Compact Routing Schemes With Low Stretch Factor
(Extended Abstract)

Tamar Eilam* Cyril Gavoille† David Peleg‡

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* (namely, the ratio between their length and the distance between their endpoints) is at most five, and whose average stretch is at most three, with routing tables of size $O(n^{3/2}\log^{3/2} n)$ bits in total. A similar routing strategy for unweighted networks which guarantees the same bounds on the stretch factor and in addition a bound of $\lceil 1.5D \rceil$ on the route lengths, where D is the diameter of the network, is also presented. Moreover, it is shown that the PIR strategy can be implemented so that the generated scheme is in the form of an *interval routing scheme (IRS)*, using at most $2\sqrt{n(1+\ln n)}$ intervals per link in the first case and $3\sqrt{n(1+\ln n)}$ in the second case. As a result, the schemes are simpler than previous ones and they imply that paths of messages are loop-free. Finally, it is shown that there is no *loop-free* routing strategy guaranteeing a memory bound of $\sqrt{n}$ bits per router 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 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 that 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 which specifies an output port for every destination. Though this strategy can guarantee routing along shortest paths, each router has to store locally $\Theta(n\log d)$ bits of memory, where d is the degree of the router (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, 2, 3, 25]) that there is a tradeoff between the memory requirements of a routing scheme and the worst case stretch factor it guarantees. In [25] it was shown that any universal routing strategy achieving stretch factor $s \geq 1$ must use a total of $\Omega(n^{1+\frac{1}{2s+4}})$ bits of routing information in the network. Stronger lower bounds hold for small stretch factors: any routing scheme with stretch factor $s < 3$ must use a total of $\Omega(n^2)$ bits [8, 16], and an optimal bound of $\Theta(n^2\log n)$ bits holds for stretch $s = 1$ [19].

On the positive side, a number of routing strategies have been proposed that achieve almost optimal efficiency-space tradeoff. Specifically, in [25] it was shown that for every graph and every integer $k \geq 1$ it is possible to construct a hierarchical routing scheme with stretch factor $O(k)$ which uses a total of $\Omega(k^3 n^{1+1/k}\log n)$ bits and names each node with $O(\log^2 n)$ bits. Though this routing scheme is almost optimal in terms of its efficiency-space tradeoff, it has a few drawbacks. First, it is not *name-independent*, that is, it relabels the nodes with new names. Secondly, this scheme does not bound the local memory requirement of a node, and finally, it assumes a unit cost on the links of the network. Other methods that overcome these problems achieve an inferior efficiency-space tradeoff; the hierarchical routing scheme presented in [1] uses $O(k\, n^{1/k}\log n)$ bits of memory per node and guarantees a stretch factor $O(k^2\, 9^k)$. The scheme presented in [3] keeps the advantages of the former one, while regaining the polynomial tradeoff. In particular it guarantees, for every $k \geq 1$, stretch factor $O(k^2)$, while using $O(k\, n^{1/k}\log n \log D)$ bits per node, where D is the weighted diameter of the network.

*Department of Computer Science, Technion, Haifa 32000, Israel. *eilam@cs.technion.ac.il*

†LaBRI, Université Bordeaux I, 351, cours de la Libération, 33405 Talence Cedex, France. *gavoille@labri.u-bordeaux.fr*

‡Department of Applied Mathematics and Computer Science, The Weizmann Institute of Science, Rehovot, 76100 Israel. *peleg@wisdom.weizmann.ac.il*. Supported in part by a grant from the Israel Science Foundation and by a grant from the Israel Ministry of Science and Art.

The major disadvantage of all the proposed hierarchical routing strategies is that they are rather complex, and thus are impractical especially for high-speed networks for which the latency in each node must be very short. Broadly speaking, the hierarchical routing strategies are based on techniques for generating a sparse cover of clusters for the underlying graph. In order to implement the routing scheme, there are $O(\log n)$ levels of covers with increasing radii. Intuitively, higher levels are responsible for routing to farther destinations. A message is first sent on the lowest level, on the hope that the destination is nearby. If this is not the case, then the transmission might fail, in which case the message will bounce back to the originator. This step may be viewed as one "phase" of the routing process. Once receiving the message back (with a notification of failure), the originator tries to send it on a higher level, and so on until the routing process succeeds in delivering the message. Clearly, the implementation of this procedure in every router results in a complex decision function; the number of the present phase has to be coded into the header of the message, thus the message header must be rewritten upon each retransmission. Moreover, intermediate routers must also change the message header, for example, in order to notify of a failure. In addition, those strategies treat the nodes of the network non-uniformly, in the sense that different nodes play different roles, thus the decision function could be substantially different at different nodes. As high-speed networks gain popularity and increase in size, these drawbacks become crucial, since the main routing bottleneck in these networks is often the decision function in the nodes and not the propagation delay. Therefore, simple routing schemes which could be implemented in hardware may be preferable in practice.

Subsequently, considerable attention is 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, 10, 13, 26, 28]). An important property common to all of them is that they are *direct*. Namely, they employ a simple "shoot 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.) As a consequence, it is easy to verify that every message traverses a loop-free path from its source to its destination. (Schemes enjoying this property are referred to in what follows as *loop-free* routing schemes.) All of these strategies involve relabeling of the nodes in order to reduce the amount of space needed for the routing, thus they are not name-independent. The most popular such scheme is the *interval routing scheme (IRS)*, presented in [26, 28] and implemented in the T9000 Transputer router chip of INMOS [21]. The idea of this scheme is to label nodes with unique integers from a contiguous range, and to label the outgoing arcs in every node with a set of intervals forming a partition of the name range. When invoking the delivery protocol, a message is sent on the unique outgoing arc labeled by an interval that contains the destination label.

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, 7, 14, 17, 22, 27]), 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 [26, 28]. This scheme uses only one interval per edge, thus the memory in a node with degree d is $O(d \log n)$, but it implies no non-trivial upper bound on the stretch factor. On the other extreme it was shown in [18] 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, [22] showed that for every graph there exists an interval routing scheme under which every message traverses a path of length at most $\lceil 1.5D \rceil$, where D is the diameter of the network, and which labels every arc with at most $\lceil \sqrt{n \ln n} \rceil + 1$ intervals. While this result implies an upper bound on the *dilation*, i.e., on the length of paths traversed by messages, it does not imply any non-trivial 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.

This unfortunate situation holds not only for interval routing but for all loop-free, compact routing strategies proposed in the literature; while there are many compact routing strategies for specific topologies (see, e.g., [11, 12]), to the best of our knowledge there is no universal strategy which generates for every graph a loop-free routing scheme with a constant stretch factor that uses significantly fewer than $O(n^2)$ memory bits.

1.2 Our Results

A basic question which arises from the above discussion is whether it is possible to combine the benefits of both worlds, and design "simple" near-optimal routing strategies, which are still compact. In particular, can "simple" routing schemes achieve a similar tradeoff between stretch factor and memory as the hierarchical routing schemes? In this paper we analyze the power of "simple" routing strategies and give both positive and negative results for this question. To do that, we must first give a precise definition that captures our intuition about "simple" routing schemes. Define 1-*phase routing schemes* as schemes in which the header is not allowed to be re-written. *Oblivious routing schemes* pose further restrictions which make the decision function in the nodes even simpler. Finally, a *direct* routing scheme is defined as one in which the decision functions depend only on the destination, which is the only information that is kept (and never changed) in the header. Our main results are two 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 $s \leq 5$ and average stretch factor $\overline{s} \leq 3$ that uses $O(n^{3/2} \log^{3/2} n)$ bits in total. Moreover, the PIR1 preprocessing algorithm actually generates for every graph an interval routing scheme which labels every arc with at most $2\sqrt{n(1 + \ln n)}$ intervals. 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, but with $3\sqrt{n(1 + \ln n)}$ intervals per arc), 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 note that our routing strategies provide the first constructive method for design-

ing for every graph an interval routing scheme with small stretch factor and $o(n)$ intervals per arc. We also prove that the bound of $\overline{s} \leq 3$ on the average stretch factor of the proposed routing strategies is tight, and yet that for almost all graphs (specifically, all but a $1/n^3$ fraction of all the n-node graphs), the average stretch factor guaranteed by the PIR1 and PIR2 routing strategies is asymptotically equal to 1.875.

When comparing the proposed strategies with the hierarchical routing strategies (which are not loop-free, direct, oblivious or 1-phase), a natural question is whether it is possible to design a family of loop-free routing strategies spanning the entire range of possible tradeoffs between different values of stretch factors and memory requirements. In other words, is it possible using loop-free routing strategies to decrease the memory requirements at the expense of a larger stretch factor? We answer this question negatively by showing that no matter which stretch factor one aims at guaranteeing, if the routing scheme is loop-free then there must be some graphs for which it requires $\sqrt{n}$ memory bits in at least one of the nodes. Hence there is no universal loop-free routing strategy that can guarantee $o(\sqrt{n})$ bits in each node, whatever the stretch factor is.

Finally, the PIR1 and PIR2 routing strategies are also compared with one of the hierarchical routing strategies proposed in [1]. It is shown that while the difference between the average stretch factor achieved by both strategies on every graph does not exceed $o(1)$, there are concrete examples in which our strategy is superior in terms of the average stretch factor.

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. A comparison between PIR and HCP_k, the hierarchical routing strategy presented in [1], is given in Section 5. Proofs are almost all deferred to the full version of the paper [4].

2 Model and Definitions

2.1 Routing Schemes and Complexity Measures

A point-to-point communication network is modeled as a symmetric, weighted, finite digraph $G = (V, E, W)$, $|V| = n$, 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 non-negative weight $w(e)$ (i.e., its cost). We assume that for every two opposite arcs e_1 and e_2, $w(e_1) = w(e_2)$. 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 multi-arcs. We assume that every node v is named with a unique identification integer $\mathrm{ID}(v)$. In what follows, we informally confuse between the node v and its name $\mathrm{ID}(v)$. Note that the identifications induce a total order on the nodes, thus for every two nodes $u, v \in V$ either $u < v$ or $v < u$.

The length $|P|$ of a directed path P in the graph is the sum of weights of the arcs in P. The distance $\mathrm{dist}_G(u, v)$ 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 $D(G) = \max\{\mathrm{dist}_G(u, v) \mid u, v \in V\}$. 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* $\mathcal{L}(v)$ of the destination node v, and the message's information field. A routing scheme is *name-independent* if node "names" do not change in the preprocessing, that is, the destination's routing label $\mathcal{L}(v)$ is simply its original name, $\mathrm{ID}(v)$. In a name-dependent routing scheme, nodes are relabeled by unique integers in the preprocessing stage, and the new labels are then used when invoking the delivery protocol. 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 $\mathrm{dist}_G(R, u, v)$.

Specifically, the delivery protocol of the routing scheme R uses a pair of integer functions (P, H) where P is the *port function* and H is the *header function*. For any two distinct nodes u and v, R produces a path of nodes $(u = u_0, u_1, \ldots, u_k = v)$, a sequence $h_0, h_1, \ldots, h_k$ of headers, and a sequence $p_0, p_1, \ldots, p_k$ of output port numbers. A message with header h_i arriving at node u_i through input port p_i is given a new header $H(u_i, p_i, h_i) = h_{i+1}$, and is forwarded to the output port $P(u_i, p_i, h_i) = p_{i+1}$. Thus, we require that for every $i \in \{1, \ldots, k-1\}$, $H(u_i, p_i, h_i) = h_{i+1}$, $P(u_i, p_i, h_i) = p_{i+1}$ and that p_{i+1} corresponds to the output port number of the link (u_i, u_{i+1}). On each router, the input and output ports numbered 0 are associated with the special link between the router and its host. This allows us to complete the description by imposing the constraints that $p_0 = 0$, $P(u_k, p_k, h_k) = 0$, as well as $h_0 = \mathcal{L}(v)$, thus fixing the initial header, which is provided to the router from a higher layer. (In a name-independent routing scheme, h_0 is set to $\mathrm{ID}(v)$, the original name of the destination node v.) Since we are interested in distributed routing strategies, we will express the delivery protocol as a collection of local routing functions $R_x = (P_x, H_x)$, one for each node x.

A 1-*phase routing scheme* is a routing scheme in which the header of the message is determined by the originator and is never modified by any intermediate node. Formally, the first header in the sequence, h_0, is equal to the destination. This is simply the mathematical formulation of the command to invoke the delivery protocol which is given from a higher layer. When the delivery protocol is invoked, the first router computes the header h_1 of the message (which is not necessarily equal to the destination). Once the header is fixed by the first router, it is never changed, i.e., 1-phase routing schemes satisfy $h_0 = \mathcal{L}(v)$ and $h_1 = h_2 = \ldots = h_k$. The fact that there is no need to recompute and rewrite the header once it is computed by the first router when invoking the delivery protocol, implies a simple and fast decision function in the nodes. This is important especially when dealing with high-speed networks.

An *oblivious routing scheme* is a 1-phase routing scheme in which for every node x, two input ports p and p' and header h, $P(x,p,h) = P(x,p',h)$, namely, the decision function in a node x depends only on the header of the message (and not on its entry port). Note that 1-phase routing schemes may route messages along paths with loops: a message might arrive at the same node of the network twice but from different input ports. In the case of oblivious routing schemes the routing path may cycle once on the originator node only, since the routing decision depends only on the header, which is changed at most once (at the first router).

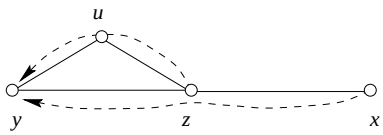

Figure 1: An oblivious routing scheme which is not direct.

We advance one step further and define a *direct routing scheme* as an oblivious routing scheme in which the decision functions in the nodes depend only on the destination of the message, namely, $h_0 = h_1 = ... = h_k = \mathcal{L}(v)$ (where $\mathcal{L}(v)$ is the destination's label). Note that every direct routing scheme is also a *loop-free* routing scheme, i.e., it guarantees that a message always traverse a loop-free path, since otherwise the message will cycle forever without reaching its destination, contradicting the definition of a routing scheme. The classical full routing tables are direct routing schemes. Another example of a direct routing scheme is interval routing, a well-known routing method presented in [26, 28]. Figure 1 is an example of an oblivious routing scheme which is not direct: a message destined to a node y, entering z on the arc from x is forwarded on one output port, while a message entering z on input port 0 (i.e., the originator is z) will be forwarded on different output port. This is possible with oblivious routing schemes since the header used for a message from z to y might be different from the header used for a message from x to y.

A *universal routing strategy* is a function that returns, for every n-node 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 required for the routing scheme. A routing strategy is *name-independent* if it produces for every graph a name-independent routing scheme.

Given a graph G, a routing scheme R on G, and a node x, the *memory requirement* of x, denoted by $\mathrm{Memory}_G(R,x)$, is the smallest number of bits that are required in order to code R_x. The *total memory requirements* of R, are $\mathrm{Memory}_G(R) = \sum_x \{\mathrm{Memory}_G(R,x)\}$. Let R be a routing scheme on an n-node graph G. The maximum and average stretch factors of R are respectively defined as follows.

$$\mathrm{Stretch}_G(R) = \max_{x \neq y} \frac{\mathrm{dist}_G(R,x,y)}{\mathrm{dist}_G(x,y)} \quad \text{and}$$

$$\mathrm{AvStr}_G(R) = \frac{1}{n(n-1)} \sum_{x \neq y} \frac{\mathrm{dist}_G(R,x,y)}{\mathrm{dist}_G(x,y)}.$$

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, $\mathrm{Dilation}_G(R) = \max_{x \neq y}\{\mathrm{dist}_G(R,x,y)\}$. Given a graph G and a set of routing schemes $\mathcal{RS}$ on G, denote

- $\mathrm{Memory}_G(\mathcal{RS}) = \min_{R \in \mathcal{RS}}\{\mathrm{Memory}_G(R)\}$
- $\mathrm{Stretch}_G(\mathcal{RS}) = \min_{R \in \mathcal{RS}}\{\mathrm{Stretch}_G(R)\}$
- $\mathrm{AvStr}_G(\mathcal{RS}) = \min_{R \in \mathcal{RS}}\{\mathrm{AvStr}_G(R)\}$
- $\mathrm{Dilation}_G(\mathcal{RS}) = \min_{R \in \mathcal{RS}}\{\mathrm{Dilation}_G(R)\}$

Hereafter, we sometimes omit the subscript G, when the graph is clear from the context.

2.2 Direct Routing Schemes and Interval Routing

In this section we give a definition of interval routing, and show its equivalence to direct routing.

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 x, the collection of sets that label all the outgoing arcs of x forms a partition of the name range (possibly excluding x itself). Formally, for every $x \in V$:

1) $\bigcup_{e \in E_x} \mathcal{I}(e) \cup \mathcal{L}(x) = \{1,\ldots,n\}$,
2) $\mathcal{I}(e) \cap \mathcal{I}(e') = \emptyset$, for every two distinct arcs $e, e' \in E_x$.

The delivery protocol is defined as follows. Given a destination node v, set the first header to be $h_1 = \mathcal{L}(v)$. Also, for every node y and port p, $H(y,p,h) = h$ and $P(y,p,h) = p'$ if and only if $h \in \mathcal{I}(x,y)$ and p' is the output port number of the arc (x,y). Namely, the message is sent on the arc which is labeled by a set that contains the destination. For any set $V' \subseteq V$, denote $\mathcal{L}(V') = \{\mathcal{L}(v) \mid v \in V'\}$.

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, we argue that every direct routing scheme could be implemented using interval routing. This is done by labeling every outgoing arc in a node with the set of destinations for which a message will be sent on that arc. This is always possible since with direct routing schemes, in every node, two messages with the same destination are forwarded on the same arc (even if their originators are different). Note that this is not the case with oblivious routing, as implied by Figure 1.

Given an integer n and a subset $I \subseteq \{1,\ldots,n\}$, define the *compactness of I w.r.t. n*, denoted $c(I)$, as the smallest integer k such that I can be represented by the union of k intervals $[a,b]$ of consecutive integers from $\{1,\ldots,n\}$, n and 1 being considered as consecutive (cyclically). The compactness of an interval routing scheme $R = (\mathcal{L},\mathcal{I})$ on G, denoted by $\mathrm{Comp}_G(R)$, is the maximum, overall arcs $e \in E$, of the compactness $c(\mathcal{I}(e))$ of the set $\mathcal{I}(e)$ labeling e. Intuitively, smaller compactness and degrees imply smaller routing tables. The interval routing strategy presented in [26], based on routing on a minimum spanning tree, has compactness 1 (for every graph) but unbounded stretch factor. Another example is the ring C_n which admits an interval routing scheme R of compactness 1 with stretch factor 1 ([26]). Therefore, $\mathrm{Memory}_{C_n}(R,x) = O(\log n)$ for every node x, as it suffices for x to store its label and the intervals assigned to the two arcs. Actually, every graph G supporting an IRS R of compactness k satisfies $\mathrm{Memory}_G(R,x) = O(\deg(x) k \log(n/k))$ [15].

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 identifications. Formally, $x \prec_v y$ if and only if either $\mathrm{dist}_G(x,v) < \mathrm{dist}_G(y,v)$ or $\mathrm{dist}_G(x,v) = \mathrm{dist}_G(y,v)$ 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 w.r.t. v. The *r-neighborhood* of a node $v \in V$ is defined as $\Gamma_v(r) = \{u \in V \mid \mathrm{dist}_G(v,u) \leq r\}$. 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. The proof can be found in [24], Page 53.

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

Consider a collection $\mathcal{H}$ of subsets of size t of elements from a set B. A set $P \subseteq B$ 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 B 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 [23]) *Let P be a greedy cover for $\mathcal{H}$. Then $|P| < |B|(\ln|\mathcal{H}| + 1)/t$.*

The second method is randomized, and takes each element of B 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 $|B| \geq 2t$ 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 $|P| \leq (2c|B|\ln|\mathcal{H}|)/t$.*

For our needs, $|\mathcal{H}| = |B| = n$. 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 [4] we prove the following.

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

3 The Pivot Interval Routing Strategy

We present two routing strategies, termed *Pivot Interval Routing*. The first one, PIR1, generates a direct routing scheme for every weighted graph (Theorem 3.1) and the second one, PIR2, generates a direct routing scheme for unweighted graphs with improved dilation (Theorem 3.2).

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

- $\mathrm{Memory}_G(R) = O(n^{3/2}\log^{3/2} n)$,
- $\mathrm{Memory}_G(R,v) = O(n^{1/2}\log^{3/2} n + \deg(v)\log n)$, *for every $v \in V$,*
- $\mathrm{Stretch}_G(R) \leq 5$,
- $\mathrm{AvStr}_G(R) \leq 3$,
- $\mathrm{Dilation}_G(R) \leq 2D$,
- $\mathrm{Comp}_G(R) \leq 2\sqrt{n(1+\ln n)}$.

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 as follows. If the destination v is in u's t-ball, then the message will traverse a shortest path from u to v. 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, 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. We now give a formal description of the PIR1 strategy. The preprocessing algorithm consists of three parts. In the first part, some preliminary structures are constructed which are used later to define the node and arc labeling functions $\mathcal{L}$ and $\mathcal{I}$. The delivery protocol of PIR1, is the usual delivery protocol for interval routing schemes.

Strategy PIR1

Input: A weighted graph $G = (V,E,W)$.

The preprocessing algorithm

a. Preliminary constructions

1. Let P be a cover for the collection of t-balls of the nodes, $\{\mathcal{B}_v(t) \mid v \in V\}$.
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 identifications).
3. For every pivot $p \in P$, let $S_p = \{v \mid p(v) = p\}$ be its set of "clients".
4. For every pivot $p \in P$, construct a minimum BFS spanning tree T_p rooted at p and spanning the entire graph G, and let T'_p be the subtree of T_p induced by p's set of clients S_p.

b. Labeling the nodes

Assume that $P = \{p_1, p_2, \ldots, p_l\}$. We start by labeling the nodes in S_{p_1}. The labeling is performed by traversing the tree T'_{p_1} (i.e., the subtree spanning S_{p_1}), and assigning the

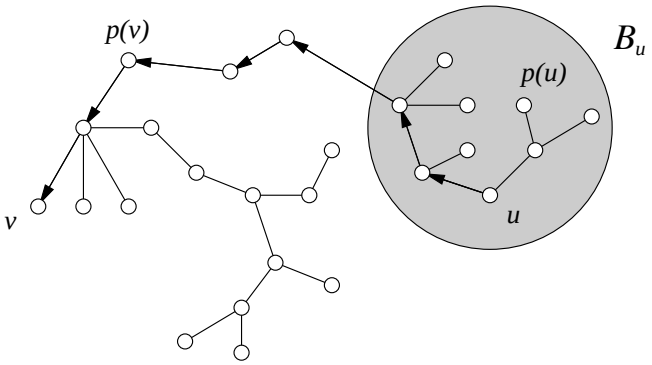

Figure 2: The route from u to a destination v which is not in u's ball.

nodes of T'_{p_1} a DFS (pre-order) numbering in sequential ascending order, starting from 1. Assuming that n_1 is the largest label assigned to a node in S_{p_1}, we continue by labeling the nodes in S_{p_2} in the same manner (traversing the tree T'_{p_2}), starting from the integer $n_1 + 1$. Then we label the nodes in $S'_{P_3}, \ldots S'_{p_l}$ in the same way.

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

0. fix $I(e) = \emptyset$, for every $e \in E_u$.
1. If u is not a leaf in the tree $T'_{p(u)}$, let $s_1, ..., s_j$ be its successors in $T'_{p(u)}$ and let T'_{s_i} be the subtree of $T'_{p(u)}$ rooted at s_i, for $1 \leq i \leq j$. Assign $\mathcal{I}(u, s_i) = \{\mathcal{L}(v) \mid v \in T'_{s_i}\}$, for every $1 \leq i \leq j$. Define $M_1 = \bigcup_{1 \leq i \leq j} \mathcal{I}(u, s_i)$.
2. Define $M_2 = \mathcal{L}(\mathcal{B}_u(t)) \setminus M_1$. For every node $v \neq u$ such that $\mathcal{L}(v) \in M_2$, let $e \in E_u$ be an arc on a shortest path from u 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 $M_3 = \mathcal{L}(V) \setminus (M_1 \cup M_2)$. For every $p \in P$, let $e_p \in E_u$ be the arc from u to its predecessor on the tree T_p. Assign $\mathcal{I}(e_p) = \mathcal{I}(e_p) \cup (\mathcal{L}(S_p) \cap M_3)$.

The delivery Protocol
In a node u, a message M with destination $v \neq u$ is sent on the unique arc $e \in E_u$, such that $\mathcal{L}(v) \in \mathcal{I}(e)$.

Figure 2 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. In Appendix [4] we prove that the PIR1 preprocessing algorithm generates a (direct) interval routing scheme that satisfies (for specific possible values of t and $|P|$) the properties stated in Theorem 3.1.

Theorem 3.2 *For every n-node unweighted graph $G = (V, E)$ with diameter D there exists a (direct) interval routing scheme $R = (\mathcal{L}, \mathcal{I})$ on G such that:*

- $\text{Memory}_G(R) = O(n^{3/2} \log^{3/2} n)$.
- $\text{Memory}_G(R, v) = O(n^{1/2} \log^{3/2} n + \deg(v) \log n)$, *for every $v \in V$.*
- $\text{Stretch}_G(R) \leq 5$.
- $\text{AvStr}_G(R) \leq 3$.
- $\text{Dilation}_G(R) \leq \lceil 1.5D \rceil$.
- $\text{Comp}_G(R) \leq 3\sqrt{n(1 + \ln n)}$.

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

Strategy PIR2

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

This change is responsible for the improvement in the dilation bound, as proved in Appendix [4].

In the rest of this section we present some additional analytic results on the efficiency of the PIR1 routing strategy. These results hold regardless of whether the cover P is constructed using the randomized or the greedy algorithm, and some of these results hold for *any* cover P of size $O(\sqrt{n \ln n})$. Moreover, most of the results apply also for the PIR2 strategy.

Let $PIR1[P]$ denote the PIR1 strategy with $t = \sqrt{n(1 + \ln n)}$ and a cover P for the t-balls, and let $PIRS1[P]$ denote the set of routing schemes generated by $PIR1[P]$.

Let us first describe a property of the dilation incurred by the PIR1 strategy. For every graph G of diameter D, denote by $\overline{D}$ the average distance in G, i.e., $\overline{D} = \frac{1}{n(n-1)} \sum_{u \neq v} \text{dist}_G(u, v)$.

Theorem 3.3 *Let G be a graph of diameter D and of average distance $\overline{D}$. Then, for every $R \in PIRS1$, and for all nodes $u, v \in V$, $\text{AvStr}_G(R) \leq 2 + D - \overline{D} - \Theta(\sqrt{(\log n)/n})$.*

It follows that the average stretch factor for graphs with $\overline{D} = D - o(1)$ is at most $2 + o(1)$. For instance, for the complete graph we get $\text{AvStr}_{K_n}(PIR1[P]) \leq 2 - \Theta(\sqrt{(\log n)/n})$.

The next result shows that the bound of Theorems 3.1 and 3.2 on the average stretch factor is tight, as an average stretch factor of 3 can be reached asymptotically with the PIR1 strategy.

Theorem 3.4 *There exists an n-node graph G and an identity assignment* ID *for its nodes such that for every set of pivots P of G computed by the greedy cover algorithm (respectively, the randomized algorithm), $\text{AvStr}_G(PIR1[P]) > 3 - o(1)$ (resp., with probability at least $1 - 1/\sqrt{n}$).*

Proof. Given two suitable integers $m, k \geq 1$, let us build an n-node graph $G = (V, E)$ defined as follows. Let $V = A \cup B$, where A is a clique of size mk, consisting of k subsets of size m, denoted $A_1, \ldots, A_k$. Likewise, B is composed of $2k$ cliques of size $t-1$, denoted $B_1, \ldots, B_k$ and $B'_1, \ldots, B'_k$. In addition, for every $i \in \{1, \ldots, k\}$, each node $u \in B_i$ is connected to all the nodes of A_i, and B'_i is connected to B_i by a matching. See Figure 3. Hence $n = |V| = mk + 2(t-1)k$ (where m and k must be selected so as to ensure that $t = \sqrt{n(1+\ln n)}$, as required from the size of the balls in the PIRS1[P] schemes.) See Figure 3.

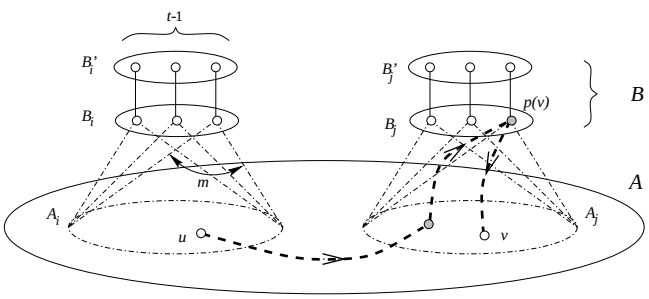

Figure 3: A example for proving an average stretch factor of $3 - o(1)$.

Consider the following identity assignment ID for the nodes of G, selected in order to enforce a specific choice of the closest pivot. The identities for nodes of B are selected arbitrarily from the set $\{1, \ldots, |B|\}$, and the identities of the nodes of A are selected arbitrarily from the set $\{|B| + 1, \ldots, n\}$.

Claim 3.5 *For every $u \in V$, $\deg(u) \geq t-1$. Moreover, if $u \in A_i$, then $\mathcal{B}_u = \{u\} \cup B_i$, and if $u \in B_i \cup B'_i$, then $\mathcal{B}_u \subseteq B_i \cup B'_i$.*

Proof. For every $u \in A$ and $v \in B$, $\deg(u) = |A| - 1 + t$, and $\deg(v) \geq t - 1$. Hence, for every $u \in V$, $\mathcal{B}_u$ consists of u and the $t-1$ lowest-ID neighbors of u. Let us consider some $i \in \{1, \ldots, k\}$. For each $u \in A_i$, the $t-1$ lowest-ID neighbors of u constitute all the nodes of B_i, so $\mathcal{B}_u \setminus \{u\} = B_i$. For each $u \in B_i \cup B'_i$, u has exactly $t-1$ neighbors in $B_i \cup B'_i$, because the identities of the nodes of B are lower than those of the nodes of A. Hence $\mathcal{B}_u \subseteq B_i \cup B'_i$. □

Let P be the set of pivots. By Claim 3.5, for every $i \in \{1, \ldots, k\}$, and for every $u \in A_i \cup B_i \cup B'_i$, $\mathcal{B}_u \subseteq A_i \cup B_i \cup B'_i$. Hence whatever P is, there exists at least one $p \in P$ such that $p \in A_i \cup B_i \cup B'_i$. Let $\mathcal{E}_{OK}$ denote the event that for every $i \in \{1, \ldots, k\}$, there is a pivot $p \in P$ such that $p \in B_i$. Then,

Claim 3.6 (1) *If P is computed by the greedy cover algorithm, then $\Pr(\mathcal{E}_{OK}) = 1$;*
(2) *if P is computed by the randomized algorithm, then $\Pr(\mathcal{E}_{OK}) > 1 - 1/\sqrt{n}$.*

Proof. (1) Let p be the first pivot selected in $A_i \cup B_i \cup B'_i$. Note that if $p \in A_i$ then p covers only the ball around p itself. Also, if $p \in B'_i$ then p covers at most $\deg(p) = |B'_i| + 1 = t$ balls. On the other hand, if $p \in B_i$ then p covers at least $|A_i| + |B_i| + 1 = m + t$. It follows that for each i, the greedy algorithm will necessarily choose a pivot in B_i.

(2) Recall that the randomized algorithm randomly picks each node of V into the cover with probability $p = (c \ln n)/t$ for constant $c > 1$. For every $i \in \{1, \ldots, k\}$, let $X_i = |B_i \cap P|$ denote the random variable counting the number of pivots in B_i. For every i,

$$\Pr(X_i = 0) \;=\; (1-p)^{|B_i|} \;=\; (1-p)^{t-1},$$

so $\Pr(\exists i, X_i = 0) \leq k(1-p)^{t-1}$. It follows that

$$\Pr(\mathcal{E}_{OK}) = \Pr(\forall i, X_i > 0) \;\geq\; 1 - k(1-p)^{t-1}\ .$$

Note that $k = n/(m + 2(t-1)) \leq \sqrt{n}/2$. In order to bound the probability we use the following known fact.

Fact 3.7 *For all $0 < \alpha \leq 1/2$ and $\beta > 0$, $(1-\alpha)^{\beta} < e^{-\alpha\beta}$.*

Hence

$$\begin{aligned}\Pr(\mathcal{E}_{OK}) &\geq 1 - k\,e^{-p(t-1)} \;\geq\; 1 - \frac{\sqrt{n}}{2}\,e^{-c(1-1/t)\ln n} \\ &> 1 - \frac{1}{2}\,n^{-(1/2-1/t)} \;>\; 1 - n^{-1/2}.\end{aligned}$$

□

By Claim 3.6, every clique B_i contains at least one pivot. Consider a routing scheme $R \in PIRS1[P]$ on G. We want to lower bound $\mathrm{AvStr}_G(R)$. Because $\mathrm{dist}(u, v) = 1$ for all $u, v \in A$, we have

$$\mathrm{AvStr}_G(R) \;>\; \frac{1}{n(n-1)} \sum_{u \neq v \in A} \mathrm{dist}_G(R, u, v).$$

For every node $u \in V$, let $p(u) \in P$ denote the closest pivot of u, according to the distance definition. For every $p \in P$, let T_p be the minimum spanning tree rooted in p, and used by R. Consider some pair $u, v \in A$, $u \neq v$. Note that if $\mathrm{dist}_G(u, p(v)) = 2$ and $\{u, v\}$ is not an edge of $T_{p(v)}$ then $\mathrm{dist}_G(R, u, v) = 3$. Given u, the number of nodes $v \in A$ such that $\{u, v\}$ is an edge of $T_{p(v)}$ is $k-1$, since all the nodes of a same component A_i share a unique pivot in B_i, hence use the same tree to route inside A_i. Moreover, $\mathrm{dist}_G(u, p(v)) = 1$ if and only if u and v are both in the same A_i. In total, the number of nodes v such that $\mathrm{dist}_G(R, u, v) \neq 3$ is at most $m + k - 1$, so

$$\begin{aligned}\sum_{v \in A \setminus \{u\}} \mathrm{dist}_G(R, u, v) &\geq 3(|A| - 1 - (m + k - 1)) \\ &= 3mk - 3(m + k).\end{aligned}$$

Summing over all $u \in A$, it follows that

$$\sum_{u \neq v \in A} \mathrm{dist}_G(R, u, v) \;\geq\; 3(mk)^2 - 3mk(m + k).$$

Letting $k = \sqrt{n}/\ln n$, we have $mk = n - 2(t-1)k = n - cn/\sqrt{\ln n}$ and $m + k < c'\sqrt{n}\ln n$, for large n and for suitable

constants $c, c' > 0$. Hence

$$\begin{aligned}
\mathrm{AvStr}_G(R) &> \frac{3(n - cn/\sqrt{\ln n})^2 - 3(n - cn/\sqrt{\ln n})c'\sqrt{n}\ln n}{n(n-1)} \\
&> \frac{3n^2 - 6cn/\sqrt{\ln n} - 3c' n^{3/2}\sqrt{\ln n}}{n^2} \\
&> 3 - \frac{6c}{n\sqrt{\ln n}} - \frac{3c'\sqrt{\ln n}}{\sqrt{n}} \\
&> 3 - O\left(\sqrt{\frac{\log n}{n}}\right) ,
\end{aligned}$$

completing the proof. □

Next we present a lower bound for the average stretch factor which is independent on the choice of the pivots and the node identities. According to Theorem 3.3, this bound is tight.

Theorem 3.8 *For every set of pivots P, and for every $R \in PIRS1[P]$, $\mathrm{AvStr}_{K_n}(PIR1[P]) = 2 - \Theta\left(\sqrt{(\log n)/n}\right)$.*

Finally, we bound the average stretch factor of PIR1 for almost all unweighted graphs.

Theorem 3.9 *For almost every n-node graph G,*

$$1.875 - \delta \leq \mathrm{AvStr}_G(PIR1[P]) \leq 1.875 + \delta$$

with

$$\delta = O\left(\sqrt{\frac{\log n}{n}}\right).$$

4 Lower Bounds on Loop-Free Routing Schemes

It is relevant to ask if an amount of $O(\sqrt{n \log n})$ bits of memory in each node (on average) for some stretch factor s is the best we can hope for with loop-free routing schemes. It turns out that a lower bound of $\Omega(\sqrt{n})$ memory bits holds for any stretch factor. To prove it, we will take advantage of the fact that loop-free schemes route optimally on trees.

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

Applying Theorem 3.1, since the schemes of PIRS1 are loop-free and of bounded stretch factor, it follows that they are shortest path routing schemes for trees.

Theorem 4.2 *There exists an n-node graph G such that for every s, and every loop-free routing scheme R on G of stretch factor s that uses names taken from $\{1, \ldots, n\}$, there is a node x such that $\mathrm{Memory}_G(R, x) \geq c\sqrt{n} - O(\log n)$, where $c \approx 3.71$.*

Proof. Consider a tree T of n nodes and root x, and let $d = \deg(x)$. Let R be any loop-free routing scheme on T, and let $l_x \in \{1, \ldots, n\}$ be the label of x under R. By Lemma 4.1, the local routing function R_x routes along shortest paths. For every $i \in \{1, \ldots, d\}$, denote by n_i the number of nodes in the subtree of T rooted at the ith child of x. Without loss of generality assume $n_1 \leq \ldots \leq n_d$. Observe that knowing n, d and l_x suffices for computing the sequence $S = (n_1, \ldots, n_d)$ with R_x. Indeed, it is sufficient to enumerate the set of labels distinct of l_x, and compute the answer of R_x for each of them. The port number must be the same for all the nodes that belong to the same subtree. We remark that $\sum_{i=1}^{d} n_i = n - 1$.

Let $K(X)$ be the minimum number of bits required in order to code an object X. By the above discussion, $K(S) \leq K(R_x) + O(\log n)$, because S can be described knowing R_x, n, d and l_x (the last three integers can be stored using $O(\log n)$ bits). A *partition* of n is an integer sequence $U = (u_1, \ldots, u_k)$ such that $1 \leq u_1 \leq \ldots \leq u_k$ and $\sum_{i=1}^{k} u_i = n$. Note that for each partition U of $n-1$ there exists a distinct n-node tree T_U, with root u of degree $\deg(u) = k$, whose children induce subtrees of size exactly u_i, for $i \in \{1, \ldots, k\}$. Hence for every loop-free routing R' on T_U, $K(R'_u) \geq \log_2(p_{n-1}) - O(\log n)$, where p_n denotes the number of partitions of n.

We now rely on the following well-known formula due to Harder and Ramanujan [20, Equation (4.2.7) page 44].

$$\ln p_n \sim c'\sqrt{n}, \quad \text{for } c' = \pi\sqrt{2/3}.$$

Thus for sufficiently large n, $K(R'_u) \geq c\sqrt{n} - O(\log n)$, where $c \approx 3.71$, completing the proof. □

Note that an upper bound of $c\sqrt{n}$ bits per node for trees has been established in [15, page 6] for shortest path loop-free 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 $k \geq 1$, is presented in [1]. Moreover, the generated schemes are name-independent. It follows that for any graph satisfying Theorem 4.2 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 $k = 3$). 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.

As done previously in Theorem 3.4, one may ask whether the stretch factor guaranteed by the PIR1 strategy is the best that can be achieved with memory requirements $O(n^{3/2} \log^{3/2} n)$. Actually, the smallest possible stretch factor one can hope for when considering a routing scheme with a memory requirements $o(n^2)$ is $s = 3$. Indeed, in [16] it is shown that there exist some graphs on which every routing scheme of stretch factor $s < 3$ requires at least $\Omega(n^2)$ bits.

5 Comparison with General Routing Schemes

This section compares the PIR1 strategy with other general strategies. Let HCP_k be the routing strategy developed in [2] called *Hierarchical Covering Pivot*.

Theorem 5.1 (Awerbuch et al. [1]) *For every n-node weighted graph G, and for every integer $k \geq 1$, and there exists $R \in \mathrm{HCP}_k$ on G such that* $\mathrm{Memory}_G(R) = O(k\, n^{1+1/k} \log^2 n)$, *and* $\mathrm{Stretch}_G(R) \leq 2^k - 1$. *Moreover R is constructible in polynomial time.*

The HCP_k strategy first selects a hierarchy of sets of pivots, $P_{k-1} \subset \ldots \subset P_1 \subset P_0 = V$, each one of size about $n^{1-i/k}$. Each pivot $v \in P_i$ is able to route on a shortest path to vertices in its neighborhood of size roughly $n^{(i+1)/k}$. When a vertex u wants to send a message to v, the message is transferred trough the chain of pivots associated with u from successively higher levels, until it reaches a pivot able to route directly to v. For concreteness, let us consider $k = 2$. Then the scheme generated by HCP_2 routes messages trough one level of pivots, P_1. Hence the route from u to v follows, successively, u, $p(u)$ (the pivot closest to u in P_1), then v. In comparison, the route generated by the PIR1 or PIR2 strategy would be u, $p(v)$, and v, allowing a loop-free routing, and bounding the compactness of its interval routing implementation.

The HCP_k strategy is name-independent. However, in the complexity memory analysis, it is assumed that the length of node identities is $O(\log n)$ bits (the pivots need to keep some translation tables storing the identities of some nodes). Note also that the space complexity analysis of the schemes can be slightly improved, to show that $\mathrm{Memory}_G(R) = O(k\, n^{1+1/k} \log^{2-1/k} n)$, for every $R \in \mathrm{HCP}_k$.

As done previously, given a set of pivots P, we denote by $\mathrm{HCP}_k[P]$ the subset of HCP_k that consists of only the covering pivot routing schemes having a pivot set P. It is easy to see that, for a given pivot set P of a graph G, $\mathrm{AvStr}_G(\mathrm{HCP}_{k+1}[P]) \geq \mathrm{AvStr}_G(\mathrm{HCP}_k[P])$. Moreover $\mathrm{Memory}(\mathrm{HCP}_1, G) = O(n^2 \log n)$ which is higher than the memory requirements of the PIR strategy. This is why, in the remainder of the paper, we are interested in the schemes of the strategy HCP_2. In [1], the set of schemes HCP_2 is called *Covering Pivot*. For simplicity we denote $\mathrm{CP} = \mathrm{HCP}_2$.

We first show that in terms of their average stretch factor, the schemes of PIR1 are asymptotically never worse than those of CP.

Theorem 5.2 *Let G be an n-node graph, let P be an arbitrary set of pivots of G, $R_{PIR1} \in PIRS1[P]$, and $R_{CP} \in \mathrm{CP}[P]$. Then* $\mathrm{AvStr}_G(R_{PIR1}) \leq \mathrm{AvStr}_G(R_{CP}) + 2\sqrt{(1+\ln n)/(n-1)}$.

As a corollary of Theorem 5.2, we have that $\mathrm{AvStr}_{K_n}(\mathrm{CP}) = 2 - O(\sqrt{(\log n)/n})$, and also that there exists some worst-case graph G with specific node labeling such that $\mathrm{AvStr}_G(\mathrm{CP}[P]) = 3 - o(1)$, for every pivot set P chosen randomly or computed with the greedy algorithm. Similarly, almost every graph G satisfies $\mathrm{AvStr}_G(\mathrm{CP}) \geq 1.875 - O(\sqrt{(\log n)/n})$.

Next we illustrate the advantage of the PIR1 strategy over the CP strategy w.r.t. average stretch factor.

Theorem 5.3 *Let $T = K_{1,n-1}$, and let r be its root. If* $\mathrm{ID}(r)$ *is not the lowest* ID, *then for every set of pivots P of T computed by the random or the greedy cover algorithm,* $\mathrm{AvStr}_T(\mathrm{CP}[P]) > 2 - o(1)$ *(w.h.p.), whereas* $\mathrm{AvStr}_T(PIR1[P]) = 1$.

6 Conclusion

In high-speed networks the bottleneck is the delay in the nodes and not the propagation time of the messages. As high-speed networks gain popularity, it is essential to design direct routing schemes, which could be implemented in hardware and at the same time, when dealing with large networks, it is important to decrease the amount of memory kept in the nodes for routing purposes. In this work we present for the first time a routing strategy that generates for every (weighted) network, a direct routing scheme with small stretch factor and modest memory requirements. On the other hand, we show that there exist n-node graphs on which any loop-free routing scheme uses $\Omega(\sqrt{n})$ bits of memory in at least one node, 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]).

Following are some directions for further research.

- Is it possible to improve the worst case stretch factor using the same amount of memory? By [16], it cannot be improved below 3.
- What is the best trade-off between the memory requirements and the (average) stretch factor for 1-phase routing schemes?

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

References

[1] Awerbuch, B., Bar-Noy, A., Linial, N., and Peleg, D. Compact distributed data structures for adaptive routing. In 21^{st} *Symposium on Theory of Computing (STOC)* (1989), vol. 2, pp. 230–240.

[2] Awerbuch, B., Bar-Noy, A., Linial, N., and Peleg, D. Improved routing strategies with succint tables. *Journal of Algorithms 11* (Feb. 1990), 307–341.

[3] Awerbuch, B., and Peleg, D. Sparse partitions. In 31^{th} *Symposium on Foundations of Computer Science (FOCS)* (1990), pp. 503–513.

[4] Eilam, T., Gavoille, C., and Peleg, D. Compact routing schemes with low stretch factor. Research Report RR-1195-98, LaBRI, University of Bordeaux, 351, cours de la Libération, 33405 Talence Cedex, France, Jan. 1998.

[5] Eilam, T., Moran, S., and Zaks, S. A lower bound for linear interval routing. In 10^{th} *International Workshop on Distributed Algorithms (WDAG)* (1996), vol. 1151 of Lecture Notes in Computer Science, Springer-Verlag.

[6] Flammini, M., Gambosi, G., Nanni, U., and Tan, R. B. Multi-dimensional interval routing schemes. In

9^{th} *International Workshop on Distributed Algorithms (WDAG)* (1995), vol. 972 of Lecture Notes in Computer Science, Springer-Verlag. To appear in TCSA.

[7] Flammini, M., and Nardelli, E. On the path length in interval routing schemes. Manuscript, 1996.

[8] Fraigniaud, P., and Gavoille, C. Memory requirement for universal routing schemes. In 14^{th} *Annual ACM Symposium on Principles of Distributed Computing (PODC)* (Aug. 1995), pp. 223–230.

[9] Frederickson, G. N., and Janardan, R. Seperator-based strategies for efficient message routing. In 27^{th} *Symposium on Foundations of Computer Science (FOCS)* (May 1986), pp. 428–437.

[10] Frederickson, G. N., and Janardan, R. Designing networks with compact routing tables. *Algorithmica 3* (Aug. 1988), 171–190.

[11] Frederickson, G. N., and Janardan, R. Efficient message routing in planar networks. *SIAM Journal on Computing 18*, 4 (Aug. 1989), 843–857.

[12] Frederickson, G. N., and Janardan, R. Space-efficient message routing in c-decomposable networks. *SIAM Journal on Computing 19*, 1 (Feb. 1990), 164–181.

[13] Gambosi, G., and Vocca, P. Topological routing schemes. In 10^{th} *International Workshop on Distributed Algorithms (WDAG)* (Oct. 1996), vol. 1151 of Lecture Notes in Computer Science, Springer-Verlag.

[14] Gavoille, C. On the dilation of interval routing. In 22^{nd} *International Symposium on Mathematical Foundations of Computer Science (MFCS)* (Aug. 1997), vol. 1300 of Lecture Notes in Computer Science, Springer-Verlag, pp. 259–268.

[15] Gavoille, C. A survey on interval routing scheme. Research Report RR-1182-97, LaBRI, University of Bordeaux, 351, cours de la Libération, 33405 Talence Cedex, France, Oct. 1997. Submitted for publication.

[16] Gavoille, C., and Gengler, M. Space-efficiency of routing schemes of stretch factor three. In 4^{th} *International Colloquium on Structural Information & Communication Complexity (SIROCCO)* (July 1997).

[17] Gavoille, C., and Guévremont, E. Worst case bounds for shortest path interval routing. Research Report 95-02, LIP, École Normale Supérieure de Lyon, 69364 Lyon Cedex 07, France, Jan. 1995. To appear in Journal of Algorithms.

[18] Gavoille, C., and Peleg, D. The compactness of interval routing. Research Report RR-1176-97, LaBRI, University of Bordeaux, 351, cours de la Libération, 33405 Talence Cedex, France, Sept. 1997. Submitted for publication.

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

[20] Hall, M. J. *Combinatorial Theory (second edition).* Wiley-Interscience Publication, 1986.

[21] Inmos. *The T9000 Transputer Products Overview Manual*, 1991.

[22] Kráľovič, R., Ružička, P., and Štefankovič, D. The complexity of shortest path and dilation bounded interval routing. In 3^{rd} *International Euro-Par Conference* (Aug. 1997), vol. 1300 of Lectures Notes in Computer Science, Springer-Verlag, pp. 258–265.

[23] Lovász, L. On the ratio of optimal integral and fractional covers. *Discrete Mathematics 13* (1975), 383–390.

[24] Peleg, D. An overview of locality-sensitive distributed computing, Unpublished Monograph, The Weizmann Institute, Rehovot, Israel, 1997.

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

[26] Santoro, N., and Khatib, R. Labelling and implicit routing in networks. *The Computer Journal 28*, 1 (1985), 5–8.

[27] Tse, S. S. H., and Lau, F. C. M. An optimal lower bound for interval routing in general networks. In 4^{th} *International Colloquium on Structural Information & Communication Complexity (SIROCCO)* (July 1997).

[28] van Leeuwen, J., and Tan, R. B. Routing with compact routing table. In *The Book of L* (1986), Springer-Verlag, pp. 259–273.