JOURNAL OF ALGORITHMS 11, 307–341 (1990)

Improved Routing Strategies with Succinct Tables

BARUCH AWERBUCH*

Laboratory for Computer Science, MIT, Cambridge, Massachusetts 02138

AMOTZ BAR-NOY†

Computer Science Department, Stanford University, Stanford, California 94305

NATHAN LINIAL

IBM Almaden Research Center, 650 Harry Road, San Jose, California 95120 and the Computer Science Department, Hebrew University, Jerusalem

AND

DAVID PELEG‡

Department of Applied Mathematics, The Weizmann Institute, Rehovot 76100, Israel

Received February 2, 1990

In designing a routing scheme for a communication network it is desirable to use as short as possible paths for routing messages, while keeping the routing information stored in the processors' local memory as succinct as possible. The efficiency of a routing scheme is measured in terms of its *stretch factor*—the maximum ratio between the cost of a route computed by the scheme and that of a cheapest path connecting the same pair of vertices. This paper presents several simple families of routing schemes for general networks, featuring some desirable properties. Our two main families are the *hierarchical covering pivots* schemes HCP_k and the *hierarchical balanced* schemes HB_k (for $k \geq 1$). The scheme HCP_k guarantees a stretch factor of $2^k - 1$ and requires storing a total of $O(k \cdot n^{1+1/k} \log^2 n)$ bits of routing information in the network. The new important

*Supported by Air Force Contract TNDGAFOSR-86-0078, ARO Contract DAAL03-86-K-0171, NSF Contract CCR8611442, and a special grant from IBM.

†Supported in part by a Weizmann fellowship, by Contract ONR N00014-88-K-0166 and by a grant of Stanford Center for Integrated Systems.

‡Part of this work was carried out while this author was visiting Stanford University. Supported in part by a Weizmann fellowship, by Contract ONR N00014-88-K-0166 and by a grant of Stanford Center for Integrated Systems.

features of these schemes are applicability to networks with arbitrary edge costs and attractive stretch factors for small values of k. The purpose of the second method is to provide balanced bounds on the memory requirements of the individual vertices. The scheme HB_k guarantees a stretch factor of $2 \cdot 3^k - 1$ and requires storing at most $O(k \log n(d + n^{1/k}))$ bits of routing information in a vertex of degree d and $O(kn^{1+1/k} \log n)$ bits overall. We also describe an efficient distributed preprocessing algorithm for this scheme, which requires same amount of space.

1. Introduction

1.1. *Background*

A central activity of any computer network is the passing of messages among the processors. This activity is performed by a routing subsystem, consisting of a collection of message forwarding mechanisms and information tables, whose quality is pivotal to the overall performance of the network. It is therefore natural that the design of efficient routing schemes was the subject of much study over the last two decades.

In this paper we study the question of designing efficient schemes with respect to two central parameters, namely, *route efficiency* and *memory requirements*. When designing a routing strategy for a network it is clearly desirable to be able to route messages with small communication cost. At the same time, the space used for the routing tables is also a significant consideration. There are a number of reasons to minimize the memory requirements of a routing scheme. The task of routing is usually performed by a special-purpose processor (an "IMP" in the ISO terminology [T, Z]) which may have limited resources. Furthermore, it is usually desirable that the routing tables be kept in fast memory (e.g., a "cache"), in order to expedite message traffic. Also, we do not want memory requirements to grow fast with the size of the network, since it means that the incorporation of new vertices to the network requires adding hardware to all the vertices in the network. It is therefore interesting to search for routing schemes that involve small communication cost and have low space requirements at the individual vertices.

Technically, the two parameters are measured as follows. We assume that there is a *cost function* associated with the network links. The cost of routing a message is simply the sum of the costs of the transmissions performed during the routing. The *route efficiency* of a routing scheme is formulated in terms of its *stretch factor*—the maximum ratio (over all possible processor pairs) between the communication cost of a route produced by the scheme and the cheapest possible cost for passing a message between the same pair of processors. The *space requirement* of a scheme is the maximum number of memory bits used by any single processor to store the routing information.

Let us demonstrate the combined behavior of these two parameters by looking at two extreme examples. The *direct* routing scheme in an n-processor network is constructed by specifying, at each vertex v, a set of $n - 1$ pointers, one pointer for each possible destination vertex $x \neq v$. Each such pointer points to some neighbor w of v, to which v will forward a message destined to x. The message is forwarded along those pointers until it eventually arrives at its destination. Clearly, it is advantageous to set up the pointers with respect to a fixed destination x in such a way that they form a tree of shortest paths towards x, based on the edge costs. Then the communication cost of such a routing is optimal, i.e., the stretch factor is 1. The disadvantage of the direct routing scheme is that each vertex has to maintain a very large ($\Omega(n)$ bit) routing table.

At the other extreme lies the *flooding* routing scheme, in which instead of forwarding a message along a shortest path, the sender simply floods (broadcasts) it through the whole network. Clearly, this scheme requires no memory overhead. On the other hand, the communication cost of such a scheme may be significantly higher than optimal, since instead of using just one link, we may be using a lot of (possibly expensive) links. Thus, the stretch factor is unbounded.

The natural question which arises here is whether one can design a routing scheme which combines low memory requirements and small communication cost.

1.2. *Existing Work*

The problem was first raised in [KK1]. However, in order to apply the methods of [KK1] and several consequent works [KK2, P, S] one needs to make some fairly strong assumptions regarding the existence of a certain partition of the network. Such a partition does not always exist, and furthermore, no algorithm is provided for constructing it when it does exist. Also, somewhat stronger technical assumptions are required in order to guarantee good performance bounds.

Most consequent work on the problem has focused on solutions for special classes of network topologies. Optimal or near-optimal routing strategies were designed for various topologies like trees [SK], rings, complete networks and grids [vLT1, vLT2], series-parallel networks, outer-planar, k-outerplanar, and planar networks [FJ1, FJ2]. (By "optimal" we mean here stretch factor 1 and a total memory requirement of $O(n \log n)$ bits in an n-processor network.)

In [PU] the problem is dealt with for general networks. Rather than designing a scheme only for some fixed stretch factor, the method presented in [PU] is parameterized and applies to the entire range of possible stretch factors. The construction yields an almost optimal behavior, as implied from a lower bound given in [PU] on the space requirement of any scheme with a given stretch factor. Specifically, the hierarchical routing schemes of [PU] guarantee a stretch factor of $12k + 3$ while requiring a total of $O(n^{1+1/k})$ bits of routing information in the network (for every fixed $k \geq 1$).

Unfortunately, the routing strategy of [PU] suffers from several deficiencies which make it impractical. One weakness of the solution of [PU] is that it deals only with unit-cost edges, while the construction of [KK1] and the separator-based strategies of [FJ2], for instance, apply also to the case of networks with costs on the edges. This property may be important for practical applications, where edge costs are sometimes used to reflect physical distances, estimated link delays and congestion conditions in the network.

A second problem is that in [PU] the initial design has to determine not only the routes but also the labels used for addressing the vertices. This is inappropriate, since it would require changing the addresses of vertices each time the routes are recomputed. In fact, it is preferable to allow each vertex to choose its own address.

A third problem is that the method bounds the *total* space requirements of the scheme, but not the *individual* memory requirements of each vertex. (This problem exists also in [FJ1, FJ2].) In particular, some vertices are designated as *centers* and are required to store more information than others. Other vertices may just happen to be the crossing point of many routes which are maintained by the scheme. It is true that such a situation frequently occurs in real networks, and in fact, some vertices are intentionally designated to play the role of communication centers. However, these centers are selected according to the presence of appropriate resources at specific vertices or edges. The method of [PU] assigns the centers based on some graph-theoretic considerations which need not necessarily coincide with the practical ones. Moreover, a good bound on the amount of memory used in routing tables is important even in major vertices where memory is not a problem, since as mentioned earlier, the special-purpose routing processor (the "IMP") may have limited resources and small fast memory.

Finally, the preprocessing algorithm proposed in [PU] is a centralized (polynomial time) algorithm. One would prefer a preprocessing algorithm which is distributed and space efficient. (Again, this issue was not considered in previous work as well.) In particular, it is preferable that the preprocessing algorithm obey the same space constraints imposed on the size of routing tables in the individual vertices. (This is of secondary importance when the update phase is allowed access to auxiliary, slower memory.)

1.3. *Contributions of This Paper*

In this paper we propose several new routing methods which attempt to overcome the problems mentioned above. In particular, all of the schemes presented below solve the problem for arbitrary networks with arbitrary (non-negative) edge costs and have efficient distributed preprocessing algorithms.

The main two methods have the following characteristics. The first method yields the family of *hierarchical covering pivots* schemes, HCP_k. For every $k \geq 1$, the scheme HCP_k uses $O(kn^{1+1/k} \log^2 n)$ bits of memory throughout the network and guarantees a stretch of at most $2^k - 1$. While this method is less efficient asymptotically than that of [PU], it is much simpler and yields attractive performance for small values of k. For example, for $k = 2$, i.e., using $O(n^{3/2} \log^2 n)$ bits of memory, the stretch is at most 3, rather than 27 in the scheme of [PU]. Further, unlike [PU], the schemes have the advantage of not having to specify special addressing labels, i.e., the original vertex names may be used for addressing purposes.

The main problem left unsolved by the first method is that the memory requirements of vertices are not balanced, and some vertices may require much more memory than the average. The second method we present overcomes this problem and yields a family of *hierarchical balanced schemes*, HB_k. For every $k \geq 1$, the scheme HB_k uses $O(k \log n(d + n^{1/k}))$ bits of memory in a vertex with degree d and $O(kn^{1+1/k} \log n)$ bits of memory overall, and guarantees a stretch of at most $2 \cdot 3^k - 1$. However, in these schemes we have to choose new names to be used by vertices for addressing purposes. For these schemes we also present an efficient distributed preprocessing algorithm for setting up the tables, that uses space which is bounded by the same bounds per vertex.

Our approach is based on the use of communication centers, or "pivots," spread throughout the network. A regular vertex can send messages directly only to a small subset of vertices in its nearby vicinity. Messages to other vertices have to be forwarded through the pivots. Our two hierarchical schemes are based on hierarchies of pivots, where pivots on higher levels control larger "zones" in the network.

This indirect forwarding design enables us to reduce the memory requirements needed for routing, since most vertices need to store routing information only for a small section of the network. On the other hand, it increases communication cost, since messages need not, in general, be moving along shortest paths towards their destination. An appropriate partition of the network into clusters and appropriate selection and assignment of pivots, can guarantee that both overheads are low.

Intuitively, in the hierarchical covering pivots schemes, the pivots played the role of "mail boxes"; that is, the hierarchical organization of the pivots centered on the process of *collecting* messages from their senders. In contrast, in the hierarchical balanced schemes, the pivots act as "post offices"; that is, the hierarchical organization is based on the process of *distributing* messages to their destinations.

The particular construction methods described here differ from that of [PU] in several important ways. To begin with, the new methods make use of inherently different *hierarchical designs*. In [PU], the scheme is composed of a *collection of independent* schemes. Each scheme $\mathcal{RES}_i$, $i \geq 1$, is based on a partition of the network into clusters of radius 2^i. Transmitting a message from a sender to a destination is based on a "trial and error" process involving repeated trials, each attempting to route the message using *one* of the individual schemes. In case of failure, the message is returned to the sender, who then tries the next scheme in the hierarchy. The route of each individual scheme is itself a complex path, composed of three segments: from the sender to its cluster leader in that partition, from this leader to the leader of the destination and finally to the destination. In contrast, the routing process described here is conceptually much simpler. A message is passed from the sender to the destination in a single try, and no retries are needed. In our first method, the path consists of two segments: from the sender to its pivot on the appropriate level (through a chain of lower-level pivots), and then to the destination itself. In our second method the path again consists of two segments: from the sender to the pivot of the destination on the appropriate level, and then to the destination itself (through a chain of lower-level pivots). Thus the hierarchical organization is utilized *internally* within the (single) routine scheme.

A second important difference is in the clustering method. The clusters described in [PU] are based on *radius* constraints, whereas the clustering structure proposed here is based on *size* constraints. This difference is responsible for the fact that the new methods are capable of handling arbitrary edge costs and that individual memory requirements can be bounded.

The rest of the paper is organized as follows. Section 2 contains necessary definitions. In Section 3 we introduce some preliminaries. In Section 4 we present three routing methods of increasing complexity, the last of which is the HCP_k family. Section 5 presents the second method giving the balanced routing schemes HB_k. Finally Section 6 handles the issue of efficient preprocessing algorithms for the balanced schemes.

2. Definition of the Problem

2.1. *Network Model*

We consider the standard model of a point-to-point communication network, described by a connected undirected interconnection graph $G = (V, E)$, $V = \{1, \dots, n\}$. The vertices represent the processors of the network and the edges represent bidirectional communication channels between the vertices. The degree (number of neighbors) of each vertex $v \in V$ is denoted by $\deg(v)$. A vertex may communicate directly only with its neighbors, and messages between nonneighboring vertices are sent along some path connecting them in the network. We also assume that messages sent and variables maintained at vertices contain $O(\log n)$ bits.

A *cost* value $\mathrm{cost}(e) > 0$ is associated with each edge $e \in E$. For two vertices u, w let $\mathrm{dist}(u, w)$ denote the cost of the cheapest path connecting them, where the cost of a path $(e_1, \dots, e_s)$ is $\Sigma_{1 \le i \le s} \mathrm{cost}(e_i)$. For two sets of vertices U, W, let $\mathrm{dist}(U, W) = \min\{\mathrm{dist}(u, w) | u \in U,\ w \in W\}$. The *diameter* of the network G is defined as

$$\mathrm{diam}(G) = \max\{\mathrm{dist}(u, w) | u, w \in V\}.$$

A *local radius* of a graph G at a vertex x is

$$r_x(G) = \max_{v \in V}\{\mathrm{dist}(x, v)\}.$$

A *center* of a graph G is any vertex c such that for every $v \in V$,

$$r_c(G) \le r_v(G).$$

The *radius* of the graph is the local radius $r_c(G)$ of some center c.

2.2. *On-line and Off-line Routing Schemes*

A *routing scheme RS* for the network G consists of two procedures, a *preprocessing* procedure $\mathrm{Pre}(RS)$ and a *delivery* protocol $\mathrm{Delivery}(RS)$. The preprocessing procedure $\mathrm{Pre}(RS)$ performs the necessary preprocessing in the network, e.g., constructs certain distributed data structures. The delivery protocol $\mathrm{Delivery}(RS)$ can be invoked at any vertex u with a *destination* parameter v, and a *message* parameter M. The protocol then delivers the message M from the sender u to the destination v via a sequence of message transmissions. This sequence of message transmissions depends on the particular data structures constructed by the preprocessing protocol. Thus the protocol specifies for each pair of vertices $u, v \in V$ a route $\rho(RS, u, v)$ in the network connecting u to v. (We sometimes omit the parameter RS when it is clear from the context.)

We consider two versions of the problem. In the *off-line* version we apply a centralized sequential algorithm that uses full information about the network and computes the appropriate routes and distributed data structures. In this version we do not limit the space used by the preprocessing protocol Pre(RS). The *on-line* version requires computing the routes and the necessary data structures in a distributed way. Further, the space used by the preprocessing protocol Pre(RS) must not exceed the space used by the delivery protocol Delivery(RS).

2.3. *Complexity Measures for Routine Schemes—Stretch and Memory*

The communication cost of the delivery protocol when invoked at a vertex u with a destination v and an $O(\log n)$-bit message is

$$|\rho(RS,u,v)| = \sum_{e\in\rho(RS,u,v)} \operatorname{cost}(e)$$

(i.e., the cost of the path $\rho(RS,u,v)$ through which the message is transmitted). Given a routine scheme RS for an n-processor network $G=(V,E)$, we say that RS *stretches* the path from u to v (for every $u,v\in V$) by $|\rho(RS,u,v)|/\operatorname{dist}(u,v)$. We define the *stretch factor* STRETCH(RS) of the scheme RS to be

$$\operatorname{STRETCH}(RS) = \max_{u,v\in V}\left\{\frac{|\rho(RS,u,v)|}{\operatorname{dist}(u,v)}\right\}.$$

Comment. STRETCH(RS) can be viewed as measuring essentially the "normalized" communication complexity of the routing scheme, i.e., the ratio of the actual communication cost to the optimal communication cost.

The *vertex memory* $\text{Memory}_{\text{vertex}}(\pi,v)$ of a protocol π at a vertex v is the number of bits maintained by the vertex. The *total memory* $\text{Memory}_{\text{total}}(\pi)$ of the protocol is the total number of memory bits maintained by all network vertices. The *vertex memory* $\text{Memory}_{\text{vertex}}(RS,v)$ of a routing scheme RS (at a vertex v) is the vertex memory requirement of the delivery protocol of RS for the off-line problem, and is the maximum between the memory requirements of the delivery protocol and the preprocessing protocol of RS for the on-line problem. Define $\text{Memory}_{\text{total}}(RS)$ similarly.

3. Technical Preliminaries

3.1. *Neighborhoods*

Our schemes are based on a notion of *neighborhood* which is defined by *volume* rather than radius (as in [PU]). The *neighborhood* of a vertex

$v \in V$ with respect to a specific set of destinations $S \subseteq V$ and a parameter $1 \leq j \leq n$, is a collection of j of the nearest vertices to v from the set S. More precisely, order the vertices of S by increasing distance from v, breaking ties by increasing vertex names. Hence $x \prec_v y$ if either $\mathrm{dist}(x, v) < \mathrm{dist}(y, v)$ or $\mathrm{dist}(x, v) = \mathrm{dist}(y, v)$ and $x < y$. Then $N(v, j, S)$ contains the first j vertices in S according to the order $\prec_v$. (In some of our schemes, e.g., the ones described in Section 4, we can do with neighborhoods defined in a more relaxed manner, in which ties are resolved arbitrarily.)

The *radius* of the neighborhood $N(v, j, S)$ is defined as

$$r(v,j,S) = \max_{x \in N(v,j,S)} \mathrm{dist}(v,x).$$

When $S = V$ we sometimes omit the third parameter and write simply $N(v, j)$ or $r(v, j)$. The basic useful fact regarding the radius is

FACT 3.1. *For every vertex w*:

1. If $w \in N(v, j, S)$ then $\mathrm{dist}(v, w) \leq r(v, j, S)$.
2. If $w \in S - N(v, j, S)$ then $\mathrm{dist}(v, w) \geq r(v, j, S)$.

One more fact we need about the sets $N(v, j, S)$ is the following:

LEMMA 3.1. *For every set $S \subseteq V$, vertices $u, w \in V$ and integer $1 \leq j \leq n$,*

$$r(u,j,S) \leq r(w,j,S) + \mathrm{dist}(u,w).$$

Proof. By contradiction, relying on the fact that $|N(u, j, S)| = |N(w, j, S)| = j$. □

3.2. *Covers*

For the construction of our routing schemes we need some basic facts concerning the concept of covers. Consider a collection $\mathscr{H}$ of subsets of size s of a set B. A set of elements $M \subseteq B$ is said to *cover* those sets of $\mathscr{H}$ that contain at least one element of M. A *cover* of $\mathscr{H}$ is an $M \subseteq B$ covering all sets in $\mathscr{H}$. A *fractional cover* for $\mathscr{H}$ is a system of nonnegative real weights $\{t_v | v \in B\}$ such that $\sum_{x \in S} t_x \geq 1$ for every set $S \in \mathscr{H}$. Let $\tau^* = \min \sum_{x \in B} t_x$ where the minimum is taken over all fractional covers. Since the weight system assigning $t_v = 1/s$ to every vertex v is a fractional cover, we get

FACT 3.2. $\tau^* \leq |B|/s$.

Consider the following two simple procedures for creating a cover for $\mathscr{H}$.

Greedy Cover Algorithm. Start with $M = \emptyset$ and iteratively add an element that increases the number of sets covered by M as much as possible. Stop when M becomes a cover. This M is called a *greedy cover* for $\mathscr{H}$.

We rely on the following lemma of Lovász.

Lemma 3.2 (L). *Let M be a greedy cover for $\mathscr{H}$. Then*

$$|M| < (\ln|\mathscr{H}| + 1)\tau^* \leq \frac{(\ln|\mathscr{H}| + 1)|B|}{s}.$$

For the random algorithm we assume that $|B| \geq 2s$ and that $\ln \mathscr{H} = o(s)$.

Random Cover Algorithm. Randomly select each element $v \in B$ into the set M with probability $p = c \ln|\mathscr{H}|/s$, where $c > 1$ is a constant.

In the proof of the next lemma we use the following propositions. The first states the Chernoff estimate [C] for the tails of the binomial distribution and the second is an easy fact.

Proposition 3.1 (Chernoff). *Let X be a random variable with a binomial distribution and expectation $E = E(X)$. Then for $\alpha > 0$,*

$$\Pr(X > (1 + \alpha)E) < e^{-\alpha^2 E/2}$$

Proposition 3.2. *For $0 < \alpha \leq \frac{1}{2}$ and $\beta > 0$,*

$$(1 - \alpha)^{\beta} < e^{-\alpha\beta}.$$

Lemma 3.3. *Let M be a set constructed by the randomized algorithm under the above assumptions. Then, with probability at least $p = 1 - 1/|\mathscr{H}|^{c-1}$,*

1. *M is a cover for $\mathscr{H}$, and*
2. *$|M| \leq 2c|B|\ln|\mathscr{H}|/s$.*

Proof. Denote by q the probability that there exists a set $S \in \mathscr{H}$ that is not covered by M. For a fixed $S \in \mathscr{H}$ the probability that no $v \in S$ has been selected is bounded by $(1 - p)^s$. It follows that $q \leq |\mathscr{H}|(1 - p)^s$. Using Proposition 3.2 we get that $q \leq |\mathscr{H}|e^{-ps}$. Plugging in the value of p, $c \ln|\mathscr{H}|/s$, we get that $q \leq 1/|\mathscr{H}|^{c-1}$, thus proving the first part of the lemma.

The expected size of M is $E = E(|M|) = p|B| = c|B|\ln|\mathscr{H}|/s$. Using Chernoff bound (Proposition 3.1) for $\alpha = 1$ we get that

$$\Pr\left(|M| \geq \frac{2c|B|\ln|\mathscr{H}|}{s}\right) \leq e^{-c|B|\ln|\mathscr{H}|/2s} = |\mathscr{H}|^{-c|B|/2s}.$$

The second part of the lemma is implied as we assumed that $|B| \geq 2s$. □

4. Pivot-Based Routing Schemes

In this section we propose several pivot-based strategies for designing routing schemes. The first subsection describes one of our basic tools, called interval tree routing. The following three subsections present and analyze three families of routing schemes of increasing complexity, namely, the single pivot schemes SP_k, the covering pivots schemes CP and the hierarchical covering pivots schemes HCP_k.

4.1. *Interval Tree Routing*

In this section we describe a routing component introduced in [PU], which is based on the routing scheme of [SK]. We call this component an *interval tree routing*. It is defined for a subgraph $G' = (V', E')$ and a designated vertex $r \in V'$, and is denoted $ITR^*(G', r)$ or $ITR^*(V', r)$.

Preprocessing. The scheme is constructed as follows:

1. Construct a BFS (shortest path) spanning tree T for G', rooted at r.

2. Assign the vertices of V' a DFS (pre-order) numbering according to T.

3. Associate with each vertex v an *interval label* in the form of a pair of integers $\text{int}(v) = (\textit{Lowest}, \textit{Highest})$ such that the interval $[\textit{Lowest}, \textit{Highest}]$ contains precisely the pre-order numbers of the vertices of V' in the subtree rooted at v. (Note that *Lowest* is simply the DFS number of v itself.) In particular, the interval attached to v is contained in that of u iff v is a descendant of u.

4. The root r of an $ITR^*(V', r)$ component stores a *translation table* TT_r, specifying for every vertex $v \in V'$ its interval label $\text{int}(v)$ in the component.

Delivery. Routing a message from a vertex u to a vertex w for $u, w \in V'$ involves sending the message upwards in the tree until it reaches the root, and then downwards to w. More specifically, this routing is done as follows:

1. The sender attaches a header to the message, containing an identifier for the routing component used (in order to enable us to use this component in a network when other routing components are present as well), the name of the final destination, w, and the name of the intermediate destination r, the root of the component.

2. The message is then sent to r (note that it is always possible to send a message *upwards* to the root in an ITR^* component).

3. The root r modifies the header by eliminating its own name and attaching the interval label $\text{int}(w)$ of w.

4. Finally, the root sends the message down the tree to w.

It should be clear that this is not necessarily an optimal route in G'. However, the length of the route is at most twice the height of the tree.

Complexity. Maintaining such a component involves the following memory requirements. Each vertex v must store its own interval label $\text{int}(v)$ as well as the interval label $\text{int}(w)$ of each of its children w in the tree. In addition it has to store the name of the edge leading to its parent and to each of its children. Finally, the root has to store the translation table (containing $|V'|$ items). Thus the overall memory requirement for maintaining $ITR^*(V', r)$ is $\text{Memory}_{\text{total}}(ITR^*(V', r)) = O(|V'| \log n)$.

4.2. *The "Single Pivot" Scheme*

Let us now describe our simplest routing strategy, called the *single pivot* scheme, or SP_k for some $k \geq 1$.

Preprocessing.

1. Fix $m = n^{1/k}$.

2. Select some center c of the graph (as defined in Section 2.1), called the *pivot*.

3. Construct a single routing component $R_c = ITR^*(V, c)$ on the entire graph.

4. For every other vertex $v \in V$ select a neighborhood $N(v, m)$ and construct a component $R_v = ITR^*(N(v, m), v)$.

Delivery. The delivery protocol operates as follows. Suppose a vertex u wishes to send a message to a vertex w. The sender u first consults its translation table TT_u to see if $w \in N(u, m)$. If w does belong to u's neighborhood then the message is sent directly using R_u. Otherwise, the routing is done using the R_c component of the scheme, i.e., u sends the message to c which forwards it to w.

Complexity. We now analyze the memory and stretch complexities of the scheme SP_k.

LEMMA 4.1. *For every $k \geq 1$, the total memory requirement of the routing scheme SP_k satisfies* $\text{Memory}_{\text{total}}(SP_k) = O(n^{1+1/k} \log n)$.

Proof. The component R_c rooted at the pivot $c \in P$ requires $O(n \log n)$ bits. Each other component requires $O(n^{1/k} \log n)$ bits, summing up to $O(n^{1+1/k} \log n)$ bits. □

Let $r_{\min}(G)$ denote the minimum radius of a neighborhood $N(v, m)$ in G, and denote the maximal degree in the graph by Δ.

FACT 4.1. $r_{min}(G) \geq 1 + \log_{\Delta-1}(m/\Delta) \geq \log_\Delta m = \log m / \log \Delta$.

LEMMA 4.2. *For every $k \geq 1$, the stretch factor of the routing scheme SP_k satisfies*

$$\text{STRETCH}(SP_k) \leq \frac{2r(G)}{r_{\min}(G)}.$$

Proof. If the destination w is in the neighborhood of u, then an optimal path is used. Otherwise, $\text{dist}(u, w) \geq r_{\min}(G)$, and the message is sent using the main ITR^* component R_c, and hence along a path of cost at most $2r(G)$. □

COROLLARY 4.1. *For every $k \geq 1$, the stretch factor of the routing scheme SP_k satisfies*

$$\text{STRETCH}(SP_k) \leq \frac{2r(G)\log \Delta}{\log m}.$$

Consider now random graphs G in the space $G_{n,n^{\alpha-1}}$ for $0 < \alpha < 1$, and apply the routing scheme with the parameter $m = n^{1/k}$ to them. (Recall that $G_{n,p}$ is the probability space of all n-vertex graphs, where each edge is chosen independently with probability p.) For such a graph, with high probability the maximal degree and the radius satisfy $\Delta \leq 2n^\alpha$ and $r(G) \leq 1/\alpha$, hence we get:

COROLLARY 4.2. *For fixed k and α and for a random graph G in $G_{n,n^{\alpha-1}}$, the routing scheme SP_k has stretch factor*

$$\text{STRETCH}(SP_k) \leq 2k + 1$$

asymptotically almost surely.

4.3. *The "Covering Pivots" Scheme*

The next step is designing a routing scheme in which there are *several* global *ITR** components in use, each rooted in a different pivot in the graph. By carefully choosing the locations for the pivots it is possible to considerably reduce the stretch. We refer to this scheme as the *covering pivots* scheme, or *CP*.

Preprocessing.

1. Fix $m = \sqrt{n}$.

2. For every vertex $v \in V$ select a neighborhood $N(v, m)$.

3. Using one of the covering algorithms of Section 3.2, select a set $P \subset V$ of *pivots* covering these neighborhoods, i.e., with the property that for every $v \in V$ there is some $u \in P$ such that $u \in N(v, m)$.

4. For every vertex v, compute the pivot that is closest to v according to our definition of distance and denote it by $p(v)$.

5. Construct a component $R_c = ITR^*(V, c)$ around each chosen pivot $c \in P$ and a component $R_v = ITR^*(N(v, m), v)$ for every other vertex $v \in V - P$.

Delivery. The delivery protocol operates as in the single-pivot scheme, except that now, whenever a sender u does not find the destination w in its translation table TT_u, it sends the message through its designated pivot $p(u)$, i.e., using the component $R_{p(u)}$.

Complexity. By Lemmas 3.2 and 3.3, the algorithms of Section 3.2 selects a covering set P of pivots whose size is at most $2cn \ln n/m$ for some constant $c > 1$. Since $m = \sqrt{n}$ we get a total of $O(\sqrt{n} \log n)$ pivots. This implies

LEMMA 4.3. *The total memory requirement of the routing scheme CP satisfies*

$$\text{Memory}_{\text{total}}(CP) = O(n^{3/2} \log^2 n).$$

Proof. The component R_c rooted at a pivot $c \in P$ requires $O(n \log n)$ bits, so all of these components together require $O(n^{3/2} \log^2 n)$ bits. Each other component requires $O(\sqrt{n} \log n)$, summing up to $O(n^{3/2} \log n)$ bits. □

As for the stretch factor of the scheme *CP*, we have the following lemma.

LEMMA 4.4. *The stretch factor of the routing scheme CP satisfies* STRETCH(CP) ≤ 3.

Proof. If the destination w is in the neighborhood of u, then an optimal path is used. Otherwise, $\text{dist}(u,w) \geq r(u,m)$, and the message is sent using the component $R_{p(u)}$, where $p(u)$ is a pivot belonging to $N(u,m)$. The first segment of the route, from u to $p(u)$, is of cost at most $r(u,m)$. The second segment of the route is along some shortest path from $p(u)$ to w. Thus by the triangle inequality it is of cost at most $r(u,m) + \text{dist}(u,w)$. Hence the total cost of the route satisfies

$$|\rho(u,w)| \leq r(u,m) + (r(u,m) + \text{dist}(u,w)) \leq 3\,\text{dist}(u,w). \quad \square$$

Combining Lemma 4.3 and Lemma 4.4 yields

THEOREM 4.1. *For every n-vertex network G it is possible to construct a covering pivots scheme CP with memory requirement*

$$\text{Memory}_{\text{total}}(CP) = O(n^{3/2}\log^2 n)$$

and stretch factor

$$\text{STRETCH}(CP) \leq 3.$$

4.4. *The "Hierarchical Covering Pivots" Scheme*

The CP strategy of Section 4.3 leads to a small stretch factor at the cost of using a total of $O(n^{3/2}\log^2 n)$ bits of memory. In this section we present a family of *hierarchical covering-pivots* schemes, HCP_k, which for every $k \geq 1$, use $O(kn^{1+1/k}\log^2 n)$ bits of memory and guarantee a stretch of at most $2^k - 1$.

Overview. Let us start with an overview of the structures and information items required in the vertices of the network. We base our schemes on a hierarchy of sets of *pivots*,

$$P_{k-1} \subset P_{k-2} \subset \cdots \subset P_1 \subset P_0 = V.$$

For every $0 \leq i \leq k-1$, $|P_i|$ is of size about $n^{1-i/k}$. On each level $0 \leq i \leq k-1$ we maintain a routing component $ITR^*(N(v, n^{(i+1)/k}), v)$ for every pivot $v \in P_i$, enabling routing from v to vertices in its $N(v, n^{(i+1)/k})$ neighborhood. This means that for each level i, every pivot v knows how to get to the $n^{(i+1)/k}$ nearest vertices in the network. Specifically, the pivots of level $1 \leq i \leq k-1$ are selected so as to have the covering property with respect to the neighborhoods $N(v, n^{i/k})$ of pivots $v \in P_{i-1}$. For every $1 \leq i \leq k-1$, we associate with every pivot $v \in P_{i-1}$ a unique pivot $p_i(v)$ such that $p_i(v) \in N(v, n^{i/k})$. Each vertex v has to know an outgoing edge toward its pivot $p_i(v)$.

With the information structure described above maintained in the network, routing is carried out according to the following schematic plan.

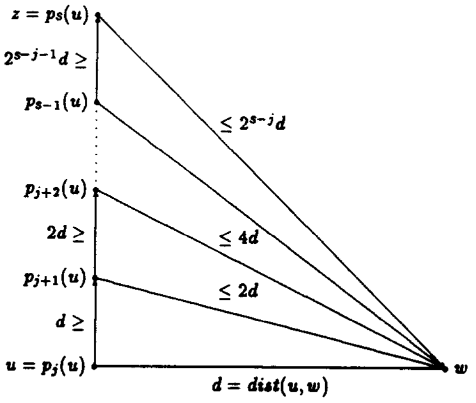

FIG. 1. The schematic description of routing from u to w and the bounds on distance relationships in HCP_k.

Suppose u wants to send a message to w. Then the message is transferred through the chain of pivots associated with u from successively higher levels, until it reaches a pivot $z \in P_s$ knowing the destination w (i.e., such that $w \in N(z, n^{(s+1)/k})$). This pivot sends the message to w directly using its component R_z (see Fig. 1).

Preprocessing. The scheme HCP_k is constructed as follows:

1. Let $P_0 = V$ and $p_0(v) = v$ for every $v \in V$.
2. **For** $i = 1$ **to** $k - 1$ **do**:
   (a) For every $v \in P_{i-1}$ select a neighborhood $N(v, n^{i/k})$ and construct a routing component $R_v = ITR^*(N(v, n^{i/k}), v)$. (If a routing component has already been constructed for v in one of the previous iterations then the new component replaces it.)
   (b) Using one of the covering algorithms of Section 3.2, select a set P_i of pivots covering these neighborhoods, i.e., with the property that for every $v \in P_{i-1}$ there is some $u \in P_i$ such that $u \in N(v, n^{i/k})$.
   (c) For every vertex $v \in P_{i-1}$, compute the pivot of P_i that is closest to v according to our definition of distance and denote it by $p_i(v)$.

   End for
3. **For every** $v \in P_{k-1}$ construct a routing component $R_v = ITR^*(V, v)$ (again, discarding previously constructed components).

Delivery. The delivery protocol proceeds as follows (Fig. 1). Suppose u wants to send a message to w, and suppose $u \in P_j$ for some $0 \leq j \leq k - 1$. If $w \in N(u, n^{(j+1)/k})$ then u sends the message to w directly using the component R_u. Otherwise, u sends the message to its pivot in the next level, $p_{j+1}(u)$. This pivot is now responsible for the message, and deals with it just as if it was originated by itself. Thus, a message originated at u is transferred through the chain of pivots associated with u from successively higher levels, until reaching a pivot $z \in P_s$ knowing the destination w (i.e., such that $w \in N(z, n^{(s+1)/k})$). This pivot sends the message to w directly using its component R_z (see Fig. 1). Since the pivots in the highest level, P_{k-1}, know the entire graph, it is clear that every message eventually arrives at its destination.

Complexity. The memory requirements of the hierarchical schemes are summarized as follows.

LEMMA 4.5. *The total memory requirement of the routing scheme HCP_k satisfies*

$$\text{Memory}_{\text{total}}(HCP_k) = O(kn^{1+1/k}\log^2 n).$$

Proof. Let us first estimate the size of the sets P_i constructed in Step 2(b) of the preprocessing algorithm. Note that the assumptions needed for the randomized algorithm hold here. Thus by the properties of the covering algorithms of Section 3.2 (Lemmas 3.2 or 3.3 as appropriate), there is a constant $c > 1$ such that

$$|P_i| \le \frac{2c|P_{i-1}|\ln|P_{i-1}|}{n^{i/k}} \le \frac{2cn\ln n}{n^{i/k}} = O(n^{1-i/k}\log n).$$

Each component R_v constructed in iteration i for $v \in P_{i-1}$ requires $O(n^{i/k}\log n)$ bits, hence the total amount of memory for components of pivots in P_{i-1} is

$$O(|P_{i-1}|n^{i/k}\log n) = O(n^{1+1/k}\log^2 n).$$

The overall result follows from summing these memory requirements over all k levels of the hierarchy. □

In order to analyze the stretch factor of these schemes we need to establish the following facts.

LEMMA 4.6. *Let $u \in P_{i-1}$. If $z \notin N(u, n^{i/k})$ then*

1. $\text{dist}(u, p_i(u)) \le \text{dist}(u, z)$ *and*
2. $\text{dist}(p_i(u), z) \le 2\,\text{dist}(u, z)$.

Proof. The first claim is immediate from Fact 3.1 and the covering algorithms of Section 3.2. The second claim follows since by the triangle inequality, $\text{dist}(p_i(u), z) \le \text{dist}(p_i(u), u) + \text{dist}(u, z)$. □

LEMMA 4.7. *Suppose the route $\rho(u, w)$ defined by the scheme for u and w goes through the chain of pivots $u = u_j, u_{j+1}, \ldots, u_s$ where $0 \leq j \leq s \leq k - 1$ and $u_i \in P_i$ for $j \leq i \leq s$. Then*

$$|\rho(u,w)| \leq (2^{s-j+1} - 1)\text{dist}(u,w).$$

Proof. First use the previous lemma to prove by induction on i, $j < i \leq s$, that

1. $\text{dist}(u_{i-1}, u_i) \leq 2^{i-j-1}\,\text{dist}(u,w)$ and
2. $\text{dist}(u_i, w) \leq 2^{i-j}\,\text{dist}(u,w)$.

As a result, we get the global distance relationships described in Fig. 1. The lemma now follows, since

$$\begin{aligned}|\rho(u,w)| &\leq \sum_{j+1 \leq i \leq s} 2^{i-j-1}\,\text{dist}(u,w) + 2^{s-j}\,\text{dist}(u,w) \\ &= (2^{s-j+1} - 1)\text{dist}(u,w) \qquad \square\end{aligned}$$

COROLLARY 4.3. *The stretch factor of the routing scheme HCP_k satisfies*

$$\text{STRETCH}(HCP_k) \leq 2^k - 1.$$

Combining Lemma 4.5 and Corollary 4.3 yields

THEOREM 4.2. *For every n-vertex network G and for every $k \geq 1$ it is possible to construct a hierarchical covering-pivots scheme HCP_k with memory requirement*

$$\text{Memory}_{\text{total}}(HCP_k) = O(kn^{1+1/k}\log^2 n)$$

and stretch factor

$$\text{STRETCH}(HCP_k) \leq 2^k - 1.$$

5. MEMORY-BALANCED ROUTING SCHEMES

There are two basic reasons why the hierarchical schemes described in the last section are not memory-balanced. The first is the use of large "communication centers" in the scheme. The root of each *ITR** component has to know of every vertex in its tree, and some trees are very large (in particular, trees on the highest level span the entire network). A second situation in which a vertex may need much space is when it happens to be the "crossroad" of many routes. This happens whenever a vertex occurs in many different *ITR** components (i.e., whenever it resides in the neighborhoods of many pivots).

Thus in order to get rid of these problems we need to devise a way of limiting the use of "communication centers" and bounding the number of occurrences of a vertex in different routing components.

The rest of this section is organized as follows. The first two subsections describe two basic routing components to be used in our schemes. The following subsection describes and analyzes the hierarchical balanced scheme.

5.1. *Decentralized Interval Tree Routing*

The *ITR** scheme as described in Section 4.1 requires the root of the tree to store a translation table specifying the interval label of every vertex in the tree. In a memory-balanced scheme this cannot be permitted. We therefore have to use a modified version of the *ITR** scheme that does not use such tables. Instead, the interval label int(v) of a vertex v has to be part of v's addressing label.

Preprocessing. The modified scheme $ITR(G', r)$ (for a subgraph $G' = (V', E')$ and a vertex $r \in V'$) is constructed just as the scheme $ITR^*(G', r)$, except we do not need the translation table TT_r at the root.

Delivery. Routing a message from a vertex u to a vertex v for $u, v \in V'$ involves sending the message upwards (i.e., towards the root) in the tree until it reaches the lowest common ancestor of u and v, and then downwards on the path leading to v. (Again, this is not necessarily an optimal route in G'.) The header of the message has to contain an identifier for the routing component used (in order to enable us to use this component in a network when other routing components are present as well) and the interval label of the destination. This clearly enables each vertex along the way to decide (using its own label and those of its children in the tree) which way to forward the message.

Complexity. Maintaining the modified component involves the same memory requirements as before, except now the root does not maintain the translation table. Consequently it is possible to bound the individual memory requirements for maintaining $ITR(V', r)$ in a vertex $v \in V'$ by

$$\text{Memory}_{\text{vertex}}(ITR(V', r), v) = O(\deg(v)\log n).$$

The overall memory requirement is still

$$\text{Memory}_{\text{total}}(ITR(V', r)) = O(|V'|\log n).$$

5.2. *The "Forwarding Routing" Scheme*

We now introduce a second type of basic routing component, which we call *forwarding routing* (*FR*). In this scheme, in order for a vertex v to be able to send a message to a vertex w, v has to store in its memory the name of a *forwarding edge* $FE(w)$, which is the first edge along some shortest path from v to w. Clearly, in order for such an approach to make sense it is necessary to ensure the consistency (and sufficiency) of the information stored along shortest paths.

Preprocessing. The partial routing component $FR(S, j)$, for a specific set of destinations $S \subseteq V$ and an integer j, is constructed as follows:

1. For every vertex $v \in V$ select a neighborhood $N(v, j, S)$.

2. For every vertex v construct a table $\{FE(w) | w \in N(v, j, S)\}$ of *forwarding edges* for the vertices in its neighborhood, such that $FE(w)$ is the first edge along some shortest path from v to w.

Delivery. Suppose that a vertex v wants to send a message to a destination w such that $w \in N(v, j, S)$. Then v consults its FE table, retrieves the edge $FE(w) = (v, z)$ and sends the message through this edge to z. The crucial consistency property maintained in this structure is that whenever this happens, w appears also in the FE table of z, so the message does arrive along a shortest path. This is guaranteed by the following lemma.

LEMMA 5.1. *If $w \in N(v, j, S)$ and x occurs on some shortest path connecting v and w then also $w \in N(x, j, S)$.*

Proof. By contradiction. Assume that $w \notin N(x, j, S)$. We claim that for every $z \in N(x, j, S)$, also $z \in N(v, j, S)$. In order to prove this it suffices to show that every $z \in N(x, j, S)$ satisfies $z \prec_v w$, since w is included in $N(v, j, S)$. Consider some $z \in N(x, j, S)$. By the triangle inequality

$$\mathrm{dist}(v, z) \leq \mathrm{dist}(v, x) + \mathrm{dist}(x, z).$$

By Fact 3.1 $\mathrm{dist}(x, z) \leq \mathrm{dist}(x, w)$. Since x is on a shortest path from v to w, $\mathrm{dist}(v, w) = \mathrm{dist}(v, x) + \mathrm{dist}(x, w)$. Put together,

$$\mathrm{dist}(v, z) \leq \mathrm{dist}(v, w).$$

There are two cases to consider. If $\mathrm{dist}(v, z) < \mathrm{dist}(v, w)$ then the claim is immediate. Now suppose $\mathrm{dist}(v, z) = \mathrm{dist}(v, w)$, then necessarily $\mathrm{dist}(x, z) = \mathrm{dist}(x, w)$ too. Since $w \notin N(x, j, S)$ and $z \in N(x, j, S)$, by definition $z \prec_x w$, so $z < w$. Therefore also $z \prec_v w$. It follows from our claim that $N(x, j, S) \subseteq N(v, j, S)$, and since both are of size j, $N(x, j, S) = N(v, j, S)$. But $w \in N(v, j, S) - N(x, j, S)$, a contradiction. □

Complexity. The memory cost of the component $FR(S, j)$ per vertex v is

$$\text{Memory}_{\text{vertex}}(FR(S, j), v) = O(j \log n),$$

and the overall memory requirement is

$$\text{Memory}_{\text{total}}(FR(S, j)) = O(jn \log n).$$

5.3. *The "Hierarchical Balanced" Scheme*

In this section we present the family of *hierarchical balanced* schemes, HB_k, which for every $k \geq 1$, use $O(kn^{1/k} \log n)$ bits of memory per vertex and guarantee a stretch of at most $2 \cdot 3^k - 1$.

Overview. Let us again start with an overview of the method. These schemes are also based on a hierarchy of sets of pivots, $P_k \subset P_{k-1} \subset \cdots \subset P_1 \subset P_0 = V$, such that for every $0 \leq i \leq k$, $|P_i|$ is of size about $n^{1-i/k}$. A vertex v is a *j-pivot* if it belongs to P_j (hence also to P_i for $0 \leq i < j$) but not to P_{j+1} (hence also not to P_i for $j + 1 < i \leq k$). For each level $0 \leq i \leq k$ we maintain a forwarding routing component $FR(P_i, m_i)$ based on the set P_i, where m_i is about $n^{1/k}$ for every i. This means that for each level i, every vertex v knows how to get to the m_i nearest pivots from P_i. Specifically, the pivots of level $1 \leq i \leq k$ are selected so as to have the covering property with respect to the neighborhoods $N(v, m_i, P_{i-1})$ of pivots $v \in P_{i-1}$.

With every vertex v we associate a unique pivot $p_i(v)$, referred to as the *i-post* of v, in every level $0 \leq i \leq k$. This i-post is selected as follows. Suppose that v is a j-pivot. For $0 \leq i \leq j + 1$, $p_i(v)$ is taken to be the smallest pivot in P_i according to $\prec_v$. In particular,

- for $0 \leq i \leq j$, $p_i(v) = v$, and
- $p_{j+1}(v) \in N(v, m_{j+1}, P_j)$.

For $j + 2 \leq i \leq k$ we define $p_i(v)$ recursively by setting $p_i(v) = p_i(p_{i-1}(v))$. Each j-pivot $v \in V$ is given a label consisting of its original name v, its highest level, j, and all its i-posts for $i \geq j + 1$. Thus v's label becomes the tuple $(j, v, p_{j+1}(v), \ldots, p_k(v))$. Finally, a j-pivot $v, j > 0$, has a *zone* $Z_i(v)$ for every $1 \leq i \leq j$, consisting of all $(i - 1)$-pivots u for whom $p_j(u) = v$. It also has an *ITR* component $R_i(v)$ covering its zone $Z_i(v)$.

With the information structure described above maintained in the network, routing is carried out as follows. Suppose u wants to send a message to w. Let i be the minimal index such that the i-post $p_i(w)$ of w is in the neighborhood $N(u, m_i, P_i)$ of u. The originator u sends the message to $p_i(w)$ using the forwarding component $FR(P_i, m_i)$. The message is then forwarded to w through the chain of j-posts $p_j(w)$, $j = i$,

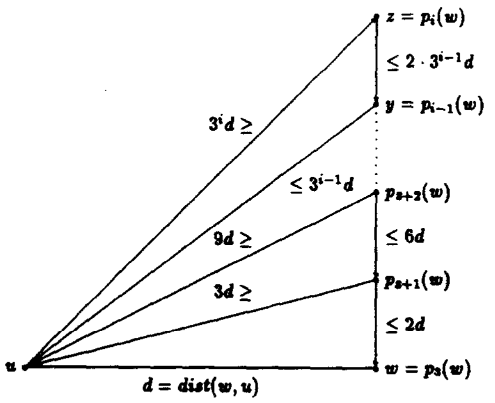

FIG. 2. The schematic description of routing from u to w and the bounds on distance relationships in HB_k.

$i - 1, \dots, s$, where $w = p_s(w)$ is an s-pivot. Each post $p_j(w)$ along the way forwards the message to the next post $p_{j-1}(w)$ on the tree component $R_j(p_j)$. See Fig. 2.

Preprocessing. The scheme HB_k is constructed as follows:

1. Let $P_0 = V$ and $p_0(v) = v$ for every $v \in V$. .
2. Fix $m = n^{1/k}$.
3. If $i < k$ then let $m_i = m$ else let $m_k = |P_k|$.
4. **For** $i = 0$ to $k - 1$ **do**:
   (a) Construct a forwarding scheme $FR(P_i, m_i)$.
   (b) Let $\mathscr{H} = \{N(v, m_i, P_i) | v \in P_i\}$.
   (c) Using one of the covering algorithms of Section 3.2, select a set $P_{i+1} \subseteq P_i$ of pivots covering the sets of $\mathscr{H}$, i.e., with the property that for every $v \in P_i$ there is some $u \in P_{i+1}$ such that $u \in N(v, m_i, P_i)$. The vertices of $P_i - P_{i+1}$ are called i-pivots.
   (d) **For every** $v \in P_i$:
      i. Associate with v an $(i + 1)$-post $p_{i+1}(v)$, chosen as the smallest such pivot $u \in P_{i+1}$ according to $\prec_v$ (in particular, if $v \in P_{i+1}$ then $p_{i+1}(v) = v$).
      ii. Construct a shortest path connecting v with $p_{i+1}(v)$.
   (e) **For every** $v \in P_{i+1}$:
      i. Let the zone of v, $Z_{i+1}(v)$, be the collection of pivots $u \in P_i$ which chose v as their $(i + 1)$-post (i.e., such that $p_{i+1}(u) = v$).
      ii. Denote by $T_{i+1}(v)$ the tree composed of the union of the shortest paths connecting v to the pivots in $Z_{i+1}(v)$.

```
        iii. Construct a component $R_{i+1}(v) = ITR(T_{i+1}(v), v)$.
    (f) Associate with each vertex $v \in V - P_i$ an $i+1$-post $p_{i+1}(v) = p_{i+1}(p_i(v))$.
    End for
5. Construct a forwarding scheme $FR(P_k, m_k)$.
6. In the end of the process, label each $j$-pivot $v$ by $(j, v, p_{j+1}(v), \dots, p_k(v))$.
```

Delivery. The delivery protocol proceeds as follows: Suppose u wants to send a message to w. Denote the j-posts of w by $w_j = p_j(w)$ for every $0 \le j \le k$. Let i be the minimal index such that $w_i \in N(u, m_i, P_i)$ and let s be the maximal index such that $w = w_s$. The originator u starts by sending the message to w_i using the forwarding component $FR(P_i, m_i)$. The message is now forwarded to w through the chain of j-posts w_j, $j = i, i-1, \dots, s$. Consider the first step along this chain. Recall that $p_i(w)$ was chosen to be $p_i(w) = p_i(w_{i-1})$. Therefore $p_i(w_{i-1}) = w_i$ too, so by the definition of zones, w_{i-1} belongs to the zone $Z_i(w_i)$. The message can therefore be forwarded from w_i on the tree component $R_i(w_i)$ to w_{i-1}. The same process repeats itself on $R_{i-1}(w_{i-1})$ and the message gets forwarded to w_{i-2}, and so on, until it arrives $w_s = w$.

Since on the last level the forwarding scheme enables every vertex u to send messages to any pivot in P_k, a message destined to w must eventually reach some i-post of w, $p_i(w)$. Since every pivot $w \in P_i$ has an ITR component covering a shortest path to every P_{i-1} pivot in its zone, and $p_{i-1}(w) \in Z_i(p_i(w))$, it is possible to get from $p_i(w)$ to $p_{i-1}(w)$. By the same argument it is possible to proceed along the chain of j-posts to w. Hence the scheme works correctly for every origin and destination.

Complexity. In order to analyze the memory requirements of the scheme HB_k we need to prove that the trees of the various pivots in a given level P_i are all disjoint. This is done in the following technical lemma.

LEMMA 5.2. *For every $1 \le i \le k$ and for every $u_1, u_2 \in P_i, T_i(u_1)$ and $T_i(u_2)$ are vertex-disjoint.*

Proof. By contradiction. Assume that $T_i(u_1)$ and $T_i(u_2)$ intersect in some vertex x for some $u_1, u_2 \in P_i$. Then for $j = 1, 2$ there is a vertex $w_j \in Z_i(u_j)$ such that x occurs on the shortest path connecting w_i to u_i. Without loss of generality, x is the first intersection of the paths, tracing the paths from the endpoints u_1 and u_2 (see Fig. 3).

By the rules of the algorithm, since $p_i(w_1)$ was set to u_1, necessarily $u_1 \prec_{w_1} u_2$. (Recall that $a \prec_c b$ if either $\mathrm{dist}(a, c) < \mathrm{dist}(b, c)$ or $\mathrm{dist}(a, c) = \mathrm{dist}(b, c)$ and $a < b$.) Since $\mathrm{dist}(u_1, w_1) = \mathrm{dist}(u_1, x) + \mathrm{dist}(x, w_1)$ and $\mathrm{dist}(u_2, w_1) \le \mathrm{dist}(u_2, x) + \mathrm{dist}(x, w_1)$, it follows that $u_1 \prec_x u_2$. Since

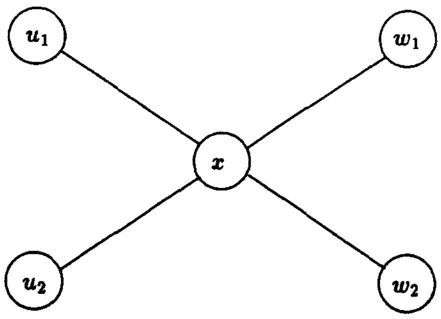

FIG. 3. Node x belonging to both $T_i(u_1)$ and $T_i(u_2)$.

$\text{dist}(u_2, w_2) = \text{dist}(u_2, x) + \text{dist}(x, w_2)$ and $\text{dist}(u_1, w_2) \leq \text{dist}(u_1, x) + \text{dist}(x, w_2)$, it follows that $u_1 \prec_{w_2} u_2$. Hence u_1 should have been chosen as $p_i(w_2)$, a contradiction. □

LEMMA 5.3. *The memory requirements of the routing scheme* HB_k *satisfy*

1. $\text{Memory}_{\text{vertex}}(HB_k, v) = O(k \log n(\deg(v) + n^{1/k}))$, *and*
2. $\text{Memory}_{\text{total}}(HB_k) = O(kn^{1+1/k} \log n)$.

Proof. By Lemma 5.2 each vertex v participates in at most one *ITR* component in each level, so the memory it uses for the trees in all levels is at most $O(k \cdot \deg(v)\log n)$. The total memory used for the *ITR* schemes is $O(kn \log n)$. In addition, v participates in the forwarding scheme of each level. For all levels but the last, this involves $O(m \log n)$ bits of memory. For the last level the cost is $O(|P_k|\log n)$. Noting again that the assumptions of the randomized algorithm hold, by Lemmas 3.2 or 3.3 there is a constant $c > 1$ such that

$$|P_i| \leq \frac{2c|P_{i-1}|\ln|P_{i-1}|}{n^{1/k}}.$$

Since $|P_0| = |V|$, we get that

$$|P_i| \leq (2c \ln n)^i \cdot n^{1-i/k}.$$

Particularly, $|P_k| = O(\ln^k n)$, and since $\log n = o(n^{1/k^2})$ for sufficiently large n, $|P_k| = O(n^{1/k})$. It follows that $O(kn \log n)$ bits are used by each vertex for the forwarding schemes *FR* on all $k + 1$ levels of the hierarchy. The required results follow from adding the memory requirements of the *ITR* and *FR* components. □

It remains to analyze the resulting stretch factor.

LEMMA 5.4. *For $1 \le i \le k-1$, if $p_{i-1}(w) \notin N(u, m_i, P_{i-1})$ then*

1. $\text{dist}(p_i(w), p_{i-1}(w)) \le 2\, \text{dist}(u, p_{i-1}(w))$, *and*
2. $\text{dist}(u, p_i(w)) \le 3\, \text{dist}(u, p_{i-1}(w))$.

Proof. Let $y = p_{i-1}(w)$ and $z = p_i(w)$ (see Fig. 2), and let $P = P_{i-1}$. Since $y \notin N(u, m_i, P)$, by Fact 3.1 $r(u, m_i, P) \le \text{dist}(u, y)$. Since $z \in N(y, m_i, P)$, by Fact 3.1 $r(y, m_i, P) \ge \text{dist}(y, z)$. By Lemma 3.1, $r(y, m_i, P) \le r(u, m_i, P) + \text{dist}(u, y)$. Put together, we get that $\text{dist}(y, z) \le 2\, \text{dist}(u, y)$, which proves the first claim. The second claim follows since by the triangle inequality, $\text{dist}(u, z) \le \text{dist}(u, y) + \text{dist}(y, z)$. □

As a result, we get the global distance relationships depicted in Fig. 2 and summarized in the following corollary:

COROLLARY 5.1. *If $p_j(w) \notin P_j(u, m)$ for every $0 \le j < i$ then*

1. $\text{dist}(p_{i-1}(w), p_i(w)) \le 2 \cdot 3^{i-1} \cdot \text{dist}(u, w)$, *and*
2. $\text{dist}(u, p_i(w)) \le 3^i \cdot \text{dist}(u, w)$.

LEMMA 5.5. *If $p_i(w) \in N(u, m_i, P_i)$ and $p_j(w) \notin N(u, m_j, P_j)$ for every $0 \le j < i$ then the cost of the route satisfies $|\rho(u, w)| \le (2 \cdot 3^i - 1)\text{dist}(u, w)$.*

Proof.

$$|\rho(u,w)| \le \text{dist}(u, p_i(w)) + \sum_{1 \le j \le i} \text{dist}(p_j(w), p_{j-1}(w))$$

$$\le \left(3^i + 2 \sum_{1 \le j \le i} 3^{j-1}\right) \text{dist}(u,w) \le (2 \cdot 3^i - 1)\text{dist}(u,w). \quad \square$$

COROLLARY 5.2. *The stretch factor of the routing scheme HB_k satisfies*

$$\text{STRETCH}(HB_k) \le 2 \cdot 3^k - 1.$$

THEOREM 5.1. *For every n-vertex network G and for every $k \ge 1$ it is possible to construct a hierarchical balanced scheme HB_k with memory requirement*

$$\text{Memory}_{\text{vertex}}(HB_k, v) = O(k \log n(\deg(v) + n^{1/k}))$$

for each vertex v, and

$$\text{Memory}_{\text{total}}(HB_k) = O(kn^{1+1/k} \log n)$$

overall, and stretch factor

$$\text{STRETCH}(HB_k) \le 2 \cdot 3^k - 1.$$

6. Distributed Preprocessing with Bounded Space

6.1. *Overview*

In this section we describe a distributed preprocessing algorithm that initializes the schemes HB_k. Its space overhead, at a single vertex with degree d, is $O(k \log n(d + n^{1/k}))$ bits, the *same* as that of the delivery protocol. The preprocessing stage includes computing all the pivots, constructing the zones and establishing the forwarding schemes *FR* and the *ITR* schemes. The latter requires also assigning appropriate names to the vertices.

We assume a *synchronous* network model in the sense that all vertices have access to a global clock. A message sent upon time τ from v to u arrives at u strictly *after* time $\lceil \tau \rceil + \text{cost}(v, u) - 1$, but no later than $\lceil \tau \rceil + \text{cost}(v, u)$. Intuitively, this means that messages can be sent only at integer times. This is why arrival times depend on $\lceil \tau \rceil$, rather than τ, and edge delays $\delta(e)$ may fluctuate as $\text{cost}(e) - \varepsilon \leq \delta(e) \leq \text{cost}(e)$, for all $\varepsilon < 1$, i.e., the delay of e is *at most* $\text{cost}(e)$.

Since we allow message transmissions at times which are not integers, the algorithm is driven by messages and by the global clock. Messages can be sent either in response to arriving messages, or in response to clock pulses, which occur at integer times.

Our algorithm employs the following four procedures.

(1) *Initialization procedure.* Appoints all the pivots and triggers all other procedures.

(2) *Forwarding-construction procedure.* Constructs the forwarding routing schemes *FR* and computes the parent and children pointers of the *ITR* schemes in the zones.

(3) *DFS-numbering procedure.* Simultaneously operates on the spanning trees of all the zones, assigning DFS numbering to the vertices of the trees for the *ITR* schemes and computing the interval labels.

(4) *Post-update procedure.* Specifies for each vertex its posts.

Each of those procedures has local variables and output variables. The local variables are recognized only inside the procedure, while the output of the procedure is written in the output variables. All the variables are appropriately initialized to *nil*, $\emptyset$, or 0. The values of k, $m = n^{1/k}$, $m_1 = m_2 = \cdots = m_{k-1} = m$, and $c > 1$ are global and known to all the vertices.

Observe that we can construct a spanning tree by invoking the algorithm of [GHS] and then use this tree for computing n, the size of the network, in case it is not globally known. This tree can later be used for other tasks, like counting the number of pivots, detecting termination of a process running in the network, etc. Each of these tasks requires only $O(\log n)$ space per vertex.

6.2. *Initialization Procedure*

The initialization procedure first randomly selects the pivots for each level and then invokes the other procedures. At each vertex, the procedure outputs variables p_i, for $0 \leq i \leq k$, so that $p_i = \mathit{self}$ if the vertex is chosen as a pivot of level i and $p_i = \mathit{nil}$ otherwise.

The pivot selection process is based on the randomized covering algorithm described in Section 3.2. Note that the algorithm is given in a "Monte-Carlo" version, i.e., there is a small probability that the algorithm might err and produce a pivot set that is not a cover. It is easy to modify our algorithm into a "Las-Vegas" algorithm, by detecting the fact that there is no pivot in a certain neighborhood; in this case the algorithm is restarted.

Every vertex is a pivot at level 0. Each pivot at level $i - 1$ becomes a pivot at level i with probability $c \ln|P_{i-1}|/m$. This selection rule guarantees that, with high probability, $|P_i| \leq 2c|P_{i-1}|\ln n/m$. Moreover, with probability $1 - O(n^{1-c})$ there exists a pivot at level i in every neighborhood $N(v, m, P_{i-1})$. To calculate $|P_i|$ we use the function Count(S) that counts the size of an arbitrary subset $S \subseteq V$. This function is easily implemented with $O(\log n)$ space per vertex, given a spanning tree of the network.

After the ith level in the selection process, each vertex triggers Procedure *Forwarding-Construction*$_i$. This procedure consists of a collection of $|P_i|$ interacting diffusing computations [DS], each propagating messages from a pivot $w \in P_i$ to other vertices. It is shown in [SF] that assuming the presence of a spanning tree, termination detection of a collection of diffusing computations can be performed without any space overhead. This procedure is named *Detect-Termination*$_i$. After Procedure *Forwarding-Construction*$_i$ is completed for all the vertices, every vertex triggers the DFS-*Numbering*$_i$ procedure. In the last step of the Initialization procedure the vertices in P_k trigger the *Post-Update* procedure.

Output variables:

- p_i: pivot of the vertex on level i, for $0 \leq i \leq k$.
- m_k: the size of P_k.

Local variables:

- *Coin*: the value of the coin flip; receives the values *head* and *tail*.
- *size*$_i$: the size of $|P_i|$, for $0 \leq i \leq k$.

```
PROCEDURE Initialization /* for a vertex self */.
  $i := 0$; $Coin := head$;
  while $i \leq k$ and $Coin = head$ do;
    $p_i := self$; /* you are a pivot of level $i$ */
    $size_i := \text{Count}(P_i)$;
    if $i = k$ then $m_k := size_k$;
    trigger Procedure Forwarding-Construction$_i$;
    trigger Procedure Detect-Termination$_i$;
    /* wait until all forwarding schemes $FR_i$ of level $i$ are completed */
    trigger Procedure DFS-Numbering$_i$;
    if $i < k$ then flip Coin so that it comes head with probability $c \ln \text{size}_i/m$;
    $i := i + 1$;
  end while /* you are an $(i - 1)$-pivot */
  if $i - 1 = k$ then trigger Procedure Post-Update;
end procedure
```

6.3. *Forwarding-Construction Procedure*

Procedure *Forwarding-Construction*$_i$ has two main tasks. The first is to construct a forwarding routing scheme $FR(P_i, m_i)$ for each level $0 \leq i \leq k$. The goal of this construction is specifying, for every vertex v, a set of pointers pointing towards the m_i pivots of level i that are closest to v according to the $\prec_v$ ordering. That is, for each vertex $w \in N(v, m_i, P_i)$, the vertex v stores a pointer $FE_i(w)$ pointing to a neighbor of v that resides along a shortest path to w. The second task of the procedure is to construct parent and children pointers, *Parent*$_i$ and *Children*$_i$, respectively, for the zones of level i.

The algorithm below takes advantage of the fact that network is synchronous and that message delays are very closely related to edge costs. Thus, the delay of a message sent over a communication path p is "essentially" the cost of that path, cost(p); to be more precise, it varies between cost(p) and cost$(p) - 1$.

Each vertex v keeps a list *List*$_i$ containing pairs (u, w) such that $u \in N(v, m_i, P_i)$ and $w = FE_i(u)$. It also keeps a variable *Parent*$_i$ initialized to *nil*.

Each vertex u that is a pivot of level i initiates a process, propagating FORWARD$_i(u)$ messages throughout the entire network. Once a vertex v receives a message FORWARD$_i(u)$ from w, it checks whether *Parent*$_i$ = *nil*; if so it sets *Parent*$_i := w$. Also, it checks whether *List*$_i$ already contains a pair $(u, \overline{w})$. If no such pair exists then v concludes that this is the first message received from u. Consequently, v adds the pair (u, w) to *List*$_i$, and propagates the message to all neighbors. In fact, v propagates only FORWARD$_i(u)$ messages for vertices $u \in N(v, m_i, P_i)$, and discards messages of vertices outside $N(v, m_i, P_i)$.

Formally, for vertices $u \in N(v, m_i, P_i)$, define $t_i(v, u)$ to be the time at which vertex v receives a FORWARD$_i(u)$ message for the first time. This time is defined as ∞ if such a message never arrives. Under the strategy described above, for all vertices $u \in N(v, m_i, P_i)$, $\lceil t_i(v,u) \rceil = \text{dist}(v,u)$. This fact is proved by induction on the length of a shortest path from v to u, using Lemma 5.1 as an induction step (the base of the induction is trivial).

The strategy described above makes the assumption that upon receipt of a FORWARD$_i(u)$ message, the vertex v can determine whether $u \in N(v, m_i, P_i)$. This is not the case. Below, we describe the modification in the algorithm needed to get around this difficulty.

Denote the radius of $N(v, m_i, P_i)$ by $r = r(N(v, m_i, P_i))$. Let us partition the set P_i into the following four subclasses:

Class A containing all vertices $u \in P_i$ such that $\text{dist}(u,v) < r$.

Class B containing the $m_i - |A|$ vertices with the smallest names among the vertices $u \in P_i$ such that $\text{dist}(u,v) = r$.

Class C containing the vertices $u \in P_i$ such that $\text{dist}(u,v) = r$ but $u \notin B$.

Class D containing vertices $u \in P_i$ such that $\text{dist}(u,v) > r$.

Clearly, $N(v, m_i, P_i)$ is the union of the classes A and B.

The algorithm maintains the invariant that by pulse q, *List*$_i$ contains all pairs (u, w), where $u \in N(v, m_i, P_i)$ and $\text{dist}(u,v) \le q$. Under this invariant, observe that:

- At clock pulse $r - 1$, *List*$_i$ contains exactly the vertices of class A. Thus, $|List_i| = |A| < m_i$ at this pulse.
- Let S be the set of vertices u such that v receives a FORWARD$_i(u)$ message for the first time between clock pulse $r - 1$ and clock pulse r. Then S is the union of the classes B and C. Also, the $m_i - |A|$ vertices of class B are those with the smallest names among the vertices of the set S.
- At clock pulse r, *List*$_i$ contains exactly the vertices of classes A and B. Thus, $|List_i| = |A| + |B| = m_i$ at this pulse.

This enables v to recognize FORWARD$_i(u)$ messages of vertices u in class D, as upon their first arrival $|List_i| = m_i$. Also, v can recognize FORWARD$_i(u)$ messages of vertices u in class A, as upon their first arrival $|List_i| < m_i$. The only problem is to distinguish between vertices in classes B and C. All the FORWARD messages of these vertices arrive (for the first time) between pulses $r - 1$ and r. However, messages of vertices in class B do not necessarily arrive before those of vertices in class C.

In order not to exceed the given size of space during this time interval, we need to dynamically delete vertices from class C as we insert vertices from class B. Towards this goal, each vertex maintains a list New_i, containing "candidates" to be inserted into $List_i$. Upon each clock pulse, the contents of New_i are added to $List_i$, and then New_i is emptied.

In general, whenever a vertex receives a FORWARD$_i(u)$ message from a neighbor w, it checks whether either $List_i$ *or* New_i contains a pair $(u, \bar{w})$, or $|List_i| = m_i$. If either one of these conditions is true, the message is discarded. Else, the vertex inserts the pair (u, w) into New_i list. Next, the vertex checks whether $|New_i| + |List_i| > m_i$. If this is indeed the case, then necessarily $|New_i| + |List_i| = m_i + 1$ as a result of the last insertion into New_i. In this case, the vertex deduces that it is now between pulses r and $r - 1$, and that one of the entries in New_i corresponds to a vertex in class C and hence should be purged. Clearly, this entry is the pair $(\bar{u}, \bar{w})$ corresponding to maximal name $\bar{u}$ among all entries in New_i. This entry is deleted from New_i, thus reducing the total size of $New_i \cup List_i$ back to m_i.

Note that even though the number r is not known to v, it can easily identify clock pulse r at the first clock pulse by which $|New_i| + |List_i| = m_i$.

For a vertex v, let (u, w) be the pair with the smallest u among all pairs in NEW$_i$ after the first clock pulse when NEW$_i$ is not empty. Then w is the parent of v in the zone of level i. At that point, v notifies w of this fact by sending it the message CHILD$_i$.

Data Structure. We assume the existence of a data structure that supports the following operations on dynamic sets S whose members are ordered pairs (a, b). These pairs are stored according to their first item and it is assumed that $a_1 \neq a_2$ for any two different pairs $(a_1, b_1), (a_2, b_2)$ in S.

- Member(S, a): returns *true* iff there exists b such that $(a, b) \in S$.
- Insert($S, (a, b)$): adds (a, b) to S.
- Delete(S, a): deletes the pair (a, b) from S, if such a pair exists.
- ExtractMax(S): returns the pair $(a, b) \in S$ with the largest a among all pairs of S.
- DeleteMax(S): abbreviation for Delete(S, ExtractMax(S)).

• ExtractMin(S): returns the pair $(a, c) \in S$ with the smallest a among all pairs of S.

• size(S): returns the number of pairs in S.

Output variables:

• $FE_i(w)$: the forwarding edge to w, kept for each w in $List_i$, for $0 \le i \le k$.

• $Parent_i$: the parent of v in the zone of level i, for $0 \le i \le k$.

• $Children_i$: the set of children of v in the zone of level i, for $0 \le i \le k$.

Local variables:

• $List_i$: A dynamic set containing the list of edges $(w, FE_i(w))$ for each $w \in N(v, m_i, P_i)$, for $0 \le i \le k$.

• New_i: A dynamic set containing a list of pairs that are candidates to enter $List_i$, for $0 \le i \le k$.

```
PROCEDURE Forwarding-Construction_i /* for a vertex v */.
upon triggering the procedure (for $v \in P_i$):
    send message FORWARD_i($v$) to all neighbors $w$;
    insert($New_i$, $(v, v)$);
upon receiving FORWARD_i($u$) message from neighbor $w$:
    if Member($List_i$, $u$) = Member($New_i$, $u$) = false and size($List_i$) < $m_i$ then do:
        insert($New_i$, $(u, w)$); /* tentative insertion */
        if size($List_i$) + size($New_i$) > $m_i$ then DeleteMax($New_i$); /* handle overflow */
    end if
upon Clock pulse:
    if size($List_i$) = 0 and size($New_i$) > 0 then do:
        $(u, w)$ := ExtractMin($New_i$);
        $Parent_i$ := $w$; /* choosing the parent in the zone of $v$ */
        send message CHILD_i to neighbor $w$;
    end if
    for all $(u, w) \in New_i$ do:
        $FE_i(u)$ := $w$;
        send message FORWARD_i($u$) to all neighbors;
        Insert($List_i$, $(u, w)$);
        Delete($New_i$, $u$);
    end for
upon receiving CHILD_i message from neighbor $w$: add $w$ to $Children_i$;
end procedure
```

6.4. *DFS-Numbering and Zone-Construction*

The procedure DFS-$Numbering_i$ described in this section sets up the zones of level i (for each $0 \leq i \leq k$) and the ITR_i schemes in each zone. Recall that the zones have been determined by the $Parent_i$ and the $Children_i$ pointers computed by Procedure $Forward\text{-}Construction_i$ of the previous subsection.

For each pivot q of level i, we now consider the tree $T_i(q)$, which defines the zone $Z_i(q)$ of level i around q. Recall that by Lemma 5.2 in every level all the zones are disjoint.

We assign a DFS (pre-order) number to each vertex $v \in T_i(q)$, and maintain counters $Lowest_i$ and $Highest_i$ as the lowest and highest pre-order DFS numbers in the sub-tree rooted at v. The interval int$(v) = (Lowest_i, Highest_i)$ contains the preorder numbers of all vertices in v's sub-tree of level i. (Recall that $Lowest_i$ is the DFS number of v itself.) Also, for each child u of v, v maintains the parameters $(Lowest_i(u), Highest_i(u))$, where $Highest_i(u)$ equals $Highest_i$ of the vertex u, and $Lowest_i(u)$ equals $Lowest_i$ of u.

The DFS procedure proceeds by forwarding a token that represents the "center of activity" along network edges. Transmission of a token is performed by sending a VISIT(p) message, where p is the highest DFS number in the sub-tree rooted at the vertex which sends the token. Returning a token is performed by sending a RETREAT(q) message, where q is the highest DFS number in the sub-tree rooted at the vertex which returns the token.

Output variables:

• $Lowest_i$: DFS (pre-order) number of v in the DFS of level i, for $0 \leq i \leq k$.

• $Lowest_i(w)$: the $Lowest_i$ of a neighbor w, for $0 \leq i \leq k$.

• $Highest_i$: the maximal DFS number assigned to a vertex in the sub-tree rooted at v, for $0 \leq i \leq k$.

• $Highest_i(w)$: the $Highest_i$ of a neighbor w, for $0 \leq i \leq k$.

Local variables:

• $Unvisited_i$: the set of unvisited children in level i, for $0 \leq i \leq k$. Initially, $Unvisited_i = Children_i$.

```
PROCEDURE DFS$_i$ /* subprocedure of procedure DFS-$Numbering_i$ */.
    if $Unvisited_i \neq \emptyset$ then do:
        pick $u \in Unvisited_i$; delete $u$ from $Unvisited_i$;
        $Lowest_i(u) := Highest_i + 1$;
        send VISIT$_i$($Highest_i$) to $u$;
        else if $Parent_i \neq nil$ then send RETREAT$_i$($Highest_i$) to $Parent_i$;
        end if
    end procedure

    PROCEDURE DFS-$Numbering_i$ (for a vertex $self$).
    upon triggering the procedure (at the root):
        $Lowest_i := 1$;
        $Highest_i := 1$;
        trigger Procedure DFS$_i$;
    upon receiving VISIT$_i$($p$) from $u$:
        $Lowest_i := p + 1$;
        $Highest_i := p + 1$;
        trigger Procedure DFS$_i$;
    upon receiving RETREAT$_i$($p$) from $u$:
        $Highest_i(u) := p$;
        $Highest_i := p$;
        trigger Procedure DFS$_i$;
    end procedure
```

6.5. *Post-Update Procedure*

The goal of Procedure *Post-Update* is to compute the posts of every vertex. At the outset, all the vertices in P_k know their post; it is simply their original name. Now, whenever an i-pivot (i.e., a vertex which is a pivot of level i but not of level $i + 1$) learns about its posts for levels $j > i$, it has to pass this information on to all the l-level pivots in its zone Z_l, for all $1 \leq l \leq i$. For that purpose, it broadcasts its posts over its *ITR* tree of level l. Then the l-level pivots in that zone broadcast their new post over their zones, etc., until all the vertices in the network know their posts.

In the code below, the message POST$(j, q_j, \ldots, q_k)$ is interpreted as saying that q_j is a j-pivot, and its i-post at level i is q_i, for $j \leq i \leq k$.

Output variables:

- *posts*: the new posts of a vertex v; it is a sequence $(j, v, q_{j+1}, \ldots, q_k)$, where v is a j-pivot.

```
    PROCEDURE Post-Update (for a vertex $self$).
    upon triggering the procedure: (for $v \in P_k$)
        $posts := (k, self)$;
        for each vertex $z$ in $Children_k$
            send message POST$(k, self)$ to $z$;
    upon receiving POST$(j, q_j, \ldots, q_k)$ from $w = Parent_i$:
        for each vertex $z$ in $Children_i$
            send message POST$(j, q_j, \ldots, q_k)$ to $z$;
        if $p_{j-1} = self$ then
        set $posts := (j - 1, self, q_j, \dots, q_k)$;
        if $j - 1 > 0$ then
          for $l := 1$ to $j - 1$ do:
            for each vertex $z$ in $Children_l$
              send message POST($l, self, \dots, self, q_j, \dots, q_k$) to $z$,
              where $self$ appears $j - l$ times;
          end for
      end if
  end procedure
```

6.6. *Complexity of the Algorithm*

It takes $dk \log n$ bits at a vertex with degree d to maintain p_i, $Parent_i$, $Children_i$, and $posts$. It takes $km \log n = kn^{1/k} \log n$ bits to maintain each of the lists $List_i$ and New_i. The memory requirements at a single vertex are thus upper-bounded by $O(k \log n(d + n^{1/k}))$ bits, summing up to $O(kn^{1+1/k} \log n)$ bits over the entire network.

References

[C] H. Chernoff, A measure of asymptotic efficiency for tests of hypothesis based on the sum of observations, *Ann. Math. Statist.* **23** (1952), 493–507.

[DS] E. W. Dijkstra and C. S. Scholten, Termination detection for diffusing computations, *Inform. Process. Lett.* **11** (1980), 1–4.

[FJ1] G. N. Frederickson and R. Janardan, Designing networks with compact routing tables, *Algorithmica* **3** (1988), 171–190.

[FJ2] G. N. Frederickson and R. Janardan, Separator-based strategies for efficient message routing, *in* "Proceedings, 27th Symp. on Foundations of Computer Science, 1986," pp. 428–437.

[GHS] R. G. Gallagher, P. A. Humblet, and P. M. Spira, A distributed algorithm for minimum-weight spanning trees, *ACM Trans. Programming Lang. Systems* **5** (1983), 66–77.

[KK1] L. Kleinrock and F. Kamoun, Hierarchical routing for large networks; Performance evaluation and optimization, *Comput. Networks* **1** (1977), 155–174.

[KK2] L. Kleinrock and F. Kamoun, Optimal clustering structures for hierarchical topological design of large computer networks, *Networks* **10** (1980), 221–248.

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

[PU] D. Peleg and E. Upfal, A tradeoff between size and efficiency for routine tables, *JACM* **36** (1989), 510–530.

[P] R. Perlman, Hierarchical networks and the subnetwork partition problem, *in* "Proceedings, 5th Conf. on System Sciences, 1982."

[SF] N. Shavit and N. Francez, A new approach to detection of locally indicative stability, *in* "Proceedings, 13th ICALP, 1986," pp. 344–358.

[SK] N. Santoro and R. Khatib, Labeling and implicit routing in networks, *Comput. J.* **28** (1985), 5–8.

[S] C. A. Sunshine, Addressing problems in multi-network systems, *in* "Proceedings, IEEE INFOCOM, 1982."

[T] A. S. Tanenbaum, "Computer Networks," Prentice-Hall, Englewood Cliffs, NJ, 1981.

[vLT1] J. van Leeuwen and R. B. Tan, Routing with compact routing tables, *in* "The Book of L" (G. Rozenberg and A. Salomaa, Eds.), pp. 259–273, Springer-Verlag, New York, 1986.

[vLT2] J. van Leeuwen and R. B. Tan, Interval routing, *Comput. J.* **30** (1987), 298–307.

[Z] H. Zimmermann, OSI reference model—The ISO model of architecture for open systems interconnection, *IEEE Trans. Commun.* **28** (1980), 425–432.