ACADEMIC PRESS

Available at WWW.MATHEMATICSWEB.ORG POWERED BY SCIENCE DIRECT®

Journal of Algorithms 46 (2003) 97–114

Journal of Algorithms

www.elsevier.com/locate/jalgor

Compact routing schemes with low stretch factor ☆

Tamar Eilam,[a] Cyril Gavoille,[b] and David Peleg [c,*,1]

[a] *IBM T.J. Watson Research Center, Yorktown Heights, NY, USA*
[b] *LaBRI, Université Bordeaux I, 351, cours de la Libération, 33405 Talence cedex, France*
[c] *Department of Computer Science and Applied Mathematics, The Weizmann Institute of Science, Rehovot, 76100 Israel*

Received 28 October 2000

Abstract

This paper presents a routing strategy called *Pivot Interval Routing* (*PIR*), which allows message routing on every weighted n-node network along paths whose *stretch factor* (namely, the ratio between the length of the routing path and the shortest path) is at most five, and whose average stretch factor is at most three, with routing tables of size $O(\sqrt{n}\log^{3/2} n)$ bits per node. In addition, the route lengths are at most $2D$ ($\lceil 1.5D \rceil$ for uniform weights) where D is the weighted diameter of the network. Moreover, it is shown that the PIR strategy can be constructed in polynomial time and can be implemented so that the generated scheme is in the form of an *interval routing scheme* (*IRS*), using at most $O(\sqrt{n}\log n)$ intervals per link. As a result, the schemes are simpler than previous ones and they imply that the paths followed by messages are *loop-free*. On the other hand, we show that there is no loop-free routing strategy guaranteeing a memory bound of at most $\sqrt{n}$ bits per node for all networks, regardless of the route lengths.

1. Introduction

1.1. Background

In point-to-point communication networks, a routing scheme is employed in order to deliver messages between processors. As networks grow in size, it becomes important to

☆ An extended abstract of this paper has appeared in the PODC '98 symposium.

* Corresponding author.
E-mail addresses: eilamt@us.ibm.com (T. Eilam), gavoille@labri.fr (C. Gavoille), peleg@wisdom.weizmann.ac.il, david.peleg@weizmann.ac.il (D. Peleg).

[1] Supported in part by grants from the Israel Science Foundation and from the Israel Ministry of Science and Art.

0196-6774/03/$ – see front matter
doi:10.1016/S0196-6774(03)00002-6

reduce the amount of memory kept in each node for routing purposes. At the same time, it is essential to route messages along paths that are as short as possible. The efficiency of a routing scheme is often measured in terms of its *stretch factor*, namely, the maximum ratio between the length of the path traversed by a message and the length of the shortest path between its source and destination.

A *universal routing strategy* is an algorithm which generates a routing scheme for every given network. One type of trivial universal routing strategy is based on schemes that keep in each node a full routing table, i.e., a table which specifies an output port for every destination. Though this strategy guarantees routing along shortest paths, each router has to store locally $\Theta(n \log d)$ bits of memory, where d is its degree (i.e., the number of output ports) and n is the number of nodes in the network. Therefore, this scheme is impractical when dealing with large networks.

It was shown in a series of papers (see, e.g., [1–3,20,22]) that there is a tradeoff between the memory requirements of a routing scheme and the worst case stretch factor it guarantees. In [20] it was shown that any universal routing strategy achieving stretch factor $s \geqslant 1$ must use a total of $\Omega(n^{1+1/(2s+4)})$ bits of routing information in the network. Stronger lower bounds hold for small stretch factors [7,13,16].

On the positive side, a number of hierarchical routing strategies have been proposed that achieve almost optimal efficiency-space tradeoffs [2,20]. In particular, the scheme presented in [2] guarantees a stretch factor of $O(k^2)$, while using $O(k n^{1/k} \log^2 n \log D)$ memory bits per node, for every $k \geqslant 1$, where D is the weighted diameter of the network. Another strategy presented in [22] guarantees a stretch factor of $O(k)$ and uses $O(n^{1/k} \log^{O(1)} n)$ memory bits per node with $o(k \log^2 n)$ bit names. The major disadvantage of all the proposed hierarchical routing strategies is that they are rather complex. Briefly, headers of messages are re-written, a message can bounce back to the originator, and the decision function in each node is complex and does not depend solely on the destination name.

Subsequently, considerable attention has been given recently to an opposing design philosophy, focusing on *simple* and *uniform* compact routing strategies. Many compact routing strategies were proposed in the last decade (see, e.g., [6,9–11,21,24]). An important property common to all of them is that they employ a simple "transmit and forget" type decision function in the nodes, depending only on the destination of the message, and the destination is the only information coded in the message header (which is determined once and for all by the originating router, and is never changed afterwards).

The most popular such scheme is the *interval routing scheme* (*IRS*). It is presented in [21,24] and implemented in the T9000 Transputer router chip of INMOS. The idea of this scheme is to label nodes with unique integers from $\{1, \ldots, n\}$, and to label the outgoing arcs in every node with a set of destination labels in the form of a set of consecutive intervals of the name segment. The collection of sets that label the outgoing arcs of a node forms a partition of the name segment. When invoking the delivery protocol, a message is sent on the unique outgoing arc labeled by a set that contains the destination label. While the preprocessing stage of such a routing scheme (which is performed once in the initialization of the network) might be complex, the delivery protocol consists of simple decision functions in every node that depends only on the destination and thus implies loop-free routing paths.

One of the desirable goals in interval routing is to minimize the maximum number of intervals that label an arc. Unfortunately, while many lower bounds are known for this problem (see, e.g., [5,14,18,23]), few trade-off results between efficiency and space are known for any of these strategies for general graphs. For interval routing, a universal strategy which is based on routing on a BFS-tree is presented in [21,24]. This scheme uses only one interval per edge, thus the memory in a node with degree d is $O(d \log n)$, it guarantees that the length of routing paths is at most $2D$, where D is the diameter, but it implies no upper bound on the stretch factor. On the other extreme, it is shown in [15] that it is possible to generate for every network an interval routing scheme which uses at most $n/4 + o(n)$ intervals per edge and guarantees stretch factor $s = 1$. Lately, [18] showed that for every graph there exists an interval routing scheme under which every message traverses a path of length at most $\lceil 1.5D \rceil$, and which labels every arc with at most $\sqrt{n \ln n} + O(1)$ intervals. While this result implies an upper bound on the *dilation*, i.e., the length of paths traversed by messages, it does not imply any nontrivial upper bound on the stretch factor. Moreover, the paper does not present any efficient (say, polynomial time) preprocessing algorithm for generating such an interval routing scheme for a given graph. The scheme presented in [4] uses $O(n^{2/3} \log^{4/3} n)$ memory bits per node and guarantees a stretch factor $s = 3$. While the scheme is loop-free, it is not an interval routing scheme, and it uses $3 \log n$ bit names.

1.2. Our results

A basic question which arises from the above discussion is whether it is possible to design "simple" near-optimal routing strategies, which are still compact. By "simple" we mean routing schemes in which the header is not allowed to be rewritten. Further, the header contains only the destination of the message (where destinations are in $\{1, \ldots, n\}$) and the decision function depends only on the header (and not on the incoming port number for example). We term routing methods that obey these restrictions *direct*. Note that all the proposed compact routing strategies (e.g., interval routing) are direct. In this paper we analyze the power of direct routing strategies and give both positive and negative results for this question.

Our main results are two polynomial time constructible direct universal routing strategies, termed *pivot interval routing* (*PIR*, in short). The first one, PIR1, generates for every weighted graph with arbitrary link costs, a routing scheme with stretch factor $s \leqslant 5$ that uses $O(\sqrt{n} \log^{3/2} n)$ bits per node, and requires $O(\log n)$ latency (defined as the time required to extract the outgoing link on which a message is to be forwarded in a node). Also, we show that the average stretch factor is $\bar{s} \leqslant 3$. Moreover, the PIR1 preprocessing algorithm actually generates for every graph an interval routing scheme which labels every arc with at most $\alpha \sqrt{n \log n}$ intervals, where $\alpha \approx 1.17$. The dilation guaranteed by the PIR1 strategy is $2D$, where D is the weighted diameter of the network. For the unweighted case, we present a slightly different universal routing strategy named PIR2 (which also generates for every graph an interval routing scheme with still $\beta \sqrt{n \log n}$ intervals per arc, where $\beta \approx 2.00$), achieving the same memory requirements, stretch factor and average stretch factor as the PIR1 algorithm while guaranteeing a better dilation bound of $\lceil 1.5D \rceil$. We also provide a lower bound of $\sqrt{n}$ on the number of bits kept locally in a node for

every loop-free routing scheme that uses names from the range $\{1, \ldots, n\}$ and for every stretch factor. Thus our routing strategy cannot be generalized to obtain a family of routing schemes with different values of stretch factor and memory such as the hierarchical routing schemes (e.g., [1,22]).

A direction remaining for future research is to establish the best trade-off between the memory requirements and the (average) stretch factor for direct routing schemes using node labels in a range $\{1, \ldots, m\}$, with $m > n$.

The rest of the paper is organized as follows. In Section 2 we give a precise definition of routing schemes and efficiency measures. We formally define interval routing and in addition we overview some covering techniques, adopted from [1]. We make use of these techniques for the proposed routing strategy. In Section 3 we present our PIR routing strategies, prove their correctness and analyze their complexity measures. The lower bound on the power of loop-free routing schemes is proved in Section 4.

2. Model and definitions

2.1. Routing schemes

A point-to-point communication network is modeled as a symmetric, weighted, finite digraph $G = (V, E, \omega)$, $|V| = n$, where the set of nodes represent the processors of the network and every pair of two opposite arcs represents a bidirectional communication link. Every arc of the network $e \in E$ is associated with a nonnegative weight $\omega(e)$ (i.e., its cost) defining a metric. We assume that for every two opposite arcs e_1 and e_2, $\omega(e_1) = \omega(e_2)$. (Our algorithm will not work correctly for the case of asymmetric weights. Whether it can be generalized to handle that case requires further research.) In the special case of uniform unit weight links, we say that the graph is *unweighted*, and denote it simply by $G = (V, E)$. Graphs are connected and do not contain self-loops or multiple arcs. We assume that every node v is named with a unique identity integer. In what follows, we informally use the node v and its unique identity integer interchangeably. Note that the identities induce a total order on the nodes, thus for every two nodes $u, v \in V$ either $u < v$ or $v < u$.

The length of a directed path in the graph is the sum of weights of its arcs. The *distance* $d_G(u, v)$ between two nodes $u, v \in V$ is the length of a shortest path connecting them. The *diameter* of the graph G is defined as $\max\{d_G(u, v) \mid u, v \in V\}$. For a node $v \in V$, let E_v denote its set of outgoing arcs, and denote its *degree* by $\deg(v) = |E_v|$.

A *routing scheme* R is a distributed algorithm whose role is to deliver messages between nodes of the network. The routing scheme consists of certain *distributed data structures* in the network, and a *delivery protocol*, which can be invoked in any node u with two parameters: a *routing label* of the destination node v, and the message's information field. The message is delivered to v via a sequence of transmissions determined uniquely by the distributed data structure.

The length of the route traversed by a message from u to v in the graph G according to the routing scheme R is denoted by $d_R(u, v)$. A *universal routing strategy* is a function that returns for every graph G a routing scheme on G. It is implemented by a *preprocessing*

algorithm, performed during set-up time in order to construct the distributed data structures and the labels required for the routing scheme.

An *interval routing scheme* R on G is a routing scheme consisting of a pair $(\mathcal{L}, \mathcal{I})$, generated in the preprocessing step, where $\mathcal{L}$ is a *node-labeling*, $\mathcal{L}: V \to \{1, \ldots, n\}$, and $\mathcal{I}$ is an *arc-labeling*, $\mathcal{I}: E \to 2^{\mathcal{L}(V)}$, that satisfy the following condition. For any node u, the collection of sets that label all the outgoing arcs of u forms a partition of the name range (possibly excluding u itself [2]). Formally, for every $u \in V$,

(1) $\bigcup_{e \in E_u} \mathcal{I}(e) \cup \mathcal{L}(u) = \{1, \ldots, n\}$;
(2) $\mathcal{I}(e_1) \cap \mathcal{I}(e_2) \subseteq \mathcal{L}(u)$ for every two distinct arcs $e_1, e_2 \in E_u$.

The delivery protocol is defined as follows. In every node u, a message with destination v ($\mathcal{L}(v)$ written in its header) is sent on the arc which is labeled by a set that contains the destination label (namely, $\mathcal{L}(v) \in \mathcal{I}(u, v)$).

We denote by IRS the class of all the interval routing schemes on arbitrary graphs. Clearly, every interval routing scheme is a direct routing scheme. Conversely, note that every direct routing scheme can be implemented using interval routing by labeling every outgoing arc in a node with the set of destinations for which a message will be sent on that arc (encoded using a set of intervals). (This is not always possible for routing schemes that are not direct since the output port does not necessarily depend only on the destination.)

2.2. Complexity measures

Let R be a routing scheme on an n-node graph G. Given a node u, the *memory requirement* of u, denoted by $\text{Memory}_G(R, u)$, is the smallest number of bits that are required in order to code R in u. The *latency* of R in u, denoted by $\text{Latency}_G(R, u)$, is the time complexity of R per node in the standard $O(\log n)$-word RAM-model. It corresponds to the time required to extract from R the outgoing link on which the message is forwarded in u. The maximum and average stretch factors of R are respectively defined as

$$\text{Stretch}_G(R) = \max_{u \neq v} \left\{ \frac{d_R(u, v)}{d_G(u, v)} \right\} \quad \text{and} \quad \text{AvStr}_G(R) = \frac{1}{n(n-1)} \sum_{u \neq v} \frac{d_R(u, v)}{d_G(u, v)}.$$

A routing scheme of stretch factor 1 is termed a *shortest path* routing scheme. The *dilation* of a routing scheme R is the maximal length of a path traversed by a message. Formally,

$$\text{Dilation}_G(R) = \max_{u \neq v} \{ d_R(u, v) \}.$$

Given an integer n and a subset $I \subseteq \{1, \ldots, n\}$, define the *compactness of I w.r.t. n*, denoted $c_n(I)$, as the smallest integer k such that I can be represented by the union of k intervals $[a, b]$ of consecutive integers from $\{1, \ldots, n\}$, with n and 1 being considered as consecutive (cyclically). The *compactness* of an interval routing scheme $R = (\mathcal{L}, \mathcal{I})$ on G, denoted by $\text{Comp}_G(R)$, is the maximum, over all arcs $e \in E$, of the compactness

[2] A labeling excluding u from its arc-labels is termed *strict*. Although nonstrict labeling may produce more compact schemes, in this paper we restrict our attention to strict labeling only.

$c_n(\mathcal{I}(e))$ of the set $\mathcal{I}(e)$ labeling e. Intuitively, smaller compactness and degrees imply smaller routing tables. For example, the interval routing strategy presented in [21], based on routing on a minimum spanning tree, has compactness 1 (for every graph) but unbounded stretch factor.

2.3. Balls, neighborhoods, and covers

Our routing scheme constructions are based on the notions of neighborhoods, balls, and covers. For every node v we can order all the nodes of the graph w.r.t. v by increasing distance from v, breaking ties by increasing node identities. Formally, $x \prec_v y$ if and only if either $d_G(x,v) < d_G(y,v)$, or $d_G(x,v) = d_G(y,v)$ and $x < y$. The *t-ball* $\mathcal{B}_v(t)$ of v, is the set of the first t nodes according to the node ordering $\prec_v$. The *r-neighborhood* of a node $v \in V$ is defined as $\Gamma(v,r) = \{u \in V \mid d_G(v,u) \leqslant r\}$. Hence intuitively, a ball is a neighborhood defined by volume rather than by radius.

Following is a simple fact which holds for both neighborhoods and balls.

Fact 2.1 (monotonicity). *If $u \in \mathcal{B}_v(t)$ (respectively, $u \in \Gamma(v,r)$) then for every node x on a shortest path from v to u, $u \in \mathcal{B}_x(t)$ (respectively, $u \in \Gamma(x,r)$).*

Consider a collection $\mathcal{H}$ of subsets of size t of elements from a set $\mathcal{V}$. A set $P \subseteq \mathcal{V}$ is said to *cover* the collection $\mathcal{H}$ if for every $A \in \mathcal{H}$, $A \cap P \neq \emptyset$. We review two techniques presented in [1] for generating relatively small covers for a given collection of sets of equal size. The first technique is by using a greedy algorithm that starts with $P = \emptyset$ and iteratively adds to the set P an element in $\mathcal{V}$ occurring at the most uncovered sets. The algorithm stops when P becomes a cover. The set P is termed a *greedy cover* for $\mathcal{H}$.

Lemma 2.2 (Lovász [19]). *Let P be a greedy cover for $\mathcal{H}$. Then $|P| < |\mathcal{V}|(\ln|\mathcal{H}| + 1)/t$.*

The second method is randomized, and takes each element of $\mathcal{V}$ to the set P with probability $(c \ln|\mathcal{H}|)/t$, for some constant $c > 1$.

Lemma 2.3 (Awerbuch et al. [1]). *Let P be the set constructed by the randomized cover algorithm under the assumptions that $|\mathcal{V}| \geqslant 2t$ and $\ln|\mathcal{H}| = o(t)$. Then with probability at least $1 - 1/|\mathcal{H}|^{c-1}$, P is a cover for $\mathcal{H}$ and $|P| \leqslant (2c|\mathcal{V}|\ln|\mathcal{H}|)/t$.*

For our needs, $|\mathcal{H}| = |\mathcal{V}| = n$. In this paper, we are interested in the r-neighborhoods of nodes only for the case $r = \lceil D/2 \rceil$, where D is the diameter of the graph, and only for unweighted graphs. In particular, we would like to use such neighborhoods in order to construct a small $\lceil D/2 \rceil$-dominating set for our graph in the case where the graph is unweighted (PIR2).

For every node v_i, $1 \leqslant i \leqslant n$, let $W_i = \Gamma(v_i, \lceil D/2 \rceil)$. Note that a cover for the set family $\mathcal{W} = \{W_1, \ldots, W_n\}$ (i.e., a set $X \subseteq V$ whose intersection with each W_i is nonempty) is a $\lceil D/2 \rceil$-dominating set for the graph. Also, note that (since the weights on the graph arcs are uniform), $W_i \cap W_j$ is nonempty for every $i \neq j$, since otherwise $d_G(v_i, v_j) > D$. It is therefore easy to verify that $\mathcal{W}$ has a cover of cardinality $O(\sqrt{n \log n})$. Note that this

cannot be deduced directly from Lemma 2.2, since $|W_i|$ is not necessary in $\Theta(\sqrt{n\log n})$. Nevertheless, such a cover X can be constructed by the following algorithm.

First, note that since $W_i \cap W_j$ is nonempty for every $i \neq j$, each set W_i is in itself a cover for the set family $\mathcal{W}$. Hence if there exists some $1 \leqslant i \leqslant n$ such that the set W_i of cardinality $|W_i| < \sqrt{n(1+\ln n)}$ then we take the set $X = W_i$ as our cover and we are done. So now suppose that $|W_i| \geqslant \sqrt{n(1+\ln n)}$ for every $1 \leqslant i \leqslant n$. In this case, the result follows from observing that Lemma 2.2 holds also for a collection of sets of nonequal sizes, provided t is a *lower bound* on the size of the sets in the collection. Hence the cover X is found by setting $t = \sqrt{n(1+\ln n)}$ and applying the above greedy algorithm. We thus have the following.

Lemma 2.4. *For every n-node unweighted graph G there exists a $\lceil D/2 \rceil$-dominating set X of cardinality $|X| < \sqrt{n(1+\ln n)}$. Moreover, this set can be constructed by a polynomial-time algorithm* (*described above*).

3. The pivot interval routing strategy

We present two routing strategies, termed *Pivot Interval Routing*. The first one, PIR1, generates an interval routing scheme for every weighted graph (Theorem 3.1) and the second one, PIR2, generates an interval routing scheme for every unweighted graph with improved dilation (Theorem 3.11).

Theorem 3.1. *For every n-node weighted graph $G = (V, E, \omega)$ with weighted diameter D there exists a interval routing scheme $R = (\mathcal{L}, \mathcal{I})$ on G such that*

(1) $\mathrm{Memory}_G(R, u) = O(\sqrt{n}\log^{3/2} n)$ *for every* $u \in V$,
(2) $\mathrm{Latency}_G(R, u) = O(\log n)$ *for every* $u \in V$,
(3) $\mathrm{Stretch}_G(R) \leqslant 5$,
(4) $\mathrm{AvStr}_G(R) \leqslant 3$,
(5) $\mathrm{Dilation}_G(R) \leqslant 2D$, *and*
(6) $\mathrm{Comp}_G(R) \leqslant \alpha\sqrt{n\log n}$, *where* $\alpha \approx 1.17$.

Moreover, R can be constructed in time polynomial in n.

Intuitively, the idea of the PIR1 strategy is first to find the collection of t-balls of all the nodes of the graph, then to cover this collection by a (comparatively small) set of nodes termed *pivots*, and finally to label the nodes and arcs of the graph in such a way that a message with source u and destination v will be routed on a shortest path if the destination v is in u's t-ball. Otherwise, it will traverse (in the worst case) a shortest path to the pivot nearest to v, and then a shortest path from that pivot to v itself.

We present the preprocessing algorithm of the PIR1 strategy for any given ball size t and for any given cover P of the collection of t-balls of nodes in the graph. The value of t is determined later (to roughly $O(\sqrt{n\log n})$), in the proof of Theorem 3.1, where

it is also shown how to construct a cover P such that PIR1 will satisfy the properties in Theorem 3.1.

The preprocessing algorithm consists of three parts. In the first part, some preliminary structures are constructed which are used later (in the second and third parts) to define the node and arc labeling functions $\mathcal{L}$ and $\mathcal{I}$.

3.1. Strategy PIR1: preprocessing and delivery protocol

We consider a weighted graph $G = (V, E, \omega)$. t, the size of the balls used, is a parameter. We construct the interval routing scheme $R = (\mathcal{L}, \mathcal{I})$ as follows.

3.1.1. The preprocessing algorithm

Preliminary constructions.

(1) Let P be a cover for the collection of t-balls of the nodes, $\{\mathcal{B}_v(t)\}_{v\in V}$. Let $\ell = |P|$.
(2) Assign to every node v its pivot $p(v) \in P$, where $p(v)$ is the nearest node to v in P (breaking ties by increasing node identities).
(3) For every pivot $p \in P$, let $S_p = \{v \mid p(v) = p\}$ be the set of nodes having p as their pivot. (Observation 3.2 shows that $\{S_p\}_{p\in P}$ is a partition.)
(4) For every pivot $p \in P$, construct a minimum weight BFS spanning tree T_p rooted at p, and spanning the entire graph G (namely, in T_p the unique path between any node and p is a shortest path). Let $\widehat{T}_p$ be the subgraph of T_p induced by p and its set S_p. (Observation 3.2 shows that the subgraph $\widehat{T}_p$ is actually a tree.)

Labeling the nodes. Assume that $P = \{p_1, \ldots, p_\ell\}$. We start by labeling the nodes in S_{p_1}. The labeling is performed by traversing the tree $\widehat{T}_{p_1}$ (i.e., the subtree spanning S_{p_1}), and assigning the nodes of $\widehat{T}_{p_1}$ a DFS (pre-order) numbering in sequential ascending order, starting from 1. In order to give an efficient implementation of the scheme with low memory and low latency (see Proposition 3.10), we impose a DFS so that, at any node x of $\widehat{T}_{p_1}$, the children of x are visited in a nondecreasing order of their number of descendants in the subtree. Once all the nodes of S_{p_1} have been labeled, we continue by labeling the nodes in S_{p_2} in the same manner (traversing the tree $\widehat{T}_{p_2}$), starting from the integer $1 + |S_{p_1}|$. Then we label the nodes of $S_{p_3}, \ldots, S_{p_\ell}$ in the same way, provided the node labeling $\mathcal{L}$.

Labeling the arcs. For every node $x \in V$, we label every arc $e \in E_x$ by a set of destinations $\mathcal{I}(e) \subseteq \{1, \ldots, n\}$ in three main steps. We start by fixing $\mathcal{I}(e) = \emptyset$ for every $e \in E_x$, and then at each step we add some node labels to the sets $\{\mathcal{I}(e)\}_{e\in E_x}$, such that labels are never deleted from the sets $\mathcal{I}(e)$, and the sets are mutually disjoint throughout the process. Define for a set $A \subseteq V$, $\mathcal{L}(A) = \{\mathcal{L}(a) \mid a \in A\}$. Formally, we label the arcs E_x of every node x as follows.

(0) Fix $\mathcal{I}(e) = \emptyset$, for every $e \in E_x$.
(1) If x is not a leaf in the tree $\widehat{T}_{p(x)}$, let $s_1, \ldots, s_j$ be its successors in $\widehat{T}_{p(x)}$ and let $\widehat{T}_{s_i}$ be the subtree of $\widehat{T}_{p(x)}$ rooted at s_i, for $1 \leqslant i \leqslant j$. Assign $\mathcal{I}(x, s_i) = \{\mathcal{L}(v) \mid v \in \widehat{T}_{s_i}\}$, for every $1 \leqslant i \leqslant j$. Define $L_1 = \bigcup_{1\leqslant i\leqslant j} \mathcal{I}(x, s_i)$. If x is a leaf, $L_1 = \emptyset$.

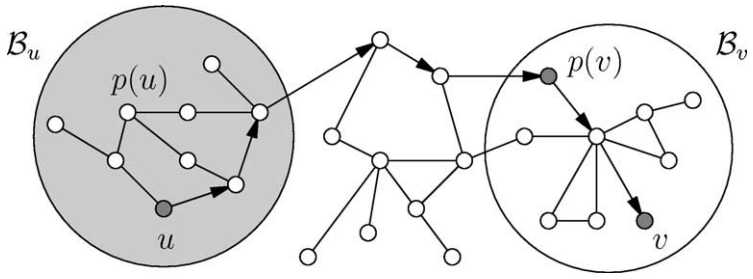

Fig. 1. The route from u to a destination v which is not in u's ball.

(2) Define $L_2 = \mathcal{L}(\mathcal{B}_x(t)) \setminus L_1$. For every node $v \neq x$ such that $\mathcal{L}(v) \in L_2$, let $e \in E_x$ be an arc on a shortest path from x to v (if there is more than one such arc, choose one arbitrarily). Assign $\mathcal{I}(e) = \mathcal{I}(e) \cup \{\mathcal{L}(v)\}$.
(3) Let $L_3 = \mathcal{L}(V) \setminus (L_1 \cup L_2)$. For every $p \in P$, let $e_p \in E_x$ be the arc from x to its predecessor on the tree T_p. Assign $\mathcal{I}(e_p) = \mathcal{I}(e_p) \cup (\mathcal{L}(S_p) \cap L_3)$.

3.1.2. The delivery protocol

In a node x, a message M with destination $v \neq x$ is sent on the unique arc $e \in E_x$, such that $\mathcal{L}(v) \in \mathcal{I}(e)$. Figure 1 depicts the path of a message on the spanning tree $T_{p(v)}$ from a source u to a destination v which is not in u's ball.

3.2. Analysis of PIR1

We now prove that the PIR1 preprocessing algorithm generates an interval routing scheme that satisfies (for specific possible values of t and ℓ) the properties stated in Theorem 3.1.

We start with some simple observations.

Observation 3.2. (1) *For every node $v \in V$, $p(v) \in \mathcal{B}_v(t)$.*

(2) *For every node $v \in V$, and for every node x on a shortest path from v to its pivot $p(v)$, $p(x) = p(v)$.*

(3) *$\{S_p\}_{p \in P}$ is a partition of the set of nodes V.*

(4) *For every pivot $p \in P$, $\widehat{T}_p$ is a connected tree.*

Proof. The first observation follows from the definition of P as a cover of the collection of t-balls and the choice of the pivot $p(v)$ as the nearest pivot for every node v. The second one is easily verified by the monotonicity property (Fact 2.1), and the choice of $p(v)$ for every v. The third observation follows from the definition of the sets S_p, and the last observation is easily verified using the definition of the tree $\widehat{T}_p$, the sets S_p, and Observation 3.2(2). □

In every node x, define a level function on the labels of all other nodes. A label $\mathcal{L}(v)$ of a node $v \neq x$ has level i in x, $i \in \{1, 2, 3\}$, if $\mathcal{L}(v) \in L_i$, i.e., if it was inserted to a set $\mathcal{I}(e)$, for some $e \in E_x$, in Step i of the arc labeling algorithm. The *level of a message M in a*

node x is the level of the label of its destination in x. We first prove that the output of the PIR1 preprocessing algorithm for every weighted graph G is an interval routing scheme on G, and then analyze the routes taken by the messages.

Proposition 3.3. *$R = (\mathcal{L}, \mathcal{I})$ is an interval routing scheme on G. Specifically,*

(a) *For every $x \in V$, $\{\mathcal{I}(e) \mid e \in E_x\} \cup \mathcal{L}(x) = \{1, \ldots, n\}$.*
(b) *For every two distinct arcs (x, y) and (x, z), $\mathcal{I}(x, y) \cap \mathcal{I}(x, z) = \emptyset$.*
(c) *For every two nodes $u, v \in V$, there exists a sequence of nodes $u = x_1, \ldots, x_r = v$ such that $\mathcal{L}(v) \in \mathcal{I}(x_i, x_{i+1})$ for every $1 \leqslant i < r$, namely, every message M with source u and destination v eventually arrives at its destination.*
Moreover, the message M will traverse a shortest path from u to v if $v \in \mathcal{B}_u(t)$. Otherwise, it will traverse a shortest path to $p(v)$ or a prefix of that path, and then a shortest path to v.

Proof. In every step of the arc labeling algorithm, every label which is added to one of the sets $\mathcal{I}(e)$ did not belong to any of the sets in any former step. In addition, it is clear that in every step a label is inserted only to one of the sets $\mathcal{I}(e)$. Note that in Step 3 we rely on the fact that $\{S_p\}_{p \in P}$ is a partition of V. Proposition 3.3(b) follows. Since every label of a node belongs to a (unique) set S_p for some $p \in P$, Step 3 of the arc labeling algorithm guarantees that every node label will be contained in one of the sets $\mathcal{I}(e)$. Proposition 3.3(a) follows.

In order to prove Proposition 3.3(c), we show that the sequence of levels of a message on the path traversed by it is nonincreasing. In addition, if the level of a message in a node is 1 then eventually the destination is reached and if it is 2 or 3 then eventually the destination is reached or the level of the message decreases.

Consider a message M with source u and destination v. Assume that its level in a node $x \neq v$ is 1. $\mathcal{L}(v) \in L_1$ in x implies that x is not a leaf in the tree $\widehat{T}_{p(x)}$. Let s_i be the successor of x in the tree $\widehat{T}_{p(x)}$ such that v belongs to the subtree $\widehat{T}_{s_i}$ of the tree $\widehat{T}_{p(x)}$. The message is delivered to s_i which is strictly closer to v than x and the same condition holds for s_i. Thus in s_i, M is closer to v and has level 1, or the destination v is reached.

Now assume that M has level 2 in a node x. It follows that $v \in \mathcal{B}_x(t)$. Assume that $\mathcal{L}(v) \in \mathcal{I}(x, y)$. Then since (x, y) is on a shortest path from x to v, by Observation 3.2, $v \in \mathcal{B}_y(t)$ and y is strictly closer to v than x. It follows that either the level of M in y is 1, and the claim follows by reduction to the former case, or the level remains 2. In any case, since we advance towards v, it is easy to verify by induction that eventually v is reached in this case as well.

Finally, assume that M has level 3 in a node x. Assume that $\mathcal{L}(v) \in \mathcal{I}(x, y)$. Then it is clear by the PIR1 preprocessing algorithm, that (x, y) is the arc in the tree $T_{p(v)}$ from x to its predecessor. It follows that y it is strictly closer to $p(v)$ and either the same conditions as in the previous step hold or the level is smaller than 3 (either 2 or 1). Thus eventually the destination is reached.

Note that if the level of a message in a node x is 2 or 1 then it is sent along an arc which is on a shortest path to the destination. Thus in case the destination belongs to the source's t-ball, the message will traverse a shortest path. If the level of a message in a node is 3 then

the message is sent on an arc which is on a shortest path to the destination's pivot, thus in this case a shortest path to the destination's pivot is traversed until the level decreases. Since when $p(v)$ is reached the level of M decreases to 1, M will traverse in this case a shortest path to $p(v)$ or some prefix of that path, and then it will traverse a shortest path to v. Proposition 3.3(c) follows. □

By Proposition 3.3(c) we have

Corollary 3.4. *For every $u, v \in V$, $d_R(u, v) = d_G(u, v)$ if $v \in \mathcal{B}_u(t)$, and $d_R(u, v) \leqslant d_G(u, p(v)) + d_G(p(v), v)$ otherwise.*

We now analyze the properties of the routing scheme R generated by the PIR1 strategy. Let D be the weighted diameter of G. By Corollary 3.4, $d_R(u, v) \leqslant 2D$ for every $u, v \in V$. Therefore we have

Proposition 3.5. $\text{Dilation}_G(R) \leqslant 2D$.

Proposition 3.6. $\text{Stretch}_G(R) \leqslant 5$.

Proof. Consider a message M with source u and destination v. By Corollary 3.4, in case $v \in \mathcal{B}_u(t)$, the stretch factor is 1. It remains to bound the stretch factor in case $v \notin \mathcal{B}_u(t)$. The length of the path traversed by the message M in this case is bounded by Corollary 3.4 as $d_R(u, v) \leqslant d_G(u, p(v)) + d_G(p(v), v)$. Since $p(u) \in \mathcal{B}_u(t)$ and $p(v) \notin \mathcal{B}_u(t)$, necessarily $d_G(u, p(u)) \leqslant d_G(u, v)$. Thus, by the triangle inequality, $d_G(v, p(u)) \leqslant d_G(v, u) + d_G(u, p(u)) \leqslant 2\,d_G(u, v)$. Since $p(v)$ is the pivot minimizing the distance to v among all pivots in P, $d_G(v, p(v)) \leqslant d_G(v, p(u)) \leqslant 2\,d_G(u, v)$. Finally, $d_G(u, p(v)) \leqslant d_G(u, v) + d_G(v, p(v)) \leqslant 3\,d_G(u, v)$. It follows that

$$d_R(u, v) \leqslant d_G\big(u, p(v)\big) + d_G\big(p(v), v\big) \leqslant 5\,d_G(u, v). \qquad \square$$

Proposition 3.7. $\text{AvStr}_G(R) \leqslant 3$.

Proof. The claim is established by showing that for every two distinct nodes $u, v \in V$,

$$\frac{d_R(u, v)}{d_G(u, v)} + \frac{d_R(v, u)}{d_G(v, u)} \leqslant 6.$$

We consider three cases.

Case 1 ($u \in \mathcal{B}_v(t)$ and $v \in \mathcal{B}_u(t)$). Then a message from u to v and a message from v to u will both traverse a shortest path. Thus, the sum of the stretch factors of both paths is 2.

Case 2 ($u \in \mathcal{B}_v(t)$ but $v \notin \mathcal{B}_u(t)$ (or vice-versa)). Then a message from u to v will traverse, by Proposition 3.6, a path of length at most $5\,d_G(u, v)$, and the message from v to u will traverse a shortest path. Thus, the sum of the stretch factors is at most 6.

Case 3 ($u \notin \mathcal{B}_v(t)$ and $v \notin \mathcal{B}_u(t)$). We bound the stretch factor of the path of a message from, say, u to v, and the same bound holds symmetrically for the path of a message from v to u. Since $u \notin \mathcal{B}_v(t)$, and recalling that $p(v) \in \mathcal{B}_v(t)$, we have that $d_G(p(v), v) \leqslant d_G(u, v)$. It follows that $d_G(u, p(v)) \leqslant d_G(u, v) + d_G(v, p(v)) \leqslant 2\,d_G(u, v)$. Thus, $d_R(u, v) \leqslant d_G(u, p(v)) + d_G(v, p(v)) \leqslant 3\,d_G(u, v)$. Symmetrically, $d_R(v, u) \leqslant 3\,d_G(u, v)$. Thus, in this case as well, the sum of the stretch factors of the path of a message from u to v and from v to u is at most 6.

It follows that

$$\begin{aligned}\mathrm{AvStr}(R) &= \frac{1}{n(n-1)} \sum_{u \neq v} \frac{d_R(u, v)}{d_G(u, v)} = \frac{1}{n(n-1)} \sum_{u<v} \left(\frac{d_R(u, v)}{d_G(u, v)} + \frac{d_R(v, u)}{d_G(v, u)} \right) \\ &\leqslant \frac{1}{n(n-1)} \sum_{u<v} 6 \leqslant 3. \qquad \square\end{aligned}$$

Proposition 3.8. $\mathrm{Comp}_G(R) \leqslant t + \ell/2 + 1$.

Proof. We use the following facts, which are easily verified, in order to bound the compactness of a set $\mathcal{I}(e)$ that labels an arc e.

Fact 3.9. *For every two subsets* $A, B \subseteq \{1, \ldots, n\}$,

(a) $c_n(A) \leqslant \min\{|A|, n/2\}$.
(b) $c_n(A \cup B) \leqslant c_n(A) + c_n(B)$.
(c) $c_n(A \setminus B) \leqslant c_n(A) + c_n(B)$.

Consider any arc $e = (x, y)$. In order to prove $c_n(\mathcal{I}(e)) \leqslant t + \ell/2 + 1$, let us define $L_i(e) = \mathcal{I}(e) \cap L_i$ to be the set of labels assigned to e with level i in x, $i \in \{1, 2, 3\}$. By construction, $c_n(\mathcal{I}(e)) = c_n(L_1(e) \cup L_2(e) \cup L_3(e))$. Thus, by Fact 3.9(b),

$$c_n\big(\mathcal{I}(e)\big) \leqslant c_n\big(L_1(e)\big) + c_n\big(L_2(e) \cup L_3(e)\big). \tag{1}$$

By the DFS ordering of $\widehat{T}_{p(x)}$, we have $c_n(L_1(e)) \leqslant 1$ and $c_n(L_1) \leqslant 1$ as well. (We have $c_n(L_1(e)) = 0$ if y is a predecessor of x in $T_{p(x)}$, and $L_1 = \emptyset$ if x is a leaf of $\widehat{T}_{p(x)}$.)

Assume that e is the arc from x to its predecessors in the trees $T_{p_{i_1}}, \ldots, T_{p_{i_r}}$, and let $I = \{i_1, \ldots, i_r\}$ be the set of pivot's indices. Note that $I \subseteq \{1, \ldots, \ell\}$. Finally, set $\mathcal{S}_e = \bigcup_{i \in I} \mathcal{L}(S_{p_i})$.

Step 3 of the arc-labeling implies that $L_3(e) = \mathcal{S}_e \cap L_3 = \mathcal{S}_e \setminus (L_1 \cup L_2) = (\mathcal{S}_e \setminus L_1) \setminus L_2$. We can also write $L_2(e) \cup L_3(e) = L_2(e) \cup ((\mathcal{S}_e \setminus L_1) \setminus (L_2 \setminus L_2(e)))$. We have $c_n(L_2 \setminus L_2(e)) \leqslant |L_2 \setminus L_2(e)| \leqslant |L_2| - |L_2(e)|$ using Fact 3.9(a), and that $L_2(e) \subseteq L_2$. Thus, by Fact 3.9, (a)–(c),

$$\begin{aligned}c_n\big(L_2(e) \cup L_3(e)\big) &= c_n\big(L_2(e) \cup \big((\mathcal{S}_e \setminus L_1) \setminus \big(L_2 \setminus L_2(e)\big)\big)\big) \\ &\leqslant c_n\big(L_2(e)\big) + c_n(\mathcal{S}_e \setminus L_1) + c_n\big(L_2 \setminus L_2(e)\big) \\ &\leqslant \big|L_2(e)\big| + c_n(\mathcal{S}_e) + c_n(L_1) + |L_2| - \big|L_2(e)\big| \\ &\leqslant c_n(\mathcal{S}_e) + 1 + |L_2|.\end{aligned}$$

Thus Eq. (1) becomes $c_n(\mathcal{I}(e)) \leqslant c_n(\mathcal{S}_e) + |L_2| + 2$. Note that $|L_2| \leqslant |\mathcal{B}_x(t) \setminus \mathcal{L}(x)| \leqslant t - 1$. Moreover, since $c_n(\mathcal{L}(S_p)) = 1$ for each $p \in P$, and $\{\mathcal{L}(S_p)\}_{p\in P}$ is a partition of $\{1,\ldots,n\}$, we have that $c_n(\bigcup_{i\in I} \mathcal{L}(S_{p_i})) = c_\ell(I)$. Thus by Fact 3.9(a), $c_n(\mathcal{S}_e) = c_\ell(I) \leqslant \ell/2$, as $I \subseteq \{1,\ldots,\ell\}$. Therefore, $c_n(\mathcal{I}(e)) \leqslant t + \ell/2 + 1$, as required. □

Proposition 3.10. $\text{Memory}_G(R,u) = O((\sqrt{n} + t + \ell)\log n)$, *and* $\text{Latency}_G(R,u) = O(\log n)$, *for every node* $u \in V$.

Proof. Let $u \in V$, and let $d = \deg(u)$. Set $K = \sum_{e\in E_u} c_n(\mathcal{I}(e))$ be the total number of intervals assigned to the outgoing arcs e of u. Naively, $\text{Memory}_G(R,u) \leqslant O(K\log n)$ as follows. We store in a node u its label $\mathcal{L}(u)$, and for each interval, the two boundaries (using $O(\log n)$ bits) and the output port number associated with the arc labeled by this interval (using $O(\log d)$ bits). In total we need $O(\log n)$ bits for every interval. Moreover, this simple data structure allows a latency of $O(\log n)$ assuming a binary search in the set of intervals that are sorted according to their left boundary.

Decompose K into $K = K_1 + K_2 + K_3$, with $K_i = \sum_{e\in E_u} c_n(L_i(e))$, where $L_i(e)$ is the set of labels assigned to e at Step i of the arc-labeling algorithm ($i = 1, 2, 3$), and $L_i = \bigcup_{e\in E_u} L_i(e)$ (cf. Proposition 3.8). By the proof of Proposition 3.8, $K_1 \leqslant d$, $K_2 \leqslant t$ and $K_3 \leqslant t + \ell$, providing a rough bound of $\text{Memory}_G(R,u) \leqslant O(d\log n) + O((t+\ell)\log n)$.

We now give a better implementation of the intervals contributing to the set K_1, especially when $d \geqslant \sqrt{n}$. Towards that, we slightly modify the delivery protocol. Specifically, upon reception of a message, we first decide whether the destination v is in $\widehat{T}_{p(u)}$ (i.e., $\mathcal{L}(v) \in L_1$), as follows. As shown in the proof of Proposition 3.8, $c_n(L_1) \leqslant 1$, thus this test can be performed in constant time, by storing $O(\log n)$ extra bits in u. If $v \in \widehat{T}_{p(u)}$ then we use the implementation described hereafter in order to find the outgoing arc on which to route the message. Otherwise, we use the usual delivery protocol which performs a binary search on the set of intervals.

We now describe the implementation that we use for the destinations in the set L_1 (i.e., the descendants in the tree $\widehat{T}_{p(u)}$). We use the fact that output port numbers of u can be chosen in advance in the set $\{1,\ldots,d\}$. Note that if output ports are fixed arbitrarily or by an adversary, no loop-free routing scheme with names in the range $\{1,\ldots,n\}$ can avoid the $\Omega(n)$ memory lower bound in a tree node (cf. [8]). An even stronger lower bound of $\Omega(n\log n)$ bits holds if output ports are fixed by an adversary after the choice of the node labels.

Note that routing a message to a destination in L_1 is equivalent to routing a message in the tree $\widehat{T}_{p(u)}$ from a node u to one of its descendants, v, such that the nodes of the tree are labeled continuously in the range $\mathcal{L}(S_{p(u)}) = \{m+1,\ldots,m+|S_{p(u)}|\}$, for some m. This task can be achieved using a compact implementation of the standard interval routing scheme on trees presented in [12, Section 2.3]. More precisely, it is shown that $O(\sqrt{n})$ bits per node suffice instead of the $O(d\log n)$ bit data structure for the naive implementation. To achieve that, we must label the nodes of the tree using a particular DFS, as done in the node-labeling preprocessing stage of the PIR algorithms; i.e., the children are recursively visited in increasing order of their number of descendants. However, in the implementation of [12], the bound on the time to extract the output port number given the label of

the destination is a priori exponential in n. Here we give another implementation using $O(\sqrt{n}\log n)$ bits per node that guarantees an $O(\log n)$ query time.

Let $v_1,\ldots,v_\delta$ be the descendants of u in $\widehat{T}_{p(u)}$ (setting $\delta = d$ if u is the root of $\widehat{T}_{p(u)}$, and $\delta = d-1$ otherwise). According to the node-labeling algorithm, the labels of the v_i's are ordered according to their number of descendants in $\widehat{T}_{p(u)}$, namely, letting n_i be the number of descendants of v_i in the subtree rooted in v_i, we have $n_i \leqslant n_{i+1}$ for $i < \delta$. Note that $n_i = \mathcal{L}(v_{i+1}) - \mathcal{L}(v_i)$ for $i < \delta$. Fix the output port number of the arc (u, v_i) to i. Clearly, to route a message M to v it suffices to find the output port p such that $\sum_{i=1}^{p-1} n_i < \mathcal{L}(v) - m \leqslant \sum_{i=1}^{p} n_i$. (Note that since $\mathcal{L}(v) \in L_1$, $\mathcal{L}(v) - m \leqslant \sum_{i=1}^{\delta} n_i$, thus p exists.)

Consider the sequences $A = (a_1,\ldots,a_k)$ and $(r_1,\ldots,r_k)$ such that the sequence $(n_1,\ldots,n_\delta)$ is described by a sequence of r_1 repetitions of a_1, followed by r_2 repetitions of a_2, and so on, imposing $a_1 < a_2 < \cdots < a_k$ (note that this is possible because $1 \leqslant n_1 \leqslant \cdots \leqslant n_\delta$). Finally, fix the sequences $Z = (z_1,\ldots,z_k)$ with $z_i = \sum_{j=1}^{i} r_j$, and $S = (s_1,\ldots,s_k)$ where $s_i = \sum_{j=1}^{z_i} n_j$, and fix $s_0 = z_0 = 0$. Storing m and the sequences A, Z, S, costs $O(k\log n)$ bits, since all the integers are taken from $\{1,\ldots,n\}$. Moreover, $k < \sqrt{2n}$ because $a_i \geqslant i$, and $\sum_{i=1}^{k} a_i \leqslant n$. Thus, the memory cost for this data structure is $O(\sqrt{n}\log n)$ bits, and it takes polynomial time to set up all the tables.

To find p in $O(\log n)$ time, it suffices to perform a binary search of $\mathcal{L}(v) - m$ in the sequence S, as $s_1 \leqslant \cdots \leqslant s_k$. Let q be such that $s_{q-1} < \mathcal{L}(v) - m \leqslant s_q$. Then the required output port number towards the destination is $p = z_{q-1} + \lceil (\mathcal{L}(v) - m - s_{q-1})/a_q \rceil$. □

Proof of Theorem 3.1. It remains to choose the size t of the balls so as to minimize the memory requirements and the compactness of the routing scheme R, and to determine the size ℓ of the cover. We use the greedy algorithm described in Section 2.3, where $|\mathcal{H}| = |\mathcal{V}| = n$. By Lemma 2.2, $\ell = |P| \leqslant n(1+\ln n)/t$. We note that the randomized algorithm described in Section 2.3 could also be used in order to construct a cover P of (asymptotically) the same size. Choosing $t = \sqrt{n(1+\ln n)/2}$, the PIR1 preprocessing algorithm will construct an interval routing scheme R whose compactness is bounded by Proposition 3.8 by

$$\mathrm{Comp}_G(R) \leqslant t + \ell/2 + 1 \leqslant \sqrt{2n(1+\ln n)} + 1 \sim \alpha\sqrt{n\log n}$$

where $\alpha = \sqrt{2\ln 2} \approx 1.17$, and whose memory requirement for every node u is bounded by Proposition 3.10 by

$$\mathrm{Memory}_G(R,u) = O\big(\big(\sqrt{n} + t + \ell\big)\log n\big) = O\big(\sqrt{n}\log^{3/2} n\big).$$

The length of the routes is bounded by Propositions 3.5–3.7. The latency is $O(\log n)$ and the time to construct R and its associated data structures is polynomial in n. □

3.3. Strategy PIR2

Theorem 3.11. *For every n-node unweighted graph $G = (V, E)$ with diameter D there exists a interval routing scheme $R = (\mathcal{L}, \mathcal{I})$ on G such that*

(1) $\mathrm{Memory}_G(R,u) = O(\sqrt{n}\log^{3/2} n)$ *for every* $u \in V$,
(2) $\mathrm{Latency}_G(R,u) = O(\log n)$ *for every* $u \in V$,

(3) $\text{Stretch}_G(R) \leqslant 5$,
(4) $\text{AvStr}_G(R) \leqslant 3$,
(5) $\text{Dilation}_G(R) \leqslant \lceil 1.5D \rceil$, *and*
(6) $\text{Comp}_G(R) \leqslant \beta\sqrt{n \log n}$, *where* $\beta \approx 2.00$.

Moreover, R can be constructed in time polynomial in n.

Strategy PIR2 is very similar to PIR1, except that the preprocessing algorithm constructs a pivot collection covering simultaneously the collection of t-balls and the collection of $\lceil D/2 \rceil$-neighborhoods around the nodes of G. Formally, the only necessary change is to replace Step 1 of the preprocessing algorithm of PIR1 by the following step.

1. Let P_1 be a cover for the collection of t-balls of the nodes, $\{\mathcal{B}_v(t)\}_{v \in V}$.
   Let P_2 be a cover for the collection of the $\lceil D/2 \rceil$-neighborhoods of the nodes, $\{\Gamma(v, \lceil D/2 \rceil)\}_{v \in V}$. Set $P = P_1 \cup P_2$.

This change is responsible for the improvement in the dilation bound, as proved next. Clearly, since P is a cover for the collection of t-balls of the nodes, the PIR2 strategy is a special case of the PIR1 strategy, thus Observation 3.2 and all the propositions in Section 3.2 hold also when the PIR2 strategy is considered. We now prove a better upper bound on the dilation of PIR2 in the case where G is an unweighted graph.

Proposition 3.12. $\text{Dilation}_G(R) \leqslant \lceil 1.5D \rceil$.

Proof. Clearly, the pivot $p(v)$ of every node $v \in V$ satisfies both $p(v) \in \mathcal{B}_v(t)$ and $p(v) \in \Gamma(v, \lceil D/2 \rceil)$. It follows that the distance between v and its pivot $p(v)$ is at most $\lceil D/2 \rceil$. By Corollary 3.4, for every two nodes $u, v \in V$,

$$d_R(u, v) \leqslant d_G\big(u, p(v)\big) + d_G\big(p(v), v\big) \leqslant \lceil 1.5D \rceil. \qquad \square$$

Proof of Theorem 3.11. It remains to select the size of the balls t and to show how to construct "good" covers P_1 and P_2. First we determine t and construct the cover P_1 as in the PIR1 preprocessing algorithm. That is, $t = \Theta(\sqrt{n \log n})$, and P_1 is a greedy cover of size $O(\sqrt{n \log n})$ (by Lemma 2.2). As in the PIR1 preprocessing algorithm, here too we can alternatively use the randomized algorithm in order to construct the cover P_1. Next, we use the same greedy algorithm in order to find the cover P_2 for the collection of $\lceil D/2 \rceil$-neighborhoods of the nodes. By Lemma 2.4, the size of such a cover satisfies $|P_2| \leqslant \sqrt{n(1 + \ln n)}$. More precisely, we have $\ell = |P| \leqslant |P_1| + |P_2|$, $|P_1| \leqslant n(1 + \ln n)/t$, and $|P_2| \leqslant \sqrt{n(1 + \ln n)}$. Therefore, choosing $t = \sqrt{n(1 + \ln n)/2}$, the compactness of R generated by the PIR2 preprocessing algorithm is bounded, by Proposition 3.8, by

$$\begin{aligned} \text{Comp}_G(R) &\leqslant t + \frac{1}{2}\left(\frac{n(1 + \ln n)}{t} + \sqrt{n(1 + \ln n)}\right) + 1 \\ &\leqslant \big(\sqrt{2} + 1\big)\sqrt{n(1 + \ln n)} + 1 \sim \beta\sqrt{n \log n} \end{aligned}$$

where $\beta = (\sqrt{2} + 1)\sqrt{\ln 2} \approx 2.00$, completing the proof. $\square$

4. Lower bounds on loop-free routing schemes

At this point, it is natural to ask whether our current bound of $O(\sqrt{n}\log^{3/2} n)$ bits of memory in each node for some stretch factor s is the best one can hope for in the case of loop-free routing schemes. It turns out that a lower bound of $\Omega(\sqrt{n})$ memory bits holds for any stretch factor for node labels in $\{1,\ldots,n\}$. To prove it, we exploit the fact that loop-free schemes (such as the PIR schemes) must route optimally on trees. Note that direct routing schemes (including interval routing) are loop-free but the reverse does not hold.

Proposition 4.1. *Let T be a tree, let $s \geqslant 1$ and let R be a loop-free routing scheme on T. Then* $\mathrm{Stretch}_T(R) \leqslant s$ *if and only if* $\mathrm{Stretch}_T(R) = 1$, *i.e., R is a shortest path routing scheme.*

Theorem 4.2. *For every $s \geqslant 1$, and for every loop-free routing scheme R of stretch factor s on n-node trees that uses names taken from $\{1,\ldots,n\}$, there exists an n-node tree T and a node u in T such that* $\mathrm{Memory}_T(R,u) \geqslant c\sqrt{n} - O(\log n)$, *where $c \approx 3.70$.*

Proof. Let $\mathcal{N}$ denote the set of all partitions of $n-1$ into d integers, i.e., sequences $S = (n_1,\ldots,n_d)$ with $1 \leqslant n_1 \leqslant \cdots \leqslant n_d$ such that $\sum_{i=1}^{d} n_i = n-1$. For every coding $\varphi : \mathcal{N} \mapsto \{0,1\}^*$ of the sequences in $\mathcal{N}$, let $\widehat{S}_\varphi$ be the sequence $S \in \mathcal{N}$ with longest coding $|\varphi(S)|$. Clearly, $|\varphi(\widehat{S}_\varphi)| \geqslant \log_2 |\mathcal{N}|$ for every coding φ. By the Hardy–Ramanujan formula [17, Eq. (4.2.7), p. 44], $\ln|\mathcal{N}| \sim \pi\sqrt{2n/3}$, and hence $|\varphi(\widehat{S}_\varphi)| \geqslant c\sqrt{n}$ for every coding φ, where $c = (\pi\sqrt{2/3})/\ln 2 \approx 3.70$.

Consider some loop-free routing scheme R on trees. Note that for each partition $S = (n_1,\ldots,n_d) \in \mathcal{N}$ there exists an n-node tree T_S with root u of degree d whose children induce subtrees of size n_i, for $i \in \{1,\ldots,d\}$ (e.g., a tree of depth two). By Proposition 4.1, the local routing function R in u routes along shortest paths, so the port number must be the same for all the destination nodes that belong to the same subtree. Hence knowing n, d and $\mathcal{L}(u)$ suffices for computing the sequence $S = (n_1,\ldots,n_d)$ using R.

It follows that R yields a coding ψ for $\mathcal{N}$, which for every $S \in \mathcal{N}$ satisfies $|\psi(S)| \leqslant \mathrm{Memory}_{T_S}(R,u) + O(\log n)$ (as n, d and $\mathcal{L}(u)$ can be stored using $O(\log n)$ bits). Looking at $\widehat{S}_\psi$, the worst sequence for ψ, and constructing the corresponding tree, $T_\psi = T_{\widehat{S}_\psi}$, we have $c\sqrt{n} \leqslant |\psi(\widehat{S}_\psi)| \leqslant \mathrm{Memory}_{T_\psi}(R,u) + O(\log n)$, completing the proof. □

Note that an upper bound of $c\sqrt{n}$ bits per node for trees has been established in [12, Section 2.3] for shortest path routing scheme.

In contrast to the theorem, a universal strategy that uses $O(k\,n^{1/k}\log n)$ bits of memory per router and yields stretch factor $O(k^2 9^k)$, for every $k \geqslant 1$, is presented in [1]. Moreover, the generated schemes are *name-independent*, i.e., node names do not change in the preprocessing, so the destination's routing label is simply its original name. It follows that there exists a routing scheme using names taken from $\{1,\ldots,n\}$ that guarantees $o(\sqrt{n})$ bits per router and a constant stretch factor (e.g., choose $k = 3$) for any graph, including graphs satisfying Theorem 4.2. However, that scheme cannot be loop-free, as the theorem indicates that a loop-free routing scheme with such an efficiency-space tradeoff does not exist for such graphs.

One may ask whether the stretch factor guaranteed by the PIR strategy is the best that can be achieved with memory requirements $O(\sqrt{n}\log^{3/2} n)$. Actually, the smallest possible stretch factor one can hope for when considering a routing scheme with memory requirements in $o(n)$ is $s = 3$, since as shown in [13], there exist some graphs on which every routing scheme of stretch factor $s < 3$ requires at least $\Omega(n)$ bits. Recently, a new sampling algorithm for selecting the pivots has been presented in [22], allowing improvements in the maximum stretch factor provided by our scheme. The new pivot set P is such that, for every node u, the set $C_u = \{w \mid d_G(u, w) < d_G(u, P)\}$ of nodes closer to u than any node of P satisfies that $|C_v| + |P| = O(\sqrt{n \log n})$. With the sets C_u playing the role of balls in our scheme, this new sampling algorithm provides a routing scheme (still in the form of an IRS) with the same performances as ours but with a lower maximum stretch factor. While the average stretch is still $\bar{s} = 3$, the triangle inequality can be used symmetrically between C_u and C_v (for a source u and a destination v) implying that the maximum stretch factor is $s = 3$ as well.

Acknowledgments

We thank Shlomo Moran, Shmuel Zaks, and Jean-Michel Couvreur for helpful discussions.

References

[1] B. Awerbuch, A. Bar-Noy, N. Linial, D. Peleg, Improved routing strategies with succinct tables, J. Algorithms 11 (1990) 307–341.
[2] B. Awerbuch, D. Peleg, Sparse partitions, in: 31st Symp. on Foundations of Computer Science (FOCS), 1990, pp. 503–513.
[3] B. Awerbuch, D. Peleg, Routing with polynomial communication-space trade-off, SIAM J. Discrete Math. 5 (1992) 151–162.
[4] L.J. Cowen, Compact routing with minimum stretch, J. Algorithms 38 (2001) 170–183.
[5] T. Eilam, S. Moran, S. Zaks, Lower bounds for linear interval routing, Networks 34 (1999) 37–46.
[6] M. Flammini, G. Gambosi, U. Nanni, R.B. Tan, Multidimensional interval routing schemes, Theoret. Comput. Sci. 205 (1998) 115–133.
[7] P. Fraigniaud, C. Gavoille, Universal routing schemes, J. Distrib. Comput. 10 (1997) 65–78.
[8] P. Fraigniaud, C. Gavoille, Routing in trees, in: 28th Int. Colloquium on Automata, Languages and Programming (ICALP), in: Lecture Notes in Comput. Sci., Vol. 2076, Springer-Verlag, 2001, pp. 757–772.
[9] G.N. Frederickson, R. Janardan, Separator-based strategies for efficient message routing, in: 27th Symp. on Foundations of Computer Science (FOCS), 1986, pp. 428–437.
[10] G.N. Frederickson, R. Janardan, Designing networks with compact routing tables, Algorithmica 3 (1988) 171–190.
[11] G. Gambosi, P. Vocca, Topological routing schemes, in: 10th Int. Workshop on Distributed Algorithms (WDAG), in: Lecture Notes in Comput. Sci., Vol. 1151, Springer-Verlag, 1996, pp. 206–219.
[12] C. Gavoille, A survey on interval routing, Theoret. Comput. Sci. 245 (2000) 217–253.
[13] C. Gavoille, M. Gengler, Space-efficiency of routing schemes of stretch factor three, J. Parallel Distrib. Comput. 61 (2001) 679–687.
[14] C. Gavoille, E. Guévremont, Worst case bounds for shortest path interval routing, J. Algorithms 27 (1998) 1–25.
[15] C. Gavoille, D. Peleg, The compactness of interval routing, SIAM J. Discrete Math. 12 (1999) 459–473.

[16] C. Gavoille, S. Pérennès, Memory requirement for routing in distributed networks, in: 15th Annual ACM Symp. on Principles of Distributed Computing (PODC), ACM Press, 1996, pp. 125–133.
[17] M.J. Hall, Combinatorial Theory, 2nd edition, Wiley–Interscience, 1986.
[18] R. Kráľovič, P. Ružička, D. Štefankovič, The complexity of shortest path and dilation bounded interval routing, Theoret. Comput. Sci. 234 (2000) 85–107.
[19] L. Lovász, On the ratio of optimal integral and fractional covers, Discrete Math. 13 (1975) 383–390.
[20] D. Peleg, E. Upfal, A trade-off between space and efficiency for routing tables, J. ACM 36 (1989) 510–530.
[21] N. Santoro, R. Khatib, Labelling and implicit routing in networks, Comput. J. 28 (1985) 5–8.
[22] M. Thorup, U. Zwick, Compact routing schemes, in: 13th Annual ACM Symp. on Parallel Algorithms and Architectures (SPAA), Hersonissos, Crete, Greece, ACM Press, 2001, pp. 1–10.
[23] S.S.H. Tse, F.C.M. Lau, An optimal lower bound for interval routing in general networks, in: 4th Int. Colloquium on Structural Information & Communication Complexity (SIROCCO), Carleton Scientific, 1997, pp. 112–124.
[24] J. van Leeuwen, R.B. Tan, Computer networks with compact routing tables, in: The Book of L, Springer-Verlag, 1986, pp. 259–273.