Laboratoire Bordelais de Recherche en Informatique
UMR 5800 - Université Bordeaux I, 351, cours de la Libération,
33405 Talence CEDEX, France

Research Report `RR-1274-02`

Improved Compact Routing Scheme for Chordal Graphs

by Yon Dourisboure and Cyril Gavoille

May, 2002

Improved Compact Routing Scheme for Chordal Graphs

Yon Dourisboure* Cyril Gavoille*

Abstract

This paper concerns routing with succinct tables in chordal graphs. We show how to construct in polynomial time, for every n-node chordal graph, a routing scheme using routing tables and addresses of $O(\log^3 n/\log\log n)$ bits per node, and $O(\log^2 n/\log\log n)$ bit not alterable headers such that the length of the route between any two nodes is at most the distance between the nodes in the graph plus two.

Keywords: Chordal graph, compact routing tables, tree-decomposition

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 some destination node.

It is naturally desirable to route messages along paths that are as short as possible. The efficiency of a routing scheme is measured in terms of its *multiplicative stretch* (or *additive stretch*), namely, the maximum ratio (or surplus) between the length of a route produced by the scheme for some pair of nodes, and their distance. A straightforward approach to achieving the goal of guarantees 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

*LaBRI, Université Bordeaux {*Yon.Dourisboure,gavoille*}*@labri.fr*

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 of u that leads to v. The decision must be taken locally in 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., [PU89, ABNLP89, ABNLP90, AP90, AP92, TZ01b]) that there is a tradeoff between the memory requirements of a routing scheme and the worst-case stretch factor it guarantees. In [PU89] it is shown that every routing strategy that guarantees a multiplicative s stretched routing scheme 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 multiplicative s stretched routing scheme must use $\Omega(\sqrt{n})$ bits for some nodes in some graphs for $s < 5$ [TZ01a], $\Omega(n)$ bits for $s < 3$ [FG97, GG01], and $\Omega(n \log n)$ bits for $s < 1.4$ [GP96]. More precisely, for $s = 1$ [GP96] showed that for every shortest path routing strategy and for all d and fixed $\epsilon > 0$ such that $3 \leqslant d \leqslant (1-\epsilon)n$, there exists a graph of degree bounded by d for which $\Omega(n \log d)$ bit 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 requirement.

These lower bounds are motivations for the design of routing strategies with compact tables on more specific class of graphs. Here we non exhaustively list some of them. Regular topologies (as hypercubes, tori, cycles, complete graphs, etc.) have specific routing schemes using $O(\log n)$ bit for addresses and for routing tables (cf. [Lei92]). For non-regular topologies and wider class of graphs, several trade-offs between the stretch and the size of the routing tables have been achieved. In particular, for c-decomposable graphs [FJ90] (including bounded tree-width graphs), planar graphs [FJ89, Lu02], and bounded pagenumber graphs and bounded genus graphs [GH99]. More recently, a multiplicative $1+\epsilon$ stretched routing scheme for every planar graph, for every $\epsilon > 0$, with only $(\log n)^{O(1)}$ bit addresses and routing tables, has been announced in [Tho01]. For more detailed presentation of these schemes and for an overview of the other strategies and techniques, see [Gav01] and [Pel00a].

In this paper we investigate *chordal* graphs, namely the class of graphs containing no induced cycles of length greater than 3. Trees are chordal graphs. A member of this class is depicted on the left side of Fig. 1. Note that from an information theory point of view, there is no way to give a compact representation of the underlying topology of such graphs, unlike other regular topologies (as hypercubes, grid, Cayley graphs, etc.). Indeed, there are at least $2^{n^2/4-o(n^2)}$ non-isomorphic chordal graphs with n nodes by considering for instance *split-graphs*, namely a complete graphs of $\lceil n/2 \rceil$ nodes and with $\lfloor n/2 \rfloor$ extra nodes whose neighborhood is randomly selected into the clique.

Previous Works

If we insist on shortest path (i.e., optimal stretch $s = 1$), no strategy better than complete routing tables is known for chordal graphs. Nevertheless, every chordal graph whose its maximal cliques are of size $k + 1$ exactly, namely every k-tree, supports shortest path routing tables such that, at each node, the destination addresses using the same outgoing edge consists of at most 2^{k+1} blocks of consecutive addresses [NN98]. Actually, as shown in [Dou02], this result can be easily extended to every chordal graph of *maximum* clique $k+1$. Derived from [NN98], every chordal graph with maximum clique k has shortest path routing tables of $O((2^k + d)\log n)$ bits per node of degree d, and using addresses $\in [1, n]$.

Relaxing the address size from $\log n$ bits[1] to $2\log n$, [Dou02] has recently showed that chordal graphs of maximum clique k have an additive 2 stretched routing strategy with $O(k \log n)$ bit routing table per node. However, $k = \Theta(n)$ is possible, yielding a space bound not better than complete routing tables in the worst-case.

The only scheme providing a space bound independent of the size of the maximum clique is due to Peleg and Upfal [PU89]. Based on the construction of a multiplicative 3-spanner with $O(n \log n)$ edges [PS89] (namely, a spanning subgraph whose the distance between any two nodes does not exceed 3 times the original distance in the graph), [PU89] have constructed a multiplicative 3 stretched routing strategy for chordal graphs using $O(n \log^2 n)$ bits in total for tables and $O(\log^2 n)$ bit addresses. In their scheme, headers do not consist of the full destination addresses as in [NN98] and in [Dou02], but are of $O(\log n)$ bits only. However headers can change several times along the route. In particular, the scheme provides routes that are not loop-free path in the graph.

Anyway, the solutions of [NN98], of [Dou02], and of [PU89] do not guarantee short routing tables for all the nodes. In the worst-case, $\Omega(n \log n)$ bits might be required at some nodes.

Our Results

In this paper, we design for chordal graphs an additive 2 stretched routing scheme with addresses and routing tables of $O(\log^3 n / \log\log n)$ bits per node. The headers are of size $O(\log^2 n / \log\log n)$ bits, and once initialized by the source node of the message, headers are never changed along the route. As a consequence the routes provided by our scheme are loop-free. On the time complexity point of view, our scheme is polynomially constructible (actually the time complexity to setup all the data structures is bounded by $O(n^2 \log n)$). Moreover, once the routing scheme is constructed, routing decisions at each node (including header initialization[2]) require a constant number of standard operations on $O(\log n)$ bit

[1]Logs are in base two.

[2]It should be clear that to get a constant time complexity, one does not take in account the time to copy headers from router's memory to the link registers. Otherwise, the time complexity depends on the header size.

word.

Obviously, an additive 2 stretched routing scheme gives also a multiplicative 3 stretched scheme, since in the worst-case the route between two adjacent nodes might be of length 3. However, the routes provided by our scheme are relevant as we can show that, in the scheme of [PU89], there are some counter-example graphs with nodes at distance two and having routes of length 5. Moreover, since our routing scheme is loop-free it guarantees shortest path for trees, whereas in [PU89] the routes are not optimal whenever the depth of the tree is at least two.

Technically, our result is achieved by the use of two main ingredients: 1) the well known tree-decomposition in maximal cliques of chordal graphs; and 2) the recent compact and distributed data structures for trees, in particular answering efficiently routing queries and ancestor queries with small labels [AR02, KM01, KMS02, FG01, TZ01b]. At this step, it is worth to observe that additive r stretched routing scheme on chordal graphs cannot be reduced to the problem of routing in a suitable spanning tree of the graph. Indeed, as mentioned in [Pri97, BCD99], for every fixed integer r there is a chordal graph without tree r-spanners (additive as well as multiplicative).

The paper is organized as follows. In Section 2, we introduce some definitions as tree-decomposition of Robertson and Seymour, and hierarchical tree. In Section 3, we sketch the routing scheme, and then we give its correctness and its space complexity analysis.

2 Preliminaries

We need the notion of *tree-decomposition* used by Robertson and Seymour in their work on graphs minors [RS86].

Definition 1 *A tree-decomposition of a graph G is a tree T whose nodes are subsets of $V(G)$, such that (see an example on Fig. 1):*

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* $X \cap Z \subseteq Y$.

Proposition 1 (cf. [Die00]) *A graph G is a chordal graph if and only if there exists a tree-decomposition of G (polynomial-time constructible) such that for all $X \in V(T)$, X induced a maximal clique in G.*

From now we consider an arbitrary connected chordal graph $\mathcal{G}$ with n nodes. According to Proposition 1, let $\mathcal{T}$ be a tree-decomposition of $\mathcal{G}$ such that each one of its nodes induced a maximal clique in $\mathcal{G}$. It is clear that the number of maximal cliques in $\mathcal{G}$ is at most n, so $\mathcal{T}$ has at most n nodes.

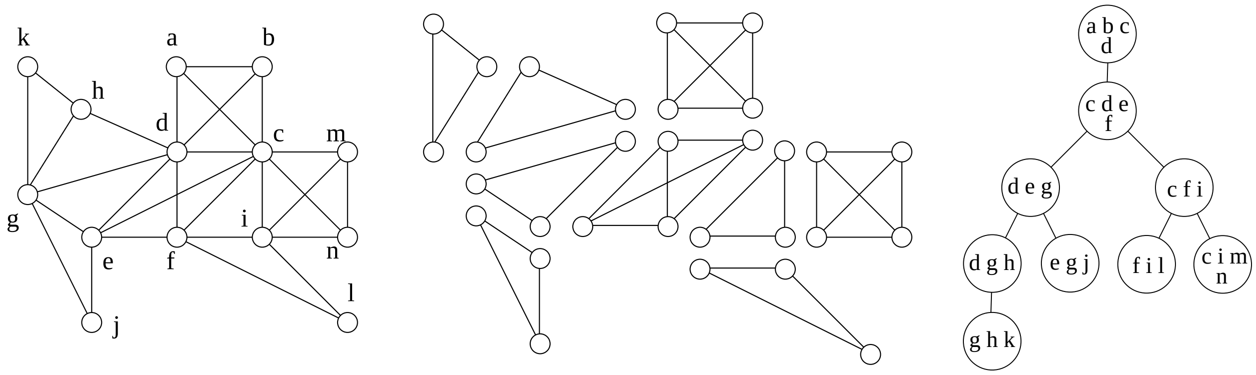

Figure 1: From left-to-right: a chordal graph G, its set of maximal cliques, and a tree-decomposition of G satisfying Proposition 1.

It is well known that every tree T with n nodes has a node s, called *separator*, such that all connected components of $T \setminus \{s\}$ have at most $n/2$ nodes. A *hierarchical tree of* T is a rooted tree H defined recursively as follows: the root of H is s, the separator of T, and the children of s are the roots of the hierarchical trees of the connected components of $T \setminus \{s\}$. Observe that H and T share the same node set, and that the depth of H is at most $\log n$. Let $\mathcal{H}$ be any hierarchical tree of $\mathcal{T}$ (cf. Fig. 2).

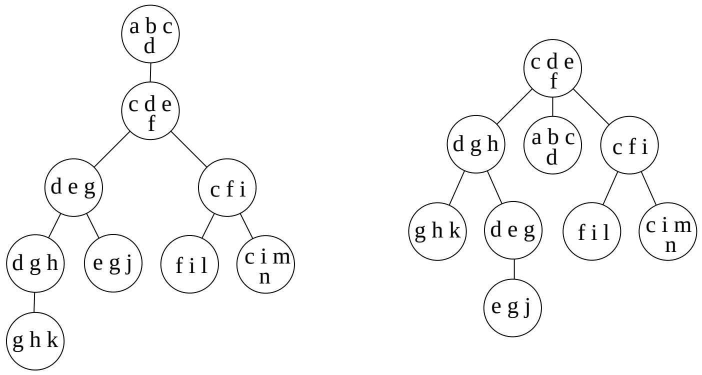

Figure 2: The tree-decomposition $\mathcal{T}$ and the hierarchical tree $\mathcal{H}$.

We use standard notions of *children*, *parent*, *ancestors*, *descendants* and *depth* in rooted trees. For simplicity we assume that a node is an ancestor of itself. We denote by $nca_T(x, y)$ the nearest common ancestor of the nodes x and y in the rooted tree T. By construction of $\mathcal{H}$ we have:

Proposition 2 *Let Q be the path in $\mathcal{T}$ from U to V, and let $Z = nca_{\mathcal{H}}(U, V)$. Then $Z \in Q$, and Z is an ancestor in $\mathcal{H}$ of all the nodes of Q.*

We assume that $\mathcal{T}$ is rooted. For every node u of $\mathcal{G}$, the *ball* of u, denoted by $B(u)$, is a node X of $\mathcal{T}$ of minimum depth such $u \in X$. Observe that, once $\mathcal{T}$ has been fixed, $B(u)$

is unique for each u by Rule 3 of Definition 1.

To each node X of $\mathcal{H}$ we associate a shortest path spanning tree of $\mathcal{G}$, denoted by $\mathcal{S}_X$, rooted at any node r_X of $\mathcal{G}$ such that $B(r_X) = X$. Observe that for each X, the node r_X exists, otherwise X would not induce a maximal clique.

Proposition 3 *Let u, v be two adjacent nodes of $\mathcal{G}$. One of these statements is true:*

1. *$B(u)$ is an ancestor of $B(v)$ in $\mathcal{T}$ and then $u \in B(v)$.*
2. *$B(v)$ is an ancestor of $B(u)$ in $\mathcal{T}$ and then $v \in B(u)$.*

Proof. Let u, v be two adjacent nodes of $\mathcal{G}$. If $B(u) = B(v)$ then the two statements are trivially trues (recall that we consider that a node is an ancestor of itself).

So, assume that $B(u) \neq B(v)$. By minimality of the depth of the balls and by definition of $\mathcal{T}$, every ball X containing a node x, is a descendant of $B(x)$ in $\mathcal{T}$. Moreover by Rule 2 of Definition 1, there exists a ball Y containing u and v. Thus Y is a descendant of $B(u)$ and of $B(v)$. Thus either $B(u)$ is an ancestor of $B(v)$ or the reverse. If $B(u)$ is an ancestor of $B(v)$ then $B(v)$ is on the path from $B(u)$ to Y in $\mathcal{T}$, by Rule 3 of Definition 1, $u \in B(v)$. Similarly if $B(v)$ is an ancestor of $B(u)$ then $v \in B(u)$. $\square$

Definition 2 *Let P be a path in $\mathcal{G}$ from u to v, and let $Z = nca_{\mathcal{H}}(B(u), B(v))$. We define $except(P) = \{w \in P \mid B(w)$ is not a descendant of Z in $\mathcal{H}\}$.*

Lemma 1 *Let P be a shortest path in $\mathcal{G}$ from u to v, $except(P) \subset nca_{\mathcal{T}}(B(u), B(v))$. Therefore, $|except(P)| \leqslant 2$.*

Proof. Let $P = x_1, x_2, \ldots, x_l$ be a shortest path in $\mathcal{G}$ from $u = x_1$ to $v = x_l$, and let $Q = X_1, X_2, \ldots, X_m$ be the path in $\mathcal{T}$ from $B(u) = X_1$ to $B(v) = X_m$. By Proposition 2 every $x_i \in P$ such that $B(x_i) \in Q$ is not in $except(P)$, because $B(x_i)$ is a descendent of $nca_{\mathcal{H}}(B(u), B(v))$. In other words, $except(P) \subseteq \{x_i \in P \mid B(x_i) \notin Q\}$. In order to prove Lemma 1, let us show that $\{x_i \in P \mid B(x_i) \notin Q\} \subset nca_{\mathcal{T}}(B(u), B(v))$. In this proof only the tree $\mathcal{T}$ is considered, so that when dealing with ancestor or descendant, it is assumed to be in $\mathcal{T}$.

Let i_0 be the smallest index such that $B(x_{i_0}) \notin Q$. Note that $i_0 \neq 1$ and $i_0 \neq l$. Thus there exists $Y \in Q$ such that Y is a separator between x_{i_0} and v. Thus there exists $j > i$ such that $x_j \in Y$. Moreover by minimality of i_0, $B(x_{i_0-1}) \in Q$. Assume that $x_{i_0} \notin nca_{\mathcal{T}}(B(u), B(v))$. By Proposition 3, either $B(x_{i_0-1})$ is an ancestor of $B(x_{i_0})$, or $B(x_{i_0})$ is an ancestor of $B(x_{i_0-1})$. If $B(x_{i_0-1})$ is an ancestor of $B(x_{i_0})$, then $B(x_{i_0-1})$ is also an ancestor of Y, and by Rule 3 of Definition 1, $x_{i_0-1} \in Y$: a contradiction because the path $x_1, \ldots x_{i_0-1}, x_j, \ldots, x_l$ would be shorter than P which is a shortest path. Thus $B(x_{i_0})$ is an ancestor of $B(x_{i_0-1})$, thus $Y = nca_{\mathcal{T}}(B(u), B(v))$ and by Rule 3 of Definition 1, $x_{i_0} \in Y$.

Let i_1 be the largest index such that $B(x_{i_1}) \notin Q$, possibly $i_1 = i_0$. By replacing x_{i_0} by x_{i_1}, and, x_{i_0-1} by x_{i_1+1}, we obtain similarly that $x_{i_1} \in nca_{\mathcal{T}}(B(u), B(v))$.

Thus we have two nodes $x_{i_0}, x_{i_1} \in nca_{\mathcal{T}}(B(u), B(v))$. They are adjacent in $\mathcal{G}$, so there is no $x_{i'} \in P$ with $i_0 < i' < i_1$. □

Corollary 1 *Let u, v be two nodes of $\mathcal{G}$, $X = nca_{\mathcal{H}}(B(u), B(v))$, and let P_X be the path from u to v in the tree $\mathcal{S}_X$. Then $|except(P_X)| \leqslant 3$.*

Proof. Let u, v be two nodes of $\mathcal{G}$, $X = nca_{\mathcal{H}}(B(u), B(v))$, and let P_X be the path from u to v in the tree $\mathcal{S}_X$. Let P_{up} and P_{down}, the paths in $\mathcal{S}_X$ from u to r_X, respectively from r_X to v. Note that P_X is composed by a sub-path of P_{up} then by a sub-path of P_{down}. Thus $except(P_X) \subseteq except(P_{up}) \cup except(P_{down})$. By Lemma 1 $|except(P_{up})| \leqslant 2$ and $|except(P_{down})| \leqslant 2$.

Moreover, Proposition 2 states that X is on the path in $\mathcal{T}$ from $B(u)$ to $B(v)$, thus X is an ancestor in $\mathcal{T}$ either of $B(u)$, or of $B(v)$. If X is an ancestor of $B(u)$, then by Lemma 1, $except(P_{up}) \subset nca_{\mathcal{T}}(B(u), X) = X$. Moreover $r_X \in X$ thus r_X and w are adjacent in $\mathcal{G}$, therefore $|except(P_{up})| \leqslant 1$. Similarly if X is an ancestor of $B(v)$, then $|except(P_{down})| \leqslant 1$. Thus $|except(P_X)| \leqslant |except(P_{up})| + |except(P_{down})| \leqslant 3$. □

3 The Routing Scheme

Let us outline the routing scheme. Each node u of $\mathcal{G}$ contains in its address the information needed to route in all trees $\mathcal{S}_X$ such that X is an ancestor in $\mathcal{H}$ of $B(u)$ ($B(u)$ included). They are $O(\log n)$ such routing trees. To send a message from a source u to a destination v, we use a route through the tree $\mathcal{S}_X$ where $X = nca_{\mathcal{H}}(B(u), B(v))$. In fact, X is a separator between u and v in $\mathcal{G}$ (Proposition 2), and both u, v contain the information about routing in $\mathcal{S}_X$. So, the standard route is done along $\mathcal{S}_X$. However that makes harder the routing process is that, along the route between u and v in $\mathcal{S}_X$, one may traverse some nodes w such that $B(w)$ is not a descendant of X in the tree $\mathcal{H}$. As w knows only trees $\mathcal{S}_Y$ for all ancestors Y of $B(w)$ in $\mathcal{H}$, it turns out that w ignores the route in $\mathcal{S}_X$, whereas $\mathcal{S}_X$ spans w. Repeating recursively the same kind of routing from w to v may result of a very long detour from the distance between u to v. (Actually the result for the length of the routes would be a multiplicative stretch $s > 1$, and not a constant additive stretch as required.) Such a node w belongs to $except(P_X)$ where P_X is the path from u to v in $\mathcal{S}_X$. As these nodes are at most 3 (Corollary 1), the solution consists to store in advance the right edge for all the nodes that are in $except(P_X)$ in order to force them to follow the path in $\mathcal{S}_X$.

3.1 Description of the labels

We assume that the outgoing edges of every node u of $\mathcal{G}$ are numbered arbitrarily by distinct integers, called *output port numbers*, and taken from $[1, \deg(u)]$. Our scheme associate to every node u of $\mathcal{G}$ two labels: its *address*, denoted by $address(u)$, and a *local routing table*, denoted by $table(u)$. The description of this latter label is given in Lemma 4, in Paragraph 3.4.

The address of u in $\mathcal{G}$ is defined by $address(u) = \langle path(u), id(u), help(u), info(u) \rangle$. Let h be the depth of $B(u)$ in $\mathcal{H}$, and $i \in \{0, \ldots, h\}$. Let X_i be the ancestor in $\mathcal{H}$ of $B(u)$ of depth i and let P_i be the path in S_{X_i} from u to r_{X_i}. The fields of $address(u)$ are as follows:

***path(u)*:** represents a sequence $(p_1, \ldots, p_h)$ of h integers. For every node X of $\mathcal{H}$, the edges between X and its children are labeled with distinct integers. $path(u)$ is the sequence of edge labels encountered along the path from the root of $\mathcal{H}$ to $B(u)$. So, $path(u)$ is an identifier of $B(u)$. If $B(u)$ is the root of $\mathcal{H}$, then $path(u)$ is defined as the empty sequence. For convenience, $path(u)[j]$ denotes p_j, and $path(u)[1 \ldots j]$ denotes the sequence $(p_1, \ldots, p_j)$. Note that the path from the root of $\mathcal{H}$ to $nca_{\mathcal{H}}(B(u), B(v))$ is the longest common prefix between $path(u)$ and $path(v)$.

***id(u)*:** is an integer taken from $[1, n]$, such that $id(u) \neq id(v)$ for all u, v of $\mathcal{G}$.

***help(u)*:** is a table with $h+1$ entries. The i-th entry is $\langle |except(P_i)|, rescue(P_i) \rangle$. Where $rescue(P_i)$ is a table with $|except(P_i)|$ entries. Let w be the j-th node of $except(P_i)$ (by Lemma 1, $j \leqslant 2$). The j-th entry of $rescue(P_i)$ is: $\langle id(w), down_{P_i}(w), up_{P_i}(w) \rangle$, where $down_{P_i}(w)$ and $up_{P_i}(w)$ are the output port numbers of the edges allowing to go in P_i from w to u, respectively from w to r_{X_i}

***info(u)*:** is a table with $h+1$ entries. The i-th entry of $info(u)$ is $route_{X_i}(u)$, where $route_{X_i}(u)$ is a routing label. More precisely, for a suitable computable function f described later, $f(route_{X_i}(u), route_{X_i}(v))$, for every $v \neq u$, returns the output port number of the first edge of the path from u to v in $\mathcal{S}_{X_i}$.

3.2 The routing algorithm

Consider u, v two nodes of $\mathcal{G}$, u the sender and v the receiver. Procedure $\textsc{init}(u, v)$ is in charge of initializing the header attached to the message sent by u. This header, denoted by $H_{u,v}$, is $(h_X, \ell, (w_1, p_1), (w_2, p_2), (w_3, p_3))$ computed as follows:

Procedure $\textsc{init}(u,v)$:

1. Let X the nearest common ancestor in $\mathcal{H}$ between $B(u)$ and $B(v)$. Set h_X be the depth in $\mathcal{H}$ of X.

2. Set $\ell := \mathit{route}_X(v)$, i.e., $\mathit{info}(v)[h_X]$. Observe that this step just requires pointer assignment[3]. Let P, P' be the paths in $\mathcal{S}_X$ from u to r_X, respectively from r_X to v.

3. For all nodes w of $\mathit{except}(P')$ (contained in $\mathit{rescue}(P')$, the second field of $\mathit{help}(u)[h_X]$), append $(\mathit{id}(w), \mathit{down}_{P'}(w))$ in $H_{u,v}$.

4. For all nodes w of $\mathit{except}(P) \setminus \mathit{except}(P')$, append $(\mathit{id}(w), \mathit{up}_P(w))$ in $H_{u,v}$.

Consider any node w of $\mathcal{G}$ that receives $H_{u,v} = (h_X, \ell, (w_1,p_1), (w_2,p_2), (w_3,p_3))$, the header computed from $\textsc{init}(u,v)$ (possibly, $w = u$). The output port number of the edge on which the message to v has to be sent from w is defined by Procedure $\textsc{send}(w, H_{u,v})$ described below. Observe that once $H_{u,v}$ is initialized by the sender, $H_{u,v}$ is never changed along the route.

Procedure $\textsc{send}(w, H_{u,v})$:

1. If $\mathit{id}(w) = w_i$, for $i = 1, 2$ or 3, then route on p_i.
2. Otherwise route in $\mathcal{S}_X$ using function $f(\mathit{route}_X(w), \ell)$. (Recall that $\ell = \mathit{route}_X(v)$. Moreover in this case, $w \notin \mathit{except}(P_i)$. Thus X is an ancestor in $\mathcal{H}$ of $B(w)$ and $\mathit{route}_X(w)$ is defined).

3.3 Correctness and Performances of the routing algorithm

We now give the correctness of the routing algorithm. Let $\rho(u,v)$ denote the length of the route produced by $\textsc{init}$ and $\textsc{send}$ from u to v. We denote by $d(u,v)$ the distance between u and v in $\mathcal{G}$. The correctness of our scheme is done proving that $\rho(u,v)$ is bounded. More precisely:

Lemma 2 *Let u, v be two nodes of $\mathcal{G}$, $\rho(u,v) \leqslant d(u,v) + 2$.*

Proof. Let $P = x_1, \ldots, x_m$ be the path from $u = x_1$ to $v = x_m$ induced by the routing scheme. Clearly P is the path from u to v in $\mathcal{S}_X$, where X is the ball chosen by $\textsc{init}(u,v)$, i.e., $X = \mathit{nca}_{\mathcal{H}}(B(u), B(v))$. Thus $\rho(u,v) \leqslant d(u, r_X) + d(r_X, v)$.

[3] The copy of the header attached to the message before forwarding is not take in account in the time complexity of $\textsc{init}(u,v)$.

Moreover, by Proposition 2, X is a separator between u and v in $\mathcal{G}$. Thus there exists $w \in X$ such that $d(u,w) + d(w,v) = d(u,v)$. As X is a clique, $d(u, r_X) \leqslant d(u,w) + 1$, and $d(r_X, v) \leqslant d(w,v) + 1$. This complete the proof. $\square$

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.

Lemma 3 *For every u, $path(u)$ and $help(u)$ can be implemented by a binary string of $O(\log n)$ bits such that* $\textsc{init}(u,v)$ *runs in constant time.*

Proof. Step 1 of $\textsc{init}(u,v)$ needs h_X, the depth in $\mathcal{H}$ of $nca_{\mathcal{H}}(B(u), B(v))$. Although this problem cannot be solved for general trees with labels shorter than $\Theta(\log^2 n)$ bits [Pel00b], $O(\log n)$ bit labels suffice for $\mathcal{H}$ as is depth is bounded by $O(\log n)$. Assume $path(u) = (p_1, \ldots, p_h)$. To compact the labels, for every i, the integer $p_i \in \{1, 2, \ldots\}$ is chosen such that it labels the edge leading to the p_i-th heaviest child of the node identified by $path(u)[1 \ldots i-1]$. It is not difficult to check that such assignment satisfies: $\prod_{i=1}^{h} p_i \leqslant n$. Thus, using variable length encoding of each p_i, $path(u)$ can be stored on $O(\log n)$ bits. Using standard techniques, the length of the longest common prefix can be extracted in a constant number of bitwise operations (see [GKK+00, pp.7-8] for a more precise implementation).

Step 2 of $\textsc{init}(u,v)$ is a pointer assignment. It assigns $info(u)[h_X]$ to ℓ, the second field of $H_{u,v}$. Steps 3 and 4 consist to extract $help(u)[h_X]$ and $help(v)[h_X]$, which are composed of a constant number of integers of $O(\log n)$ bits. Recall that by Corollary 1, $|except(P)| + |except(P')| \leqslant 3$. Then, tests and assignments on theses integers can be done in constant time. So, $\textsc{init}(u,v)$ runs in constant time. $\square$

Lemma 4 *For every w, $address(w)$, and $table(w)$ can be implemented by a binary string of $O(\log^3 n / \log\log n)$ bits (polynomial-time constructible) such that* $\textsc{send}(w, H_{u,v})$ *runs in constant time. Moreover the length of $H_{u,v}$ is $O(\log^2 n/\log\log n)$ bits.*

Proof. Each node w of $\mathcal{G}$ has a finite set of algorithms (including the function f) representing a constant number of bits, and its table label.

To implement the routing in trees $\mathcal{S}_X$ we use the scheme presented in [FG01]. This scheme uses addresses and tables of length $O(\log^2 n/\log\log n)$ bits (for an arbitrary edge labeling of the graph), and runs in constant time. As $\mathcal{H}$ is of depth at most $\log n$, the length of $info(w)$ and $table(w)$ is bounded by $O(\log^3 n/\log\log n)$ bits. Therefore, the length of $address(w)$ is bounded by $O(\log^3 n/\log\log n)$ bits.

Clearly, if $w = w_1, w_2$ or w_3, $\textsc{send}(w, H_{u,v})$ runs in constant time. Otherwise, let $\ell_1 = \mathit{route}_X(w)$ and $\ell_2 = \mathit{route}_X(v)$. By construction, ℓ_1 is $\mathit{info}(w)[h_X]$, and h_X, ℓ_2 are given by $H_{u,v}$. As computing $f(\ell_1, \ell_2)$ requires a constant time, cf. [FG01], it follows that $\textsc{send}(w, H_{u,v})$ runs in constant time.

Finally, the length of $H_{u,v}$ is $O(\log^2 n/\log\log n)$ bits since it contains only one routing label plus some constant terms on $O(\log n)$ bits. □

From all the previous lemmas it follows that:

Theorem 1 *There exists an additive* 2 *stretched loop-free routing scheme for chordal graphs using addresses and routing tables of size* $O(\log^3 n/\log\log n)$ *bits, and headers of size* $O(\log^2 n/\log\log n)$ *bits. Once computed by the sender, headers never change. Moreover, this scheme is polynomial-time constructible, and the routing function is computable in constant time for each node.*

References

[ABNLP89] Baruch Awerbuch, Amotz Bar-Noy, Nathan Linial, and David Peleg. Compact distributed data structures for adaptive routing. In 21^{st} *Annual ACM Symposium on Theory of Computing (STOC)*, pages 479–489, Seattle, WA, May 1989.

[ABNLP90] Baruch Awerbuch, Amotz Bar-Noy, Nathan Linial, and David Peleg. Improved routing strategies with succinct tables. *Journal of Algorithms*, 11(3):307–341, September 1990.

[AP90] Baruch Awerbuch and David Peleg. Sparse partitions. In 31^{th} *Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 503–513. IEEE Computer Society Press, October 1990.

[AP92] Baruch Awerbuch and David Peleg. Routing with polynomial communication-space trade-off. *SIAM Journal on Discrete Mathematics*, 5(2):151–162, May 1992.

[AR02] Stephen Alstrup and Theis Rauhe. Improved labeling scheme for ancestor queries. In 13^{th} *Symposium on Discrete Algorithms (SODA)*. ACM-SIAM, January 2002.

[BCD99] Andreas Brandstädt, Victor Chepoi, and Feodor Dragan. Distance approximating trees for chordal and dually chordal graphs. *Journal of Algorithms*, 30:166–184, 1999.

[Die00] Reinhard Diestel. *Graph Theory (second edition)*, volume 173 of Graduate Texts in Mathematics. Springer, February 2000.

[Dou02] Yon Dourisboure. An additive stretched routing scheme for chordal graphs. In 28^{th} *International Workshop, Graph - Theoretic Concepts in Computer Science (WG)*, volume Lecture Notes in Computer Science. Springer, June 2002.

[FG97] Pierre Fraigniaud and Cyril Gavoille. Universal routing schemes. *Journal of Distributed Computing*, 10:65–78, 1997.

[FG01] Pierre Fraigniaud and Cyril Gavoille. Routing in trees. In Fernando Orejas, Paul G. Spirakis, and Jan van Leeuwen, editors, 28^{th} *International Colloquium on Automata, Languages and Programming (ICALP)*, volume 2076 of Lecture Notes in Computer Science, pages 757–772. Springer, July 2001.

[FJ89] Greg N. Frederickson and Ravi Janardan. Efficient message routing in planar networks. *SIAM Journal on Computing*, 18(4):843–857, August 1989.

[FJ90] Greg N. Frederickson and Ravi Janardan. Space-efficient message routing in c-decomposable networks. *SIAM Journal on Computing*, 19(1):164–181, February 1990.

[Gav01] Cyril Gavoille. Routing in distributed networks: Overview and open problems. *ACM SIGACT News - Distributed Computing Column*, 32(1):36–52, March 2001.

[GG01] Cyril Gavoille and Marc Gengler. Space-efficiency of routing schemes of stretch factor three. *Journal of Parallel and Distributed Computing*, 61:679–687, 2001.

[GH99] Cyril Gavoille and Nicolas Hanusse. Compact routing tables for graphs of bounded genus. In Jiří Wiedermann, Peter van Emde Boas, and Mogens Nielsen, editors, 26^{th} *International Colloquium on Automata, Languages and Programming (ICALP)*, volume 1644 of Lecture Notes in Computer Science, pages 351–360. Springer, July 1999.

[GKK$^+$00] Cyril Gavoille, Michal Katz, Nir A. Katz, Christophe Paul, and David Peleg. Approximate distance labeling schemes. Research Report RR-1250-00, LaBRI, University of Bordeaux, 351, cours de la Libération, 33405 Talence Cedex, France, December 2000.

[GP96] Cyril Gavoille and Stéphane Pérennès. Memory requirement for routing in distributed networks. In 15^{th} *Annual ACM Symposium on Principles of Distributed Computing (PODC)*, pages 125–133. ACM PRESS, May 1996.

[KM01] Haim Kaplan and Tova Milo. Short and simple labels for small distances and other functions. In 7^{th} *International Workshop on Algorithms and Data Structures (WADS)*, volume 2125 of Lecture Notes in Computer Science, pages 32–40. Springer, August 2001.

[KMS02] Haim Kaplan, Tova Milo, and Ronen Shabo. A comparison of labeling schemes for ancestor queries. In 14^{th} *Symposium on Discrete Algorithms (SODA)*. ACM-SIAM, January 2002.

[Lei92] Frank Thomson Leighton. *Introduction to Parallel Algorithms and Architectures: Arrays - Trees - Hypercubes.* Morgan Kaufmann, 1992.

[Lu02] Hsueh-I Lu. Improved comact routing tabels for planar networks via orderly spanning trees. In 8^{th} *Annual International Computing & Combinatorics Conference (COCOON)*, volume Lectures Notes in Computer Science. Springer, August 2002.

[NN98] Lata Narayanan and Naomi Nishimura. Interval routing on k-trees. *Journal of Algorithms*, 26(2):325–369, February 1998.

[Pel00a] David Peleg. *Distributed Computing: A Locality-Sensitive Approach.* SIAM Monographs on Discrete Mathematics and Applications, 2000.

[Pel00b] David Peleg. Informative labeling schemes for graphs. In 25^{th} *International Symposium on Mathematical Foundations of Computer Science (MFCS)*, volume 1893 of Lecture Notes in Computer Science, pages 579–588. Springer, August 2000.

[Pri97] Erich Prisner. Distance approximating spanning trees. In 14^{th} *Annual Symposium on Theoretical Aspects of Computer Science (STACS)*, volume 1200 of Lecture Notes in Computer Science, pages 499–510. Springer, 1997.

[PS89] David Peleg and Alejandro A. Schäffer. Graph spanners. *Journal of Graph Theory*, 13(1):99–116, 1989.

[PU89] David Peleg and Eli Upfal. A trade-off between space and efficiency for routing tables. *Journal of the ACM*, 36(3):510–530, July 1989.

[RS86] Neil Robertson and Paul D. Seymour. Graph minors. II. Algorithmic aspects of tree-width. *Journal of Algorithms*, 7:309–322, 1986.

[Tho01] Mikkel Thorup. Compact oracles for reachability and approximate distances in planar digraphs. In 42^{th} *Annual IEEE Symposium on Foundations of Computer Science (FOCS)*. IEEE Computer Society Press, October 2001.

[TZ01a] Mikkel Thorup and Uri Zwick. Approximate distance oracles. In 33^{rd} *Annual ACM Symposium on Theory of Computing (STOC)*, pages 183–192, Hersonissos, Crete, Greece, July 2001.

[TZ01b] Mikkel Thorup and Uri Zwick. Compact routing schemes. In 13^{th} *Annual ACM Symposium on Parallel Algorithms and Architectures (SPAA)*, pages 1–10, Hersonissos, Crete, Greece, July 2001. ACM PRESS.