura. Interval routing on -trees. *J. Algorithms*, 26(2):325–369, 1998.

[NSR08] N. Nisse, K. Suchan, and I. Rapaport. Distributed computing of efficient routing schemes in generalized chordal graphs. Technical Report CMM-B-08/10-220, CMM, oct 2008. http://www-sop.inria.fr/members/Nicolas.Nisse/publications/distribRouting.pdf.

[PU89] D. Peleg and E. Upfal. A trade-off between space and efficiency for routing tables. *J. ACM*, 36(3):510–530, 1989.

[SK85] N. Santoro and R. Khatib. Labelling and implicit routing in networks. *Comput. J.*, 28(1):5–8, 1985.

[Tho04] M. Thorup. Compact oracles for reachability and approximate distances in planar digraphs. *J. ACM*, 51(6):993–1024, 2004.

[TZ01] M. Thorup and U. Zwick. Compact routing schemes. In *Proceedings of the 13th Annual ACM Symposium on Parallel Algorithms and Architectures (SPAA)*, pages 1–10, 2001.