Routing with Polynomial Communication-Space Tradeoff

Baruch Awerbuch * David Peleg †

April 19, 1993

Abstract

This paper presents a family of memory-balanced routing schemes that use relatively short paths while storing relatively little routing information. The *hierarchical schemes* $\mathcal{H}_k$ (for every integer $k \geq 1$) guarantee a stretch factor of $O(k^2)$ on the length of the routes and require storing at most $O(kn^{\frac{1}{k}} \log n \log D)$ bits of routing information per vertex in an n-processor network with diameter D. The schemes are name-independent and applicable to general networks with arbitrary edge weights. This improves on previous designs whose stretch bound was exponential in k.

Key words:

Communication networks, routing tables, communication-space trade-offs, graph covers.

*Dept. of Mathematics and Lab. for Computer Science, M.I.T., Cambridge, MA 02139; ARPANET: baruch@theory.lcs.mit.edu. Supported by Air Force Contract TNDGAFOSR-86-0078, ARO contract DAAL03-86-K-0171, NSF contract CCR8611442, DARPA contract N00014-89-J-1988, and a special grant from IBM.

†Department of Applied Mathematics, The Weizmann Institute, Rehovot 76100, Israel. Supported in part by an Allon Fellowship and by a Bantrell Career Development Fellowship. Part of the work was done while visiting IBM T.J. Watson Research Center.

* * * * * * This paper has appeared in SIAM Journal on Discrete Math., Vol.5, pp.151—162, May 1992. Preliminary version appeared in Proc. of 31'st IEEE Symp. on Foundations of Computer Science, October 1990, St. Louis * * * * * *

1 Introduction

Delivering messages between pairs of processors is a primary activity of any distributed communication network. This task is performed using a routing scheme, which is a mechanism for routing messages in the network. It can be invoked at any origin vertex and be required to deliver a message to some destination vertex.

Naturally, it is desirable to route messages along short paths. The straightforward approach of storing a complete routing table in each vertex v in the network (specifying for each destination u the first edge along some shortest path from v to u) guarantees optimal routes, but is too expensive for large systems since it requires a total of $O(n^2 \log n)$ memory bits in an n-processor network. Thus, one basic goal in large scale communication networks is the design of routing schemes that produce efficient routes and have relatively low memory requirements. The efficiency of a routing scheme is measured in terms of its *stretch*, namely, the maximum ratio between the length of a route produces by the scheme for some pair of processors and their distance.

The problem of efficiency-memory tradeoffs for routing schemes was first raised in [KK1]. The solution given there and in several consequent papers [BJ, KK2, Pr, S, FJ1, FJ2, SK, vLT1, vLT2] applies only under some special assumptions or for special classes of network topologies. In [PU] the problem is dealt with for general networks. The paper presents a family of hierarchical routing schemes (for every integer $k \geq 1$) which guarantee stretch $O(k)$ and require storing a total of $O(k^3 n^{1+1/k} \log n)$ bits of routing information in the network. This behavior is almost optimal, as implied from a lower bound given in [PU] on the space requirement of any scheme with a given stretch.

While the method of [PU] is appropriate for designing a static routing scheme, it is not suitable for situations in which the routing scheme needs to be updated from time to time. Such updates may be needed when the topology of the network changes (due to failures or recoveries or other reasons). Each such update involves recomputing the edge costs in the network and deciding on the new routes accordingly. (In this paper we ignore the issue of determining the link costs and concentrate on the step of setting up a routing tables.)

In such situations it is necessary that the routing scheme be able to handle arbitrary edge costs (as well as arbitrary network topologies.) Another crucial requirement is *name-independence*. By this we mean that the addresses used to describe the destination of messages should be permanently fixed and independent of the routing scheme. This precludes routing strategies in which the route design involves also selecting an addressing label for each vertex (typically encrypting partial information necessary for determining the routes leading to it). This approach clearly violates the principle that routing-related system activities need be transparent to the user. Finally, in the non-static case, a bound on the total (or average) memory requirements is insufficient. This is because in such setting, vertices may play different roles and thus require varying amounts of space. For instance, a vertex may be designated as a "communication center" or just happen to be the crossing point of many routes in a particular scheme. This forces every vertex to have sufficient memory for performing the most demanding role it may ever need to perform. It is thus necessary to guarantee a bound on the worst-case memory requirements of each vertex. (See [ABLP] for a more detailed discussion of these issues.)

Unfortunately, the routing strategy of [PU] lacks all of these three properties. It deals only with unit-cost edges, it is name-dependent, and there is no bound on the worst-case memory requirements of each vertex. Consequently, these problems were tackled in [ABLP, P2]. The schemes proposed in these two papers succeed in achieving these desirable properties but at the cost of an inferior efficiency-space tradeoff. In an n-processor network of diameter D, the schemes HS_k of [ABLP], for $k \geq 1$, use $O(k \cdot \log n \cdot n^{\frac{1}{k}})$ bits of memory per vertex and guarantee a stretch of $O(k^2 \cdot 9^k)$, while the schemes R_k of [P2], for $k \geq 1$, use $O(\log D \cdot \log n \cdot n^{\frac{1}{k}})$ bits of memory per vertex and guarantee a stretch of $O(4^k)$. Thus the stretch becomes exponential in k, in contrast with the polynomial dependency achieved in [PU].

The schemes of [P2] are based on solving a certain graph problem (handled previously in [P1]) involving the construction of sparse covers. The schemes presented in the current paper are basically the same as those of [P2], except that they employ a new, more efficient solution to the underlying cover problem, and thus improves the tradeoff. Thus for every graph G and every integer $k \geq 1$ we construct a hierarchical routing scheme $\mathcal{H}_k$ with stretch $O(k^2)$ using $O(kn^{1/k} \log n \log D))$ memory bits per vertex. Thus the new scheme regains the polynomial dependency of the stretch factor on k.

Let us comment that the schemes of [ABLP] still have two advantages over the new schemes. First, their space complexity is purely combinatorial, i.e., it does not depend on the edge weights. This may be significant if we allow super-polynomial edge weights. Secondly, that scheme has an efficient distributed preprocessing algorithm for setting the routes, while the preprocessing stage of the current schemes seems to be inherently sequential.

The new cover algorithm described herein seems to have several other useful applications in the area of distributed network algorithms. For some of these applications see [AKP, AP1, AP2].

The rest of the paper is organized as follows. Section 2 formally defines the problem. Section 3 presents the hierarchical routing scheme. Finally, Section 4 gives the new solution to the sparse cover problem.

2 Definition of the problem

We consider the standard model of a point-to-point communication network, described by an undirected graph $G = (V, E)$, $V = \{1, \ldots, n\}$. The vertices V represent the processors of the network and the edges E represent bidirectional communication channels between the vertices. A vertex may communicate directly only with its neighbors, and messages between non-adjacent vertices are sent along some path connecting them in the network.

We assume the existence of a *weight* function $w : E \to \mathcal{R}^+$, assigning an arbitrary positive weight $w(e)$ to each edge $e \in E$. Also, there exists a *name* function $name : V \to U$, which assigns to each vertex $v \in V$, an arbitrary name $name(v)$ from some universe U of names. We sometime abuse notation, referring to $name(v)$ simply by v.

For two vertices u, w in a graph G let $dist_G(u, w)$ denote the (weighted) length of a shortest path in G between those vertices, i.e., the cost of the cheapest path connecting them, where the cost of a path $(e_1, \ldots, e_s)$ is $\sum_{1 \le i \le s} w(e_i)$. (We sometimes omit the subscript G where no confusion arises.)

A *routing scheme* RS for the network G is a mechanism for delivering messages in the network. It can be invoked at any origin vertex u and be required to deliver a message M to some destination vertex v (which is specified by its *name*) via a sequence of message transmissions. An elementary message is allowed to contain $O(\log n)$ bits.

We now give precise definitions for our complexity measures for stretch and memory. The *communication cost* of transmitting a message over edge e is the weight $w(e)$ of that edge. The communication cost of a *protocol* is the sum of the communication costs of all message transmissions performed during the execution of the protocol. Let $C(RS, u, v)$ denote the communication cost of the routing scheme when invoked at an origin u, w.r.t. a destination v and an elementary ($O(\log n)$ bit) message, i.e., the total communication cost of all message transmissions associated with the delivery of the message. Given a routing scheme RS for an n-processor network $G = (V, E)$, we say that RS *stretches* the path from u to v by $\frac{C(RS,u,v)}{dist(u,v)}$. We define the *stretch factor* of the scheme RS to be

$$Stretch(RS) \;=\; \max_{u,v \in V} \left\{ \frac{C(RS, u, v)}{dist(u, v)} \right\}.$$

The *memory requirement* of a protocol is the maximum amount of memory bits used by the protocol in any single processor in the network. We denote the memory requirement of a routing scheme RS by $Memory(RS)$.

Next let us define some basic graph notation. The *j-neighborhood* of a vertex $v \in V$ is defined as

$$N_j(v) = \{w \mid dist(w,v) \leq j\}.$$

Given a subset of vertices $R \subseteq V$, denote $\mathcal{N}_m(R) = \{N_m(v) \mid v \in R\}$. Let $D = Diam(G)$ denote the *diameter* of the network, i.e., $\max_{u,v\in V}(dist(u,v))$. For a vertex $v \in V$, let

$$Rad(v,G) = \max_{w\in V}(dist_G(v,w)).$$

Let $Rad(G)$ denote the *radius* of the network, i.e., $\min_{v\in V}(Rad(v,G))$. A *center* of G is any vertex v realizing the radius of G (i.e., such that $Rad(v,G) = Rad(G)$). In order to simplify some of the following definitions we avoid problems arising from 0-diameter or 0-radius graphs, by defining $Rad(G) = Diam(G) = 1$ for the single-vertex graph $G = (\{v\}, \emptyset)$. Observe that for every graph G, $Rad(G) \leq Diam(G) \leq 2Rad(G)$. (Again, in all of the above notations we may sometimes omit the reference to G where no confusion arises.)

Finally, let us introduce some definitions concerning covers. Given a set of vertices $S \subseteq V$, let $G(S)$ denote the subgraph induced by S in G. A *cluster* is a subset of vertices $S \subseteq V$ such that $G(S)$ is connected. We use $Rad(v,S)$ (respectively, $Rad(S)$, $Diam(S)$) as a shorthand for $Rad(v,G(S))$ (resp., $Rad(G(S))$, $Diam(G(S))$). A *cover* is a collection of clusters $\mathcal{S} = \{S_1,\ldots,S_m\}$ such that $\bigcup_i S_i = V$. Given a collection of clusters $\mathcal{S}$, let $Diam(\mathcal{S}) = \max_i Diam(S_i)$ and $Rad(\mathcal{S}) = \max_i Rad(S_i)$. For every vertex $v \in V$, let $deg_{\mathcal{S}}(v)$ denote the degree of v in the hypergraph $(V,\mathcal{S})$, i.e., the number of occurrences of v in clusters $S \in \mathcal{S}$. The *maximum degree* of a cover $\mathcal{S}$ is defined as $\Delta(\mathcal{S}) = \max_{v\in V} deg_{\mathcal{S}}(v)$.

Given two covers $\mathcal{S} = \{S_1,\ldots,S_m\}$ and $\mathcal{T} = \{T_1,\ldots,T_k\}$, we say that $\mathcal{T}$ *subsumes* $\mathcal{S}$ if for every $S_i \in \mathcal{S}$ there exists a $T_j \in \mathcal{T}$ such that $S_i \subseteq T_j$.

3 The routing scheme

In this section we describe our routing strategy and the structures it uses in the network. Our approach is based on constructing a *hierarchy* of covers in the network, and using this hierarchy for routing. In each level, the graph is covered by clusters (namely, connected subgraphs), each managed by a center vertex. Each cluster has its own internal routing mechanism (described in the following subsection), enabling routing to and from the center. Messages are always transferred to their destinations using the internal routing mechanism of some cluster, along a route going through the cluster center. It is clear that this approach reduces the memory requirements of the routing schemes, since one has to define routing paths only for cluster centers, but it increases the communication cost, since messages need not be moving along shortest paths. Through an appropriate choice of the cluster cover we guarantee that both overheads are low.

3.1 Tree routing

In this subsection we discuss the basic routing component used inside clusters. This component is based on a shortest path tree T rooted at a vertex r and spanning the cluster. Routing messages to the root is a straightforward task. We need a mechanism enabling us to route a message from the root, where the destination is not necessarily in the tree (in which case the message is to be returned to the root with an appropriate notification).

This subproblem was treated in previous papers using a simple scheme called the *interval routing scheme* or ITR [SK,PU]. This scheme is based on assigning the vertices $v \in T$ a DFS numbering $DFS(v)$ and storing at a vertex u its DFS number and the DFS numbers $DFS(w)$ of each of its children w in the tree. Then routing a message from the root r to a vertex v (assuming that r knows the value $DFS(v)$) involves propagating the message from each intermediate vertex u with children $w_1, \ldots, w_k$ (ordered by increasing DFS numbering) to the child w_i such that $DFS(w_i) \leq DFS(v) < DFS(w_{i+1})$ (setting $DFS(w_{k+1})$ to ∞).

Using the ITR scheme for our purposes poses some new technical problems. First, the scheme requires using the DFS numbers as routing labels, which interferes with the name-independence requirement. Secondly, the memory stored at a vertex depends linearly on its degree, and thus may be as high as $\Omega(n)$ in the worst case. This interferes with the balanced-memory requirement.

The two problems are solved in [ABLP] as follows. In order to avoid the need to know the DFS labels in advance by the origins, the scheme uses a distributed data structure enabling the root to retrieve the DFS label of a vertex using its original name as a key. The second problem is handled by proving that one can embed into any tree a tree of "small" degrees, without paying too high a price in memory and without increasing the depth of the tree "too much", where the degrees and depth are controlled by some parameter k.

In fact, the scheme as described in [ABLP] is aimed at solving a somewhat harder task then ours. It therefore uses a special type of stratified tree structure, which increases the resulting complexity. This feature is not needed here. Consequently, we may use a simplified version of the mechanism of [ABLP]. We do not describe the mechanism in detail, but rather state the complexity of the resulting tree routing component. Define the depth of the tree

T, $depth(T)$, as the maximum distance of any vertex in T from the root. The memory requirements of the tree routing scheme for T are $O(n^{1/k} \log n)$ bits per vertex. The length of the resulting paths is up to $4k \cdot depth(T)$ for any $u \in V$, whether or not $u \in T$. (In the scheme as presented in [ABLP] both complexities are higher by a factor of k due to the stratified structure.)

3.2 Regional routing schemes

Each level in our hierarchy constitutes a *regional* (C, m)*-routing scheme*, which is a scheme with the following properties. For every two processors u, v, if $dist(u, v) \leq m$ then the scheme succeeds in delivering messages from u to v. Otherwise, the routing might end in failure, in which case the message is returned to u. In either case, the communication cost of the entire process is at most C.

In this subsection we describe how to construct a regional $(O(k^2 m), m)$-routing scheme, for any integers $k, m \geq 1$. We rely on the following Theorem, to be proved in Section 4.

Theorem 3.1 Given a graph $G = (V, E)$, $|V| = n$, a cover $\mathcal{S}$ and an integer $k \geq 1$, it is possible to construct a cover $\mathcal{T}$ that satisfies the following properties:

(1) $\mathcal{T}$ subsumes $\mathcal{S}$,

(2) $Rad(\mathcal{T}) \leq (2k-1) Rad(\mathcal{S})$, and

(3) $\Delta(\mathcal{T}) = O(k|\mathcal{S}|^{1/k})$.

The routing scheme is constructed as follows. We start by setting $\mathcal{S} = \mathcal{N}_m(V)$ and constructing a cover $\mathcal{T}$ as in the Theorem. Next, we provide internal routing services in each cluster T by selecting a center $\ell(T)$ and constructing a tree routing component for T rooted at this center. By Property (1) of the Theorem, the cover $\mathcal{T}$ subsumes $\mathcal{N}_m(V)$, that is, for every vertex $v \in V$ there is a cluster $T \in \mathcal{T}$ such that $N_m(v) \subseteq T$. Consequently, we associate with every vertex $v \in V$ a *home cluster*, $home(v) \in \mathcal{T}$, which is the cluster containing $N_m(v)$. (In case there are several appropriate clusters, select one arbitrarily.) A processor v routes a message by sending it to its home cluster leader, $\ell(home(v))$. The leader uses the tree routing mechanism to forward the message to its destination. If that destination is not found in the cluster, the message is returned to the root and from there to the originator.

The correctness and complexity of the constructed regional scheme are dealt with in the following two lemmas, relying on the properties of the tree routing mechanism and the constructed cover.

Lemma 3.2 The mechanism described above is a regional $(O(k^2 m), m)$-routing scheme.

Proof: Suppose that $dist(u,v) \leq m$ for some processors u, v. By definition, $v \in N_m(u)$. Let T be the home cluster of u. Then $N_m(u) \subseteq T$, so $v \in T$. Hence the tree routing on T will succeed in passing the message from u to v. Furthermore, as discussed in the previous subsection, the routing proceeds along a path of length at most $8k \cdot depth(T) = 8k \cdot Rad(\ell(T), T)$. By Property (2) of Theorem 3.1, $8k \cdot Rad(\ell(T), T) \leq 8k(4k+1)m$. ▮

Lemma 3.3 For every graph G and integers $m, k \geq 1$, the regional $(O(k^2 m), m)$-routing scheme described above can be implemented using $O(n^{2/k} \log^2 n)$ memory bits per vertex.

Proof: Implementing the routing scheme involves the following space requirements. Each vertex v needs to store up to $O(n^{1/k} \log n)$ bits for every $\mathcal{T}$ cluster to whom it belongs, (i.e., for $deg_{\mathcal{T}}(v)$ clusters). By Property (3) of Theorem 3.1 this degree is $O(n^{1/k} \log n)$, since $|\mathcal{N}_m(V)| = n$. This gives a total of $O(n^{2/k} \log^2 n)$ bits per vertex. ▮

3.3 The hierarchical routing scheme

Finally we present our family of *hierarchical routing schemes*. For every fixed integer $k \geq 1$, construct the hierarchical scheme $\mathcal{H}_k$ as follows. Let $\delta = \lceil \log Diam(G) \rceil$. For $1 \leq i \leq \delta$ construct a regional $(O(k^2 m_i), m_i)$-routing scheme R_i where $m_i = 2^i$. Each processor v participates in all δ regional routing schemes R_i. In particular, v has a home cluster $home_i(v)$ in each R_i, and it stores all the information it is required to store for each of these schemes.

The routing procedure operates as follows. Suppose a vertex u wishes to send a message to a vertex v. Then u first tries using the lowest-level regional scheme R_1, i.e., it forwards the message to its first home cluster leader, $\ell(home_1(v))$. If this trial fails, u retries sending its message, this time using regional scheme R_2, and so on, until it finally succeeds.

Lemma 3.4 The hierarchical routing scheme $\mathcal{H}_k$ has $Stretch(\mathcal{H}_k) = O(k^2)$.

Proof: Suppose that a processor u sends a message to some other processor v. Let $d = dist(u,v)$ and $j = \lceil \log d \rceil$ (i.e., $2^{j-1} < d \leq 2^j$). The sender u successively tries forwarding the message using the regional schemes R_1, R_2 etc., until the message finally succeeds in arriving v. By Lemma 3.2, the regional scheme R_j will necessarily succeed, if no previous level did. (Note that the highest-level scheme, R_δ, has $m_\delta = 2^\delta \geq Diam(G) \geq d$, and therefore will always succeed.)

Denote the total length of the combined path traversed by the message by ρ. By Lemma 3.2), ρ satisfies (for some constant $c > 0$)

$$\rho \;\leq\; \sum_{i=1}^{j} ck^2 2^i \;\leq\; ck^2 2^{j+1} \;\leq\; O(k^2) dist(u,v).$$

▮

Theorem 3.5 For every graph G and every fixed integer $k \geq 1$ it is possible to construct (in polynomial time) a hierarchical routing scheme $\mathcal{H}_k$ with $Stretch(\mathcal{H}_k) = O(k^2)$ using $Memory(\mathcal{H}_k) = O(n^{1/k} \log^2 n \log Diam(G))$ bits per vertex.

Proof: Construct the $\delta = \lceil \log Diam(G) \rceil$ regional schemes R_i as in the previous subsection. By Lemma 3.3, the total memory requirements are

$$\sum_{i=1}^{\delta} O\left(n^{1/k} \log^2 n\right) \;=\; O(n^{1/k} \log^2 n \log Diam(G)).$$

▮

4 Constructing a sparse cover

4.1 The construction of the cover

In this subsection we present the algorithm `Main`, whose task is to construct a cover as in Theorem 3.1. The construction revolves on covering the clusters of $\mathcal{S}$. The input to the algorithm is a graph $G = (V, E)$, $|V| = n$, a cover $\mathcal{S}$ and an integer $k \geq 1$. The output collection of cover clusters, $\mathcal{T}$, is initially empty. The algorithm maintains the set of "remaining" clusters $\mathcal{R}$. These are the clusters not yet subsumed by the constructed cover. Initially $\mathcal{R} = \mathcal{S}$, and the algorithm terminates once $\mathcal{R} = \emptyset$. The algorithm operates in at most $k|\mathcal{S}|^{1/k}$ phases. Each phase consists of the activation of the procedure `Cover`($\mathcal{R}$), which adds a subcollection of clusters $\mathcal{DT}$ to $\mathcal{T}$ and removes a set of subsumed original clusters $\mathcal{DR}$ from $\mathcal{R}$. An important property of the procedure `Cover` is that the partial cover $\mathcal{DT}$ constructed by it consists of *disjoint* clusters, hence each phase contributes at most one to the degree of each vertex.

Algorithm `Main` is formally described in Figure 1.

The role of Procedure `Cover`($\mathcal{R}$) is to construct a sparse partial collection $\mathcal{DT}$ of clusters covering part of $\mathcal{R}$. This collection is used as part of the output cover $\mathcal{T}$. Each cluster

$\mathcal{R} \leftarrow \mathcal{S}$ /* $\mathcal{R}$ is the collection of remaining (unsubsumed) clusters */
$\mathcal{T} \leftarrow \emptyset$ /* $\mathcal{T}$ is the output cover */
repeat
    $(\mathcal{DR}, \mathcal{DT}) \leftarrow$ `Cover`$(\mathcal{R})$ /* invoke procedure `Cover` */
    $\mathcal{T} \leftarrow \mathcal{T} \cup \mathcal{DT}$
    $\mathcal{R} \leftarrow \mathcal{R} \setminus \mathcal{DR}$
until $\mathcal{R} = \emptyset$

Figure 1: Algorithm `Main`.

$\mathcal{U} \leftarrow \mathcal{R}$
$\mathcal{DT} \leftarrow \emptyset$
$\mathcal{DR} \leftarrow \emptyset$
repeat
    Select an arbitrary cluster $S \in \mathcal{U}$.
    $\mathcal{Z} \leftarrow \{S\}$
    repeat
        $\mathcal{Y} \leftarrow \mathcal{Z}$
        $Y \leftarrow \bigcup_{S \in \mathcal{Y}} S$
        $\mathcal{Z} \leftarrow \{S \mid S \in \mathcal{U},\ S \cap Y \neq \emptyset\}$.
    until $|\mathcal{Z}| \leq |\mathcal{R}|^{1/k} |\mathcal{Y}|$
    $\mathcal{U} \leftarrow \mathcal{U} - \mathcal{Z}$
    $\mathcal{DT} \leftarrow \mathcal{DT} \cup \{Y\}$
    $\mathcal{DR} \leftarrow \mathcal{DR} \cup \mathcal{Y}$
until $\mathcal{U} = \emptyset$
Output $(\mathcal{DR}, \mathcal{DT})$.

Figure 2: Procedure `Cover`$(\mathcal{R})$.

$Y \in \mathcal{DT}$ is constructed by merging together some clusters of $\mathcal{R}$. During the construction, the procedure keeps also the collection $\mathcal{Y}$ containing the original clusters merged into Y.

The general structure of the procedure is similar to the algorithms presented in [P1, P2]. We start by setting $\mathcal{U}$, the collection of *unprocessed* sets, to equal $\mathcal{R}$. The procedure operates in iterations. Each iteration begins by arbitrarily picking a cluster S in $\mathcal{U}$ and designating it as the kernel of a cluster to be constructed next. The cluster is then repeatedly merged with intersecting clusters from $\mathcal{R}$. This is done in a layered fashion, adding one layer at a time. At each stage, the original cluster is viewed as the internal kernel Y of the resulting cluster Z. The merging process is carried repeatedly until reaching a certain sparsity condition. The procedure then adds the kernel Y of the resulting cluster Z to a collection $\mathcal{DT}$. It is important to note that the newly formed cluster consists of only the kernel Y, and not the entire cluster Z, which contains an additional "external layer" of $\mathcal{R}$ clusters. The role of this external layer is to act as a "protective barrier" shielding the generated cluster Y, and providing the desired disjointness between the different clusters Y added to $\mathcal{DT}$.

Throughout the process, the clusters Y, Z are kept in an "unmerged" form $\mathcal{Y}, \mathcal{Z}$, i.e., as collections of original $\mathcal{R}$ clusters. At the end of the iterative process, when Y is completed, every cluster in the collection $\mathcal{Y}$ is added to $\mathcal{DR}$, and every cluster in the collection $\mathcal{Z}$ is removed from $\mathcal{U}$. Then a new iteration is started. These iterations proceed until $\mathcal{U}$ is exhausted. The procedure then outputs the sets $\mathcal{DR}$ and $\mathcal{DT}$.

Note that each of the original clusters in $\mathcal{DR}$ is covered by some cluster $Y \in \mathcal{DT}$ constructed during the execution of the procedure. However, some original $\mathcal{R}$ clusters are thrown out of consideration without being subsumed by any cluster in $\mathcal{DT}$; these are precisely the clusters merged into some external layer $\mathcal{Z} - \mathcal{Y}$. Therefore, there may be clusters left in $\mathcal{R}$ after the main algorithm removes the elements of $\mathcal{DR}$. This is why a single application of Procedure `Cover` is not enough, and many phases are necessary. (In contrast, the algorithms of [P1, P2] can essentially be thought of as based on a single application of a variant of this process, in which entire Z clusters are taken into the output cover, yielding possibly wide overlaps.)

Procedure `Cover` is formally described in Figure 2.

4.2 Correctness and analysis of the regional cover

The properties of Procedure `Cover` are summarized by the following lemma.

Lemma 4.1 Given a graph $G = (V, E)$, $|V| = n$, a collection of clusters $\mathcal{R}$ and an integer k, the collections $\mathcal{DT}$ and $\mathcal{DR}$ constructed by Procedure `Cover`($\mathcal{R}$) satisfy the following properties:

(1) $\mathcal{DT}$ subsumes $\mathcal{DR}$,

(2) $Y \cap Y' = \emptyset$ for every $Y, Y' \in \mathcal{DT}$,

(3) $|\mathcal{DR}| \geq |\mathcal{R}|^{1-1/k}$, and

(4) $Rad(\mathcal{DT}) \leq (2k-1)Rad(\mathcal{R})$.

Proof: First let us note that since the elements of $\mathcal{U}$ at the beginning of the procedure are clusters (i.e., their induced graphs are connected), the construction process guarantees that every set Y added to $\mathcal{DT}$ is a cluster. Property (1) now holds directly from the construction.

Let us now prove Property (2). Suppose, seeking to establish a contradiction, that there is a vertex v such that $v \in Y \cap Y'$. Without loss of generality suppose that Y was created in an earlier iteration than Y'. Since $v \in Y'$, there must be a cluster S' such that $v \in S'$ and S' was still in $\mathcal{U}$ when the algorithm started constructing Y'. But every such cluster S' satisfies $S' \cap Y \neq \emptyset$, and therefore the final construction step creating the collection $\mathcal{Z}$ from Y should have added S' into $\mathcal{Z}$ and eliminated it from $\mathcal{U}$; a contradiction.

Property (3) is derived as follows. It is immediate from the termination condition of the internal loop that the resulting pair $\mathcal{Y}, \mathcal{Z}$ satisfies $|\mathcal{Z}| \leq |\mathcal{R}|^{1/k}|\mathcal{Y}|$. Therefore

$$|\mathcal{R}| = \sum_{\mathcal{Z}} |\mathcal{Z}| \leq \sum_{\mathcal{Y}} |\mathcal{R}|^{1/k}|\mathcal{Y}| = |\mathcal{R}|^{1/k}|\mathcal{DR}|$$

which proves Property (3).

Finally we analyze the increase in the radius of clusters in the cover. Consider some iteration of the main loop of Procedure `Cover` in Figure 2, starting with the selection of some cluster $S \in \mathcal{R}$. Let J denote the number of times the internal loop was executed. Denote the initial set $\mathcal{Z}$ by $\mathcal{Z}_0$. Denote the set $\mathcal{Z}$ (respectively, $Y, \mathcal{Y}$) constructed on the i'th internal iteration $(1 \leq i \leq J)$ by $\mathcal{Z}_i$ (resp., $Y_i, \mathcal{Y}_i$). Note that for $1 \leq i \leq J$, $\mathcal{Z}_i$ is constructed on the basis of $\mathcal{Y}_i$, $\mathcal{Y}_i = \mathcal{Z}_{i-1}$ and $Y_i = \bigcup_{S \in \mathcal{Y}_i} S$. We proceed along the following chain of claims.

Claim 4.2 $|\mathcal{Z}_i| \geq |\mathcal{R}|^{i/k}$ for every $0 \leq i \leq J-1$, and strict inequality holds for $i \geq 1$.

Proof: By induction on i. The claim is immediate for $i = 0$. Assuming the claim for $i - 1 \geq 0$, it remains to prove that $|\mathcal{Z}_i| > |\mathcal{R}|^{1/k}|\mathcal{Z}_{i-1}|$, which follows directly from the fact that the termination condition of the internal loop was not met. ∎

Corollary 4.3 $J \le k$.

Claim 4.4 For every $1 \le i \le J$, $Rad(Y_i) \le (2i-1)Rad(\mathcal{R})$.

Proof: By straightforward induction on i. The base case is immediate since $Y_1 = S \in \mathcal{R}$. The inductive step follows from the fact that for $2 \le i \le J$, $Rad(Y_i) \le Rad(Y_{i-1}) + 2Rad(\mathcal{R})$, since $\mathcal{Z}_{i-1}$ is created from Y_{i-1} by adding into it all $\mathcal{R}$ clusters intersecting it, and Y_i is simply a merge of all the clusters in $\mathcal{Z}_{i-1}$. ▮

It follows from Corollary 4.3 and Claim 4.4 that $Rad(Y_J) \le (2k-1)Rad(\mathcal{R})$, which completes the proof of the last property of the Lemma. ▮

We are now ready to prove Theorem 3.1.

Proof of Theorem 3.1: We need to prove that given a graph $G = (V, E)$, $|V| = n$, a cover $\mathcal{S}$ and an integer $k \ge 1$, the cover $\mathcal{T}$ constructed by Algorithm `Main` satisfies the three properties of the Theorem.

Let $\mathcal{R}^i$ denote the contents of the set $\mathcal{R}$ at the beginning of phase i, and let $r_i = |\mathcal{R}^i|$. Let $\mathcal{DT}^i$ denote the collection $\mathcal{DT}$ added to $\mathcal{T}$ at the end of phase i, and let $\mathcal{DR}^i$ be the set $\mathcal{DR}$ removed from $\mathcal{R}$ at the end of phase i.

Property (1) follows from the fact that $\mathcal{T} = \bigcup_i \mathcal{DT}^i$, $\mathcal{S} = \bigcup_i \mathcal{DR}^i$ and by Property (1) of Lemma 4.1 $\mathcal{DT}^i$ subsumes $\mathcal{DR}^i$ for every i. Property (2) follows directly from Property (4) of Lemma 4.1. It remains to prove Property (3). This property relies on the fact that by Property (2) of Lemma 4.1, each vertex v participates in at most one cluster in each collection $\mathcal{DT}^i$. Therefore it remains to bound the number of phases performed by the algorithm. This bound relies on the following observations. By Property (3) of Lemma 4.1, in every phase i, at least $|\mathcal{DR}^i| \ge |\mathcal{R}^i|^{1-1/k}$ clusters of $\mathcal{R}^i$ are removed from the set $\mathcal{R}^i$, i.e., $r_{i+1} \le r_i - r_i^{1-1/k}$.

Claim 4.5 Consider the recurrence relation $x_{i+1} = x_i - x_i^\alpha$, for $0 < \alpha < 1$. Let $f(n)$ denote the least index i such that $x_i \le 1$ given $x_0 = n$. Then $f(n) = O((1-\alpha)^{-1} n^{1-\alpha})$.

Proof: It follows from the definition of $f(n)$ that

$$f(n) \le \frac{n/2}{(n/2)^\alpha} + f(n/2).$$

¿From this we get

$$f(n) \le n^{1-\alpha} \sum_{j=1}^{\log n} \left(2^{\alpha-1}\right)^j < n^{1-\alpha} \int_{x=0}^{\infty} \left(2^{\alpha-1}\right)^x dx = ((1-\alpha)\ln 2)^{-1} n^{1-\alpha}.$$

▮

Consequently, since $r_0 = n$, $\mathcal{S}$ is exhausted after no more than $O(k|\mathcal{S}|^{1/k})$ phases of Algorithm `Main`, and hence $\Delta(\mathcal{T}) = O(k|\mathcal{S}|^{1/k})$. This completes the proof of the Theorem. ∎

Acknowledgments

The authors thank Steve Ponzio for pointing out an error in a previous version of the paper.

References

[ABLP] B. Awerbuch, A. Bar-Noy, N. Linial, and D. Peleg, Compact Distributed Data Structures for Adaptive Routing, *21st ACM Symp. on Theory of Computing*, Seattle, WA, May 1989, 479–489.

[AKP] B. Awerbuch, S. Kutten and D. Peleg, Efficient Deadlock-Free Routing, Manuscript, January 1990.

[AP1] B. Awerbuch and D. Peleg, Online Tracking of Mobile Users, Technical Report MIT/LCS/TM-410, Aug. 1989.

[AP2] B. Awerbuch and D. Peleg, Local Synchronizers, Manuscript, March 1990.

[BJ] A.E. Baratz and J.M. Jaffe, Establishing Virtual Circuits in Large Computer Networks, *Computer Networks* **12**, (1986), pp. 27–37.

[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, *27th Symp. on Foundations of Computer Science*, 1986, 428–437.

[KK1] L. Kleinrock and F. Kamoun, Hierarchical Routing for Large Networks; Performance Evaluation and Optimization, *Computer Networks* **1**, (1977), pp. 155–174.

[KK2] L. Kleinrock and F. Kamoun, Optimal Clustering Structures for Hierarchical Topological Design of Large Computer Networks, *Networks* **10**, (1980), pp. 221–248.

[P1] D. Peleg, Sparse Graph Partitions, Report CS89-01, Dept. of Applied Math., The Weizmann Institute, Rehovot, Israel, February 1989.

[P2] D. Peleg, Distance-Dependent Distributed Directories, Report CS89-10, Dept. of Applied Math., The Weizmann Institute, Rehovot, Israel, May 1989.

[PU] D. Peleg and E. Upfal, A Tradeoff between Size and Efficiency for Routing Tables, *J. of the ACM* **36**, (1989), 510–530.

[Pr] R. Perlman, Hierarchical Networks and the Subnetwork Partition Problem, *Proc. 5th Conf. on System Sciences*, Hawaii, 1982.

[SK] N. Santoro and R. Khatib, Labelling and implicit Routing in Networks, *The Computer Journal* **28**, (1985), pp. 5–8.

[S] C.A. Sunshine, Addressing Problems in Multi-Network Systems, *Proc. IEEE INFOCOM*, Las-Vegas, 1982.

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

[vLT2] J. van Leeuwen and R.B. Tan, Interval Routing, *The Computer Journal* **30**, (1987), 298–307.