[BLT91] Bakker, E.M., van Leeuwen, J., Tan, R.B.: Linear Interval Routing. Algorithms Review **2** (1991) 45–61.

[E79] Even, S.: Graph Algorithms. Computer Science Press, Inc., (1979).

[EMZ96] Eilam, T., Moran, S., Zaks, S.: Lower Bounds for Linear Interval Routing. Proc. 10th int. Workshop on Distributed Algorithms (WDAG) (1996) 191–205.

[FG94] Fraigniaud, P., Gavoille, C.: Interval Routing Schemes. Research Rep. 94-04, LIPS-ENS Lyon (1994).

[FGNT95] Flammini, M., Gambosi, G., Nanni, U., Tan, R.: Multi-Dimensional Interval Routing Scheme. Proc. 9th Int. Workshop on Distributed Algorithms (WDAG) (1995) 131–144.

[FGS93] Flammini, M., Gambosi, G., Salomone, S.: Boolean Routing. Proc. 7th Int. Workshop on Distributed Algorithms (WDAG) (1993) 219–233.

[FG96] Fraigniaud, P., Gavoille, C.: Local Memory Requirement of Universal Routing Schemes. Proc. 8th Annual ACM Symp. on Parallel Algorithms and Architecture (SPAA) (1996).

[GP96a] Gavoille, C., Perennes, S.: Memory Requirements for Routing in Distributed Networks. Proc. 15th Annual ACM Symp. on Principles of Distributed Computing (PODC) (1996) 125–133.

[GP96b] Gavoille, C., Perennes, S.,: Lower Bounds for Shortest Path Interval Routing. Proc. 3rd Colloq. on Structural Information and Communication Complexity (SIROCCO) (1996).

[I91] The T9000 Transputer Products Overview Manual, INMOS (1991).

[LT83] van Leeuwen, J., Tan, R.B.: Routing with Compact Routing Tables. Tech. Rep. RUU-CS-83-16, Dept. of Computer Science, Utrecht University (1983). Also as: Computer Networks with Compact Routing Tables. In: G. Rozenberg and A. Salomaa (Eds.) The book of L, Springer- Verlag, Berlin (1986) 298–307.

[LT86] van Leeuwen, J., Tan, R.B.: Computer Network with Compact Routing Tables. In: G. Rozenberg and A. Salomaa (Eds.), The Book of L., Springer-Verlag, Berlin (1986) 259–273.

[PU89] Peleg, D., Upfal, E.: A Trade-Off between Space and Efficiency of Routing Tables. Journal of the ACM **36** (1989) 510–530.

[R91] Ružička, P.: A Note on the Efficiency of an Interval Routing Algorithm. The Computer Journal **34** (1991) 475–476.

[SK82] Santoro, N., Khatib, R.: Routing Without Routing Tables. Tech. Rep. SCS-TR-6, School of Computer Science, Carleton University (1982). Also as: Labeling and Implicit Routing in Networks. The Computer Journal **28** (1) (1985) 5–8.

[TL94] Tse, S. S.H., Lau, F. C.M.: A Lower Bound for Interval Routing in General Networks. Networks **29** (1997) 49–53. Also as: Tech. Rep. TR-94-09, Department of Computer Science, University of Hong Kong (1994).

This article was processed using the LaTeX macro package with LLNCS style

Linear-Label so that it will guarantee the same property also for any two separation nodes $t_{i_1}^{j_1}$ and $t_{i_2}^{j_2}$ even if they are in the same 2-edge-connected component (that is, $i_1 = i_2$). For example, consider the nodes s and t in Figure 6. The refined algorithm will guarantee that the sub-path between the separation nodes t_1^1 and t_1^3 of either of the paths $Path(s, t, \mathcal{L}_G)$ and $Path(t, s, \mathcal{L}_G)$ is a shortest length path (although s and t are in the same 2-edge-connected component).

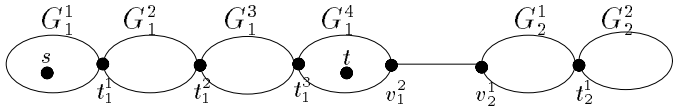

Fig. 6. A decomposition of a graph whose superstructure is a line. Every G_i^j is a 2-node-connected graph (with leaves)

.

The refinement of Algorithm Linear-Label in order for it to imply the stronger property for this family of graphs is as follows. When traversing the graph in the first phase in order to construct a DFS tree, we start in any node r in the last 2-node-connected component, $G_r^{k\,r}$, (G_2^2 in Figure 6). We traverse first a shortest length path P from it to the separation node t_1^1 and then continue the DFS traversal arbitrarily. We label the nodes, as in the general case, such that all the nodes in the path P are leftmost nodes. It should be clear, by the same considerations as in the general case, that the stronger property is satisfied by this refined version of Algorithm Linear-Label.

It is interesting to note that the graphs for which this refinement could be applied (graph that satisfy that the superstructure of their reduced graph is a line) are all the graphs that are not *Petal graphs*. In [EMZ96], Petal graphs were introduced and a lower bound of the total_2-diameter (a quantity that can be as large as the square of the diameter) on the efficiency of Linear-Interval Routing was proved for all Petal graphs. Thus, the fact that the refinement of Algorithm Linear-Label applies to all non-petal graphs is not surprising.

References

[ABLP89] Awerbuch, B., Bar-Noy, A., Linial, N., Peleg, D.: Improved Routing Strategies with Succinct Tables. Journal of Algorithms, **11** (1990) 307–341.

[AP92] Awerbuch, B., Peled, D.: Routing with Polinomial Communication-Space Tradeoff. SIAM Journal on Discrete Math., **5** (1992) 151–162.

[BLT90] Bakker, E.M., van Leeuwen, J., Tan, R.B.: Prefix Routing Schemas in Dynamic Networks. Tech. Rep. RUU-CS-90-10, Dept. of Computer science, Utrecht University, Utrecht (1990), and, Computer Networks and ISDN Systems **26** (1993) 403–421.

to s. Since $L(t) > L(s)$ then if s is not in T_t, the message will traverse a path in G_j until an ancestor of s, x, is reached and then it will traverse the unique path on T between x and s, and, again, the sub-path between v_j^1 and v_i^2 is a shortest length path.

Algorithm Linear-Label could be easily modified so that every link of the network (at least in one direction) will be used for routing. This property is desirable since it implies a more balanced load on the links of the network. Consider an LIRS $\mathcal{L}_G$ generated by Algorithm Linear-Label, and a node $x \in V$. Let $Back(x) = \{(x, v_1), ..., (x, v_k)\}$ be the set of fronds in G w.r.t T from x to its ancestors. At most one of these edges could be the back edge, $back(x)$, assigned to x by Algorithm Linear-Label. Algorithm Linear-Label assigns a null interval to all other edges in $Back(x)$. We can thus distribute the load on the edge $(x, P_T(x))$ from x to its predecessor in T also between the edges in the set $Back'(x) = Back(x) - \{back(x)\}$. We will divide the interval $\langle max(T_x) + 1, max\rangle$ assigned to $(x, P_T(x))$ to $|Back'(x)|+1$ approximately equal subintervals $\langle max(T_x) + 1, n_1\rangle, \langle n_1, n_2\rangle, ..., \langle n_{k'-1}, max\rangle$ and assign every subinterval to a different edge in $Back'(x) \cup \{(x, p_T(x))\}$. It is easily seen that the algorithm remains correct after the modification although a message from node x to node y with larger label will not necessarily traverse the path on the tree T from x to y; the path on which the message will traverse up to a common ancestor may contain some fronds. Since every edge which is not an edge of T is of the form (x, v_i) where v_i is an ancestor of x, that is, the size of the interval initially assigned to $(x, P_T(x))$ is at least $|Back(x)'|+1$, then after the modification every edge is assigned a non-empty interval in at least one direction. We assume here that there are no multiple edges (that is, the graph is simple). Indeed, if the graph is not simple then any deterministic labeling algorithm cannot guarantee that each link will be used for routing since there could be more links than destinations.

4.2 Refinement of Algorithm Linear-Label for Subsets of Graphs

We show in this section how to refine Algorithm Linear-Label for a family of graphs which is a subset of the non-lithium graphs.

Consider a graph G such that the superstructure (defined in [E79]) of its reduced graph $R(G)$ is a line. These graphs can be decomposed in two levels; one level is by their 2-edge-connected components (as in the general case). A refined level is to further decompose every 2-edge-connected component to its 2-node-connected components. This decomposition could be represented as a sequence; $(G_1^1, t_1^1, G_1^2, t_1^2, ...t_1^{k_1-1}, G_1^{k_1}), (v_1^2, v_2^1), ..., (v_{r-1}^2, v_r^1), (G_r^1, t_r^1, ..., t_r^{k_r-1}, G_r^{k_r})$, where every G_i^j is the jth 2-node-connected component in the ith 2-edge-connected component, t_i^j is a separation node common to the 2-node-connected components G_i^j and G_i^{j+1}, and (v_i^2, v_{i+1}^1) is the bridge connecting two 2-edge-connected components. (The decomposition is demonstrated in Figure 6). Algorithm Linear-Label guarantees that a path traversed by any message between any two bridge points is a shortest length path. For this family of graphs we can refine Algorithm

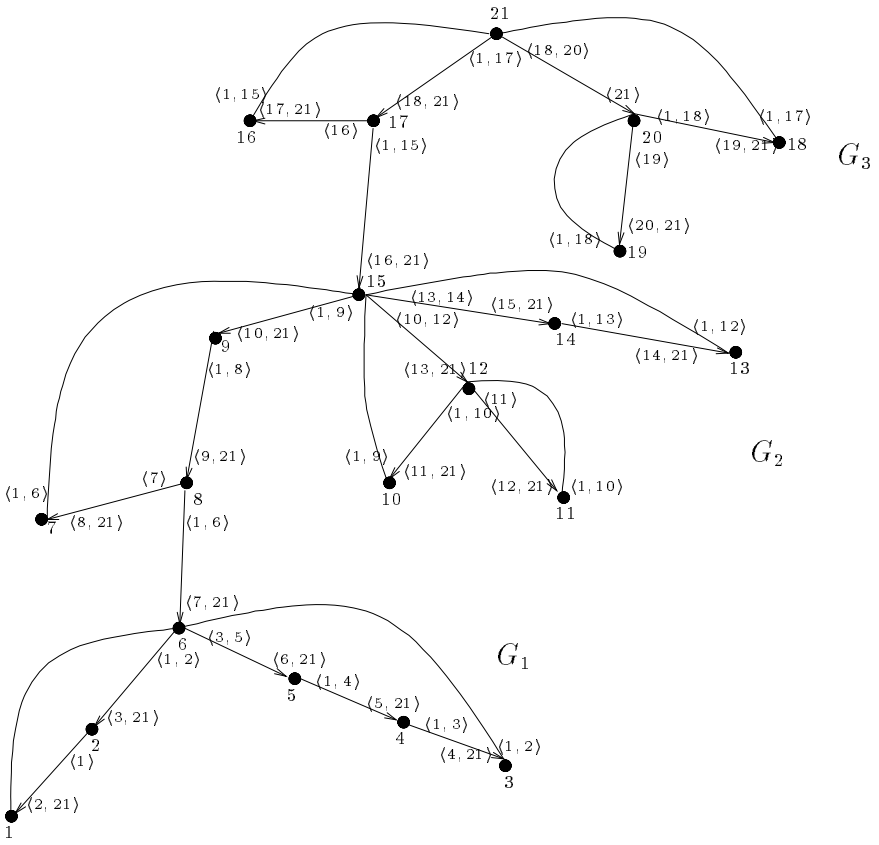

Fig. 5. A graph G together with an LILS for it, generated by Algorithm Linear-Label.

4 Properties and Extensions of Algorithm Linear-Label

4.1 Properties of the Algorithm

Let $s \in V_i$ and $t \in V_j$ be any two nodes in G_i and G_j, respectively, $i < j$. We show that the sub-path between the bridge points v_i^2 and v_j^1 of either of the paths $Path(s, t, \mathcal{L}_G)$ and $Path(t, s, \mathcal{L}_G)$, where $\mathcal{L}_G$ is the LILS generated by Algorithm Linear-Label, is a shortest length path.

The path P in the tree T (which was constructed by the algorithm) between its root r and the node v_1^2 is a shortest length path which passes through the nodes $r, v_n^1, v_{n-1}^2, ..., v_1^2$. By Algorithm Linear-Label, $L(t) > L(s)$ and thus a message from s to t will traverse the unique path between s and t in the tree T. Since the sub-path of this path between v_i^2 and v_j^1 is a sub-path of the shortest length path P, it is also a shortest length path. Now consider a message from t

Phase 2:

A. Assigning labels to nodes

Traverse T in a depth first style; when a node x is reached for the first time,

1. If x is a node in the path P then let $y_1 \in S_T(x)$ be its successor in the path P. Recursively traverse first T_{y_1} and then each T_z, $z \in S_T(x) - \{y_1\}$.
   Otherwise, let $back(x) = (u, v)$. If $u = x$ then recursively traverse each T_z, $z \in S_T(x)$, in an arbitrary order.
   Otherwise ($back(x) = (u, v)$, $u \neq x$), let $y_1 \in S_T(x)$ be the successor of x such that u is a node in T_{y_1}. Recursively traverse first T_{y_1} and then each T_z, $z \in S_T(x) - \{y_1\}$ (in an arbitrary order).
2. Assign an integer $L(x)$ to x.
3. Integers are assigned in ascending order.

B. Assigning labels to edges

For a node $x \in V$ we denote by $max(T_x)$ and $min(T_x)$ the maximal and minimal label of a node in T_x.

Let $x \in V$ and let $y_1, ..., y_l$ be its successors in an ascending order (of their labeling).

• If x is a node in the path P then $I_x(x, y_1) = \langle min, max(T_{y_1})\rangle$, $I_x(x, y_i) = \langle min(T_{y_i}), max(T_{y_i})\rangle$ for every $2 \leq i \leq l$, $I_x(x, P_t(x)) = \langle max(T_x) + 1, max\rangle$

Otherwise, let $back(x) = e = (u, v)$.

• If $u = x$ then $I_x(e) = \langle min, min(T_x) - 1\rangle$, $I_x(x, y_i) = \langle min(T_{y_i}), max(T_{y_i})\rangle$ for every $1 \leq i \leq l$, $I_x(x, P_t(x)) = \langle max(T_x) + 1, max\rangle$

• Otherwise ($x \neq u$) the labeling of the edges in this case is similar to the first case.

If the algorithm does not specify how to label an edge e at one of its end nodes u (that is, if $e = (u, v)$ is neither an edge of the tree nor the back edge of Node u) then $L_u(e) = \langle\rangle$ (the null interval).

Example 2. Figure 5 is an example of a graph G that was labeled according to Algorithm Linear-Label (null intervals of edges are not marked). The graph G has three components which are 2-edge-connected graphs G_1, G_2 and G_3. We refer to the nodes in G by the integers assigned to them as labels. The edges $(6, 8)$ and $(15, 17)$ are the strong bridges that connect G_1 with G_2 and G_2 with G_3, respectively. Edges of the DFS spanning tree found by Algorithm Linear-Label are directed. A shortest length path P in T which passes through all the bridges is the path $\langle 21, 17, 15, 9, 8, 6\rangle$. Note that all the nodes in P are leftmost nodes. Consider for example the path a message from node 19 to node 2 will traverse. Since 19 is larger then 2 but node 2 is not a descendant of node 19, the message will go up the back edge of 19 to node 20. Still 2 is not a descendant of 20 thus the message go down to 18 and then up the back edge of node 20 (and 18) to node 21. Now 2 is a descendant of 21 thus the message will go down the tree to $17, 15, 9, 8, 6$ and 2. Note that the sub-path of this path between 17 and 6 is a shortest length path in G. A message from node 2 to node 19 will traverse the unique path on T between them. Again, the sub-path of this path between node 6 and node 17 is a shortest length path.

Consider a labeled DFS tree T. A node v is termed *a leftmost node* if every node u with label smaller than the label of v is a successor of v (that is, u is a node in the sub-tree T_v). Note that a back edge for a leftmost node in a DFS tree which was labeled by Algorithm Linear-Label is not needed (since the interval of nodes with labels smaller than $L(v)$ that are not in T_v is empty).

The structure of a non-lithium graph is a sequence of 2-edge-connected graphs (with leaves) connected by bridges. In a DFS spanning tree of the reduced graph of a non-lithium graph a back edge could be found to every node except for some of the bridge points. Therefore, the extension of Algorithm Linear-Label for non-lithium graphs will guarantee that every end-node of a bridge will be a left-most node in the tree (thus, a back edge for it is not needed).

Following is an informal description of the algorithm. Let $G = (V, E)$ be a non-lithium graph represented by the sequence of components $G_1, (v_1^2, v_2^1)$, $G_2, (v_2^2, v_3^1)$, $G_3, ..., (v_{n-1}^2, v_n^1), G_n$, where $G_i = (V_i, E_i)$ is a 2-edge-connected graph with leaves. We first find a strict-LIRS for the reduced graph of G, $R(G)$, and then transform it into an LIRS of G, as explained in Section 3.3. To simplify the notations, in the sequel G will stand for $R(G)$. In the first phase we traverse G in order to find a DFS spanning tree; for this, we first traverse a shortest length simple path P from the root - an arbitrary node $r \in V_n$ - to the node v_1^2 (such a path must pass through all the bridge points), then we continue to span G in a depth-first style arbitrarily. In this phase, we also assign a back edge to every node that is not on the path P. After a DFS spanning tree T is constructed, we traverse T in a depth-first style in order to label the nodes; when we reach a node $x \in P$ we first traverse recursively T_y, where $y \in S_T(x)$ is the successor of x in P, thus guaranteeing that all the nodes in P are leftmost nodes in the tree. Other nodes are treated as in the original algorithm.

Note that for the correctness of the algorithm the path P does not have to be a shortest length path. We construct P as a shortest length path in order to guarantee that the path that any message traverses between any two bridge points is a shortest length path (a property that will be analyzed in Section 4.1). The only difference between the two versions of Algorithm Linear-Label is that in the new version some nodes are not assigned a back edge. Note however that all such nodes are nodes in the path P, that is, leftmost nodes for which a back edge is not needed. The path P contains all the bridge points, thus for every node which is not contained in it, there exists a back edge.

Following is the formal code of Algorithm Linear-Label for any non-lithium graph together with an example.

Algorithm Linear-Label
Phase 1:
Start from any node r in G_n and construct a DFS spanning tree T of G; traverse first a shortest length path P between r and v_1^2 and then the rest of the graph. Assign to every node x that is not in the path P a back edge, $back(x)$, while keeping the following invariant.
$\mathcal{INV}$: For every node x with $back(x) = (u, v)$ and for every node y on the path in T from x to u, $back(y) = back(x)$.

2. Assign an integer $L(x)$ to x.
3. Integers are assigned in ascending order.

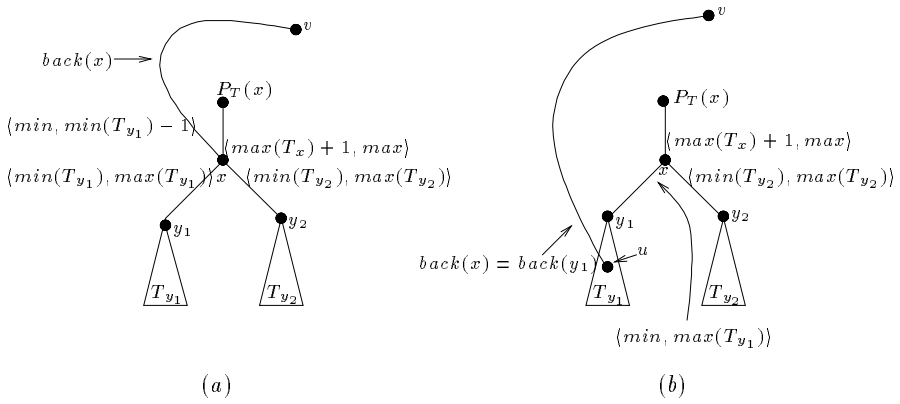

Fig. 4. The labeling of the edges outgoing a node x where (a) $back(x)$ is an edge outgoing x (b) Otherwise ($back(x) = (u, v)$ and u is in T_{y_1}).

B. Assigning labels to edges

For a node $x \in V$ we denote by $max(T_x)$ and $min(T_x)$ the maximal and minimal label of a node in T_x (note that $max(T_x) = L(x)$).
Let $x \in V$ and let $y_1, ..., y_l$ be its successors in an ascending order (of their labeling). Let $back(x) = e = (u, v)$.

1. If $x = u$ then $I_x(e) = \langle min, min(T_x) - 1\rangle$, $I_x(x, y_i) = \langle min(T_{y_i}), max(T_{y_i})\rangle$ for each $1 \leq i \leq l$, $I_x(x, P_t(x)) = \langle max(T_x) + 1, max\rangle$ (see Figure 4(a)).
2. Otherwise ($x \neq u$), the only difference in labeling in this case is: $I_x(x, y_1) = \langle min, max(T_{y_1})\rangle$ (the back edge of x is not labeled yet) (see Figure 4(b)).

If the algorithm does not specify how to label an edge e at one of its end nodes u (that is, if $e = (u, v)$ is neither an edge of the tree nor the back edge assigned to node u) then $L_u(e) = \langle\rangle$ (the null interval).

Algorithm Linear-Label generates a strict-LIRS for any 2-edge-connected graph. As explained in [FG94], it is straightforward to generate from a strict-LIRS of any graph G, an LIRS for a graph G' whose reduced graph is equal to G (i.e., $R(G') = G$; Definition 8).

3.4 Labeling any Non-Lithium Graph

We now extend Algorithm Linear-Label to any non-lithium graph.

ancestor of y is reached. Note that the third case is the only one in which the path traversed by a message is not the unique path in the tree T.

There are two phases in Algorithm Linear-Label. In the first phase a DFS tree T is constructed and a back edge is assigned to every node. In the second phase we traverse the tree in a (specific) depth-first search and label the nodes (in a post-order fashion) and the edges. Specifically, consider any node x with a set of successors $y_1, ..., y_k$ and let $e = (u, v)$ be the back edge assigned to x in the first phase. If $u \neq x$ (that is, the back edge of x emanates from a successor u of x which is not equal to x) then assume that T_{y_i} is the sub-tree to which u belongs. Then the subtree T_{y_i} is traversed first and then all other subtrees T_{y_j}, $y_j \in S_T(x)$, are traversed in an arbitrary order. This traversal guarantees that the interval of labels of nodes in T_{y_i} is smaller then the interval of labels of nodes in any other sub-tree T_{y_j}, $y_j \in S_T(x)$. This property enables us to send both messages destined to nodes in T_{y_i} and messages destined to nodes not in T_x but with smaller labels, on the edge from x to y_i (towards the back-edge assigned to x) since the union of both sets of labels form a contiguous interval. If the back edge assigned to x emanates from x (that is, $u = x$), then there is no restriction on the order of the traversal of the subtrees T_{y_j}, $y_j \in S_T(x)$.

The formal code of Algorithm Linear-Label for any 2-edge-connected graph with leaves is presented in Section 3.3. The generalization of Algorithm Linear-Label to any non-lithium graph is in Section 3.4.

3.3 Labeling a 2-Edge-Connected Graph with Leaves

The input to Algorithm Linear-Label is a 2-edge-connected graph and the output is a strict-LIRS for it. Given a graph G, a strict-LIRS for it $\mathcal{L}_G$, and a graph G' such that $R(G') = G$ (see Definition 8) it is straightforward to generate an LIRS $L_{G'}$ for G' (see, [FG94]). Algorithm Linear-Label does not specify how to find back edges; however this task can be easily performed while doing the DFS traversal (see, e.g., [E79]).

Algorithm Linear-Label

Phase 1: Construct a DFS spanning tree T rooted at any node r of the graph G. Assign to every node x a back edge, $back(x)$, while keeping the following invariant.

$\mathcal{INV}$: For every node x with $back(x) = (u, v)$ and for every node y on the path in T from x to u, $back(y) = back(x)$.

Phase 2:

A. Assigning labels to nodes

Traverse T in a depth first style; when a node x is reached for the first time,

1. If $x = r$ or $back(x) = (u, v)$ and $u = x$ then recursively traverse each T_y, $y \in S_T(x)$, in an arbitrary order.
   Otherwise ($back(x) = (u, v)$, $u \neq x$), let $y_1 \in S_T(x)$ be the successor of x such that u is a node in T_{y_1}. First recursively traverse T_{y_1} and then each T_z, $z \in S_T(x) - \{y_1\}$, in an arbitrary order.

an edge of the graph which is not an edge of the tree. A DFS spanning tree of a graph satisfies the property that any frond is an edge that connects a node with one of its ancestors in the tree. Additionally, it is well known (see, e.g., [E79]) that in a DFS spanning tree of a 2-edge-connected graph for every node x (except for the root) there is a frond which connect a descendant of x and an ancestor of it in the tree.

3.2 An Informal Description of Algorithm Linear-Label

We start by giving an informal description of Algorithm Linear Label in 2-edge-connected graphs. It is rather simple to generate an ILS (Interval Labeling Scheme) for any (undirected) graph G by using any spanning tree of G (such a scheme was presented in [SK82]). In order to label the nodes, we traverse the tree in a depth-first-style and label the nodes in a post-order fashion. Let T denote the resulted directed tree. For every node x, the labels of the nodes in T_x form a contiguous interval, and $L(x) > L(u)$ for every node u in T_x. The label $I_v(e)$ for an edge in E_v, for any node v, is defined as follows: if $e = (x, y)$, and y is a successor of x in T, then $I_x(e)$ is the interval of labels of nodes in T_y, otherwise, if y is the predecessor of x in T, then the interval $I_x(y)$ is the cyclic complement interval of the interval of labels of the nodes in T_x. The path a message traverses under such IRS is the unique path in the tree T between its source and its destination.

When considering LIRSs, cyclic intervals are not allowed. As a consequence, an interval of an edge from a node to its predecessor on the tree cannot contain both labels that are smaller than the labels of the nodes in T_x and labels that are larger than them. To avoid this problem we utilize the special structure of a non-lithium graphs which is a sequence of 2-edge-connected components (with leaves) connected by bridges. We first present a labeling algorithm for any 2-edge-connected graph (with leaves) and then generalize it to a labeling algorithm for any non-lithium graph (Section 3.4).

By Algorithm Linear-Label, a DFS spanning tree T of a 2-edge-connected graph G is found and each node x ($x \neq r$) is assigned a *back edge*, i.e., a frond between a descendant of x and an ancestor of it (note that since G is 2-edge-connected, such back edge exists for every node). A message from x to a node y in T_x will traverse the path in T from x to y. A message to a node y with $L(y) > L(x)$ will traverse the unique path in T from x to a common ancestor z of x and y, and then from z down to y. The interesting case is when x has a message destined to a node y that is not in the tree T_x (i.e., y is not a descendant of x) and such that $L(y) < L(x)$. In this case we use the back edges to go up the tree until a common ancestor of x and y is reached. More specifically, let $e = (u, v)$ be the back edge assigned to x; a message from x to y will traverse the path in T from x to u, then will continue on e to v, which is an ancestor of x. If v is an ancestor of y then we get to a previous case (and the message will continue down T towards y), otherwise, it will continue through another back edge (the back edge assigned to v) to an "upper" node in T, and so on, until an

3 A Labeling Algorithm

3.1 Preliminaries

In this section we present a new algorithm, Algorithm Linear-Label, that generates an LIRS for every graph which admits one (i.e., for every connected non-lithium graph). Our algorithm, as the one of [FG94], does not imply any non-trivial upper bound for general non-lithium graphs (this question is still open), but it has the following advantages. First, it is based on a DFS-spanning tree of the graph, thus, it resembles the algorithms for IRS mentioned in Section 1 and is easier to understand and to implement. Second, it guarantees that the path that any message traverses between any two bridge points is a shortest length path (see Example 1). Last, it could be easily modified (as will be shown in the sequel) such that every link of the network (at least in one direction) will be used for routing.

Example 1. As an example consider the graph G depicted in Figure 3. Each G_i, $1 \leq i \leq n$, is a simple cycle of length n, with nodes $u_1^i, u_2^i, ..., u_n^i$, and an additional edge $(u_2^i, u_{n/2+1}^i)$. The edge $(u_{n/2}^i, u_1^{i+1})$ is the bridge that connects G_i and G_{i+1}. Under Algorithm Linear-Label a message between $s \in G_i$ and $t \in G_j$, $i < j$, will traverse a path of length at most $2n + 3(j - i - 1) + (j - i)$ (since the shortest length path between each u_1^i and $u_{n/2}^i$ is of length 3). The algorithm of [FG94] however, will route a messages between each u_1^i and $u_{n/2}^i$ on a cycle that contains both of the nodes. As a consequence, a message between s and t will traverse a path of length at least $n/2 \cdot (j - i - 1) + (j - i)$.

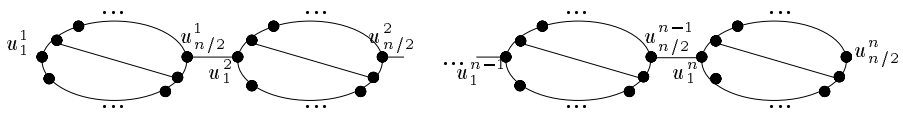

Fig. 3. An example.

Algorithm Linear-Label is based on the theory of DFS (Depth First Search) spanning trees. A DFS spanning tree of a graph is constructed by traversing the graph in a depth first style, backtracking only when there is no unvisited adjacent node. The DFS spanning tree T that is constructed throughout this traversal is a directed tree, rooted at the origin of the traversal. For any node x, we denote by T_x the sub-tree of T rooted at x. Each node in T_x including x is a *descendant* of x. Every node on the path from the root r to x except for x is an *ancestor* of x. Every node x (except for the root) has a predecessor $P_T(x)$ and every non-leaf node has a (non-empty) set of successors $S_T(x)$. A *frond* is

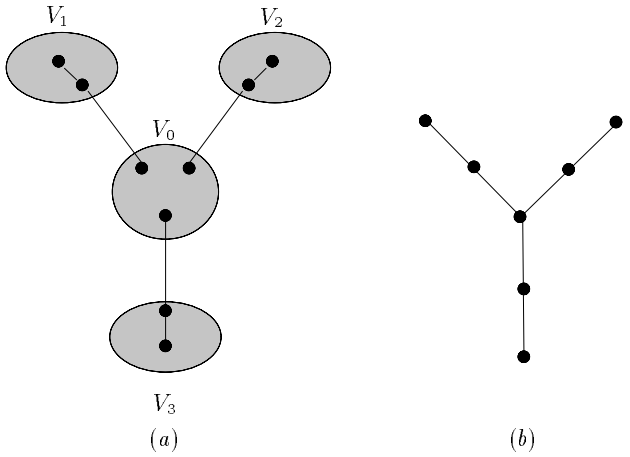

Fig. 1. (a) The general form of a lithium graph. (b) The simplest lithium graph.

Definition 8.

1. The *reduced graph* of a graph G, $R(G)$, is the graph obtained from G by removing all of its leaves and the edges adjacent to them (where a node v is a *leaf* if it has only one adjacent node, that is, $|E_v| = 1$).
2. A 2-*edge-connected graph with leaves* is a graph G for which the reduced graph $R(G)$ is 2-edge-connected (a graph is 2-edge-connected if it does not contain any bridge, note that a graph that contains a single node is 2-edge-connected).

A non-lithium graph $G = (V, E)$ can be represented as a sequence $G_1, (v_1^2, v_2^1), G_2, (v_2^2, v_3^1), G_3, ..., (v_{n-1}^2, v_n^1), G_n$, $n \geq 1$, where $G_i = (V_i, E_i)$ is a 2-edge-connected graph with leaves (Definition 8), $v_1^2 \in V_1$, $v_n^1 \in V_n$, $v_i^1, v_i^2 \in V_i$ for every $1 < i < n$, $V = \cup_{i=1}^{n} V_i$, $E = \cup_{i=1}^{n} E_i \cup \cup_{i=1}^{n-1} \{(v_i^2, v_{i+1}^1)\}$ and each (v_i^2, v_{i+1}^1) is a bridge that connects G_i and G_{i+1}. The vertices $\{v_i^1, v_i^2\}_{1 \leq i \leq n}$ are termed *bridge points*. (See Figure 2.)

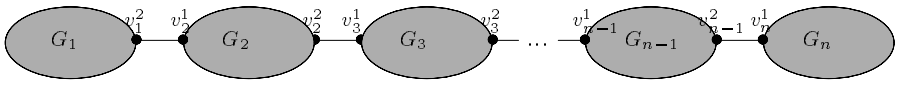

Fig. 2. A non-lithium graph. The G_is are 2-edge-connected graph with leaves and the v_i^js are the bridge points.

(In other words, the non-empty intervals associated with all the edges adjacent to any node v form a partition of N or of $N - \{L(v)\}$.)
For $e = (u, v)$ we will write $I_u(u, v)$ instead of $I_u((u, v))$.

For simplicity, from now on, given an ILS and a node u, we will not distinguish between the node u and its node number $L(u)$.

Two variants of interval labeling scheme are defined as follows.

Definition 3. A *linear interval labeling scheme* (LILS) $\mathcal{L}_G$ of a graph G is an interval labeling scheme in which all the intervals $I_v(e)$ - for every $v \in V$ and every $e \in E_v$ - are either linear or null.

Definition 4. A *strict-linear* interval labeling scheme (strict-LILS) $\mathcal{L}_G$ of a graph $G = (V, E)$ is a linear interval labeling scheme in which $v \notin I_v(e)$ for every $v \in V$ and every $e \in E_v$.

By a *labeling scheme* we mean ILS or any of its two variants. Given a graph with any labeling scheme, the routing of messages is performed as follows. If node u has a message destined to node v, $v \neq u$, then u will send the message on the *unique* edge e such that $v \in I_u(e)$. Obviously, if the labeling scheme is arbitrary, the routing cannot ensure that every message will eventually arrive at its destination; though a message from u to v cannot be stuck at any node, it still can cycle forever without getting to v. We thus introduce the following definition.

Definition 5. A *valid* labeling scheme of a graph is a labeling scheme that guarantees that every message will eventually arrive at its destination.

Given a valid labeling scheme $\mathcal{L}_G$ of G, and two nodes u and v, $Path_G(u, v, \mathcal{L}_G)$ denotes the path along which a message, destined to v, will traverse starting from u, under the labeling scheme $\mathcal{L}_G$. From the definitions it is clear that if the routing is done according to a valid labeling scheme, each message will follow a simple path (in which all nodes are distinct), which implies a trivial upper bound of $n-1$ on the length of such paths. We denote by IRS (Interval Routing Scheme) a valid ILS and correspondingly LIRS and strict-LIRS. A precise characterization of the graphs that admit LIRS was given in [FG94]. To state it, we first need the following definition.

Definition 6. A *lithium graph* is a connected graph $G = (V, E)$, $V = V_0 \cup V_1 \cup V_2 \cup V_3$, such that the sets V_i are disjoint, $|V_1|, |V_2|, |V_3| \geq 2$ and there is a unique edge connecting V_0 with each of the V_i's, $i = 1, 2, 3$, and no edge between any two different V_i and V_j, $1 \leq i, j \leq 3$ (see Figure 1).

Theorem 7. *([FG94]) A (connected) graph G admits an LIRS iff G is not a lithium graph.*

In order to define a general representation of a non-lithium graph we first define:

paths traversed by messages (which depend on the structure of the network and will be completely characterized) are shortest length paths.

In Section 2 we present the model, previous results and a precise description of the routing methods under discussion. In Section 3 we present the algorithm and analyze its correctness. Properties and extensions are discussed in Section 4. Most of the proofs are only briefly sketched or omitted in this Extended Abstract.

2 Preliminaries

We assume a point-to-point asynchronous communication network where processors communicate with their neighbors by exchanging messages along the communication links, which are bidirectional. The network topology is modeled by a (connected) undirected graph $G = (V, E)$, $|V| = n$, where the set V of nodes corresponds to the processors, and the set E of edges corresponds to the communication links connecting them. An edge e connecting u and v is denoted by $e = (u, v)$. For a node v, E_v denotes the set of edges adjacent to v.

Each message has a header that includes its destination. When a message arrives at an intermediate node then the edge on which it will continue is determined by a local routing decision function.

The routing methods under discussion involve a labeling of the nodes and edges of the graph. Each node is labeled with a unique integer in $N = \{1, ..., n\}$, termed *node number*, and at each node v, each of the edges in E_v is labeled with an *interval of* N, defined as follows.

Definition 1. An *interval of* N is one of the following:

1. A *linear interval* $\langle p, q\rangle = \{p, p+1, ...q\}$, where $p, q \in N$ and $p \leq q$.
2. A *wrap-around interval* $\langle p, q\rangle = \{p, p+1, ..., n, 1, ..., q\}$, where $p, q \in N$ and $p > q$.
3. The *null interval* $\langle\rangle$ which is the empty set.

We say that a node $u \in V$ is *contained* in an interval $\langle p, q\rangle$ if $u \in \langle p, q\rangle$. Note that the null interval does not contain any node.

Definition 2. An *interval labeling scheme* (ILS) $\mathcal{L}_G = (L, \{I_v\}_{v \in V})$ of a graph $G = (V, E)$ is defined by:

1. A one-to-one function $L : V \rightarrow N$ that labels the nodes of V.
2. A set of edge labeling functions $I_v : E_v \rightarrow I$, for every $v \in V$, where I is the set of all intervals of N, that satisfy the following properties for every $v \in V$:
   union property the union of all the intervals corresponding to the edges in E_v is equal to N or to $N - \{L(v)\}$.
   disjunction property the intervals $I_v(e_1)$ and $I_v(e_2)$ corresponding to any two edges $e_1, e_2 \in E_v$ are disjoint.

A popular compact routing method is *Interval Routing* which was introduced in [SK82], discussed together with other compact routing methods in [LT83] and implemented on INMOS transputer C104 Router chips [I91]. Under Interval Routing the nodes are labeled with unique integers from the set $\{1, ..., n\}$, where n is the number of nodes in the network, and at each node, each of its adjacent edges is labeled with one interval of $\{1, ..., n\}$. Cyclic intervals (i.e., intervals that wrap-around over the end of the name segment) are allowed. Under such Interval Labeling Scheme (ILS), at each node i, messages destined to node j are sent on the unique edge that is labeled with an interval that contains j. A *valid* ILS of a network is an ILS under which every message will eventually arrive at its destination. A simple algorithm that generates a valid ILS for every network was presented in [SK82]. The algorithm is based on a BFS spanning tree of the network, it implies an upper bound of $2D$ on the length of a path a message traverses, where D is the diameter of the network, but it has the disadvantage that all routings are performed on a tree. [LT83] presented a different algorithm for ILS, based on a DFS spanning tree; though the algorithm does not imply any upper bound on the length of a path traversed by a message, it has the advantage that every link of the network is used for routing. In [R91] a lower bound of $1.5D + 0.5$ was proved for the maximal length of a path a message traverses under Interval Routing, and this bound was improved in [TL94] to $1.75D - 1$.

Linear Interval Routing ([BLT91]) is a restricted variant of Interval Routing which uses only linear intervals (i.e., intervals with wrap-around are not allowed). It is also the simplest form of Prefix Routing which is another compact routing method (introduced in [BLT90]). It was noted in [BLT91] that not every network has a valid Linear Interval Labeling Scheme (LILS). A complete characterization of the networks that admit a valid LILS was presented in [FG94], together with an algorithm that generates a valid LILS in case one exists. The algorithm in [FG94] is incremental in the sense that it starts by labeling a small sub-network and then in each iteration it labels an unlabeled path in the network until all nodes and edges are labeled. The algorithm does not imply any upper bound on the length of a path traversed by a message; indeed, the question of finding a non-trivial upper bound for Linear Interval Routing is still open. In [EMZ96] a lower bound of $\Omega(D^2)$ on the length of a path a message traverses in a network under a valid LILS was presented.

In this paper we present a new algorithm that generates an LILS for every network that admits one. Our algorithm is based on a DFS spanning tree of the network and thus is "in the spirit" of the algorithms for Interval Routing ([SK82, LT83]). In fact, by utilizing the special structure of networks which admit valid LILS, it accomplishes the same benefits as the algorithm in [LT83] for Interval Routing but it does not use wrap-around intervals. Our algorithm has the following advantages over the algorithm of [FG94]: (1) It utilizes the well-known theory of DFS spanning trees, and is thus simpler to understand and to implement, (2) it uses all links of the network (at least in one direction) for routing, thus it better distributes the load, and (3) it guarantees that some

A Simple DFS-Based Algorithm for Linear Interval Routing

T. Eilam, S. Moran*, S. Zaks

Department of Computer Science
The Technion,
Haifa 32000, Israel
email: {eilam,moran,zaks}@cs.technion.ac.il

Abstract. Linear Interval Routing is a space-efficient routing method for point-to-point communication networks. It is a restricted variant of Interval Routing where the routing range associated with every link is represented by an interval with no wrap-around. It was noted in [BLT91] that not every network has a valid Linear Interval Labeling Scheme (LILS). A complete characterization of the networks that admit a valid LILS was presented in [FG94], together with an algorithm that generates a valid LILS in case one exists. We present a new algorithm that generates an LILS for every network that admits one. Our algorithm is based on a DFS spanning tree of the network, and is "in the spirit" of the algorithms for Interval Routing. Our algorithm has few advantages over the algorithm of [FG94]: it utilizes the well-known theory of DFS spanning trees and is thus simpler to understand and to implement, it uses all links of the network for routing (thus it better distributes the load), and it guarantees that some paths traversed by messages are shortest length paths.

1 Introduction

In a communication network, where communication between nodes is accomplished by sending and receiving messages, a routing algorithm is employed to ensure that every message will reach its destination. An *optimal* routing method routes every message to its destination in the shortest way. Usually optimal routing is achieved by keeping in each node a table with n entries and such that the i-th entry in the table determines the edge to be traversed by a message destined to node i. For large networks it is practical to consider routing methods in which less than $\Omega(n)$ space is used in each node, though such routing methods may not be optimal. The trade-off between space and efficiency was extensively studied in the past decade (see, e.g., [PU89, ABLP89, AP92, GP96a, GP96b]) and many *compact* routing methods were developed and analyzed (see, e.g.,[SK82, LT83, BLT90, BLT91, FGS93, FGNT95]).

* This reseaech was supported by the fund for promoting reseach in the Technion, and by the Bernard Elkin Chair in Computer Science.