Fast Network Decomposition

(Extended Abstract)

Baruch Awerbuch * Bonnie Berger † Lenore Cowen ‡ David Peleg §

Abstract

This paper obtains the first deterministic sublinear-time algorithm for *network decomposition.* The second contribution of this paper is an in-depth discussion and survey of all existing definitions of network decomposition. We also present a new reduction that efficiently transforms a weak-diameter version of the problem to a strong one. Thus our algorithm speeds up *all* alternate notions of network decomposition. Most importantly for the network applications, we obtain the first fast algorithm for constructing a sparse neighborhood cover of a network, thereby improving the distributed preprocessing time for all-pairs shortest paths, load balancing, broadcast, and bandwidth management.

*Dept. of Mathematics and Lab. for Computer Science, M.I.T., Cambridge, MA 02139. 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.

†Dept. of Mathematics and Lab. for Computer Science, M.I.T., Cambridge, MA 02139. Supported by an NSF Postdoctoral Research Fellowship.

‡Dept. of Mathematics and Lab. for Computer Science, M.I.T., Cambridge, MA 02139. Supported in part by DARPA contracts N00014-87-K-0825 and N00014-89-J-1988, Air Force Contract OSR-89-02171, Army Contract DAAL-03-86-K-0171 and Navy-ONR Contract N00014-19-J-1698

§Department of Applied Mathematics and Computer Science, The Weizmann Institute, Rehovot 76100, Israel. Supported in part by an Allon Fellowship, by a Bantrell Fellowship and by a Walter and Elise Haas Career Development Award.

PoDC '92-8/92/B.C.
© 1992 ACM 0-89791-496-1/92/0008/0169...$1.50

1 Introduction

Sparse neighborhood covers. This paper is concerned with fast deterministic algorithms for constructing sparse neighborhood covers in the distributed network model. Given an undirected (weighted) graph, a *neighborhood cover* is a collection of sets of nodes (also called *clusters*) which cover the neighborhoods of all nodes in the network. A *high-quality* or *sparse* cover (see Section 2) is one that has an optimal tradeoff between the diameter of each cluster and the cluster overlap at single nodes.

The method of representing networks by *sparse neighborhood covers* has recently been identified [13, 12, 3, 15] as the key to the modular design of efficient network algorithms. Using this method as a basic building block leads to dramatic performance improvements for several fundamental network control problems (such as shortest paths [2], job scheduling and load balancing [10], broadcast and multicast [4], deadlock prevention [9], bandwidth management in high-speed networks [7], and database management [15]), as well as for classical problems in sequential computing (such as finding small edge cuts in planar graphs [19] and approximate all-pairs shortest paths [5]). In most of these applications, sparse neighborhood covers yield the first polylogarithmic-overhead solution to the problem. Thus, in a sense, the impact of efficient sparse neighborhood cover algorithms on distributed computing is analogous to the impact of efficient data structures (like balanced search trees or 2-3 trees) on sequential computation.

Our results. This paper presents the first deterministic sublinear-time distributed algorithm (for static synchronous networks) that constructs a high-quality network decomposition. The algorithm runs in time $O(n^\epsilon)$, for any $\epsilon > 0$, improving on the best-known deterministic running time of $O(n \log n)$ for this problem in [13]. We additionally show how to efficiently transform the weak-diameter version of the problem to a strong-diameter sparse 1-neighborhood cover (see Section 2). We get the analogous improvement for t-neighborhood covers, where all algorithmic running times blow up by a factor of t. The distributed algorithm can be adapted to a more realistic dynamic asynchronous environment using the existing transformer techniques in [1, 13, 11]. The applications that use sparse covers as a data structure often run in time $O(t)$, where $t \ll Diam(G) \leq n$, in which case our improvement is particularly significant.

Our results versus existing work. In addition to obtaining the first deterministic sublinear-time algorithm for high-quality network decomposition, we emphasize that we additionally produce clusters that are more useful for applications. The definition of sparse neighborhood covers considered here is equivalent to that in [13], which employs a *strong* notion of the diameter of a cluster (see Section 2). The definition in [13] is related to yet distinct from the notion of network decomposition defined in [8, 18, 16]. Network decomposition as utilized in [8, 18, 16, 17, 6] employs only a *weak* notion of low diameter. This means that the network decomposition clusters might not even be connected within the clusters. Thus they are not sufficient to support, for example, local routing, where the path between two nodes in the same cluster should consist entirely of nodes within that cluster.

We emphasize that the new fast algorithms in this paper speed up *all* alternative notions of network decomposition, including the sparse neighborhood cover definition needed for all the distributed applications. (See Section 2 for precise definitions.)

Other work on related problems. The (weak diameter, low quality) notion of "network decomposition" was first defined in [8, 18]. Awerbuch et. al. [8] gave a fast algorithm for obtaining $O(n^\epsilon)$-diameter clusters, for any $\epsilon > 0$. Their algorithm requires $O(n^\epsilon)$ time in the distributed case, and $O(nE)$ sequential operations. Unfortunately, the construction of [8] is very inefficient in terms of the quality of the decomposition. Roughly speaking, the inefficiency factor is $O(n^\epsilon)$, and this factor carries over to all but some of the graph-theoretic applications, rendering the decomposition of [8] absolutely unacceptable in any practical context. The construction of [8] is, however, sufficient for the two main applications they site in that paper: the maximal independent set problem and $(\Delta + 1)$ coloring. This is because to construct a MIS or a $(\Delta + 1)$ coloring, one needs to traverse the $O(n^\epsilon)$-diameter clusters only a constant number of times. The network control applications, such as routing, online tracking of mobile users, and all-pairs shortest paths, however, need to traverse the clusters many times. A higher-quality decomposition is needed to avoid a large blowup in the running time for these latter applications.

For the remainder of this paper, when we refer to network decomposition, we mean any of the formulations of high-quality decomposition, and not the large diameter, large number of colors obtained by [8].

Subsequent to our work, Pasconesi and Srinivasan [17] slightly reduced the running time for the poor-quality and weak-diameter construction in [8]. While [8] obtained a running time of $O(n^\epsilon)$, where $\epsilon = O(\sqrt{\log \log n}/\sqrt{\log n})$, [17] reduced ϵ to $\tilde{\epsilon} = O(1/\sqrt{\log n})$. As a consequence of the better running time achieved in [17] (smaller ϵ) and the techniques in this paper, the running time of our algorithm for high-quality network decomposition can be slightly improved (see Corollary 3.4).

The randomized algorithm of Linial and Saks [16] achieves a high-quality decomposi-

tion by introducing randomization. The results are stated in terms of a weak notion of low-diameter clusters, but can be modified to produce a strong diameter decomposition as well, using, for example, the reduction techniques in this paper (or [11]). The resulting algorithm is efficient. (We comment that the distributed algorithm is valid only for static synchronous networks, but the efficient transformer techniques of [11] extends it to a (more realistic) dynamic asynchronous model.) Since we are concerned here with using neighborhood covers as a data structure and running various applications on top, a randomized solution is not acceptable in many cases. We need a fast deterministic algorithm that *guarantees* a good underlying neighborhood cover.

2 Definitions

Notions of network decomposition. We survey the different formulations of network decomposition, and discuss their relations. Within each family of definitions, we also discuss what it means to have a *high-quality* decomposition or cover, in terms of the optimal tradeoffs between low diameter and sparsity. The sparse neighborhood cover formulation is the one that is useful for all the applications. We stress that the algorithms in this paper achieve all alternative notions of network decomposition.

Definition 2.1 Consider a graph G whose vertices appear in sets $S_1, \ldots, S_r$. The *weak* distance between $u, v \in S_i$, denoted $dist_G(u,v)$, is the length of the shortest path between u and v in G. Namely, the path is allowed to shortcut through vertices not in S_i. The *weak diameter* of S_i, $diam(S_i) = \max_{u,v \in S_i}(dist_G(u,v))$

Definition 2.2 Consider a graph G whose vertices appear in sets $S_1, \ldots, S_r$. The *strong* distance between $u, v \in S_i$, denoted $dist_{S_i}(u,v)$, is the length of the shortest path between u and v, on the induced subgraph S_i of G. Namely, all vertices on the path connecting u and v are also in S_i. The *strong diameter* of S_i, $Diam(S_i) = \max_{u,v \in S_i}(dist_{S_i}(u,v))$.

The *square* of a graph, G^2, is defined to be the graph G with additional edges if there exists a w s.t. (u,w) and (w,v) are in G. Similarly, G^t is the graph with an edge between any two vertices that are connected by a path of length $\leq t$ in G. The *j-neighborhood* of a vertex $v \in V$ is defined as $N_j(v) = \{w \mid dist_G(w,v) \leq j\}$. Similarly, the j-neighborhood of a set, V, is defined to be $N_j(V) = \cup_{v \in V} N_j(v)$,

We are now ready to define the alternate notions of network decomposition. First, we give the weak diameter definition.

Definition 2.3 For an undirected graph $G = (V,E)$, a *(χ, d, λ)-decomposition* is defined to be a χ-coloring of the nodes of the graph that satisfies the following properties:

1. each color class is partitioned into an arbitrary number of disjoint *clusters;*
2. the *weak* diameter of any *cluster* of a single color class is at most d.
3. clusters of the same color are at least distance $\lambda+1$ apart.

A *(χ, d, λ)-decomposition* is said to be *high-quality* if when $d = O(k\lambda)$, χ is at most $kn^{1/k}$.

We make several remarks about the Definition 2.3, which is equivalent to the definitions in [8, 16, 17].

- The high-quality decomposition as defined above achieves the optimal tradeoff; there are graphs for which χ must be $\Omega(kn^{1/k})$ to achieve a decomposition into clusters of diameter bounded by $O(k\lambda)$ and separation λ [16].
- When $\lambda = 1$, we will abbreviate this as a (χ, d)-decomposition. Typically, we are most concerned with the case of a high-quality decomposition when χ and d are both $O(\log n)$. This optimal decomposition tradeoff is not achieved in the clusters of [8, 17], but is achieved by randomized methods in [16]. The algorithms in

this paper are the first to achieve the optimal tradeoff deterministically in sublinear time.

- The main application known for this structure in "symmetry-breaking"– it can be used to construct a maximal independent set or a $\Delta+1$ coloring fast in the distributed domain [8, 16, 17]. In this paper, we use the structure for symmetry breaking as follows: the recursive algorithm in Section 3 constructs a (χ, d, λ)-decomposition on the power of the graph inside the recursion first, and later uses this to obtain a *strong-diameter* decomposition.

For *strong-diameter* network decomposition, the definition is the same as Definition 2.3, except in Step 2, substitute strong for weak diameter. As with the weak definition, the "high-quality" tradeoffs are optimal. A *strong-diameter* (χ, d)-decomposition can be thought of as a generalization of the standard graph coloring problem, where χ is the number of colors used, and the clusters are supernodes of diameter d.

We now present the definition for sparse neighborhood covers. Notice that this is a strong diameter definition.

Definition 2.4 A (k,t)-*neighborhood cover* is a collection of sets (also called *clusters*) of nodes $S_1, \ldots, S_r$, with the following properties:

1. $\forall v, \exists i$ s.t. $N_t(v) \subseteq S_i$, where $N_t(v) = \{u | dist_G(u,v) \leq t\}$.
2. $\forall i, Diam(S_i) \leq O(kt)$, where $Diam(S_i) = \max_{u,v \in S_i}(dist_{S_i}(u,v))$.

A (k,t)-neighborhood cover is said to be *sparse*, if each node is in at most $kn^{1/k}$ sets.

Setting $k = 1$, the set of all balls of radius t around each node is a sparse neighborhood cover. Setting $k = Diam(G)/t$, the graph G itself is a sparse neighborhood cover. In the first case, the diameter of a ball is t, but each node could appear in every ball. In the second case, each node appears only in G, but the diameter of G could be as high as n. Setting $k = \log n$ (the typical and useful setting, for all the applications), a sparse $(\log n, t)$-neighborhood cover is a collection of sets S_i with the following properties: the sets contain all t-neighborhoods, the diameter of the sets is bounded by $O(t \log n)$, and each node is contained in at most $c \log n$ sets, where $c > 0$. We remark that this bound is tight to within a constant factor; there exist graphs for which any $(\log n, t)$-neighborhood cover places some node in at least $\Omega(\log n)$ sets [16]. When $k = \log n$, we find that sparse neighborhood covers form a useful data structure to *locally* represent the t-neighborhoods of a graph.

Our new fast distributed algorithm achieves deterministically a structure which is simultaneously a (strong, and therefore also weak) diameter decomposition *and* a sparse neighborhood cover.

3 Weak Diameter Network Decomposition

In this section, we introduce the new distributed algorithm `Color`, which recursively builds up a $(kn^{1/k}, 2k, 1)$-decomposition. It calls on a procedure, `Create_New_Color`, which runs a modified version of the Awerbuch-Peleg [14] greedy algorithm on separate clusters.

Note that all distances in the discussion below, including those in the same cluster, are assumed to be *weak* distances, and the diameter of the clusters is always in terms of weak diameter (see Section 2).

`Color` is implicitly taking higher and higher powers of the graph, where recall that we define the graph G^t to be the graph in which an edge is added between any pair of nodes that have a path of length $\leq t$ in G. Notice that to implement the graph G^t in a distributed network G, since the only edges in the network are still the edges in the underlying graph G, to look at all our neighbors in the graph G^t, we might have to traverse paths of length t. Therefore the time

for running an algorithm on the graph G^t, blows up by a factor of t. The crucial observation is that a $(\chi, d, 1)$-decomposition on G^t is a (χ, dt, t)-decomposition on G. Choosing t well at the top level of the recursion, guarantees that nodes in different clusters of the same color are always separated by at least twice the maximum possible distance of their radii. We can thus use procedure `Greedy_Create_Color` to in parallel *recolor* these separate clusters without collisions. (The leader of each cluster does all the computation for its cluster.)

The recursive algorithm has two parts:

1. Find a (χ, dt, t)-decomposition, where $\chi = xkn^{1/k}$, $d, t = 2k$, on each of x disjoint subgraphs.
2. Merge these together by recoloring, as just described, to get a $(kn^{1/k}, 2k, 1)$-decomposition.

Algorithm: `Color`(G)

Input: graph $G = (V, E)$, $|V| = n$, and integer $k \geq 1$.
Output: A $(kn^{1/k}, 2k, 1)$-decomposition of G.

1. Compute G^{2k}.
2. If G has less than x nodes, run the Linial-Saks [16] or Awerbuch-Peleg [14] simple greedy algorithm on G^{2k}, and go to step 6.
3. Partition nodes of G into x subsets, $V_1, \ldots, V_x$ (based on the last $\log x$ bits of node IDs, which are then discarded).
4. Define G_i to be the subgraph of G^{2k} induced on V_i.
5. In parallel, for i, `Color`(G_i).
   (every node of G is now colored recursively)
6. For each $v \in V$, color v with the color $<i, \text{color}(v) \in G_i>$.
   (this gives an $xkn^{1/k}$ coloring of G with separation $2k$)
7. Do sequentially, for $i = 1$ to $kn^{1/k}$, `Create_New_Color`(G, i)
   (this gives a $kn^{1/k}$ coloring of G with separation 1)

Algorithm: `Create_New_Color`(G, i)
(this colors a constant fraction of the old-colored nodes remaining with new color i)

Input: graph G with new and old colored nodes such that there is a $(xkn^{1/k}, (2k)^2, 2k)$-decomposition on the old-colored nodes of G and a $(i-1, 2k, 1)$-decomposition on the new-colored nodes of G
Output: graph G with new and old colored nodes such that there is a $(xkn^{1/k}, (2k)^2, 2k)$-decomposition on the old-colored nodes of G and a $(i, 2k, 1)$-decomposition on the new-colored nodes of G

1. $W \leftarrow V$.
2. Do sequentially, for $j = 1$ to $xkn^{1/k}$, "Look at nodes with old color j":
   (a) Do in parallel for color j clusters,
   - Elect a leader for each cluster.
   - The leader learns the identities, U, of all the nodes in W within k distance from the border of its cluster (i.e. this is graph G for that cluster).
   - The leader calls procedure `Greedy_Create_Color`(R, U), where R is the set of old-colored j nodes in both the leader's cluster and in W.
   - `Greedy_Create_Color` returns (DR, DU). The leader colors the nodes in DR with new color i, and sets $W \leftarrow W - DU$.

`Greedy_Create_Color` is the procedure of the Awerbuch-Peleg [14] greedy algorithm that determines what nodes will be given the current new color. The algorithm identifies a constant fraction of the nodes in the cluster R to be colored. The algorithm picks an arbitrary node in R (call it a *center* node) and greedily grows a ball around it of minimum

radius r, such that a constant fraction of the nodes in the ball lie in the interior (i.e. are in the ball of radius $r-1$ around the center node). It is easy to prove that there always exists an $r \leq k|R|^{1/k}$ for which this condition holds. Note that although the centers of the balls grown out are always picked (arbitrarily) from the nodes in R, the interiors and borders of the balls which are then claimed, include any of the nodes in U (not just those in R) within the ball. Then another arbitrary node is picked, and the same thing is done, until all nodes in R have been processed. Procedure `Create_New_Color` will then color the interiors of the balls (set DR) with new color i, and remove each entire ball from the working graph W.

Algorithm: `Greedy_Create_Color`(R, U)

Input: sets of nodes R and U, where R is the set of nodes in the cluster and U is a superset of nodes that contains R.

Output: (DR, DU). This returns a constant fraction of the nodes in R in set DR and the 1-neighborhoods of the clusters of DR in set DU.

1. $DR \leftarrow \emptyset$; $DU \leftarrow \emptyset$.
2. While $R \neq \emptyset$ do
   (a) $S \leftarrow \{v\}$ for some $v \in R$.
   (b) While $|N_1(S) \cap U| > |R|^{1/k}|S|$ do $S \leftarrow S \cup (N_1(S) \cap U)$.
   (c) $DR \leftarrow DR \cup S$.
   (d) $DU \leftarrow DU \cup (N_1(S) \cap U)$.
   (e) $R \leftarrow R - S - (N_1(S) \cap R)$.
   (f) $U \leftarrow U - S$.

Lemma 3.1 If $x = 2^{\sqrt{\log n}\sqrt{1+\log k}}$, the running time of the procedure `Color` is $n^{2\sqrt{1+\log k}/\sqrt{\log n}+2/k}(2k)^2$.

Proof The branching phase of the recursion takes time $T'(n) \leq 2kT'(n/x)+x$. The merge takes time $\chi kn^{1/k}td = x(kn^{1/k})^2(2k)^2$, where $\chi kn^{1/k}$ is the number of iterations overall and td is the number of steps per iteration. Overall, we have

$$\begin{aligned} T(n) &\leq 2kT(n/x) + x(kn^{1/k})^2(2k)^2 \\ &\leq (2k)^{\log n/\log x} x(kn^{1/k})^2(2k)^2 \\ &\leq n^{2\sqrt{1+\log k}/\sqrt{\log n}+2/k}(2k)^2, \end{aligned}$$

when $x = 2^{\sqrt{\log n}\sqrt{1+\log k}}$. □

Theorem 3.2 There is a deterministic distributed asynchronous algorithm which given a graph $G = (V, E)$, finds a $(kn^{1/k}, 2k, 1)$-decomposition of G in $n^{2\sqrt{1+\log k}/\sqrt{\log n}+2/k}(2k)^2$ time.

Corollary 3.3 There is a deterministic distributed asynchronous algorithm which given $G = (V, E)$, finds a $(O(\log n), O(\log n), 1)$-decomposition of G in $n^{O(\sqrt{\log\log n}/\sqrt{\log n})}$ time, which is n^{ϵ} for any $\epsilon > 0$. We remark that the constant on the big-oh in the running time is 3.

As a corollary to our theorem and [17], we can obtain a slightly better running time.

Corollary 3.4 There is a deterministic distributed asynchronous algorithm which given $G = (V, E)$, finds a $(O(\log n), O(\log n), 1)$-decomposition of G in $O(n^{1/\sqrt{\log n}})$ time, which is n^{ϵ} for any $\epsilon > 0$.

4 Strong Diameter Network Decomposition

The algorithm in the previous section produced a weak-diameter network decomposition. While this is a nice problem, the strong-diameter form is the one we want in order to successfully run most distributed applications. In this section, we give a reduction that given a weak diameter decomposition, constructs a structure that is simultaneously both a strong diameter decomposition and a sparse neighborhood cover. The algorithm as written outputs the cover: the associated strong decomposition consists of the interiors of the clusters in the sparse neighborhood cover.

We introduce an algorithm **Sparse**, which takes as input a procedure **Decomp**, which given a graph $G = (V, E)$, finds a $(kn^{1/k}, 2k, 1)$-decomposition of G. In actuality, we will bind **Decomp** to procedure **Color** of Section 3. **Sparse** first calls procedure **Decomp** with G^{8kt}. Of course, this will yield an $O(kt)$ blowup in the running time of **Decomp**, say τ.

Once **Decomp** is called, the remaining running time for **Sparse** is $O(k^2 n^{2/k})$, times a t blowup for traversing t-neighborhoods. Then, in sum, **Sparse** is able to obtain a t-neighborhood cover in the original graph G in time $O(kt\tau + k^2 t n^{2/k})$. Recall that k is typically $\log n$.

Notice that the code for **Sparse** is similar to the last pass of procedure **Color** (Section 3); however, **Sparse** has an additional level of complexity. To obtain a t-neighborhood cover, we must modify the Awerbuch-Peleg [13] coarsening algorithm, called as a subroutine, so that we can recolor clusters in parallel without interference.

Notation. In the algorithms below, we use roman capital letters for names of sets, and calligraphic letters for names of collections of sets. In particular, corresponding to a set W, by convention we will denote by $\mathcal{W}$ the collection consisting of the sets $\{N_t(v) | v \in W\}$.

Algorithm: **Sparse**(G,**Decomp**)

Input: graph $G = (V, E)$, $|V| = n$, and integer $k \geq 1$, and a procedure **Decomp**, that finds a $(kn^{1/k}, 2k, 1)$-decomposition of G.
Output: $\mathcal{T}$, a sparse (k, t)-neighborhood cover of G.

1. **Decomp**(G^{8kt}).
   (*returns a $(kn^{1/k}, 2k, 1)$-decomposition of G^{8kt} which is a $(kn^{1/k}, 16k^2 t, 8kt)$-decomposition of G.*)
   (a) $\mathcal{T} \leftarrow \emptyset$.
   (*$\mathcal{T}$ is the cover.*)
   (b) Do sequentially, for $i = 1$ to $kn^{1/k}$,
   (*find a $kn^{1/k}$-degree t-neighborhood cover of G.*)
   i. $\mathcal{U} \leftarrow \{N_t(v) | v \in V\}$.
   (*$\mathcal{U}$ is the collection of all unprocessed t-neighborhoods.*)
   ii. Do sequentially, for $j = 1$ to $kn^{1/k}$,
   "Look at nodes with old color j":
   A. Do in parallel for color j clusters,
   - Elect a leader for each cluster.
   - The leader learns the identities, of all the t-neighborhoods of nodes within a $4kt$ distance from the border of its cluster.
   - The leader calls procedure **Cover**($\mathcal{R}$,$\mathcal{U}$) on G, where $\mathcal{R}$ is the collection of t-neighborhoods of old-colored j nodes in both the leader's cluster and in U.
   - **Cover** returns $(\mathcal{DR}, \mathcal{DU})$. The leader colors the nodes in $\mathcal{DR}$ with new color i, and sets $\mathcal{U} \leftarrow \mathcal{U} - \mathcal{DU}$.
   iii. $\mathcal{T} \leftarrow \mathcal{T} \cup \mathcal{DR}$

Cover is our modification of the Awerbuch-Peleg [13] coarsening algorithm that determines what nodes will be given the current new color. The actual code for this procedure follows a description of the algorithm below. The key to our fast simulation of their coarsening algorithm, is that we keep track of neighborhoods within and outside of the old-colored j clusters separately, in order to recolor clusters in parallel without collisions.

Procedure **Cover**($\mathcal{R}$,$\mathcal{U}$) 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 $\mathcal{U} \cap \mathcal{R}$ 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 $\mathcal{U} \cap \mathcal{R}$ is exhausted. The procedure then outputs the sets $\mathcal{DR}$ and $\mathcal{DT}$.

Procedure Cover is formally described in Figure 1. Its properties are summarized by the following lemma. We comment that our modifications do not change the lemma.

Lemma 4.1 ([13]) 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($\mathcal{R}$,$\mathcal{U}$) operates in iterations. satisfy the following properties:

(1) All clusters in $\mathcal{DR}$ have their t-neighborhood contained in some cluster in $\mathcal{DT}$.

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

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

(4) $\max_{T \in \mathcal{DT}} Diam(T) \leq (2k-1) \max_{R \in \mathcal{R}} Diam(R)$.

```
DT ← ∅; DR ← ∅
repeat
   Select an arbitrary cluster S ∈ U ∩ R.
   Z ← {S}
   repeat
      Y ← Z
      Y ← ⋃_{S∈Y} S
      Z ← {S | S ∈ U, S ∩ Y ≠ ∅}.
   until |Z| ≤ |R|^{1/k} |Y|
   U ← U − Z
   DT ← DT ∪ {Y}
   DR ← DR ∪ Y
until U ∩ R = ∅
Output (DR, DT).
```

Figure 1: Procedure Cover($\mathcal{R}$,$\mathcal{U}$).

Theorem 4.2 There is a deterministic distributed algorithm, e.g. Sparse(G,Color), that given a graph $G = (V, E)$, $|V| = n$, and integers $k, t \geq 1$, constructs a t-neighborhood cover of G in $k^2 t n^{O(\sqrt{\log k}/\sqrt{\log n})}$ time in the asynchronous model, where each node is in at most $O(kn^{1/k})$ clusters, and the maximum cluster diameter is $O(kt)$.

Finally, we remark that if we color only the *interiors* of the new color i clusters, the above construction produces a strong diameter high-quality network decomposition from a weak diameter high-quality network decomposition, as well as a sparse neighborhood cover. This is because our construction is such that each node in the cover lies in precisely one new-colored interior.

5 Acknowledgment

Thanks to Tom Leighton for helpful discussions.

References

[1] Y. Afek, B. Awerbuch, and E. Gafni. Applying static network protocols to dynamic networks. In *Proc. 28th IEEE Symp. on Foundations of Computer Science*, pages 358–370, Oct. 1987.

[2] Y. Afek and M. Ricklin. Sparser: A paradigm for running distributed algorithms. Unpublished manuscript, 1991.

[3] Y. Afek and M. Riklin. Sparser: A paradigm for running distributed algorithms. *J. of Algorithms*, 1991. Accepted for publication.

[4] B. Awerbuch, A. Baratz, and D. Peleg. Efficient broadcast and light-weight spanners. Unpublished manuscript, Nov. 1991.

[5] B. Awerbuch, B. Berger, L. Cowen, and D. Peleg. Fast deterministic cover algorthms. Unpublished manuscript, Nov. 1991.

[6] B. Awerbuch, B. Berger, L. Cowen, and D. Peleg. Low-diamter graph decomposition is in NC. In *Proc. 3'rd Scandinavian Workshop on Algorithm Theory*, July 1992. to appear.

[7] B. Awerbuch, I. Cidon, I. Gopal, M. Kaplan, and S. Kutten. Distributed control for paris. In *Proc. 9th ACM Symp. on Principles of Distributed Computing*, pages 145–160, 1990.

[8] B. Awerbuch, A. Goldberg, M. Luby, and S. Plotkin. Network decomposition and locality in distributed computation. In *Proc. 30th IEEE Symp. on Foundations of Computer Science*, May 1989.

[9] B. Awerbuch, S. Kutten, and D. Peleg. On buffer-economical store-and-forward deadlock prevention. In *Proc. of the 1991 INFOCOM*, 1991.

[10] B. Awerbuch, S. Kutten, and D. Peleg. Online load balancing in a distributed network. In *Proc. 24th ACM Symp. on Theory of Computing*, pages 571–580, 1992.

[11] B. Awerbuch, B. Patt, D. Peleg, and M. Saks. Adapting to asynchronous dynamic networks with polylogarithmic overhead. In *Proc. 24th ACM Symp. on Theory of Computing*, pages 557–570, 1992.

[12] B. Awerbuch and D. Peleg. Network synchronization with polylogarithmic overhead. In *Proc. 31st IEEE Symp. on Foundations of Computer Science*, pages 514–522, 1990.

[13] B. Awerbuch and D. Peleg. Sparse partitions. In *Proc. 31st IEEE Symp. on Foundations of Computer Science*, pages 503–513, 1990.

[14] B. Awerbuch and D. Peleg. Routing with polynomial communication-space tradeoff. *SIAM J. Disc. Math*, 5(2):151–162, 1992.

[15] Y. Bartal, A. Fiat, and Y. Rabani. Competitive algorithms for distributed data management. In *Proc. 24th ACM Symp. on Theory of Computing*, pages 39–50, 1992.

[16] N. Linial and M. Saks. Decomposing graphs into regions of small diameter. In *Proc. 2nd ACM-SIAM Symp. on Discrete Algorithms*, pages 320–330. ACM/SIAM, Jan. 1991.

[17] A. Pasconesi and A. Srinivasan. Improved algorithms for network decompositions. In *Proc. 24th ACM Symp. on Theory of Computing*, pages 581–592, 1992.

[18] D. Peleg. Distance-preserving distributed directories and efficient routing schemes. unpublished manuscript, 1989.

[19] S. Rao. Finding small edge cuts in planar graphs. In *Proc. 24th ACM Symp. on Theory of Computing*, pages 229–240, 1992.