Sparse Partitions

(Extended Abstract[1])

Baruch Awerbuch * David Peleg †

Abstract: This abstract presents a collection of clustering and decomposition techniques enabling the construction of sparse and locality preserving representations for arbitrary networks. These new clustering techniques have already found several powerful applications in the area of distributed network algorithms. Two of these applications are described in this abstract, namely, routing with polynomial communication-space tradeoff and online tracking of mobile users.

1 Introduction

1.1 Motivation

As networks grow larger, various control and management functions become increasingly more complex and expensive. Traditional protocols, based on a global approach, require all sites to participate in their activities, and to maintain considerable amounts of global information (e.g. topological data, status tables etc). This becomes problematic due to space considerations, the complexity of maintaining and updating this global information and the increasing work loads on participating sites.

This realization has led to the development of systems based on *clustered* (and hierarchical) representations of networks. Such methods allow processors to maintain only limited (local) knowledge regarding the state of the network. In particular, the representation method considered in this paper is based on breaking the network $G(V,E)$ into connected regions, or *clusters*, and thus obtaining a *cover* for the network, i.e., a collection of clusters that covers the entire set of vertices V. (A *partition* is a cover consisting of disjoint clusters.) Informally speaking, a protocol based on such a cover will require a vertex to participate only in those activities that occur in the cluster(s) it belongs to. Consequently, the cost of a task tends to depend on its locality properties *in the clustered representation*: a task involving a single cluster is relatively cheap, whereas a task requiring cooperation between sites from different clusters would be more expensive.

A potential problem with a naive implementation of the cluster cover approach is that the resulting representation might not necessarily conform with the topological structure of the network, and particularly, that it might have different locality properties. For instance, it is possible that two neighboring nodes will find themselves in different clusters in the cover (which would make tasks requiring information exchange between these nodes more complex). Clearly, it is desirable that the performance of a task depend on its locality properties *in the real network*. This translates into the following requirements. First, when the given task concerns only a subset of the sites, located in a small region in the network, one would like the execution of the task to involve only sites in or around that region. Further, it is desirable that the cost of the task be proportional to its locality level.

The above discussion motivates the need for *locality preserving* network covers, whose structure faithfully captures the topology of the network itself. This abstract supplies the graph-theoretic tools for achieving this goal, by presenting a collection of clustering and decomposition techniques enabling the construction of efficient cover-based locality preserving representations for arbitrary networks. Some applications for these techniques are described as well.

*Department 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; BITNET: peleg@wisdom. Supported in part by an Allon Fellowship, by a Walter and Elise Haas Career Development Award and by a Bantrell Fellowship. Part of the work was done while visiting IBM T.J. Watson Research Center.

[1]Full details for most of the results of this abstract can be found in [AP89b, AP89a, Pel89, Pel90]

1.2 Basic parameters

Let us first attempt to identify the appropriate criteria for evaluating the quality of a cluster cover. There are two inherent criteria of interest for cluster design. The first is the "size" of the clusters; intuitively, small clusters require less internal communication and provide faster response. The second criterion is the level of "interaction" between clusters. This notion is meant to capture the extent to which individual vertices belonging to one cluster need to be involved in the affairs of other (neighboring or overlapping) clusters.

Next, let us discuss the technical manifestations of the two qualitative criteria defined above in our cover-based representations. The "size" of the cluster is captured by its *radius*. As for the "interaction level" criterion, we refer to the technical measures related to this notion as the *sparsity* of the cover. There are several plausible definitions for this sparsity measure, and the ones considered in this abstract are those proved useful in various applications. All of these definitions rely on the *degrees* of vertices or clusters in either the cover, the graph itself or the induced cluster graph. For one concrete example, a possible sparsity measure for a cover $\mathcal{S}$ (with cluster overlaps), denoted $\Delta(\mathcal{S})$, is the maximum number of occurrences of a vertex in clusters $S \in \mathcal{S}$, (i.e., the maximum degree of a vertex in the hypergraph $(V, \mathcal{S})$). In a sense, this measure captures the extent of the overlaps existing between the clusters in the given cover.

The crucial point we would like to stress is that the criteria of cluster radius and sparsity are tightly related to the significant complexity measures of distributed network algorithms. In all applications considered in the sequel, smaller cluster radii in a given cover directly translate into lower time complexities, due to the fact that computations are performed inside clusters, on shortest path trees spanning the clusters. At the same time, low sparsity (to be specific, low degrees of vertices in the cover) guarantees low memory requirements, since each vertex is actively involved in (and thus stores information for) only the clusters it belongs to. Finally, and perhaps most importantly, the communication complexity of cover-based protocols strongly depends on both parameters (typically in their product).

It is therefore worthwhile to attempt to devise graph covers that are efficient in both parameters. The key problem we are faced with lies in the fact that the two parameters apear to be inherently conflicting, and improving one of them usually results in degrading the other. Our task therefore becomes that of striking a balance between the two parameters, and seeking the appropriate break-point for each of our applications.

Note that if one ignores the sparsity requirement, it is a simple matter to design a locality preserving cover even for arbitrary networks: for any desirable radius m (serving as the "locality" parameter), select the cover $\mathcal{N}_m$ containing all the radius m neighborhoods of every node in the network. This guarantees that if two nodes are at distance at most m of each other, there exists a common cluster containing both of them. The obvious problem with this cover is that it might not be sparse; it may be that some nodes occur in many clusters, incurring high costs.

1.3 Constructions

All the algorithms presented in this abstract for constructing good (namely, sparse) covers are based on the central idea of *coarsening*. These algorithms share the following overall structure. We are given some initial cover $\mathcal{S}$, dictated by the application. This will usually be some natural cover (for instance, the neighborhood cover $\mathcal{N}_m$ for some m), that is not necessarily sparse. The goal of our algorithm is to construct, via repeated cluster merging, a *coarsening* cover $\mathcal{T}$ (namely, with the property that each original cluster is fully subsumed in one of the clusters of $\mathcal{T}$) that is relatively sparse on the one hand, and whose cluster radii are not much larger than those of $\mathcal{S}$ on the other hand. (We refer to the ratio between the respective radii of $\mathcal{S}$ and $\mathcal{T}$ clusters as the *radius ratio* of the coarsening.) Our results exhibit a tradeoff between the two parameters: better sparsity implies worse radius ratio, and vice versa. For example, one may get coarsening covers with radius ratio $O(k)$ and average degree $O(n^{1/k})$, for every $1 \le k \le \log n$. (Throughout, n denotes the number of vertices in the network.) The lower bounds established in Thm. 5.4 imply that these tradeoffs are nearly tight.

When addressing our sparsity criteria, it is possible to consider either the *maximum* or the *average* sparsity. The algorithms for reducing the average sparsity are simpler, and in fact, for some applications they suffice to enable considerable reduction in space and communication complexities. However, our strongest algorithms manage to bound also the maximum sparsity measure (albeit with somewhat inferior bounds). The special significance of bounding the maximum degrees is that this typically enables us to bound the space and communication costs of *individual processes*, rather than just the total costs, and therefore enable us to *balance* both the memory loads and the work invested in communication among the processors in a satisfactory manner.

Clustering techniques become most potent when used in conjunction with a hierarchical structure. The idea is to construct a hierarchy of covers, with higher levels using larger radii clusters. This enables communication and data structures to be organized at the appropriate level according to the "locality level" of the application itself: an application that only calls for neighboring vertices to communicate with each other can be carried out at the lowest level of the hierarchy, incurring the lowest costs possible. On the other hand, tasks involving distant vertices will be performed on higher levels of the hierarchy, and will cost accordingly. These ideas are demonstrated in the applications presented in the sequel.

We shall also discuss several other graph-theoretic structures that are strongly related to covers. These include sparse spanners (cf. [PS89]), tree covers of graphs (cf. [AKP90]), and the new concepts of regional matchings and diameter-based separators. All of these structures are constructible using one of our clustering algorithms, and each of them has proved to provide a convenient representation for handling certain network applications.

Another related graph structure is the *network decomposition* defined in [AGLP89]. Linial and Saks [LS90] have given the best known sequential algorithm for constructing such a decomposition, which is related to diameter-based separators, and also devised an ingenious randomized distributed implementation.

1.4 Applications

The new clustering techniques described herein have already found several applications in the area of distributed network algorithms. One such application is that of maintaining locality preserving distributed data structures, particularly directories. This application is handled in [Pel90] using the average cover algorithm `AV_COVER` described in Section 5.

Another application involves the classical problem of deadlock-free routing. In [AKP90], it is shown how to use the tree covers of Section 6.4 in order to devise a deadlock-free routing scheme using fewer buffers at the cost of increasing the route length. Specifically, the resulting routing requires $O(k \cdot n^{1/k} \cdot \log Diam(G))$ buffers per vertex, where $Diam(G)$ denotes the diameter of the network, and the routes are at most $O(k)$ longer than optimal.

In [AP] we use regional matchings to construct a novel synchronizer, with only $O(\log^3 n)$ communication overhead; this is a significant improvement over previous solutions which have $\Theta(n)$ overhead. A straightforward distributed implementation of algorithm $\mathtt{MAX\_PART}_c$ can also be used in order to yield a fast (polylog time) preprocessing algorithm for setting up synchronizer γ of [Awe85].

Other applications related to our covering techniques are described in [AR90, LS90].

In the sequel we present in detail two powerful applications, for designing efficient routing and tracking mechanisms.

Routing

Delivering messages between pairs of processors is a primary activity of any distributed communication network. 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 guarantees optimal routes, but requires a total of $O(n^2 \log n)$ memory bits in the 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. Evidently, locality-preserving representations of the network play an important role in this task.

The problem of efficiency-memory tradeoffs for routing schemes was first raised in [KK77], and later studied extensively (e.g. [BJ86, Per82, FJ88, SK85, vLT86, PU89a, ABLP89, Pel90]). It turns out that this efficiency-memory tradeoff is strongly related to the radius-sparsity tradeoff of neighborhood covers. This observation served as the basis of the previous solutions to the routing problem in [PU89a, ABLP89, Pel90], although the particular clustered representations used in these papers, as well as the resulting routing procedures, are considerably different than our current ones.

Let us briefly compare the properties of our new scheme with previous ones. 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 family of hierarchical routing schemes (for every integer $k \geq 1$) presented in [PU89a] guarantee stretch $O(k)$ and require storing a total of $O\big(k^3 n^{1+1/k} \log n\big)$ bits of routing information in the network. This behavior is almost optimal as far as total memory requirements are concerned, as implied from a lower bound given in [PU89a] on the space requirement of any scheme with a given stretch. However, this scheme does not bound the individual memory requirements of each vertex, and it is also limited to unit cost edges. The schemes proposed in [ABLP89, Pel90] bound the worst-case individual memory requirements of vertices, but at the cost of an inferior efficiency-space tradeoff. In particular, in an n-processor network G of weighted diameter $Diam(G)$, the schemes HS_k of [ABLP89], 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 [Pel90], for $k \geq 1$, use $O(\log Diam(G) \cdot \log n \cdot n^{\frac{1}{k}})$ bits of memory per vertex and guarantee a stretch of $O(4^k)$. Thus the stretch becomes exponential in k, in contrast with the linear dependency achieved in [PU89a].

Our new schemes remedy this situation. For every (weighted) network G and every integer $k \geq 1$, we construct a name independent hierarchical routing scheme $\mathcal{H}_k$ with stretch $O(k^2)$ using $O(kn^{1/k} \log n \log Diam(G)))$ memory bits per vertex. Thus the new scheme regains the polynomial dependency of the stretch factor on k. We comment that the schemes of [ABLP89] have the advantage of a purely combinatorial space complexity, i.e., complexity that does not depend on the edge weights.

Online tracking of mobile users

When users in a distributed communication network are *mobile*, i.e., are allowed to move from one network vertex to another, it is necessary to have a mechanism enabling one to keep track of them and contact them at their current residence. Our purpose is to design efficient tracking mechanisms, based on distributed directory structures (cf. [LEH85]), minimizing the communication redundancy involved.

There are many types of network activities that may fall under the category of mobile users. A prime example is that of cellular phone networks. In fact, one may expect that in the future, all telephone systems will be based on "mobile telephones numbers," i.e., ones that are not bound to any specific physical location. Another possible application is a system one may call "distributed yellow pages," or "distributed matchmaking" [MV88]. Such a system is necessary in an environment consisting of mobile "servers" and "clients." The system has to provide means for enabling clients in need of some service to locate the whereabouts of the server they are looking for.

In essence, the tracking mechanism has to support two operations: a "move" operation, causing a user to move to a new destination, and a "find" operation, enabling one to contact the current address of a specified user. However, the task of minimizing the communication overheads of the "move" and "find" operations simultaneously appears difficult, as can be realized by examining the following two extreme strategies (considered in [MV88]). The *full-information* strategy requires every vertex in the network to maintain a complete directory containing up-to-date information on the whereabouts of every user. This results in cheap "find" operations, but very expensive "move" operations. In contrast, the *no-information* strategy performs no updates following a "move," thus abolishing altogether the concept of directories and making the "move" operations cheap. However, establishing a connection via a "find" operation becomes very expensive, as it requires a global search over the entire network. Alternatively, it is possible to require that whenever a user moves, it leaves a "forwarding" pointer at the old address, pointing to its new address. Unfortunately, this heuristic still does not guarantee any good worst-case bound for the "find" operations.

Our purpose is to design some intermediate "partial-information" strategy, that will perform well for any find/move pattern. This problem was tackled also by [MV88]. However, their approach considers only the worst-case performance, and the schemes designed there treat all requests alike, and ignore locality considerations. Our goal is to design more refined strategies that take into account the inherent costs of the particular requests at hand, which in many cases may be lower than implied by the worst-case analysis. In particular, we would like moves to a near-by location, or searches for near-by users, to cost less. Thus we are interested in the worst case *overhead* incurred by a particular strategy. This overhead is evaluated by comparing the total cost invested in a sequence of "move" and "find" operations against the inherent cost (namely, the cost incurred by the operations themselves, assuming full information is available for free.) This comparison is done over all sequences of "move" and "find" operations. The strategy proposed guarantees overheads that are *polylogarithmic* in the size and the diameter of the network. Again, our strategy is based on a locality-preserving hierarchical representation of the network, which forms the structural skeleton for the data structures maintained by the tracking mechanism. The overheads of the "move" and "find" operations grows as the product of the radius and the maximum degree of the underlying hierarchy of neighborhood covers.

2 Definitions

We consider an arbitrary weighted graph $G(V, E, w)$, where V is the set of vertices, E is the set of edges and $w : E \rightarrow R^+$ is a *weight* function, assigning a non-negative weight $w(e)$ to every edge $e \in E$.

For two vertices u, w in G, let $dist_G(u, w)$ denote the (weighted) length of a shortest path in G between those vertices, where the length of a path $(e_1, \ldots, e_s)$ is $\sum_{1 \leq i \leq s} w(e_i)$. (We sometimes omit the subscript G where no confusion arises.) This definition is generalized to sets of vertices U, W in G in the natural way, by letting

$dist_G(U, W) = \min\{dist_G(u, w) \mid u \in U, w \in W\}$.

The *j-neighborhood* of a vertex $v \in V$ is defined as $N_j(v) = \{w \mid dist(w,v) \le j\}$. Given a subset of vertices $R \subseteq V$, denote $\mathcal{N}_m(R) = \{N_m(v) \mid v \in R\}$. Let $Diam(G)$ denote the (weighted) *diameter* of the network, i.e., $\max_{u,v \in V}(dist_G(u,v))$. 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., $\min_{v \in V}(Rad(v,G))$. 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, $Rad(G) \le Diam(G) \le 2Rad(G)$.

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. Throughout we denote clusters by capital P, Q, R etc., and collections of clusters by calligraphic type, $\mathcal{P}, \mathcal{Q}, \mathcal{R}$ etc. 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 $\bigcup_i S_i = V$. A *partition* of G is a cover $\mathcal{S}$ with the additional property that $S \cap S' = \emptyset$ for every $S, S' \in \mathcal{S}$. Given a collection of clusters $\mathcal{S}$, let $Diam(\mathcal{S}) = \max_i Diam(S_i)$ and $Rad(\mathcal{S}) = \max_i Rad(S_i)$.

We use the following measures for the sparsity (or "interaction level") of covers and partitions. First consider a cover $\mathcal{S}$. For every vertex $v \in V$, let $deg(v, \mathcal{S})$ 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 $\Delta(\mathcal{S}) = \max_{v \in V} deg(v, \mathcal{S})$. The *average degree* of a cover $\mathcal{S}$ is defined as $\bar{\Delta}(\mathcal{S}) = (\sum_{v \in V} deg(v, \mathcal{S}))/n$.

For partitions we may be interested in several different measures, based on the neighborhood relations among clusters or between clusters and individual vertices. Given a partition $\mathcal{S}$ and a cluster $S \in \mathcal{S}$, let us define the *vertex-neighborhood* of S as $\Gamma_v(S) = N_1(S)$, and the *cluster-neighborhood* of S as $\Gamma_c(S) = \{S' \mid S' \in \mathcal{S},\ dist(S, S') = 1\}$. The maximum and average *vertex-degree* and *cluster-degree* of the partition $\mathcal{S}$ are defined accordingly as $\Delta_v(\mathcal{S}) = \max_{S \in \mathcal{S}} |\Gamma_v(S)|$, $\bar{\Delta}_v(\mathcal{S}) = (\sum_{S \in \mathcal{S}} |\Gamma_v(S)|)/n$, $\Delta_c(\mathcal{S}) = \max_{S \in \mathcal{S}} \frac{|\Gamma_c(S)|}{|S|}$, and $\bar{\Delta}_c(\mathcal{S}) = (\sum_{S \in \mathcal{S}} |\Gamma_c(S)|)/|\mathcal{S}|$.

Given two covers $\mathcal{S} = \{S_1, \ldots, S_m\}$ and $\mathcal{T} = \{T_1, \ldots, T_k\}$, we say that $\mathcal{T}$ *coarsens* $\mathcal{S}$ if for every $S_i \in \mathcal{S}$ there exists a $T_j \in \mathcal{T}$ such that $S_i \subseteq T_j$. In this case, we refer to the ratio $Rad(\mathcal{T})/Rad(\mathcal{S})$ as the *radius ratio* of the coarsening.

3 Sparse coarsening covers

This section describes an algorithm for the construction of a sparse coarsening cover, i.e., a cover with low maximum degree. The main result is:

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

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

(2) $\Delta(\mathcal{T}) \le 2k|\mathcal{S}|^{1/k}$.

Let us remark that it is possible to replace the degree bound of Property (2) with $O(k \cdot n^{1/k})$. This requires a more complex algorithm and analysis, and therefore we prefer to state the theorem as above. In most of our applications there is no real difference, as $|\mathcal{S}| = n$. We also mention that this result is close to optimal in some cases, as implied from Thm. 5.4.

The coarsening problem is handled by reducing it to the sub-problem of constructing a *partial cover*. The input of this problem is a graph $G = (V, E)$, $|V| = n$, a collection of (possibly overlapping) clusters $\mathcal{R}$ and an integer $k \ge 1$. The output consists of a collection of *disjoint* clusters, $\mathcal{DT}$, that subsume a subset $\mathcal{DR} \subseteq \mathcal{R}$ of the original clusters. The goal is to subsume "many" clusters of $\mathcal{R}$ while maintaining the radii of the output clusters in $\mathcal{DT}$ relatively small. We now describe a procedure `Cover`($\mathcal{R}$) achieving this goal.

Procedure `Cover`($\mathcal{R}$) starts by setting $\mathcal{U}$, the collection of *unprocessed* clusters, to equal $\mathcal{R}$. The procedure operates in iterations. Each iteration constructs one output cluster $Y \in \mathcal{DT}$, by merging together some clusters of $\mathcal{U}$. The 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 $\mathcal{U}$. 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 (specifically, until the next iteration increases the number of clusters merged into Z by a factor of less than $|\mathcal{R}|^{1/k}$). 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 procedure keeps also the "unmerged" collections $\mathcal{Y}, \mathcal{Z}$ containing the original $\mathcal{R}$ clusters merged into Y and Z. 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 $\mathcal{Z} - \mathcal{Y}$.

Procedure `Cover` is formally described in Figure 2. Its properties are summarized by the following lemma.

Lemma 3.2 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) $\mathcal{DT}$ coarsens $\mathcal{DR}$,

(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` 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. We first observe that $|\mathcal{Z}_i| \geq |\mathcal{R}|^{i/k}$ for every $0 \leq i \leq J-1$, and strict inequality holds for $i \geq 1$. This implies $J \leq k$. Finally, we prove that for every $1 \leq i \leq J$, $Rad(Y_i) \leq (2i-1)Rad(\mathcal{R})$. It follows from the last two claims that $Rad(Y_J) \leq (2k-1)Rad(\mathcal{R})$, which completes the proof of the last property of the Lemma. ∎

$$
\begin{array}{l}
\mathcal{R} \leftarrow \mathcal{S};\ \mathcal{T} \leftarrow \emptyset \\
\textbf{repeat} \\
\quad (\mathcal{DR}, \mathcal{DT}) \leftarrow \texttt{Cover}(\mathcal{R}) \\
\quad \mathcal{T} \leftarrow \mathcal{T} \cup \mathcal{DT} \\
\quad \mathcal{R} \leftarrow \mathcal{R} \setminus \mathcal{DR} \\
\textbf{until } \mathcal{R} = \emptyset
\end{array}
$$

Figure 1: Algorithm `MAX_COVER`.

We now present the algorithm `MAX_COVER`, whose task is to construct a cover as in Theorem 3.1. 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 phases. Each phase consists of the activation of the procedure `Cover`$(\mathcal{R})$, which adds a subcollection of output clusters $\mathcal{DT}$ to $\mathcal{T}$ and removes the set of subsumed original clusters $\mathcal{DR}$ from $\mathcal{R}$.

Algorithm `MAX_COVER` is formally described in Fig. 1.

We are now ready to prove Theorem 3.1 itself. We need to prove that given a graph $G = (V, E)$, $|V| = n$, a cover $\mathcal{S}$ and an integer $k \geq 1$, the cover $\mathcal{T}$ constructed by Algorithm `MAX_COVER` coarsens $\mathcal{S}$ and satisfies the two 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.

The fact that $\mathcal{T}$ coarsens $\mathcal{S}$ 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 3.2, $\mathcal{DT}^i$ coarsens $\mathcal{DR}^i$ for every i. Property (1) follows directly from Property (4) of Lemma

$\mathcal{U} \leftarrow \mathcal{R}$; $\mathcal{DT} \leftarrow \emptyset$; $\mathcal{DR} \leftarrow \emptyset$
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($\mathcal{R}$).

3.2. It remains to prove Property (2). This property relies on the fact that by Property (2) of Lemma 3.2, 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 3.2, in every phase i, at least $|\mathcal{DR}^i| \geq |\mathcal{R}^i|^{1-1/k}$ clusters of $\mathcal{R}^i$ are removed from the set $\mathcal{R}^i$, i.e., $r_{i+1} \leq r_i - r_i^{1-1/k}$.

Claim 3.3 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 $x_i \leq 1$ given $x_0 = n$. Then $f(n) < ((1-\alpha)\ln 2)^{-1} n^{1-\alpha}$.

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

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

From this we get

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

Consequently, since $r_0 = n$, $\mathcal{S}$ is exhausted after no more than $\frac{k}{\ln 2}|\mathcal{S}|^{1/k}$ phases of Algorithm MAX_COVER, and hence $\Delta(\mathcal{T}) \leq 2k|\mathcal{S}|^{1/k}$. This completes the proof of Theorem 3.1. $\blacksquare$

Algorithm MAX_COVER as described here requires $O(n^2)$ steps, and its straightforward distributed implementation has communication cost $O(n^2)$. In [AP90] we describe a more efficient distributed algorithm.

4 Sparse coarsening partitions

In this section we discuss algorithms for the construction of sparse coarsening partitions, according to the maximum vertex- and cluster-degree measures.

Our first result concerns a maximum vertex-degree partition algorithm MAX_PART$_v$. This is the only algorithm in which our tradeoff deviates considerably from the known lower bound, in that the radius ratio is exponential in k.

Theorem 4.1 Given a graph $G = (V, E)$, $|V| = n$, a partition $\mathcal{S}$ and an integer $k \geq 1$, it is possible to construct a coarsening partition $\mathcal{T}$ that satisfies the following properties:

(1) $Rad(\mathcal{T}) \leq 2 \cdot 5^k \cdot Rad(\mathcal{S})$, and

(2) $\Delta_v(\mathcal{T}) = O(k \cdot n^{1/k})$.

The algorithm and the proof of the theorem will be presented in the full paper (see also [Pel89]).

Our next result concerns a maximum cluster-degree partition algorithm MAX_PART$_c$. The polynomial bound obtained for the radius ratio here is the result of the weaker sparsity criterion.

Theorem 4.2 Given a graph $G = (V, E)$, $|V| = n$, a partition $\mathcal{S}$ and an integer $k \geq 1$, it is possible to construct a coarsening partition $\mathcal{T}$ that satisfies the following properties:

(1) $Rad(\mathcal{T}) \leq 2 \cdot k^{\log 7} Rad(\mathcal{S})$, and

(2) $\Delta_c(\mathcal{T}) = O(\log k \cdot n^{1/k})$.

The algorithm and the proof of the theorem will be presented in the full paper.

5 Average degree clustering

In this section we consider algorithms for constructing coarsening covers and partitions with low *average* degree. Generally speaking, we expect to get better tradeoffs in this case. Let us first consider the case of covers.

Theorem 5.1 Given a graph $G = (V, E)$, $|V| = n$, a cover $\mathcal{S}$ and an integer $k \geq 1$, it is possible to construct a coarsening cover $\mathcal{T}$ that satisfies the following properties:

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

(2) $\bar{\Delta}(\mathcal{T}) = O(n^{1/k})$.

The algorithm AV_COVER can be thought of as based on a single application of Procedure Cover (used in algorithm MAX_COVER), taking the entire clusters Z produced by the procedure (including the external layers) as output clusters. A formal description of the algorithm and a proof of the lemma are omitted from the abstract (see [Pel89, Pel90]).

Next let us consider partitions. For the cluster-degree measure, a minor modification of Algorithm AV_COVER yields Algorithm AV_PART$_c$ for this problem. This algorithm is in fact a natural extension of the algorithm for constructing synchronizer γ in [Awe85]. We thus have

Theorem 5.2 Given a graph $G = (V, E)$, $|V| = n$, a partition $\mathcal{S}$ and an integer $k \geq 1$, it is possible to construct a coarsening partition $\mathcal{T}$ that satisfies the following properties:

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

(2) $\bar{\Delta}(\mathcal{T}) = O(n^{1/k})$.

The situation is again harder with the vertex degree measure, since the radius ratio guaranteed by algorithm MAX_PART$_v$ is exponential. In the full paper we describe a simple variant of algorithm AV_COVER, named AV_PART$_v$, that solves a weaker problem, in which the output clusters are allowed to be disconnected. (It is necessary to define cluster radius in this case based on distances in the entire graph G.) We refer to this problem as the problem of "weak partitions" with low average vertex-degree. We get

Theorem 5.3 Given a graph $G = (V, E)$, $|V| = n$, a partition $\mathcal{S}$ and an integer $k \geq 1$, it is possible to construct a weak coarsening partition $\mathcal{T}$ that satisfies the following properties:

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

(2) $\bar{\Delta}(\mathcal{T}) = O(n^{1/k})$.

Let us now turn our attention to the question of lower bounds for clustering algorithms. Relying on the lower bound of [PS89] for spanners (see Sec. 6.2), and on the relationships between spanners and covers, we show the following.

Theorem 5.4 For every $k \geq 3$, there exist unweighted n-vertex graphs $G = (V, E)$ for which
(a) for every cover $\mathcal{T}$ coarsening $\mathcal{N}_1(V)$, if $Rad(\mathcal{T}) \leq k$ then $\bar{\Delta}(\mathcal{T}) = \Omega(n^{1/k})$.
(b) for every partition $\mathcal{T}$ coarsening $\mathcal{N}_0(V)$, if $Rad(\mathcal{T}) \leq k$ then $\bar{\Delta}_c(\mathcal{T}) = \Omega(n^{1/k})$.

These bounds clearly imply similar bounds for the average vertex-degree partition problem, as well as for all maximum degree problems.

6 Related graph structures

6.1 Regional Matchings

In [MV88], Mullender and Vitányi proposed a general paradigm for distributed match-making between clients and servers in a distributed network. Intuitively, a match-making system is a specification of rendezvous locations in the network, enabling users to locate and communicate with one another.

Since our goal is to reduce storage and communication costs, it is desirable to keep topological considerations in mind, and devise a match-making mechanism taking locality into account. In this subsection we introduce the concept of a regional matching, geared towards this goal.

The basic components of our construction are a read set $Read(v) \subseteq V$ and a write set $Write(v) \subseteq V$, defined for every vertex v. Consider the collection $\mathcal{RW}$ of all pairs of sets, namely
$\mathcal{RW} = \{ Read(v), Write(v) \mid v \in V \}$.

Definition 6.1 The collection $\mathcal{RW}$ is an *m-regional matching* (for some integer $m \geq 1$) if for all $v, u \in V$ such that $dist(u, v) \leq m$, $Write(v) \bigcap Read(u) \neq \emptyset$.

The relevant parameters of a regional matching are its *radius*, which the maximal distance from a vertex to any other vertex in its read or write set, and its *degree*, which is the maximal number of vertices in any read or write set. Formally, for any m-regional matching $\mathcal{RW}$ define the following four parameters:
$Deg_{read}(\mathcal{RW}) = \max_{v \in V} |Read(v)|$,
$Rad_{read}(\mathcal{RW}) = \frac{1}{m} \max_{u,v \in V} \{dist(u, v) \mid u \in Read(v)\}$,
and $Deg_{write}(\mathcal{RW})$, $Rad_{write}(\mathcal{RW})$ are defined analogously based on the sets $Write(v)$. Again, there appears to be a trade-off between these two parameters, making simultaneous minimization of both of them a nontrivial task. Using algorithm MAX_COVER we get the following result, whose proof is deferred to the full paper (see also [AP89a]).

Theorem 6.2 For all $m, k \geq 1$, it is possible to construct an m-regional matching $\mathcal{RW}_{m,k}$ with

$$\begin{aligned} Deg_{read}(\mathcal{RW}_{m,k}) &\leq 2k \cdot n^{1/k} \\ Deg_{write}(\mathcal{RW}_{m,k}) &= 1 \\ Rad_{read}(\mathcal{RW}_{m,k}) &\leq 2k+1 \\ Rad_{write}(\mathcal{RW}_{m,k}) &\leq 2k+1 \end{aligned}$$

6.2 Spanners

Spanners [PU89b, PS89, ADDJ90] appear to be the underlying graph structure in various constructions in distributed systems and communication networks.

Definition 6.3 Given a connected simple graph $G = (V, E)$, a subgraph $G' = (V, E')$ is a *t-spanner* of G if for every $u, v \in V$, $dist_{G'}(u, v) \leq t \cdot dist_G(u, v)$. We refer to t as the *stretch factor* of the spanner G'.

Using the average cluster-degree partition algorithm $\texttt{AV\_PART}_c$ (or in fact, its variant from [Awe85]), it is shown in [PS89] that

Lemma 6.4 [PS89] For every unweighted n-vertex graph G and for every fixed $k \geq 1$, there exists a (polynomial-time-constructible) $(4k + 1)$-spanner with $O(n^{1+1/k})$ edges.

Using algorithm `AV_COVER`, it is possible to derive a similar result for weighted graphs, although with an additional logarithmic factor based on the edge weights. Recently, a more efficient algorithm for constructing spanners for weighted graphs was proposed in [ADDJ90].

6.3 Low diameter separators

Separators (cf. [LT79]) are traditionally based on cardinality considerations. For arbitrary networks it is not always possible to construct separators of this nature. However, for various distributed applications it may be useful to have separators based on diameter parameters.

Definition 6.5 Given a graph $G(V, E)$, a subset of the vertices $V'' \subseteq V$ is an *(m, ρ, γ)-separator* if the subgraph G' induced by $V' = V - V''$ in G has the following two properties:

1. every two connected components in G' are at distance m or more of each other.
2. every connected component in G' has diameter of $O(m \cdot \rho)$ in G, and
3. The fraction of nodes in V' is at least $O(\frac{1}{\gamma})$.

Intuitively, one should view the set V'' as the small cardinality separator and set V' as the "nicely-packed" interior of the network. The typical goal is to remain with most of the nodes being in the interior, and at the same time to keep ρ, γ small for all fixed m.

Using a single application of the inner loop of Procedure `Cover` in Figure 2, and taking the vertices in all the output clusters in $\mathcal{DT}$ as V' (and the rest of the vertices as V''), we get

Lemma 6.6 For every graph G, for all $m > 0$ and $k \geq 1$, it is possible to construct an $(m, k, n^{1/k})$ separator.

6.4 Tree Covers

Another useful structure involves constructing a sparse tree collection in a graph.

Definition 6.7 Given an undirected graph $G(V, E)$, an *(r, m)-tree cover* is a collection $\mathcal{F}$ of trees in G, that satisfies the following properties:

1. For every two nodes $u, v \in V$, there exists a tree $F \in \mathcal{F}$ such that $dist_F(u, v) \leq r \cdot dist_G(u, v)$.
2. Every node belongs to at most m different trees.

Thm. 3.1 is used in [AKP90] to prove

Lemma 6.8 For every undirected graph $G(V, E)$ and integer $k \geq 1$, it is possible to construct an (r, m)-tree cover $\mathcal{F}_k$ for G, with $r = 8k$ and $m = k \cdot n^{1/k} \cdot \log Diam(G)$.

7 Applications

7.1 Routing

A *routing scheme* RS for the (weighted) 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 to some destination vertex v (specified by a fixed name) via a sequence of message transmissions.

We now give precise definitions for our complexity measures for stretch and memory. The *communication cost* of transmitting an $O(\log n)$ bit message over edge e is the weight $w(e)$ of that edge. Let $Cost(RS, u, v)$ denote the communication cost of the routing scheme when invoked at an origin u, w.r.t. a destination v and an $O(\log n)$ 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 define the *stretch factor* of the scheme RS to be

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

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 $Memory(RS)$.

Our solution 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, each managed by a center vertex. Each cluster has its own internal routing mechanism 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.

The routing component used inside clusters is based on a variant of the *interval routing scheme*, or ITR [SK85, PU89a, ABLP89], that uses a shortest path tree rooted at a vertex r and spanning the cluster.

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.

We construct a regional $(O(k^2 m), m)$-routing scheme, for any integers $k, m \geq 1$. The main stage of the construction involves an application of Theorem 3.1. We start by setting $\mathcal{S} = \mathcal{N}_m(V)$ and constructing a coarsening 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. 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 ITR 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.

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 $(O(k^2 2^i), 2^i)$-routing scheme R_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.

In the full paper (see also [AP89b]) we analyze the scheme and prove (relying on Thm. 3.1):

Theorem 7.1 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 $\mathtt{Stretch}(\mathcal{H}_k) = O(k^2)$ using $Memory(\mathcal{H}_k) = O(n^{1/k} \log^2 n \log Diam(G))$ bits per vertex.

7.2 Online tracking of mobile users

Denote by $Addr(\xi)$ the current address of a specific user ξ. A directory $\mathcal{D}$ with respect to the user ξ is a distributed data structure which enables the following two operations.

$\mathtt{Find}(\mathcal{D}, \xi, v)$: invoked at the vertex v, this operation delivers a search message from v to the location $s = Addr(\xi)$ of the user ξ.

$\mathtt{Move}(\mathcal{D}, \xi, s, t)$: invoked at the current location $s = Addr(\xi)$ of the user ξ, this operation moves ξ to a new location t and performs the necessary updates in the directory.

We are interested in measuring the communication complexity overheads incurred by our `Find` and `Move` algorithms, compared to the minimal "inherent" costs associated with these operations. Let $Cost(F)$ (respectively, $Cost(M)$) denote the actual communication cost of the operation F (resp., M). For a `Find` instruction $F = \mathtt{Find}(\mathcal{D}, \xi, v)$, define the *optimal cost* of F as $Opt\_cost(F) = dist(v, Addr(\xi))$. For a `Move` instruction $M = \mathtt{Move}(\mathcal{D}, \xi, s, t)$, let $Reloc(\xi, s, t)$ denote the actual relocation cost of the user ξ from s to t. We define the optimal cost of M as $Opt\_cost(M) = Reloc(\xi, s, t)$, which is the inherent cost assuming no extra operations, such as directory updates, are taken. This cost depends on the distance between the old and new location, and we assume it satisfies $Reloc(\xi, s, t) \geq dist(s, t)$. (In fact, the relocation of a server is typically much more expensive than just sending a single message between the two locations.)

We would like to define the "amortized overhead" of our operations, compared to their optimal cost. For that purpose we consider mixed sequences of `Move` and `Find` operations. Given such a sequence $\bar{\sigma} = \sigma_1, \ldots, \sigma_\ell$, let $\mathcal{F}(\bar{\sigma})$ denote the subsequence obtained by picking only the `Find` operations from $\bar{\sigma}$. Define the optimal cost and the cost of the subsequence $\mathcal{F}(\bar{\sigma}) = (F_1, \ldots, F_q)$ in the natural way, setting $Opt\_cost(\mathcal{F}(\bar{\sigma})) = \sum_{i=1}^{q} Opt\_cost(F_i)$ and $Cost(\mathcal{F}(\bar{\sigma})) = \sum_{i=1}^{q} Cost(F_i)$.

The *find-stretch* of the directory with respect to a given sequence of operations $\bar{\sigma}$ is defined as

$$\texttt{Stretch}_{find}(\bar{\sigma}) = \frac{Cost(\mathcal{F}(\bar{\sigma}))}{Opt\_cost(\mathcal{F}(\bar{\sigma}))} .$$

The find-stretch of the directory, denoted $\texttt{Stretch}_{find}$, is the least upper bound on $\texttt{Stretch}_{find}(\bar{\sigma})$, taken over all finite sequences $\bar{\sigma}$.

Define the subsequence $\mathcal{M}(\bar{\sigma})$, the costs $Opt\_cost(\mathcal{M}(\bar{\sigma}))$ and $Cost(\mathcal{M}(\bar{\sigma}))$ and the *move-stretch* factors $\texttt{Stretch}_{move}(\bar{\sigma})$ and $\texttt{Stretch}_{move}$ analogously.

Our tracking mechanism is based on a hierarchy of 2^i-regional matchings plus several additional control mechanisms. In the full paper (see also [AP89a]) we provide a detailed description of the solution and prove (using Thm. 6.2):

Theorem 7.2 For every graph G and every fixed integer $k \geq 1$ it is possible to construct (in polynomial time) a directory $\mathcal{D}$ that satisfies $\texttt{Stretch}_{find} = O(\log^2 n)$ and $\texttt{Stretch}_{move} = O(\log^2 n)$ and uses a total of $O(N \cdot \log^2 n + n \cdot \log^3 n)$ memory bits for handling N users.

Acknowledgments

We are grateful to Yehuda Afek, Steve Ponzio, Moti Ricklin and an anonymous referee for pointing out some errors in previous versions of the paper. We also thank Michael Fischer for stimulating discussions, and Oded Goldreich and Richard Karp for their helpful comments.

References

[ABLP89] Baruch Awerbuch, Amotz Bar-Noy, Nati Linial, and David Peleg. Compact distributed data structures for adaptive network routing. In *Proc. 21st ACM Symp. on Theory of Computing*, pages 230–240, ACM, May 1989.

[ADDJ90] I. Althöfer, G. Das, D. Dobkin, and D. Joseph. Generating sparse spanners for weighted graphs. In *Proc. 2nd Scandianvian Workshop on Algorithm Theory*, July 1990.

[AGLP89] Baruch Awerbuch, Andrew Goldberg, Michael Luby, and Serge Plotkin. Network decomposition and locality in distributed computation. In *Proc. 30th IEEE Symp. on Foundations of Computer Science*, IEEE, May 1989.

[AKP90] Baruch Awerbuch, Shay Kutten, and David Peleg. On buffer-economical store-and-forward deadlock prevention. March 1990. Unpublished manuscript.

[AP] Baruch Awerbuch and David Peleg. Network synchronization with polylogarithmic overhead. These proceedings.

[AP89a] Baruch Awerbuch and David Peleg. *Online Tracking of mobile users.* Technical Memo TM-410, MIT, Lab. for Computer Science, August 1989.

[AP89b] Baruch Awerbuch and David Peleg. *Routing with Polynomial Communication - Space Trade-Off.* Technical Memo TM-411, MIT, Lab. for Computer Science, September 1989.

[AP90] Baruch Awerbuch and David Peleg. *Efficient Distributed Construction of Sparse Covers.* Technical Report CS90-17, The Weizmann Institute, July 1990.

[AR90] Y. Afek and M. Ricklin. Sparser: a paradigm for running distributed algorithms. April 1990. Unpublished manuscript.

[Awe85] Baruch Awerbuch. Complexity of network synchronization. *J. of the ACM*, 32(4):804–823, October 1985.

[BJ86] Alan E. Baratz and Jeffrey M. Jaffe. Establishing virtual circuits in large computer networks. *Computer Networks*, :27–37, December 1986.

[FJ88] Greg N. Frederickson and Ravi Janardan. Designing networks with compact routing tables. *Algorithmica*, 3:171–190, August 1988.

[KK77] L. Kleinrock and F. Kamoun. Hierarchical routing for large networks; performance evaluation and optimization. *Computer Networks*, 1:155–174, 1977.

[LEH85] K.A. Lantz, J.L. Edighoffer, and B.L. Histon. Towards a universal directory service. In *Proceedings of 4th PODC*, pages 261–271, Calgary, Alberta, Canada, August 1985.

[LS90] N. Linial and M. Saks. Finding low-diameter graph decompositions distributively. April 1990. Unpublished manuscript.

[LT79] Richard J. Lipton and Robert E. Tarjan. A separator theorem for planar graphs. *SIAM J. on Applied Math.*, 36(2):177–189, April 1979.

[MV88] S.J. Mullender and P.M.B. Vitányi. Distributed match-making. *Algorithmica*, 3:367–391, 1988.

[Pel89] D. Peleg. *Sparse Graph Partitions.* Technical Report CS89-01, The Weizmann Institute, February 1989.

[Pel90] D. Peleg. Distance-dependent distributed directories. *Info. and Computation*, 1990. To appear. Also in Tech. Report CS89-10, The Weizmann Institute, May 89.

[Per82] R. Perlman. Hierarchical networks and the subnetwork partition problem. In 5^{th} *Conference on System Sciences*, 1982.

[PS89] David Peleg and Alejandro A. Schäffer. Graph spanners. *J. of Graph Theory*, 13:99–116, 1989.

[PU89a] D. Peleg and E. Upfal. A tradeoff between size and efficiency for routing tables. *J. of the ACM*, 36:510–530, 1989.

[PU89b] David Peleg and Jeffrey D. Ullman. An optimal synchronizer for the hypercube. *SIAM J. on Comput.*, 18(2):740–747, 1989.

[SK85] N. Santoro and R. Khatib. Labelling and implicit routing in networks. *The Computer Journal*, 28:5–8, 1985.

[vLT86] J. van Leeuwen and R.B. Tan. Routing with compact routing tables. In G. Rozenberg and A. Salomaa, editors, *The Book of L*, pages 259–273., Springer-Verlag, New York, New York, 1986.