

Compact Routing Schemes for Generalised Chordal Graphs

Yon Dourisboure

IIT - CNR, Italy
yon.dourisboure@iit.cnr.it

Abstract

In this paper, we show how to use the notion of layering-tree introduced in [5], in order to obtain polynomial time constructible routing schemes. We describe efficient routing schemes for two classes of graphs that include the class of chordal graphs. For k -chordal graphs, i.e., graphs containing no induced cycle of length greater than k , the routing scheme uses addresses and local memories of size $O(\log^2 n)$ bits per node, and the length of the route between all pairs of vertices never exceeds their distance plus $k + 1$ (deviation at most $k + 1$). For tree-length δ graphs, i.e., graphs admitting a tree-decomposition in which the diameter of any bag is at most δ , the routing scheme uses addresses and local memories of size $O(\delta \log^2 n)$ bits per node, and its deviation is at most $6\delta - 2$. Observe that for chordal graphs, for which $\delta = 1$ and $k = 3$, both schemes produce a deviation 4, with addresses and local memories of size $O(\log^2 n)$ bits per node.

Article Type	Communicated by	Submitted	Revised
regular paper	Alon Itai	October 2004	November 2005

1 Introduction

Delivering messages between pairs of processors is a basic activity of any distributed communication network. This task is performed using a routing scheme, which is a mechanism for routing messages in the network. The routing mechanism can be invoked at any origin node and be required to deliver a message to any destination node.

It is naturally desirable to route messages along paths that are as short as possible. Routing scheme design is a well-studied subject. The efficiency of a routing scheme is measured in terms of its *deviation* or in terms of its *stretch*. The deviation of a routing scheme is d if it guarantees that the length of the route between all pairs of vertices never exceeds their distance plus d . Similarly, the stretch is s if lengths of routes are bounded by distances multiplied by s . A straightforward approach to achieving the goal of guaranteeing optimal routes is to store a *complete routing table* in each node u in the network, specifying for each destination v the first edge (or an identifier of that edge, indicating the output port) along some shortest path from u to v . However, this approach may be too expensive for large systems since it requires $O(n \log d)$ memory bits for a node of degree d in an n -node network. Thus, an important problem in large-scale communication networks is the design of routing schemes that produce efficient routes and have relatively low *memory requirements*.

The routing problem can be presented as requiring to assign two kinds of labels to every node of a graph. The first is the *address* of the node, whereas the second label is a data structure called the *local routing table*. The labels are assigned in such a way that at every source node u and given the address of any destination node v , one can decide the output port of an edge outgoing from u that leads to v . The decision must be taken locally at u , based solely on the two labels of u and with the address label of v . In order to allow each intermediate node to proceed similarly, a *header* is attached to the message to v . This header consists either of the destination label, or of a new label created by the current node.

It was shown in a series of papers (see, e.g., [1, 2, 3, 4, 18, 21]) that there is a trade-off between the memory requirements of a routing scheme and the worst-case stretch factor it guarantees. In [18] it is shown that every routing strategy that guarantees a routing scheme of stretch s for every n -node graph requires $\Omega(n^{1+1/(2s+4)})$ bits in total, so $\Omega(n^{1/(2s+4)})$ for local routing tables, for some worst-case graphs. Stronger lower bounds hold for small stretch factors. In particular, any routing scheme of stretch s must use $\Omega(\sqrt{n})$ bits for some nodes in some graphs for $s < 5$ [20], $\Omega(n)$ bits for $s < 3$ [11, 13], and $\Omega(n \log n)$ bits for $s < 1.4$ [15]. More precisely, for $s = 1$ [15] showed that for every shortest path routing strategy and for all d and fixed $\epsilon > 0$ such that $3 \leq d \leq (1 - \epsilon)n$, there exists a graph of degree bounded by d for which $\Omega(n \log d)$ bits routing tables are required simultaneously on $\Theta(n)$ nodes, matching with the memory requirements of complete routing tables. All the lower bounds presented above assume that routes and addresses can be computed and optimized by the routing strategy in order to decrease the memory requirements.

If we insist on *chordal* graphs, namely the class of graphs containing no induced cycles of length greater than 3, no strategy better than complete routing tables is known for an optimal stretch factor $s = 1$. Nevertheless, in [8], it is shown that every chordal graph admits a routing scheme of deviation 2 with addresses and local memories of size $O(\log^3 n / \log \log n)$ bits per node.

Our contribution

In this paper, we show how to use the notion of layering-tree introduced in [5], in order to obtain polynomial time constructible routing schemes. Then, we describe efficient routing schemes for two classes of graphs which both include the class of chordal graphs.

The first generalisation of chordal graphs is based on the very rich concept of Tree-decomposition, introduced by Robertson and Seymour [19] and widely used to solve various graph problems. In particular, efficient algorithms exist for graphs having a tree-decomposition into subgraphs (or *bags*) of bounded size: for bounded *tree-width* graphs. The *tree-length* of a graph G is the smallest integer δ for which G admits a tree-decomposition into bags of diameter at most δ . It has been formally introduced in [9], and extensively studied in [7]. Chordal graphs are exactly the graphs of tree-length 1, since a graph is chordal if and only if it has a tree-decomposition in cliques (cf. [16]). AT-free graphs, permutation graphs, and distance-hereditary graphs are of tree-length 2. More generally, [14] showed that k -chordal graphs have tree-length at most $k/2$. However, there are graphs with bounded tree-length and unbounded chordality¹, like the wheel. So, the class of bounded tree-length graphs is larger than the class of bounded chordality graphs. For graphs of tree-length bounded by δ , we obtain a routing scheme of deviation $6\delta - 2$ with addresses and local memories of size $O(\delta \log^2 n)$ bits per node, moreover headers attached to the message are of size $O(\delta \log n)$ bits.

The second generalisation of chordal graphs for which our routing scheme improves the best previous one, is the class of k -chordal graphs, namely the class of graphs containing no induced cycles of length greater than k . For such graphs, the best previous routing scheme has a deviation $2 \lfloor k/2 \rfloor$ with addresses and local memories of size $O(\log^3 n / \log \log n)$ bits per node [10]. In this paper we show that by relaxing a bit the deviation, $k + 1$ instead of $2 \lfloor k/2 \rfloor$, it is possible to obtain a routing scheme with addresses and local memories of size $O(\log^2 n)$ bits per node, moreover headers attached to the message are of size $\Theta(\log^n)$ bits.

Observe that for chordal graphs, both schemes have a deviation 4, with addresses and local memories of size $O(\log^2 n)$ bits per node.

¹The *chordality* is the smallest k such that the graph is k -chordal.

2 Basic notions and notation

All graphs occurring in this paper are connected, finite and undirected. Let $G = (V, E)$ be any graph, let X, Y be two subsets of V and let u be vertex of G . Then, the distance in G between u and X , denoted $d_G(u, X)$ is: $d_G(u, X) = \min_{v \in X} d_G(u, v)$. Moreover, the distance in G between X and Y is: $d_G(X, Y) = \min_{u \in X} d_G(u, Y)$. A *shortest path spanning tree* T of a graph G is a rooted tree having the same vertex set as G and such that for every vertex u , $d_G(u, r) = d_T(u, r)$ where r is the root of T . In the following, we will use the standard notions of *parent*, *children*, *ancestor*, *descendant* and *depth* in trees. The *nearest common ancestor* between two vertices u, v in a tree T is denoted by $nca_T(u, v)$.

2.1 Layering-tree

In this paper all routing schemes we describe are strongly based on the notion of Layering-tree we define in this subsection. Let G be a graph with a distinguished vertex s . Then we partition $V(G)$ into *layers*: for every integer $i \geq 0$, $L^i = \{u \in V(G) \mid d_G(s, u) = i\}$. Then, each layer L^i is partitioned into $L_1^i, \dots, L_{p_i}^i$, such that two vertices stay in a same part if and only if they are connected by a path visiting only vertices at distance at least i from s . A *Layering-tree* of G , denoted LT, is the graph whose vertex set is the collection of all the parts L_j^i . In LT, two vertices L_j^i and $L_{j'}^{i'}$ are adjacent if and only if there exists $u \in L_j^i$ and $v \in L_{j'}^{i'}$, such that u and v are adjacent in G (see Figure 1 for an example). The vertex s is called the *source* of LT.

Lemma 1 [5] *For any graph G , LT is a tree computable in linear time.*

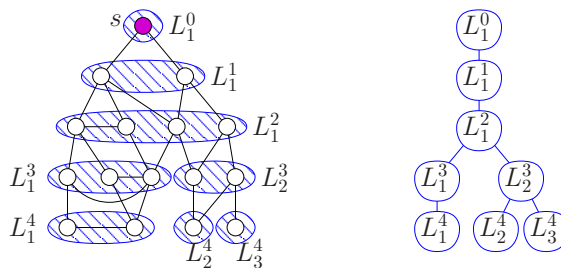


Figure 1: A graph G and a layering-tree of it.

Given a layering-tree LT of a graph G , for every vertex u we define the *part* of u : $\text{part}(u)$ that denotes the part of LT which contains u . By construction, one can prove that Layering-trees have this property:

Property 1 *Let LT be a layering-tree of a graph G , let u, v be two vertices of G , and let $X = nca_{\text{LT}}(\text{part}(u), \text{part}(v))$. Then, any path in G from u to v has*

to use at least one vertex of X . Moreover $d_G(u, v) \geq d_{LT}(\text{part}(u), \text{part}(v)) = d_S(u, X) + d_S(v, X)$

2.2 Hierarchical-tree

All routing schemes we describe in this paper need also the notion of hierarchical-tree we define hereafter. It is well known that every tree T has a vertex u , called *median*, such that each connected component of $T \setminus \{u\}$ has at most $\frac{1}{2}|V(T)|$ vertices. A *hierarchical-tree* of T is a rooted tree H defined as follows: the root of H is the median u of T and its children are the roots of the hierarchical trees of the connected components of $T \setminus \{u\}$. Observe that T and H share the same vertex set, and the depth of H is at most² $\log |V(T)|$. By construction of hierarchical-trees, one can prove the following property:

Property 2 *Let H be a hierarchical-tree of a tree T . Then let u, v be two vertices of T , then $\text{nca}_H(u, v)$ separates u and v in T .*

2.3 Tree-length δ graphs

The notion of tree-length we define hereafter will be useful in section 4. In their work on graph minors [19], Robertson and Seymour introduce the notion of *tree-decomposition*. A tree-decomposition of a graph G is a tree T whose nodes, called *bags*, are subsets of $V(G)$ such that:

1. $\bigcup_{X \in V(T)} X = V(G)$;
2. for all $\{u, v\} \in E(G)$, there exists $X \in V(T)$ such that $u, v \in X$; and
3. for all $X, Y, Z \in V(T)$, if Y is on the path from X to Z in T then $X \cap Z \subseteq Y$.

The *length* of a tree-decomposition T of G is $\max_{X \in V(T)} \max_{u, v \in X} d_G(u, v)$, and the *tree-length* of G is the minimum of the length, over all tree-decompositions of G .

A well-known invariant related to tree-decompositions of a graph G is the *tree-width*, defined as minimum of $(-1 + \max_{X \in V(T)} |X|)$ over all tree-decompositions T of G . We stress that the tree-width of a graph is not related to its tree-length. For instance cliques have unbounded tree-width and tree-length 1, whereas cycles have tree-width 2 and unbounded tree-length.

2.4 k -chordal graphs

The notion of k -chordal graphs we define hereafter will be useful in section 5. A graph G is *k -chordal* if the length of the longest induced cycle of G is at most k . This class of graphs is also discussed under the name *k -bounded-hole graphs* in [17]. Chordal graphs are 3-chordal graphs. The *chordality* of G is

²All the logs are in base two.

the smallest integer k such that G is k -chordal. Trees are, by convention, of chordality 2.

2.5 BFS-ordering and BFS-tree

These two notions we define hereafter, will be useful in the last optimisation in section 5. A *Breadth-First-Search-ordering* σ of a graph G is a function that orders vertices of G by assigning numbers from n to 1 in the following way. Let s any vertex of G , s is called the *source*, then s gets the number n . Then each next available number is assigned to the unnumbered vertex having the neighbor with the largest number.

A *BFS-tree* of G based on σ is the spanning tree of G rooted at the source of σ such that for any vertex u of G , the parent of u , $p(u)$, is the neighbor of u having the largest number in σ . Clearly a BFS-tree is a shortest path spanning tree of G , moreover for all vertices u, v , if $u >_{\sigma} v$ then either $p(u) >_{\sigma} p(v)$ or $p(u) = p(v)$.

3 The main routing scheme

In this section, G denotes an arbitrary graph, LT denotes a layering-tree of G , H denotes a hierarchical-tree of LT, and S denotes a shortest path spanning tree of G rooted at the source s of LT.

Let us outline the routing scheme. Each vertex u , contains in its address the information needed to route optimally in the tree S . In every part X of LT, we choose arbitrarily a vertex, r_X . For every part X which is an ancestor of $\text{part}(u)$ in H ($\text{part}(u)$ included), let us define $Z = \text{nca}_{LT}(\text{part}(u), X)$. Then the address of u contains all the information of a shortest path in G between the ancestor in S of r_X that belongs to Z , and the ancestor in S of u that belongs to Z . Observe that, since the depth of H is at most $\log n$, u contains the information of at most $1 + \log n$ paths.

To send a message from any vertex u to any other v , the solution we propose consists on finding the nearest common ancestor in H between $\text{part}(u)$ and $\text{part}(v)$: $X = \text{nca}_H(\text{part}(u), \text{part}(v))$. By Property 2, X belongs to the path in LT from $\text{part}(u)$ to $\text{part}(v)$. So, $Y = \text{nca}_{LT}(\text{part}(u), \text{part}(v))$ is either $\text{nca}_{LT}(\text{part}(u), X)$, or $\text{nca}_{LT}(\text{part}(v), X)$. Assume that $Y = \text{nca}_{LT}(\text{part}(u), X)$, then the route from u to v is depicted in Figure 2, where rescue_1 (Resp. rescue_2) is the path contained in the address of u (Resp. of v) associated to X . Moreover, by Property 1, we know that $d_G(u, v) \geq d_S(u, Y) + d_S(Y, X) + d_S(X, v)$, so the deviation is at most $\text{diam}_G(Y) + \text{diam}_G(X)$.

3.1 Description of labels

In this paper we consider that the outgoing edges of every vertex u of G , distinct integers called *output port numbers*, can be chosen from $[1, \text{deg}(u)]$. Our scheme associate to every vertex u of G two labels: its *address*, denoted by $\text{address}(u)$

and a *local routing table*, denoted by $\text{table}(u)$. The local routing table of u in G , $\text{table}(u)$, is set to $\langle \text{id}(u), \text{route}(u) \rangle$ (defined hereafter); The address of u in G , is set to $\langle \text{id}(u), \text{route}(u), \text{path}(u), \text{help}(u) \rangle$, where:

- $\text{id}(u)$ is the identifier of u (an integer in $\{1, \dots, n\}$);
- $\text{route}(u)$ is a binary label depending on the tree S and on u such that the route from u to any vertex v in S can be determined from the labels $\text{route}(u)$ and $\text{route}(v)$ only. More precisely, for a suitable computable function f (so independent of the tree), $f(\text{route}(u), \text{route}(v))$, for every $v \neq u$, returns the output port number of the first edge of the path from u to v in S . An implementation of these labels is discussed in Lemma 3.
- $\text{path}(u)$ is a binary label allowing to determine, given $\text{path}(u)$ and $\text{path}(v)$, the depth of the nearest common ancestor between $\text{part}(u)$ and $\text{part}(v)$ in H ;
- $\text{help}(u)$ is a table with $1 + \text{depth}_H(\text{part}(u))$ entries. Let X be an ancestor of $\text{part}(u)$ in H ($X = \text{part}(u)$ is possible), $\text{help}(u)[\text{depth}_H(X)] = \langle \text{route}(r'_X), \text{rescue} \rangle$, defined as follows (see Figure 2):

Let $Z = \text{nca}_{LT}(\text{part}(u), X)$ and r_X be a special vertex of X arbitrarily chosen in advance, then:

- r'_X is the ancestor of r_X in S which belongs to Z , $\text{route}(r'_X)$ is its routing label in S .
- Let P be a shortest path in G between the ancestor of u which belongs to Z , say u' , and r'_X , then for every vertex of P , rescue contains its identifier, the port numbers to reach its successor and its predecessor in P .

3.2 The routing algorithm

Let u, v be two vertices of G , u the sender and v the receiver. Procedure $\text{init}(u, v)$ is in charge of initializing the header attached to the message sent by u . This header, denoted by h_{uv} , contains: $h_{uv} = \langle \text{route}(v), \text{route}(r), \text{rescue}_1, \text{rescue}_2 \rangle$ as describe below.

Consider any node w of G that receives a message with a header h_{uv} , the header computed from $\text{init}(u, v)$ (possibly, $w = u$). The output port number of the edge on which the message to v has to be sent from w is computed by the function $\text{send}(w, h_{uv})$ described below. Observe that once h_{uv} is initialized by the sender, h_{uv} is never changed along the route.

Algorithm init

Input: Two addresses: $\text{address}(u)$ and $\text{address}(v)$

Result: h_{uv} , the header of a message to v

begin

$h_X \leftarrow \text{depth}_H(\text{nca}_H(\text{part}(u), \text{part}(v)));$

$\text{rescue}_1 \leftarrow \text{help}(u)[h_X].\text{rescue};$

$\text{rescue}_2 \leftarrow \text{help}(v)[h_X].\text{rescue};$

$\text{route}(r) \leftarrow \text{help}(v)[h_X].\text{route}(r'_X);$

end

Algorithm send

Input: The local routing table of a vertex w , a header h_{uv}

Result: The port number of an outgoing edge from w

begin

if w is an ancestor or a descendant of v in S **then**
 return ($f(\text{route}(w), \text{route}(v))$);

if $\text{id}(w) \in \text{rescue}_2$ **then**
 return (the port number associated to w);

if w is an ancestor or a descendant of r in S **then**
 return ($f(\text{route}(w), \text{route}(r))$);

if $\text{id}(w) \in \text{rescue}_1$ **then**
 return (the port number associated to w);

return (the port number between w and its parent);

end

3.3 Correctness and performances of the routing algorithm

We now prove the correctness of the routing algorithm. Let $\rho(u, v)$ denotes the length of the route produced by init and send from u to v . We denote by $d_G(u, v)$ the distance between u and v in G . The correctness of our scheme is given by the following lemma:

Lemma 2 *Let u, v be two vertices of G and let λ be the maximum distance between two vertices of a part of LT , then $\rho(u, v) \leq d_G(u, v) + 2\lambda$.*

Proof: Let u, v be two vertices of G , u is the sender and v the receiver. Let $Y = \text{nca}_{LT}(\text{part}(u), \text{part}(v))$ and $X = \text{nca}_H(\text{part}(u), \text{part}(v))$. Then, by Property 2, X separates $\text{part}(u)$ and $\text{part}(v)$ in LT , thus X is a descendant of Y in LT (or $X = Y$). Moreover, by Property 1, every vertex of X has an ancestor in S which belongs to Y . In particular it is true for r_X , so r'_X is well defined.

Assume that X is not an ancestor in LT of $\text{part}(u)$, i.e., X is an ancestor in LT of $\text{part}(v)$ (see Figure 2). In this case, clearly $\text{help}(u)[h_X].\text{rescue}$ contains

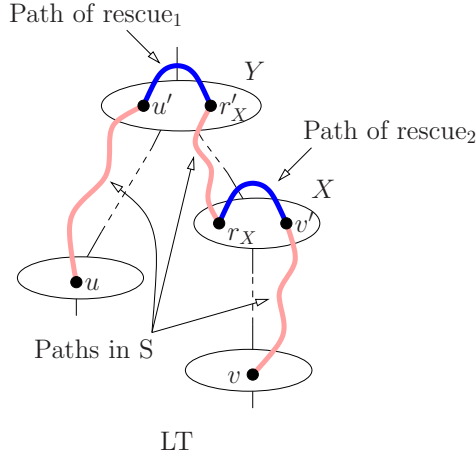


Figure 2: Example of the route induced by the scheme

a shortest path in G from u' to r'_X . Moreover, since $X = \text{nca}_{LT}(\text{part}(u), X)$, we have $\text{help}(v)[h_X].\text{route}(r'_X) = \text{route}(r_X)$, and $\text{help}(v)[h_X].\text{rescue}$ contains a shortest path in G from r_X to v' . Thus, Function `send` guarantees that the message follows the path depicted in Figure 2, or a subpath of it, if it contains loops. Then, by Property 1, we have $d_G(u, v) \geq d_G(u, Y) + d_G(Y, v) = d_S(u, u') + d_S(r'_X, r_X) + d_S(v', v)$. So we can conclude:

$$\rho(u, v) \leq d_G(u, v) + d_G(u', r'_X) + d_G(r_X, v') \leq d_G(u, v) + 2\lambda.$$

The proof of the case where X is an ancestor in LT of $\text{part}(u)$, is similar, replacing u by v , and v by u . \square

3.4 Implementation of the scheme

We assume that the standard bitwise operations (like addition, xor, shift, etc.) on $O(\log n)$ bit words run in constant time. Moreover we consider that output port numbers can be chosen during the construction of the labels.

Lemma 3 *For every vertex u , $\text{address}(u)$ can be implemented with a binary string of size $O(\lambda \log^2 n)$ bits, such that $\text{init}(u, v)$ runs in constant time. Moreover headers can be implemented with a binary string of size $O(\lambda \log n)$ bits such that for all vertex w , $\text{send}(w, h_{uv})$ runs in $O(\log \lambda)$ time.*

Proof: Let u be an arbitrary vertex of G . To implement the routing in the tree S , we use the shortest path routing scheme proposed in [12]. Since output port numbers can be chosen, this scheme produces binary labels of size $O(\log n)$ bits per vertex and such that for every u, v , the routing function $f(\text{route}(u), \text{route}(v))$ is computable in constant time. Thus, the size of $\text{route}(u)$ is $O(\log n)$ bits.

To implement $\text{path}(u)$ we use another known result, which produces labels of size $O(\log n)$, and a decodable function running in constant time [14]. Then, for every ancestor of $\text{part}(u)$ in H , $\text{help}(u)[h_X].\text{rescue}$ contains $O(\lambda)$ entries, and each entry contains three integers between 0 and n : the identifier of the vertex, the port number to reach its predecessor and the port number to reach its successor. Since the depth of H is $O(\log n)$, $\text{help}(u)$ contains $O(\lambda \log^2 n)$ bits. Since a header contains one entry of $\text{help}(u)$, it contains $O(\lambda \log n)$ bits.

In the function init , we only make some constant number of copies of pointers. Thus init runs in constant time.

In the function send , knowing if w belongs to rescue_1 or rescue_2 , can be done in $O(\log \lambda)$ time, thanks to a dichotomic procedure because rescue_1 and rescue_2 can be sorted. Others tests can be done in constant time using the routing function f . \square

From the previous lemmas, it follows that:

Theorem 1 *Every n -vertex graph admits a loop-free routing scheme of deviation 2λ such that the address and the local table have at most $O(\lambda \log^2 n)$ bits size per vertex, where λ is the maximum distance between two vertices of a same part of a layering-tree of G . Once computed by the sender, headers of size $O(\lambda \log n)$ bits never change. Moreover the scheme is polynomial time constructible and the routing decision is performed in $O(\log \lambda)$ time at every vertex.*

4 Routing scheme for tree-length δ graphs

In this section G denotes a tree-length δ graph, LT denotes a layering-tree of G and H denotes a hierarchical-tree of LT , and S denotes a shortest path spanning tree of G rooted at the source s of LT .

4.1 Preliminaries

Let us show the relation between tree-length and distances in a part of a layering-tree:

Lemma 4 *Let LT be a layering-tree of a tree-length δ graph G . Then, for every part W of LT , there is a vertex c of G , called the center of W , such that, for all $u, v \in W$, $d_G(u, v) \leq 3\delta$ and $d_G(u, c) \leq 2\delta$. Moreover, for every δ , these bounds are best possible.*

Proof: Let T be a tree-decomposition of G of length δ , w.l.o.g. T is supposed to be rooted at a bag containing s , the source of LT (see figure 3(a)). Let S be a shortest path spanning tree of G rooted at s .

Now, let W be a part of LT at distance i from s . Let X be the bag of T that is the nearest common ancestor in T of all the bags containing a vertex of W , and let $d_X = \max_{u, v \in X} d_G(u, v)$ be its diameter. Let us prove that for every $u \in W$, $d_G(u, X) \leq \delta$. In this way, we obtain:

- $\forall u, v \in W, d_G(u, v) \leq d_G(u, X) + d_X + d_G(v, X) \leq 3\delta$;
- $\forall u \in W$ and $\forall c \in X, d_G(u, c) \leq d_G(u, X) + d_X \leq 2\delta$.

Let u be an arbitrary vertex of W , and let v be a vertex of W such that $X = \text{nca}_T(\text{B}(u), \text{B}(v))$ (observe that v is well defined). Let P be the path in S from s to u , P must intersect X at a vertex x . Since u, v are both in W , there exists a path Q in G from u to v in which every vertex w is such that $d_G(s, w) \geq i$. Q intersects X at a vertex z (see Figure 3(a)).

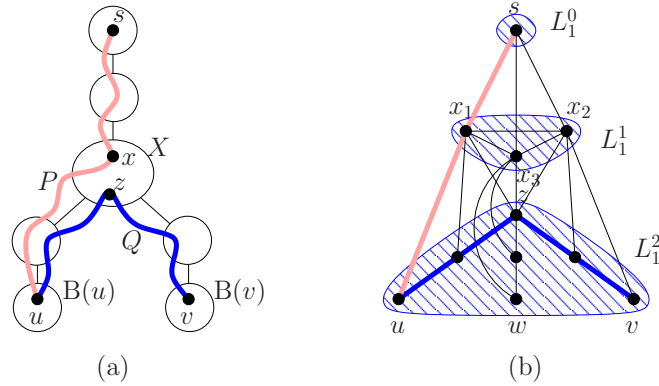


Figure 3: A part of LT is of diameter at most 3δ and of radius at most 2δ .

Note that $d_G(s, u) = i = d_G(s, x) + d_G(x, u)$ and $d_G(s, z) \leq d_G(s, x) + \delta$. So, $d_G(s, z) \leq i - d_G(x, u) + \delta$. If $d_G(x, u) > \delta$ then $d_G(s, z) < i$: a contradiction since $z \in Q$. So, $d_G(u, X) \leq d_G(x, u) \leq \delta$ as claimed.

These bounds are best possible for each $\delta \geq 1$. Indeed, for $\delta = 1$, the graph depicted on Figure 3(b) is chordal, the vertices u, v, w belong to the same part and $d_G(u, v) = d_G(u, w) = d_G(v, w) = 3$. By replacing each edge by a path of length δ , the tree-length increases to δ , vertices u, v, w still belong to the same part and they are at distance 3δ . We check that a center c for L_1^2 can be chosen arbitrarily among $\{x_1, x_2, x_3, z, s\}$ and attains a radius 2δ . Moreover, if $c \notin \{x_1, x_2, x_3, z, s\}$, one can prove that either $d_G(u, c) > 2\delta$, or $d_G(v, c) > 2\delta$, or $d_G(w, c) > 2\delta$. Thus, the radius of L_1^2 is exactly 2δ . \square

Observation 1 *Note that the choice of the center of any part W can be enhanced. Indeed, it can be set to any vertex c which belongs to X and is an ancestor in S of a vertex of W . Thus, let $u \in W$ such that for all $v \in W, d_S(u, X) \geq d_S(v, X)$. Then the center of W can be set to the vertex $c \in X$ such that $d_S(u, c) = d_S(u, X)$, and we obtain:*

$$\forall v \in W, d_G(v, c) \leq \delta + d_S(u, c) \leq 2\delta.$$

In the graph depicted on Figure 3(b), vertices satisfying this observation are x_1, x_2 and x_3 .

Lemma 4 and Theorem 1 imply the following corollary:

Corollary 1 *Every n -vertex graph of tree-length δ admits a loop-free routing scheme of deviation 6δ such that the address and the local table have at most $O(\delta \log^2 n)$ bits size per vertex. Once computed by the sender, headers of size $O(\delta \log n)$ bits never change. Moreover the scheme is polynomial time constructible and the routing decision is performed in $O(\log \delta)$ time at every vertex.*

4.2 The routing scheme

In this subsection, we will adapt the main routing scheme for the case of graphs with tree-length bounded by δ , in order to improve Corollary 1. Indeed, we will prove the following theorem:

Theorem 2 *Every n -vertex graph of tree-length δ admits a loop-free routing scheme of deviation $6\delta - 2$ such that the address and the local table have at most $O(\delta \log^2 n)$ bits size per vertex. Once computed by the sender, headers of size $O(\delta \log n)$ bits never change. Moreover the scheme is polynomial time constructible and the routing decision is performed in $O(\log \delta)$ time at every vertex.*

Proof: The routing scheme which satisfies this theorem is very similar to the general routing scheme presented in section 3. There are only two differences: let $X = \text{nca}_H(\text{part}(u), \text{part}(v))$ and $Y = \text{nca}_{LT}(\text{part}(u), X)$, then:

- the special vertex r_X is not chosen arbitrarily, it is set to a center of X which satisfies Observation 1;
- since $r_X \notin X$, $\text{part}(r_X)$ can be an ancestor of Y in LT and so the definition of r'_X has to be changed: if $\text{part}(r_X)$ is an ancestor of Y in LT then $r'_X = r_X$ else r'_X is the ancestor of r_X in S which belongs to Y .

Then the proof of Theorem 2 is completed by Lemma 5, given below. \square

Lemma 5 *Let u, v be two vertices of G , then:*

- either $\text{nca}_H(\text{part}(u), \text{part}(v)) = \text{nca}_{LT}(\text{part}(u), \text{part}(v))$, and then $\rho(u, v) \leq d_G(u, v) + 4\delta$;
- or $\text{part}(r_X)$ separates $\text{nca}_{LT}(\text{part}(u), \text{part}(v))$ and $\text{nca}_H(\text{part}(u), \text{part}(v))$ in LT , and then $\rho(u, v) \leq d_G(u, v) + 4\delta$;
- or $\text{part}(r_X)$ is an ancestor in LT of $\text{nca}_{LT}(\text{part}(u), \text{part}(v))$ and then $\rho(u, v) \leq d_G(u, v) + 6\delta - 2$;

Proof: Let u, v be two vertices of G , let $X = \text{nca}_H(\text{part}(u), \text{part}(v))$, and let $Y = \text{nca}_{LT}(\text{part}(u), X)$. In the proof we assume, w.l.o.g., that X is an ancestor in LT of $\text{part}(v)$. Let u' be the ancestor of u in S which belongs to Y and v' be the one of v which belongs to X , then:

- Case 1 (Figure 4(a)), $\text{nca}_H(\text{part}(u), \text{part}(v)) = \text{nca}_{LT}(\text{part}(u), \text{part}(v))$. By Observation 1, $\text{part}(r_X)$ is an ancestor of $\text{nca}_{LT}(\text{part}(u), \text{part}(v))$, so the path from u' to r_X is contained in the address of u and the path from r_X to v' in the address of v . So the route from u to v is well defined and by Lemma 4 the deviation is at most 4δ .
- Case 2 (Figure 4(b)), $\text{part}(r_X)$ separates Y and X in LT. Then the path from u' to r'_X is contained in the address of u , and the path from r_X to v' in the address of v . So the route is well defined and we have: $\rho(u, v) \leq d_G(u, Y) + d_G(u', r'_X) + d_G(Y, \text{part}(r_X)) + d_G(r_X, v') + d_G(X, v)$. Moreover, by Observation 1 we have $d_G(r_X, v') \leq \delta + d_G(\text{part}(r_X), X)$. Thus, we conclude $\rho(u, v) \leq d_G(u, Y) + d_G(Y, \text{part}(r_X)) + d_G(\text{part}(r_X), v') + d_G(X, v) + 4\delta$, which is less or equal to $d_G(u, v) + 4\delta$.
- Case 3 (Figure 4(c)), $X \neq Y$ and $\text{part}(r_X)$ is an ancestor in LT of Y . Then $r'_X = r_X$, so the path from u' to r_X is contained in the address of u , and the path from r_X to v' in the address of v , so the route is well defined. Moreover, we have: $\rho(u, v) \leq d_G(u, u') + d_G(u', r_X) + d_G(r_X, v') + d_G(v', v)$. Then observe that, by Property 1 and Lemma 4, we have: $d_G(u', r_X) \leq d_G(u', \text{part}(r_X)) + 3\delta$. Then, by Observation 1, we have $d_G(r_X, v') \leq \delta + d_G(v', \text{part}(r_X)) \leq 2\delta$, thus $d_G(Y, \text{part}(r_X)) \leq \delta - 1$. Finally we obtain:

$$\begin{aligned} \rho(u, v) &\leq d_G(u, u') + (\delta - 1) + 3\delta + 2\delta + d_G(v', v) \\ &\leq d_G(u, v) - d_G(X, Y) + 6\delta - 1 \\ &\leq d_G(u, v) + 6\delta - 2. \end{aligned}$$

□

5 Routing scheme for k -chordal graphs

In this section G denotes a k -chordal graph, LT denotes a layering-tree of G and H denotes a hierarchical-tree of LT, and S denotes a shortest path spanning tree of G rooted at the source s of LT.

5.1 Preliminaries

Let us show, thanks to the following lemma, the relation between chordality and distances in a part of a layering-tree. The first point of this lemma is already proved in [5] and the second one in [6].

Lemma 6 *Let LT be a layering-tree of a k -chordal graph G and S be a shortest path spanning tree of G rooted at s . Let L_j^i be an arbitrary part of LT, then for all vertices u, v of L_j^i we have:*

1. $d_G(u, v) \leq \lfloor k/2 \rfloor + 2$;

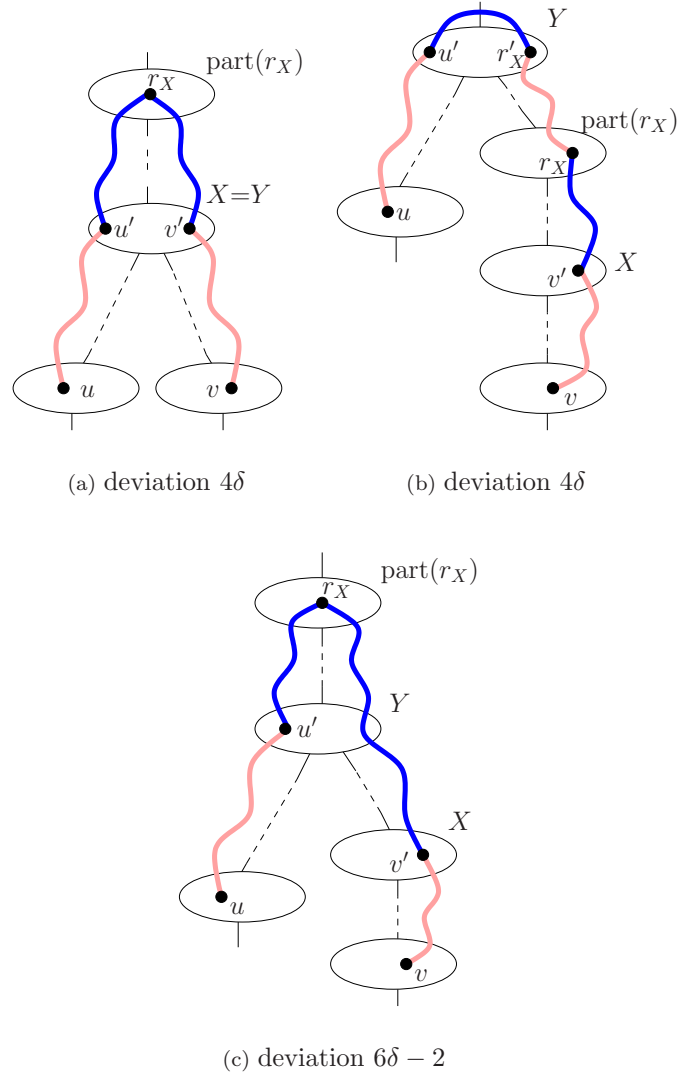


Figure 4: The three possible situations in case of tree-length δ graphs

2. there exists a path of length at most k from u to v that contains ancestors of u in S , then ancestors of v in S .

Proof: Let G be a k -chordal graph, let LT be a layering-tree of G , let S be a shortest path spanning tree of G rooted at s , and let u, v be two vertices of G which belong to a same part: L_j^i .

Then, by definition of L_j^i , there exists a path from u to v using only vertices which are at distance at least i from s . Let P be one such chordless path (see Figure 5). Moreover, let P_u (resp. P_v) be the path in S from u (resp. v) to s , then let $a(u) \in P_u$ be the nearest ancestor of u such that $a(u)$ has a neighbour, say $a(v)$, that belongs to P_v . Note that $a(u)$ exists and in the extreme case, $a(v) = s$, see Figure 5.

Then, if there is no chord between P and P_u or between P and P_v , then $P, u, \dots, a(u), a(v), \dots, v$ is an induced cycle containing both u and v , and thus $d_G(u, v) \leq k/2$, otherwise, by definition of S and as shown in Figure 5, chords can exist only between the parent of u : $p(u)$ (or the parent of v : $p(v)$) and vertices of P .

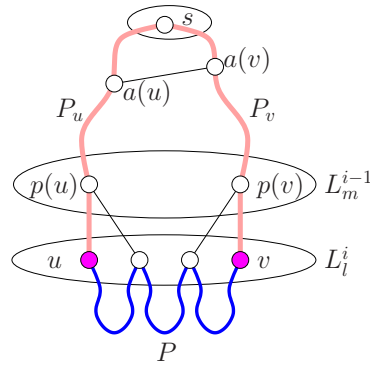


Figure 5: Relation between chordality and distances in a part of LT.

In the extreme case, there is a chord from both $p(u)$ and $p(v)$. Nevertheless there is an induced cycle containing $p(u)$ and $p(v)$. Thus $d_G(p(u), p(v)) \leq k/2$ and $d_G(u, v) \leq k/2 + 2$.

The path claimed by the second point of Lemma 6 is composed by: the subpath of P_u from u to $a(u)$, the edge $\langle a(u), a(v) \rangle$, and the subpath of P_v from $a(v)$ to v . \square

The first point of Lemma 6 implies the following corollary of Theorem 1:

Corollary 2 *Every k -chordal graph with n vertices admits a loop-free routing scheme of deviation $2 \lceil k/2 \rceil + 4$ such that the addresses and the local tables have at most $O(k \log^2 n)$ bits size per vertex. Once computed by the sender, headers of size $O(k \log n)$ bits never change. Moreover the scheme is computable in polynomial time and the routing decision is performed in $O(\log k)$ time at every vertex.*

5.2 The routing scheme

In this subsection, we will adapt the main routing scheme for the specific case of k -chordal graphs, in order to improve Corollary 2. Indeed, we will prove the following theorem:

Theorem 3 *Every k -chordal graph with n vertices admits a loop-free routing scheme of deviation $k + 1$ such that the addresses and the local tables have at most $O(\log^2 n)$ bits size per vertex. Once computed by the sender, headers of size $\Theta(\log n)$ bits never change. Moreover the scheme is computable in polynomial time and the routing decision is performed in constant time at every vertex.*

Proof: The routing scheme which satisfies this theorem is very similar to the main routing scheme presented in section 3. In this proof, we just present the differences:

1. At first, for every vertex u of G and every part X of LT that is an ancestor of $\text{part}(u)$ in H , we reduce the path contained in the field rescue of $\text{help}(u)[\text{depth}_H(X)]$ as follows. Let Y be the nearest common ancestor of $\text{part}(u)$ and X in LT and let r'_X be the ancestor of r_X in S that belongs to Y . Then, let P be the path satisfying the second point of Lemma 6, from u to r'_X . Then, $\text{help}(u)[\text{depth}_H(X)] = \langle \text{route}(a(r'_X)), \text{rescue} \rangle$, defined by:
 - $a(r'_X)$ is the first vertex of P that is an ancestor of r'_X in S .
 - rescue contains information of the edge of P between $a(r'_X)$ and its predecessor: $a(u')$. Thus, rescue contains four integers: two identifiers and two port numbers.

This is enough to prove that the routing scheme can be implemented with addresses and local tables of size at most $O(\log^2 n)$ bits per vertex and with headers of size $\Theta(\log n)$ bits. Moreover, it is easy to see (cf. Figure 6) that the route from u to v is well defined. Moreover, by the second point of Lemma 6 we have $\rho(u, v) \leq d_G(u, v) + k + 2$.

2. Now we describe how to reduce the deviation from $k + 2$ to $k + 1$.
 - Instead of setting S to an arbitrary shortest path spanning tree of G , we set S to a BFS-tree based on a BFS-ordering σ of source s .
 - For every part X of LT, instead of setting r_X to an arbitrary vertex of X , we set r_X to the vertex of X with minimum number in σ .

Then, we need the following lemma already proved in [6]:

Lemma 7 *Let S be a BFS-tree based on a BFS-ordering σ of source s , then for every vertex x of any part X , the path satisfying the second point of Lemma 6 from x to r_X is such that $d_S(x, a(x)) \leq d_S(r_X, a(r_X))$*

Proof: Let x be any vertex of any part X . Let $a(x)$ and $a(r_X)$ be the vertices satisfying the second point of Lemma 6. We already know that $|d_S(x, a(x)) - d_S(r_X, a(r_X))| \leq 1$.

Assume now that $d_S(x, a(x)) > d_S(r_X, a(r_X))$ as depicted in figure 7. Then let w be the child of $a(x)$ that is an ancestor in S of x . Then, by

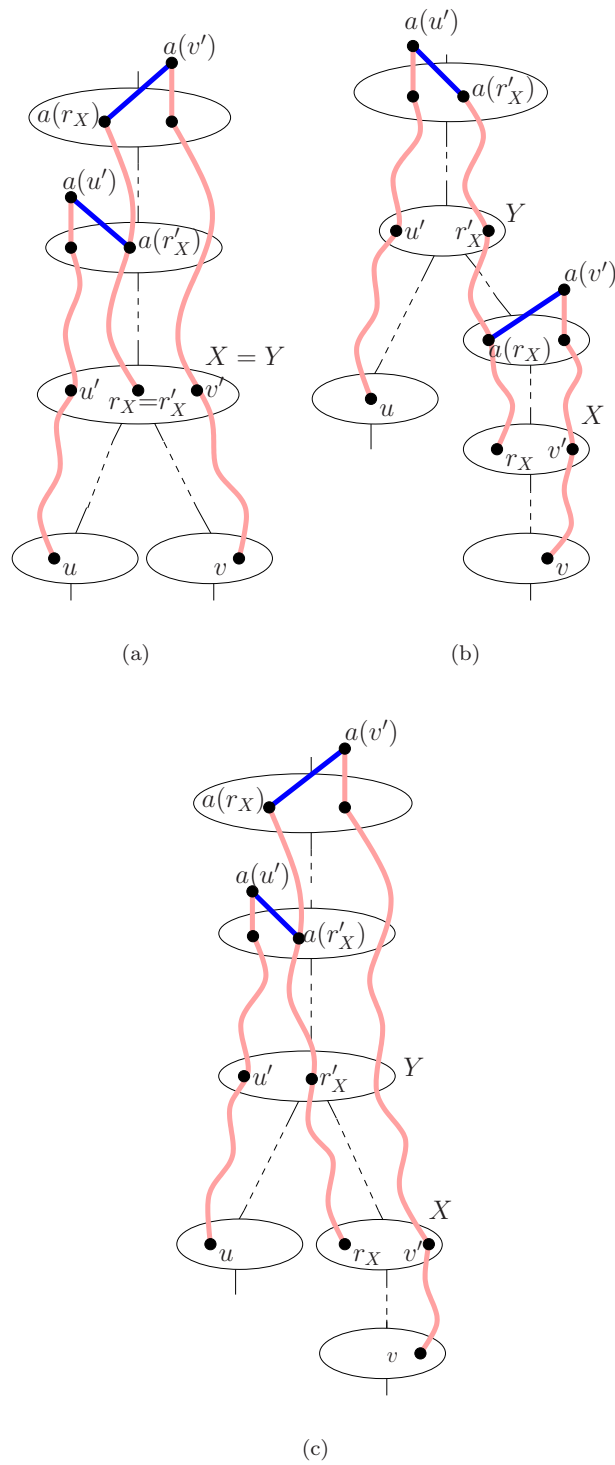


Figure 6: The three possible situations in case of k -chordal graphs

choice on r_X , $x >_\sigma r_X$, thus, by definition of S , $w >_\sigma a(r_X)$. Thus, either $a(x) = p(a(r_X))$ or $a(x) >_\sigma p(a(r_X))$. If $a(x) = p(a(r_X))$ then w has a neighbor in G that is an ancestor in S of r_X , a contradiction because by definition of $a(x)$, $a(x)$ is the nearest ancestor of x having this property. So $a(x) >_\sigma p(a(r_X))$, that is also a contradiction because by the definition of S , $p(a(r_X))$ is the neighbor of $a(r_X)$ having the largest number in σ . \square

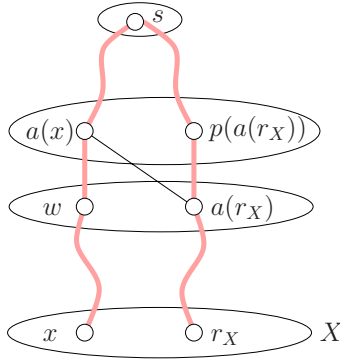


Figure 7: Impossible case when S is a BFS-tree

Lemma 7 shows that some cases depicted in figure 6 are impossible:

- in figure 6(a), $\text{part}(a(u'))$ cannot be above $\text{part}(a(r'_X))$;
and $\text{part}(a(v'))$ cannot be above $\text{part}(a(r_X))$
- in figure 6(b) and 6(c), $\text{part}(a(v'))$ cannot be above $\text{part}(a(r_X))$.

Therefore, in all cases, we obtain that $\rho(u, v) \leq d_G(u, v) + k + 1$ as claimed in Theorem 3.

\square

6 Conclusion

In this paper, we show how to use the notion of layering-tree to construct efficient compact routing schemes for tree-length δ graphs and for k -chordal graphs. It would be interesting to find other classes of graphs for which distances in parts of a layering-tree are related to some parameters of the class. In this way, we would be able to obtain a very fast constructible and efficient routing scheme, even if the class of graphs is hard to construct. Indeed, producing a tree-decomposition of length minimum is probably NP-complete for general graphs. Nevertheless, we propose an efficient routing scheme, constructible in very short time.

Moreover, the study of the case of graphs with tree-length bounded by δ can be continued. As we have done in case of k -chordal graphs, it would be interesting to remove the δ factor of the memory requirements.

Acknowledgments

This work has been done during the author's stay at Kent State University. The author is thankful to Feodor Dragan and Chenyu Yan for helpful discussions, specifically about Theorem 3. The author is also thankful to referees that allowed many improvements to this paper.

References

- [1] B. Awerbuch, A. Bar-Noy, N. Linial, and D. Peleg. Compact distributed data structures for adaptive routing. In *21st Symposium on Theory of Computing (STOC)*, volume 2, pages 230–240, May 1989.
- [2] B. Awerbuch, A. Bar-Noy, N. Linial, and D. Peleg. Improved routing strategies with succinct tables. *Journal of Algorithms*, 11(3):307–341, Sept. 1990.
- [3] B. Awerbuch and D. Peleg. Sparse partitions. In *31th Symposium on Foundations of Computer Science (FOCS)*, pages 503–513. IEEE Computer Society Press, 1990.
- [4] B. Awerbuch and D. Peleg. Routing with polynomial communication-space trade-off. *SIAM Journal on Discrete Mathematics*, 5(2):151–162, May 1992.
- [5] V. D. Chepoi and F. F. Dragan. A note on distance approximating trees in graphs. *Europ. J. Combinatorics*, 21:761–768, 2000.
- [6] V. D. Chepoi, F. F. Dragan, and C. Yan. Additive spanner for k -chordal graphs. In *Algorithms and Complexity: 5th Italian Conference (CIAC 2003)*, volume 2653 of Lecture Notes in Computer Science, pages 96–107. Springer-Verlag Heidelberg, May 2003.
- [7] Y. Dourisboure. *Routage compact et longueur arborescente*. PhD thesis, Université Bordeaux 1, 2003.
- [8] Y. Dourisboure and C. Gavoille. Improved compact routing scheme for chordal graphs. In *16-th International Symposium on Distributed Computing (DISC 2002)*, volume 2508 of Lecture Notes in Computer Science, pages 252–264. Springer-Verlag, Oct. 2002.
- [9] Y. Dourisboure and C. Gavoille. Tree-decomposition of graphs with small diameter bags. In *European conference on Combinatorics, Graph Theory and Applications (EUROCOMB 2003)*, pages 100–104. ITI, Sept. 2003.
- [10] F. F. Dragan, C. Yan, and I. Lomonosov. Collective tree spanners of graphs. In T. Hagerup and J. Katajainen, editors, *9th Scandinavian Workshop on Algorithm Theory (SWAT)*, volume 3111 of *Lecture Notes in Computer Science*, pages 64–76. Springer, 2004.
- [11] P. Fraigniaud and C. Gavoille. Universal routing schemes. *Journal of Distributed Computing*, 10:65–78, 1997.
- [12] P. Fraigniaud and C. Gavoille. Routing in trees. In F. Orejas, P. G. Spirakis, and J. v. Leeuwen, editors, *28th International Colloquium on Automata, Languages and Programming (ICALP)*, volume 2076 of *Lecture Notes in Computer Science*, pages 757–772. Springer, July 2001.

Y. Dourisboure, *Routing in gen. chordal graphs*, JGAA, 9(2) 277–297 (2005)297

- [13] C. Gavoille and M. Gengler. Space-efficiency of routing schemes of stretch factor three. *Journal of Parallel and Distributed Computing*, 61:679–687, 2001.
- [14] C. Gavoille, M. Katz, N. A. Katz, C. Paul, and D. Peleg. Approximate distance labeling schemes. In F. M. auf der Heide, editor, *9th Annual European Symposium on Algorithms (ESA)*, volume 2161 of Lecture Notes in Computer Science, pages 476–488. Springer, Aug. 2001.
- [15] C. Gavoille and S. Pérennès. Memory requirement for routing in distributed networks. In *15th Annual ACM Symposium on Principles of Distributed Computing (PODC)*, pages 125–133. ACM PRESS, May 1996.
- [16] F. Gavril. The intersection graphs of a path in a tree are exactly the chordal graphs. *Journal Comb Theory*, 16:47–56, 1974.
- [17] F. Gavril. Algorithms for maximum weight induced paths. *Inf. Process. Lett.*, 81(4):203–208, 2002.
- [18] D. Peleg and E. Upfal. A trade-off between space and efficiency for routing tables. *Journal of the ACM*, 36(3):510–530, July 1989.
- [19] N. Robertson and P. D. Seymour. Graph minors. II. Algorithmic aspects of tree-width. *Journal of Algorithms*, 7:309–322, 1986.
- [20] M. Thorup and U. Zwick. Approximate distance oracles. In *33rd Annual ACM Symposium on Theory of Computing (STOC)*, pages 183–192, Hersonissos, Crete, Greece, July 2001.
- [21] M. Thorup and U. Zwick. Compact routing schemes. In *13th Annual ACM Symposium on Parallel Algorithms and Architectures (SPAA)*, pages 1–10, Hersonissos, Crete, Greece, July 2001. ACM PRESS.