CHAPTER

EIGHTEEN

SPARSE NETWORK COVERS

Most of Parts II and III concentrated on issues of solvability and ignored issues of complexity; in this final chapter, we return to complexity issues in message passing systems.

We discuss *sparse network covers*, a graph theoretic concept with several applications in distributed algorithms. Very roughly, given a collection of subgraphs that cover all the nodes of the graphs, we conduct computations inside each subgraph, and between neighboring subgraphs. Loosely speaking, a cover is *sparse* if each node is covered by few subgraphs; a sparse network cover induces communication structures that can be tuned to save significantly on message and time complexity as well as on local memory requirements.

In this chapter, we discuss the mathematical background of this method and present two of its applications: for routing messages, and to obtain efficient synchronizers. Other applications are mentioned in the chapter notes.

Throughout this chapter, we consider asynchronous message passing systems, with arbitrary network topology.

18.1 Sparse Network Covers

We start by recalling some graph theoretic notions and terminology, and then defining sparse network covers; finally, we present a *sequential* algorithm for finding sparse network covers. Since sparse network covers are typically constructed during a pre-processing stage, the sequential algorithm can be applied in a central node, once it has collected all the topological information; other possibilities are discussed below.

Consider an undirected graph $G(V, E)$ describing the underlying topology of a message passing system. As before, we denote $n = |V|$, the number of nodes

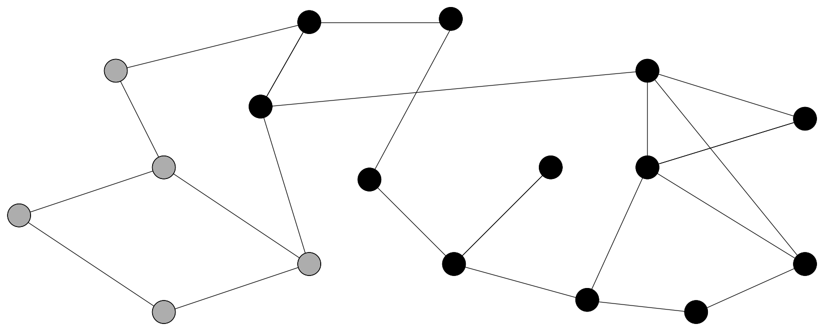

Figure 18.1: The nodes of a cluster S are drawn in grey.

(processors), and $m = |E|$, the number of edges (communication links). We assume G is connected and finite.

For any pair of nodes v and u in V, dist(v, u) denotes the *distance* between v and u, that is, the number of edges on the shortest path between them in G. The *diameter* of the graph is the maximum distance between any two nodes, that is, diam$(G) = \max_{v,u \in V}$ dist(v, u). Since G is connected, dist(v, u) is finite for every pair of nodes v and u; since G is finite, diam(G) is finite as well.

Consider a set of nodes S, and the subgraph it induces, i.e., the edges of G with both endpoints in S. S is a *cluster* in G if its nodes are connected to each other in this subgraph. The *size* of a cluster S, denoted $|S|$, is the number of nodes in S. A cluster induces a connected subgraph of G; the diameter of a cluster S, diam(S), is calculated in the connected subgraph induced by S. Figure 18.1 shows a cluster S; $|S| = 5$, and diam$(S) = 3$.

Consider some collection of clusters, $\mathcal{C} = \{S_1, \ldots, S_k\}$. The *diameter* of $\mathcal{C}$, denoted diam$(\mathcal{C})$, is the maximal diameter of a cluster in $\mathcal{C}$, that is, $\max\{$diam$(S_1), \ldots,$ diam$(S_k)\}$. The *volume* of $\mathcal{C}$, denoted vol$(\mathcal{C})$, is the sum of the sizes of its clusters, that is, $\sum_{i=1}^{k} |S_i|$. If a node appears in two clusters then it is counted twice in this summation, and contributes two to the total sum.

$\mathcal{C}$ is a *cover* of G if the union of its clusters contains all the nodes of G, that is, $\cup_{i=1}^{k} S_i = V$; a cover is also called a *network cover*. If $\mathcal{C}$ covers G then vol$(\mathcal{C}) = \sum_{i=1}^{k} |S_i| \geq n$, since $\cup_{i=1}^{k} S_i = V$. The relation between the volume of a cover and n counts the number of 'repetitions' in $\mathcal{C}$. Specifically, vol$(\mathcal{C})/n$ is the average number of occurrences of any node in all the clusters of $\mathcal{C}$. Figure 18.2 shows a cover with three clusters; the volume of this cover is 20 (while the number of nodes in the graph is only 17).

As we shall see in the examples below, the cost incurred by algorithms using network covers, for example, the number of messages or the local memory require-

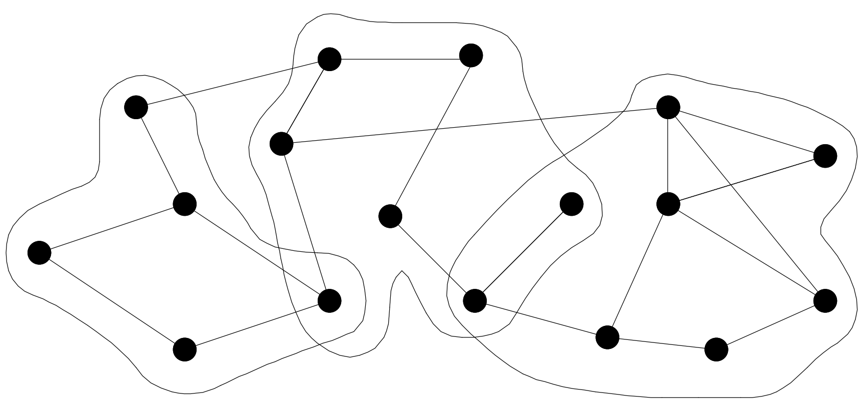

Figure 18.2: A cover.

ments, is proportional to the number of clusters to which each node belongs. Thus, the total cost of an algorithm using a specific network cover is often proportional to the volume of the cover.

Therefore, our objective is to find a cover with low volume and diameter, while ensuring that the cover has specific properties, e.g., that each node is in a cluster with all its neighbors. A useful method to achieve this goal is to start with a cover having the desired properties, regardless of its volume and diameter, and then 'coarsen' it by combining clusters, thus reducing its volume. The key issue is how to do so without increasing the diameter too much.

More specifically, a cover is coarser than another cover if its clusters contain the clusters of the other cover. Formally, given two collections of clusters, $\mathcal{C} = \{S_1, \ldots, S_h\}$ and $\mathcal{C}' = \{S'_1, \ldots, S'_r\}$, $\mathcal{C}'$ is a *coarsening* of $\mathcal{C}$ if for every $S_i \in \mathcal{C}$ there exists $S'_j \in \mathcal{C}'$ such that $S_i \subseteq S'_j$. A single cluster of $\mathcal{C}'$ may contain all clusters of $\mathcal{C}$.

Given a cover $\mathcal{C}$ of G we would like to obtain a cover of G with a smaller volume, in order to reduce the average number of clusters a node belongs to. A simple way to do so is to coarsen $\mathcal{C}$ by merging clusters; however, this tends to increase the diameter of the cover.

We next show how to trade off the volume of the cover with its diameter, by presenting a sequential algorithm that takes as input a cover $\mathcal{C}$ and generates a coarsening $\mathcal{C}'$, such that for some integer parameter k, vol$(\mathcal{C}') \leq n^{1+\frac{1}{k}}$, and diam$(\mathcal{C}') \leq (k+1) \cdot$ diam$(\mathcal{C})$.

To understand the trade-offs, consider a few typical choices for k; let d denote diam$(\mathcal{C})$. For $k = 1$, vol$(\mathcal{C}') \leq n^2$ while diam$(\mathcal{C}') \leq 2d$; for $k = 2$,

Algorithm 18.1 A sequential algorithm for coarsening covers; input is cover $\mathcal{C}$ and integer k, output is cover $\mathcal{C}'$.

```
1:  $\mathcal{C}' := \emptyset$
2:  while $\mathcal{C} \neq \emptyset$ do                         // create a new cluster for $\mathcal{C}'$
3:      pick some cluster $S$ from $\mathcal{C}$         // an arbitrary cluster not yet covered
4:      remove $S$ from $\mathcal{C}$
5:      $T := S$
6:      repeat                          // increase $T$ as long as the gain is significant
7:          $prevT := T$
8:          $\mathcal{Q} := \{S \mid S \in \mathcal{C}$ and $S \cap T \neq \emptyset\}$   // all clusters in $\mathcal{C}$ intersecting $T$
9:          $T := T \cup \bigcup_{S \in \mathcal{Q}} S$                              // merge into $T$
10:         $\mathcal{C} := \mathcal{C} - \mathcal{Q}$                  // remove the clusters in $\mathcal{Q}$ from $\mathcal{C}$
11:     until $|T| \leq n^{\frac{1}{k}} \cdot |prevT|$ or $|T| \geq n$
12:     add $T$ to $\mathcal{C}'$
```

vol$(\mathcal{C}') \leq n^{3/2}$ while diam$(\mathcal{C}') \leq 3d$; for $k = \log n$, vol$(\mathcal{C}') \leq n^{1+1/\log n}$ while diam$(\mathcal{C}') \leq (\log n + 1)d$. As k increases, the volume decreases (with a limit of n) and the diameter increases.

The pseudocode appears in Algorithm 18.1. The algorithm proceeds in iterations; in each iteration (the *while* loop), one cluster S from $\mathcal{C}$ is selected and is merged with other clusters from $\mathcal{C}$ to form a new cluster T that is added to $\mathcal{C}'$. T itself is created in iterations (the *repeat* loop); in each iteration, the current cluster is merged with all clusters in $\mathcal{C}$ that intersect it, until the increase in size is too small. This happens when T increases by no more than a factor of $n^{\frac{1}{k}}$, as captured by the condition in Line 11.

Theorem 18.1 *For any graph $G(V, E)$, a cover $\mathcal{C}$ of G, and an integer $k \geq 1$, Algorithm 18.1 creates a coarsening $\mathcal{C}'$ of $\mathcal{C}$ such that*

- *vol$(\mathcal{C}') \leq n^{1+\frac{1}{k}}$, and*
- *diam$(\mathcal{C}') \leq (k+1) \cdot$ diam$(\mathcal{C})$.*

Proof. Clearly, the resulting collection of clusters, $\mathcal{C}'$, is a coarsening of $\mathcal{C}$ and therefore, also a cover of G. We now analyze the volume and the diameter of $\mathcal{C}'$.

For any cluster $T \in \mathcal{C}'$, let *before*(T) be the value of *prevT* when T is finalized (i.e., before Line 11 is executed for the last time). By the condition evaluated at the end of the repeat loop (Line 11), $|T| \leq n^{\frac{1}{k}}|before(T)|$.

Moreover, we can show that, for every T and T' in $\mathcal{C}'$, *before*$(T) \cap$ *before*$(T') = \emptyset$. Assume T is added to $\mathcal{C}'$ before T'; in the last iteration of the *repeat* loop in which T is constructed, all the clusters intersecting *before*(T) are removed from $\mathcal{C}$ (Lines 8 and 10). Therefore, *before*(T') does not contain clusters that intersect *before*(T). Therefore,

$$\begin{aligned}
\text{vol}(\mathcal{C}') = \sum_{T\in\mathcal{C}'} |T| &\le \sum_{T\in\mathcal{C}'} n^{\frac{1}{k}} |\textit{before}(T)| && \text{(by the condition at Line 11)} \\
&\le n^{\frac{1}{k}} \sum_{T\in\mathcal{C}'} |\textit{before}(T)| \\
&\le n^{\frac{1}{k}} n && \text{(since the sets are disjoint)} \\
&= n^{1+\frac{1}{k}} .
\end{aligned}$$

This bounds the volume of $\mathcal{C}'$.

We now bound the diameter. By Line 8, the clusters of $\mathcal{C}$ added to T in an iteration of the repeat loop must intersect at least one of the clusters of $\mathcal{C}$ already in T; thus, the diameter of $\mathcal{C}'$ is bounded by the number of times of the repeat loop is executed times the diameter of $\mathcal{C}$. Therefore, it suffices to show that each cluster T is constructed within at most $k+1$ iterations of the repeat loop, in order to bound the diameter of $\mathcal{C}'$.

To bound the number of iterations of the repeat loop, we prove that at the beginning of the ith iteration of the repeat loop, $|T| \ge n^{\frac{i-1}{k}}$. Therefore, after $k+1$ iterations of the repeat loop, the size of T is at least n^1, and the algorithm terminates.

The proof is by induction on i. The base case, $i = 1$, is immediate since at the beginning of the first iteration T is not empty and thus, $|T| \ge 1$. For the inductive step, assume that at the beginning of the $(i-1)$th iteration, $|T| \ge n^{\frac{i-2}{k}}$. Since the condition of the repeat loop (Line 11) is satisfied, the size of T is multiplied by (at least) $n^{\frac{1}{k}}$, and thus, $|T| \ge n^{\frac{i-1}{k}}$ at the beginning of the ith iteration. $\square$

When sparse network covers are used, coarsening is typically used as a pre-processing stage, in which a cover with certain properties and relatively low diameter and volume is constructed. Since Algorithm 18.1 is sequential, all information has to be collected to a specific node to apply it in a distributed system. Alternatively, the pre-processing stage can be done by running a distributed coarsening algorithm; several such algorithms are mentioned in the notes at the end of the chapter.

18.2 Routing with Low Memory Overhead

The first application of sparse network covers is to the problem of routing messages in a communication network.

18.2.1 The Problem

Assume some processor p_i wishes to send a message to another processor p_j. This is done along some *route*, which is a sequence of adjacent communication links in the network, over which the message is forwarded. A *routing algorithm* specifies the route by telling each intermediate node on the route on which outgoing edge the message should be sent, depending on the destination.

We are interested in the number of messages that are required for transferring the message, typically dominated by the length of the route, and the amount of memory dedicated in each processor to the routing information. There is a trade-off between these two measures, as demonstrated by the following two simple solutions.

One solution is to route messages on the shortest path between the originator of the message and its destination. Specifically, each processor p_i maintains a *routing table* which contains the identity of p_i's neighbor that is on the shortest path to p_j, for every processor p_j. When p_i has to send (or forward) a message addressed to p_j, it uses the table to find on which link to send the message. Clearly, this guarantees optimal routes. However, it has high memory requirements since the routing table maintained by each processor requires $O(n \log n)$ bits, yielding a total of $O(n^2 \log n)$ bits in the whole network.

Another solution is to send the message by *flooding*, employing Algorithm 2.2, for example. That is, if p_i wants to send a message M to p_j it sends M to all its neighbors, which forward it to all their neighbors, etc. If p_i and p_j are connected, then the message will eventually arrive at p_j. This solution requires no memory, but it has high message complexity, as $O(m)$ messages are sent even if the source and the destination are very close.

Below, we describe a scheme which yields a better trade-off between memory and messages, using sparse network covers. In order to describe it, we need to define our cost measure more precisely.

For a fixed network G and routing algorithm A, let $\text{cost}_A(v, u)$ be the number of messages sent by A in order to deliver a message from v to u. The best we could hope for is that $\text{cost}_A(v, u) = \text{dist}(v, u)$. Therefore, a good measure for the quality of the routing performed by the algorithm is the *stretch factor* of A, defined as:

$$\max_{v,u} \frac{\text{cost}_A(v, u)}{\text{dist}(v, u)} ,$$

that is, the worst-case ratio between $\text{cost}_A(u, v)$ and $\text{dist}(u, v)$.

A small stretch factor implies that the algorithm uses routes that are not much longer than the shortest paths between nodes. For example, the stretch factor of the algorithm using routing tables is 1, while the stretch factor of the algorithm using flooding is m.

We next present a routing algorithm with stretch factor $O(k)$, which uses $O(n^{1+\frac{1}{k}} \log^2 n)$ total memory bits, for any value of a trade-off parameter $k \geq 1$.

This algorithm provides a trade-off: a reduction in the stretch factor at the cost of an increase in the memory requirements, and vice versa. In order to get a constant stretch factor, k should be $O(1)$ implying that $O(n^2 \log^2 n)$ total memory bits are required. This is a factor of $O(\log n)$ bits larger than the memory bound given by the algorithm using routing tables.

18.2.2 Overview of the Routing Algorithm

Intuitively, the algorithm maintains detailed information on close-by nodes and less information on nodes that are far away. When the distance between sender and receiver is large, even an algorithm using long routes, like the flooding algorithm, has a good stretch factor. This is achieved by a hierarchy of routing algorithms, each restricted to a region of a certain diameter.

Specifically, a *regional (s, d)-routing algorithm* provides $O(s)$ stretch factor inside regions of diameter d. That is, for any pair of nodes v and u, any message sent from v to u is delivered by the regional (s, d)-routing algorithm if $\text{dist}(v, u) \leq d$; otherwise, v is notified; in both cases, $O(s \cdot d)$ messages are sent.

Let R_i be the regional $(s, 2^i)$-routing algorithm; we use a hierarchy of routing algorithms, $R_1, \ldots, R_{\lceil \log D \rceil}$, concurrently, where $D = \text{diam}(G)$.

In more detail, $\lceil \log D \rceil$ regional routing algorithms, for regions of diameter $d = 2, 4, \ldots, 2^{\lceil \log D \rceil}$, are running at the same time in the network, all with $O(s)$ stretch factor. To send a message from v to u, routing is attempted in R_1; if routing is unsuccessful, then routing is attempted in R_2, and so on, proceeding to regional routing algorithms with higher indices $(R_3, \ldots)$ until the message is delivered. The properties of the regional routing algorithms imply that routing succeeds in $R_{\lceil \log d \rceil}$, at the latest, where $d = \text{dist}(v, u)$. This implies that each message is eventually delivered and also implies that the total number of messages sent is at most

$$\sum_{i=0}^{\lceil \log d \rceil} s \cdot 2^i = s \sum_{i=0}^{\lceil \log d \rceil} 2^i = O(s \cdot d) .$$

That is, the stretch factor of the general algorithm is $O(s)$.

18.2.3 The Regional Routing Algorithm

We now describe how sparse network covers are used in a regional (s, d)-routing algorithm to yield $O(s)$ stretch factor, while using a total of $O(n^{1+\frac{1}{s}} \log n)$ memory bits.

Recall that the *d-neighborhood* of a node v is the set of processors whose distance from v is at most d; the d-neighborhood of an arbitrary node is clearly a cluster.

Let $\mathcal{D}$ be the set containing the d-neighborhoods of all nodes in the graph. $\mathcal{D}$ is clearly a cover of the graph, $|\mathcal{D}| = n$ and $\text{diam}(\mathcal{D}) \leq 2d$. By applying

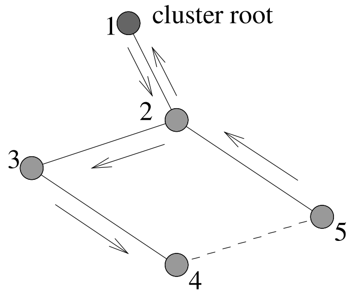

Figure 18.3: DFS numbering of a cluster; the edges of the BFS tree of S are solid.

Theorem 18.1 to $\mathcal{D}$ with the parameter $k = s$, we obtain a cover $\mathcal{D}'$ which is a coarsening of $\mathcal{D}$, diam$(\mathcal{D}') < (s+1) \cdot 2d$, and vol$(\mathcal{D}') \leq n^{1+\frac{1}{s}}$.

For each node v, there is a cluster in $\mathcal{D}'$ which contains its d-neighborhood, since $\mathcal{D}'$ is a coarsening of $\mathcal{D}$. In $\mathcal{D}'$, there may be several clusters containing the d-neighborhood of v; we designate one of them as the *home cluster* of v.

In each cluster[1] in $\mathcal{D}'$ we choose some node as a root, and construct from it a *cluster tree*, which is the shortest paths tree (BFS tree) from the root to all nodes in the cluster.

Inside the cluster, routing is done based on the depth first search (DFS) numbering of the cluster tree, as follows.

The nodes of each cluster are numbered by running DFS on the cluster tree (see Algorithm 2.3). Figure 18.3 presents the DFS numbering of a tree in the cluster S from Figure 18.1. Denote the DFS number of a node v by *dfs*(v). Every node records its own DFS number, as well as the DFS numbers of its children in the cluster tree. The root maintains a table with the DFS number of each node v in the cluster.

Routing is done through the root of the cluster. To send a message to another node u, a node v sends the message up to the cluster root. If u is not in the cluster, then the root has no DFS number for it, and it notifies v; otherwise, the root sends the message to u by propagating it down the tree. At node w with children $w_1, \ldots, w_l$ (ordered by increasing DFS numbers) the message is sent to the child w_i such that *dfs*$(w_i) \leq$ *dfs*$(u) <$ *dfs*(w_{i+1}). (Let *dfs*$(w_0) = -\infty$ and *dfs*$(w_{l+1}) = \infty$.) Exercise 18.8 asks you to write pseudocode for this algorithm.

For example, the arrows in Figure 18.3 show the routing from the node whose DFS number is 5, through the cluster root, to the node whose DFS number is 4.

Clearly, at most $2 \cdot$ diam$(\mathcal{D}')$ messages are sent, since at the worst case, the longest path in a cluster tree is used twice. This implies that this is a regional

[1]We remove from $\mathcal{D}'$ clusters that were not designated as home clusters.

(s, d)-routing algorithm.

The number of bits of information maintained in the root is proportional to the number of nodes in the cluster times $\log n$. This also dominates the total amount of information maintained at other nodes in the cluster. Therefore, the total amount of information maintained in all nodes is proportional to the volume of $\mathcal{D}'$ times $\log n$, that is $O(n^{1+\frac{1}{s}} \log n)$ bits. Therefore, the total memory overhead of all $\log n$ regional routing schemes is $O(n^{1+\frac{1}{s}} \log^2 n)$ bits.

18.3 Application to Synchronizers

In this section we present synchronizer ZETA, based on sparse network covers. This synchronizer provides an interesting trade-off between time and messages: for a given trade-off parameter k, $2 \leq k \leq n$, ZETA sends the messages of a round within $O(\log_k n)$ time after the previous round and sends $O(k \cdot n)$ messages per round. Recall that synchronizer ALPHA, presented in Chapter 11, sends the messages of a round within $O(1)$ time after the previous round and sends $O(m)$ messages per round, where m is the number of communication links.

Let $\mathcal{C}$ be the cover containing the 1-neighborhoods of all the nodes in the network; clearly, diam$(\mathcal{C}) = 1$. We construct a coarsening of $\mathcal{C}$, by applying Theorem 18.1 with parameter $\log_k n$. We obtain a cover $\mathcal{C}'$ such that the 1-neighborhood of every node is contained in some cluster of $\mathcal{C}'$; moreover, diam$(\mathcal{C}') = O(\log_k n)$, and vol$(\mathcal{C}') = n^{1+1/\log_k n} = O(k \cdot n)$.

For each node v, we designate one of the clusters containing its 1-neighborhood as its *home cluster*.In each cluster[2] we choose some node as a root, and construct a *cluster tree* around it, which is the BFS tree between the root and all the nodes in the cluster.

Recall that in synchronizer ALPHA (Chapter 11), a node was safe when all its messages were received by its neighbors. A node can learn it is safe by waiting to receive acknowledgments on all the messages it sent.

A convergecast algorithm (Section 2.2) is used to collect the safety information up each cluster tree. For each cluster tree it belongs to, a node executes the following algorithm:

If it is a leaf in the tree, it sends a $\langle$*safe*$\rangle$ message to its parent; if it is an internal node in the tree, it waits for $\langle$*safe*$\rangle$ messages from all its children in this tree, and then sends a $\langle$*safe*$\rangle$ message to its parent.[3] When the root of a cluster tree receives a $\langle$*safe*$\rangle$ message from all its children, it broadcasts a $\langle$*pulse*$\rangle$ message down the tree. When a node receives a $\langle$*pulse*$\rangle$ message from its parent in its home cluster's tree, it enables the sending of the next round's messages.

Clearly, each node sends messages for a round exactly once. Furthermore, when a node v sends round r messages it knows that all the nodes in the node's

[2] Once again, we remove from $\mathcal{C}$ all clusters that were not designated as home clusters.

[3] Messages should be tagged with the cluster's id.

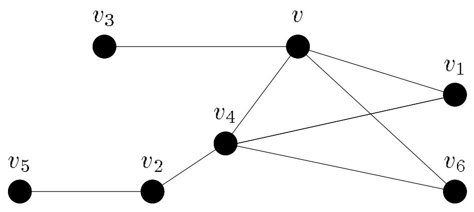

Figure 18.4: Cluster for Exercise 18.6; v is the cluster root.

home cluster are safe. Since v's home cluster includes its 1-neighborhood, this implies that all v's neighbors are safe. Therefore, all the messages sent to v at the previous round were received, as needed.

Note that the time between two consecutive rounds is proportional to the diameter of the cover, i.e., $O(\log_k n)$. The number of messages sent per round is proportional to the total size of the cluster trees, which is, in turn, proportional to the volume of the cover, i.e., $O(k \cdot n)$.

Exercises

18.1 Calculate the diameter of the cover presented in Figure 18.2.

18.2 Calculate the diameter and the volume of $\{V\}$, the cover with a single cluster containing all nodes of G.

18.3 Give an example of two covers $\mathcal{C}$ and $\mathcal{C}'$, such that $\mathcal{C}'$ is a coarsening of $\mathcal{C}$, vol$(\mathcal{C}') >$ vol$(\mathcal{C})$, and diam$(\mathcal{C}') >$ diam$(\mathcal{C})$.

18.4 Prove that if $\mathcal{C}$ is a cover of G, and $\mathcal{C}'$ is a coarsening of $\mathcal{C}$, then $\mathcal{C}'$ is a cover of G.

18.5 Consider Algorithm 18.1, and show that diam$(\mathcal{C}') \leq l \cdot$ diam$(\mathcal{C})$, where l is the maximal number of times the repeat loop is executed for a single cluster added to $\mathcal{C}'$.

18.6 Write the routing table for the nodes in the cluster shown in Figure 18.4.

18.7 Modify Algorithm 2.3 to assign DFS numbers to nodes.

18.8 Write pseudocode for routing inside a cluster, using DFS numbers.

18.9 Write pseudocode for synchronizer ZETA.

18.10 Prove that ZETA locally simulates the synchronous message passing model.

Hint: Follow the proof of Theorem 11.2.

Chapter Notes

The simple sequential algorithm for finding sparse network covers and the proof of Theorem 18.1 are due to Peleg [200]. This algorithm can be made distributed using simple synchronization mechanisms (see Peleg [200]), but the resulting algorithm does not have good message and time complexities. Efficient distributed algorithms for this problem we given by Awerbuch, Berger, Cowen and Peleg [36] (optimizing the time) and by Awerbuch, Patt-Shamir, Peleg and Saks [37] (optimizing the number of messages).

Finding routing algorithms with good memory requirements is one of the original applications of sparse network covers, presented by Awerbuch and Peleg [39]. The routing algorithm presented in Section 18.2 only bounds the *total* memory requirements of the routing scheme. However, some nodes, e.g., the cluster roots or nodes that belong to many clusters, maintain much more information than others. Awerbuch and Peleg [39] show how to optimize also the maximal memory requirements per node.

Synchronizer ZETA was suggested by Segall and Shabtay [233]. They also present an efficient distributed algorithm for constructing the sparse network cover needed for ZETA, together with the cluster trees; see also a correction by Moran and Snir [183]. Awerbuch [32] originally presented a synchronizer, GAMMA, with similar complexities. ZETA improves the constants of GAMMA, and is also simpler to understand and analyze.

Another application of sparse network covers is to reduce the message complexity of distributed algorithms. Afek and Ricklin [7] present a simulation that takes any distributed algorithm whose message complexity is $O(c \cdot m)$ and produces an algorithm solving the same problem with message complexity $O(c \cdot n \log n + m \log n)$, where, as always, n is the number of nodes and m is the number of edges.

Roughly speaking, the transformation uses a sparse network cover; in each cluster there is a central node and there are simple paths connecting the centers of pair of overlapping clusters (sharing a node). To run the algorithm, each central node executes the algorithm for all nodes in its cluster. Messages sent within the cluster need not be sent at all. Messages between nodes in different clusters are sent between the corresponding central nodes. The cost of sending messages from a specific node to all its neighbors is bounded by the number of clusters in which the node has neighbors times the distance between the corresponding central nodes. Theorem 18.1 is used to bound these quantities. For example, taking the cover consisting of the 1-neighborhood of every node and $k = \log n$, we obtain a cover in which each node is (on the average) in $O(n^{1/\log n})$ clusters, and the distance between central nodes of two neighboring clusters is $O(\log n)$.

An interesting application of this transformation is to find all the shortest paths in the network; this problem is the key for many routing and topology maintenance algorithm in wide-area networks. The message complexity of a standard algorithm for finding all shortest paths is $O(n \cdot m)$ (see Gallager [115] and Segall [232]). Applying the above simulation gives an algorithm whose message complexity is $O(n^2 \log n)$. The sparse network cover is constructed with $O(m + n \log n)$ messages using an algorithm of Awerbuch [32]; the same algorithm also distinguishes the cluster root and constructs a BFS cluster tree around it.

Other applications of sparse network covers are for tracking mobile users (by Awerbuch and Peleg [40]), and for distributed directories (by Peleg [201]).