Sparser: A paradigm for running distributed algorithms

Yehuda Afek Moty Ricklin

Computer Science Department,

Raymond and Beverly Sackler Faculty of Exact Sciences,

Tel Aviv University

Ramat-Aviv 69978

Israel

Abstract

This paper introduces a transformer for improving the communication complexity of several classes of distributed algorithms. The transformer takes a distributed algorithm whose message complexity is $O(f \cdot m)$ and produces a new distributed algorithm to solve the same problem with $O(f \cdot n \log n + m \log n)$ message complexity, where n and m are the total number of nodes and links in the network, and f is an arbitrary function of n and m.

Applying our paradigm to the standard *all shortest paths algorithm* [Gal76, Gal82, Seg83] yields a new algorithm which solves the problem in $O(n^2 \log n)$ messages (The previous best that we know of is $O(m \cdot n)$ messages). When applied to the $O(m \cdot \log^3 n)$ breadth-first search algorithm of Awerbuch and Peleg [AP90a] our paradigm yields an $O(m + n \cdot \log^4 n)$ messages algorithm.

1 introduction

One way to run a distributed algorithm is to collect all its inputs to one node, run a sequential algorithm on all the inputs at this node, and then distribute the outputs to the nodes. For many applications this is inefficient, since the message complexity of such a process is usually bounded by $\Theta(nm)$ messages, where n and m are the total number of nodes and links in the network. For most applications, truly distributed algorithms which keep the inputs and outputs of every node only at that node, are more efficient. In this paper we observe that an intermediate combination of the two approaches yields distributed algorithms that are sometimes more efficient than any of the two extremes.

The idea of our technique is to use a particular partition of the network into subsets of nodes, and select a center in each set. Each center simulates the algorithm execution on behalf of each node in its set. That is, messages sent in the original algorithm between pairs of nodes in the same set are eliminated. On the other hand, a message sent from a node in one set to a node in another set, is now sent from the center of the first set to the center of the other set. The algorithm is based on the existence of a partition such that the intra cluster communication, and the radius of every cluster, are small. The existence of a suitable partition was constructively proved by Awerbuch and Peleg in [AP90c].

The main contribution of this paper is a distributed algorithms transformer that reduces the message complexity of many distributed algorithms from $O(f \cdot m)$ to $O(f \cdot n \log n + m \log n)$ message complexity, where f is an arbitrary function of n and m. Perhaps one of the most prominent examples where such a technique might be useful is the all pairs shortest paths problem. A distributed algorithm to solve this problem is repeatedly executed all the time in the ARPANET and in other networks (ARPANET [MRR80] and others e.g. [JM82]). In this problem a routing table at each node, with one entry for each other node in the network is computed. Node u's entry in node v's table contains the length of the shortest path from v to u and the name of the first link on this path. This information is used in order to rout data messages between nodes.

Though the worst case message complexity ($O(mn)$) of the all pairs shortest paths problem has not been improved for several years, this fundamental task has been the subject of many papers [Gal76, MRR80, JM82, Eph86, Pap74, Gal82, Seg83, BG87, Spi86]. This paper produces an $O(n^2 \log n)$ messages distributed algorithm for the all pairs shortest paths problem (from scratch, i.e., including the construction of the sparse partition).

Another application of this paper is an $O(m + n \log^4 n)$ messages breadth first search (BFS) algorithm derived by Awerbuch and Peleg in [AP90a], where they have incorporated our method to their new $O(m \cdot \log^3 n)$ messages BFS algorithm. The message complexity of BFS has drawn considerable attention in recent years; In 1982 Gallager gave an $O(n^2)$ messages algorithm [Gal82], in 1985 Awerbuch and Gallager gave an $O(m^\epsilon)$ messages algorithm [AG87, AG85], and in an earlier draft of this paper we have presented an $O(m\sqrt{\log n} + n^\epsilon)$

messages algorithm [AR90]. A break through in the message complexity of BFS has been recently reported in the elegant work [AP90a] where an $O(m \cdot \log^3 n)$ messages algorithm is given. Being aware of our technique Awerbuch and Peleg have showed in [AP90a] that the combination of [AR90] and [AP90a] gives an $O(m + n \log^4 n)$ messages algorithm.

Furthermore, as suggested by our paper, once such a partition exists in the network other algorithms could be run more efficiently, e.g. the DFS in $O(n \log n)$ messages, by applying our technique to the algorithm of [Awe85b].

We believe that the above discussion shows that the sparser technique belongs to the set of elementary tools in distributed computing, such as the snapshot [CL85], the termination detection of diffusing computation [DS80] the synchronizer [Awe85a, AP90b], the resource controller [AAPS87], and the reset procedure [AAG87], which are used as building blocks in the design and implementation of other algorithms.

The rest of the paper is organized as follows: Section 2 gives an overview, Section 3 describes the sparser, which is the structure over which the simulation is performed, and in Section 4 the basic simulation technique is presented. In Section 5 extensions of the technique are presented.

Model: Throughout the paper the standard model of asynchronous networks is assumed [GHS83]. The network is modeled by an undirected graph $G = (V, E)$, $|V| = n$, $|E| = m$, where each vertex of the graph corresponds to a node (processor) in the network and each edge to a bi-directional communication link between the corresponding nodes. Nodes communicate only by exchanging messages, and message delay over a link is finite but unbounded.

Complexity measures: In this paper we are mainly concerned with the *message complexity* of distributed algorithms. The message complexity of an algorithm is the total number of messages sent during the execution of the algorithm in the network in the worst case, i.e., for each link that a message traverses, we charge one unit of cost. Messages size is at most $O(\log n)$ bits. Another standard complexity measure of distributed algorithms is their *time complexity*. The time complexity is the worst case total amount of time from the first step of the algorithm taken by any processor in a run until the last step taken by any processor. The time complexity is measured under the assumption that the delay of any message is at most one time unit.

The time complexity of the distributed algorithms generated by our technique is increased by at most $O(\log n)$ factor if we ignore the bound on the number of messages that can be transmitted over a link in one unit of time (in pipeline). However, if only one message can be transmitted into a link in one unit of time (i.e. inter message delay is $O(1)$), then due to congestion problems the message complexity is the best bound on the time complexity that we know of.

1.1 Relation to other works

Our work uses a particular partition from the sparse graph partitions of [AP90c] and combines it with a simple new idea: let the centers of clusters in the partition do the work for all the nodes in their cluster (computation and communication). It is important to note that applying our new idea with known partitions (e.g. synchronizer γ, and [AGLP89]) would not yield any savings in the communication complexity.

Our usage of the sparse graph partition is somewhat different from the applications introduced by Awerbuch and Peleg [AP90c, AP90b] for tracking mobile users, constructing compact routing tables, and synchronizing an asynchronous network with polylogarithmic overhead. Technically all their applications use the partitioning to localize global information in a hierarchical way. In this paper the partitioning is used to show that the overhead of protocols in dense networks is not much larger than that in sparse networks.

A seemingly similar idea was presented in [AGLP89], where a network is partitioned into clusters and a problem is solved by first solving it in the subnetwork of each cluster and then piecing together the partial solutions. This is in contrast to our technique where the centers of the clusters continuously communicate and cooperatively solve the problem in a distributed manner. Moreover, [AGLP89] is geared towards reducing the locality of a given problem while we are concerned with the message complexity of distributed algorithms.

2 Overview

The central idea in this paper is to execute distributed algorithms by communicating only between the nodes of a subset of the nodes. That is, to partition the network into a number of subsets of nodes, each with a distinguished central node, and establish a simple path connecting the two central nodes of each pair of neighboring subsets. To run the algorithm each central node executes the algorithm for each node in its subset. Whenever a message of the simulated algorithm is sent from a node in one subset to a subset of the nodes in another subset, that message is sent between the corresponding centers. Thus the cost of sending the same message from one node to all its neighbors is bounded by the number of subsets in which that node has neighbors, times the distance between the corresponding centers.

In order for the execution of a distributed algorithm to be communication efficient, the partition should have special properties. Roughly speaking, the sum over all nodes v, of the number of subsets in which v has neighbors, should be small (e.g. $O(n)$), and second, the radius of the connected graph spanning each subset must also be small (e.g. $O(\log n)$). The next section defines the structure of such a partition, and the section after specifies the simulation technique more precisely.

3 The sparser

The notations and definitions of [AP90c] are used in this paper. We repeat those definitions and notations that are used herein.

Let $G = (V, E)$ be an undirected graph. For two vertices $u, v \in V$, let $dist(u, v)$ denote the number of links on a shortest path from u to v in G. Define the radius of a graph to be

$$Rad(G) \stackrel{\text{def}}{=} min_{v \in V}(max_{u \in V} dist(v, u)).$$

Let the *1-neighborhood* of R be $N_1(R) \stackrel{\text{def}}{=} \{w | dist(w, v) \leq 1, v \in R\}$. Given a set of vertices $S \subset V$, let $G(S)$ denote the subgraph induced by S in G. We use $Rad(S)$ as a shorthand for $Rad(G(S))$.

Throughout sets of nodes are denoted by capital P, R, S etc., and collections of sets by calligraphic type, $\mathcal{P}, \mathcal{R}, \mathcal{S}$ etc. A set $\mathcal{P}$ is a *partition cover* of V if $\mathcal{P}$ is a set of pairwise disjoint nonempty sets and $\cup \mathcal{P} = V$. The elements of a partition are called **cells**[1]. The radius of a set of nodes S is based on their distances in the entire graph, i.e.,

$$Rad(S) \stackrel{\text{def}}{=} min_{v \in S}(max_{u \in S} dist_{G(V)}(v, u))$$

For a node $v \in V$ we define $\delta_v(\mathcal{S})$, the degree of v relative to $\mathcal{S}$, to be the number of cells in $\mathcal{S}$ which are at distance 1 or less from v. Formally:

$$\delta_v(\mathcal{S}) = |\{S | v \in N_1(S), S \in \mathcal{S}\}|.$$

Let the *density* of a partition cover $\mathcal{S}$ be:

$$dens(\mathcal{S}) = \sum_{S \in \mathcal{S}} |N_1(S)| \left(= \sum_{v \in V} \delta_v(\mathcal{S}) \right).$$

The *radius* of $\mathcal{S}$ is defined as, $Rad(\mathcal{S}) \stackrel{\text{def}}{=} \max_i Rad(S_i)$.

Definition 1 *A* **sparser** *is a partition cover $\mathcal{S}$. The radius and density of a sparser are the corresponding parameters of $\mathcal{S}$. A sparser whose radius is r and density is d is an (r, d)-sparser.*

In each cell of the sparser the following is defined: One node is distinguished as its center, *and a breadth first search tree rooted at the center spans the cell (if necessary using nodes not in the cell). Each pair of neighboring cells selects one inter-cell link, called the* preferred link. *The collection of breadth-first search trees and the preferred links is the structure spanning the network that is used by our simulation to pass messages between centers (similar to synchronizer γ in [Awe85a]).*

[1] What we call partition cover is called *weak partition cover* in [AP90c].

The existence of a sparser with low density (e.g., $O(n)$) and low radius (e.g., $O(\log n)$), follows from either Theorem 5.1 or Theorem 5.3 in [AP90c]. Few more definitions are necessary to repeat these theorems here (we will only discuss the relation between Theorem 5.1 and the sparser, since this theorem is more widely used and the efficient constructions of Theorem 5.3 have not been discussed in [AP90c]).

A *cluster* is a subset $S \subset V$ such that $G(S)$ is connected. A *cover* is a collection of clusters $\mathcal{S} = \{S_1, \ldots, S_m\}$ such that $\bigcup_i S_i = V$. Given a cover $\mathcal{S}$, its *degree*, and *average degree* are defined as follows:
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 maximum overlap between clusters of $\mathcal{S}$. The *average degree* of a cover $\mathcal{S}$ is defined as $\bar{\Delta}(\mathcal{S}) \stackrel{\text{def}}{=} (\sum_{v \in V} deg(v, \mathcal{S}))/n$.

Given a cover $\mathcal{T} = \{T_1, \ldots, T_k\}$, and the cover $\mathcal{N} = \{N_1(v) | \forall v \in V\}$, $\mathcal{T}$ is said to *coarsen* $\mathcal{N}$, if for every $N_i \in \mathcal{N}$ there exists a $T_j \in \mathcal{T}$ such that $N_i \subseteq T_j$.

Theorem 1 *[AP90c] Given a graph $G = (V, E)$, $|V| = n$, the cover $\mathcal{N}$, and an integer $k \geq 1$, it is possible to construct a coarsening cover $\mathcal{T}$ that satisfies the following properties:*

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

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

A sparser $\mathcal{S}$ is constructed from the coarsening cover $\mathcal{T}$ of the theorem as follows: each node selects one cluster $T \in \mathcal{T}$ that contains its 1-neighborhood as its home cell (the cells are not necessarily anymore clusters since nodes in one cluster may join different home cells). One can easily check that the density of the constructed sparser is $\leq n \cdot \bar{\Delta}(\mathcal{T})$ and its radius is $\leq (2k + 1)$.

Algorithms for constructing coarsening covers are discussed in [AP90a]. Note that a $(k, n^{1+1/k})$-sparser can be easily constructed by a distributed algorithm similar to the one in [Awe85a] with $O(m + nk)$ messages. Moreover, the distributed implementation readily produces the additional structures: distinguishes a center in each cell and constructs a breadth-first search tree spanning each cell, rooted at the center. All of this is achieved with the same message complexity, $O(m + nk)$. One may consider the randomized techniques of Linial and Saks [LS91] to device a fast (sub linear) randomized algorithm for the construction of a sparser, as described in [ABCP91].

4 The simulation technique

Assuming that an (r, d)-sparser $\mathcal{S}$ is given in the network, the sparser simulation technique proceeds in three major steps: First, collect the topological information of all the nodes in each cell in $\mathcal{S}$ to the center of the cell. This information includes the identity of all the cells that are incident to nodes in the cell and the links leading to these cells. Second, simulate the algorithm by exchanging messages between the center nodes. Third, the outputs of the algorithm are distributed by the center of each cell to nodes in the cell along the breadth-first search tree. The crucial step is the second step in which centers have to exchange messages on behalf of their nodes. The first step requires $O(rm)$ messages since information about m links is sent to distance r (assuming the topological information dominates the size of the input data to the algorithm). If the size of the output of the algorithm at each node is at most $O(y \log n)$ bits, then the third phase, the output distribution phase, costs $O(yrn)$ messages. [2]

To describe the simulation let us define a cycle of computation for an asynchronous distributed algorithm. A distributed asynchronous algorithm proceeds at each node in cycles of three steps:

1. *message receipt*,
2. *local computation of a new local state*, and
3. *message transmission to a subset of the neighbors.*

(Some cycles might consist only of steps 1 and 2.)

Lemma 1 considers asynchronous distributed algorithms in which nodes in the third step of each cycle send either the same message to all their neighbors, or a message to only one neighbor, or no message at all. Call such distributed algorithms *type α distributed algorithms.* (Lemma 3 is Section 5 consideres more general classes of algorithms).

Definition 2 *The* cycle complexity *of an asynchronous algorithm A, is the maximum, over all the nodes, of the number of cycles a node goes through during the execution of the algorithm, in the worst case.*

Consider a message M of a type α algorithm A that is sent from node v to node u. In the sparser simulation of A, if v and u are in the same cell then no message has to be sent by the simulation. If however $v \in S_1$ and $u \in S_2$, $S_1, S_2 \in \mathcal{S}$, then S_1's center c_1, sends the

[2]If the algorithm produces outputs at the nodes several times during a run then $O(yrn \log n)$ small messages might be necessary.

message to the preferred link that connects S_1 with S_2. Then the message is sent over the preferred link and through the parent links of the breadth-first search tree of S_2 to c_2. Since algorithm A is of type α then M is either sent to all the neighbors of v in S_2 or to exactly one. In either case this information can be encoded in the message from c_1 to c_2 in at most $\log n$ bits.

Lemma 1 *Given a network with an (r,d)-sparser $\mathcal{S}$ and a K cycle complexity distributed algorithm A of type α, then A can be run in the network in $O(mr + Kdr + yrn)$ messages, where $y \log n$ is the size of the output of the algorithm at each node in bits.*

Proof: The term $O(mr)$ is the cost of collecting the topology of the neighborhood of each cell to the center of the cell. The $O(yrn)$ term is the cost of distributing the outputs of A to the nodes.

In each cycle of computation at node v the simulation sends at most $2r+1$ messages for each neighboring cell of v. Thus at most $\sum_v \delta_v(\mathcal{S}) \cdot (2r+1)$ messages are sent, per cycle, resulting in a total of $O(Kdr)$ over the entire run of the algorithm. ∎

Example 1 *Consider the following algorithm for the all pairs shortest paths algorithm, which is also given in [Gal76, Gal82, Seg83]. Each node starts by sending its identity to all its neighbors. When a node receives the identity of all its neighbors, it marks them to be in distance one from it, and starts the second step. In the i'th step every node builds a message which consists of all the nodes which it marked to be in distance $i-1$ from it, and sends the message to all its neighbors. The node waits until it receives all the messages sent by its neighbors during their i'th step, mark the nodes which it receives their identities for the first time, to be in distance i from it, and passes to the next step. A node terminates when it receives no new identities in an entire round. Segall in [Seg83], gave a correctness proof for this algorithm. Note that for this algorithm both cycle complexity, and the size of the output at each node are $O(n)$.*

Using a $(\log n, n)$-sparser (by setting $k = \log n$ in Theorem 1) and Lemma 1 we get a $O(n^2 \cdot \log n)$ messages distributed algorithm to solve the all pairs shortest paths problem. (The algorithm to construct a $(\log n, n)$-sparser costs $O(n \log n + m)$ messages).

Since the simulated all pairs shortest path algorithm works as well in the weighted case, i.e. when each link has a real length in each direction, also our algorithm solves the weighted problem with the same communication complexity. (The length corresponds to the queue length in the entry to the link [MRR80], however the messages of the algorithm do not incur this delay because they are of the highest priority). That is, each $O(\log n)$ bits message of the shortest paths algorithm still incur one unit of cost. In this case, in the ith step node v sends to all its neighbors the i-th closest node to v.

Corollary 2 *The upper bound on the message complexity of the (weighted and unweighted) all pairs shortest paths problem is $O(n^2 \log n)$ messages (each message is of size $O(\log n)$ bits).*

Another example is the application of our technique to the BFS algorithm of [AP90a] which results in $O(m + n \log^4 n)$ messages BFS algorithm as noted in [AP90a].

5 Generalizations

Although Lemma 1 applies only to a certain class of algorithms a similar result can be derived to a much wider class. Lemma 3 in the sequel applies to any algorithm that in step 3 of the computation cycle sends the same message from a node to any subset of its neighbors.

Lemma 1 relied on the following fact about distributed algorithms of type α: whenever a node sends a message to a subset of its neighbors, the information about the destinations of the message can be encoded in no more than $O(\log n)$ bits. To enable a generalization of Lemma 1 we relax this assumption by introducing two modifications in algorithms in which nodes send a message to an arbitrary subset of their neighbors, as follows: First, whenever a node makes a "relevant" change in its local state it sends a message to this effect to all its neighbors (where "relevant" is defined in the sequel). Second, each message that is sent to a subset of the neighbors is now sent to all the neighbors. Knowing the local state of the sender, each neighbor determines whether the message was addressed to it or not.

Let us define things more formally. Let A be a distributed protocol, and let $S(A)$ be the set of local states of the protocol at some node. We define a relation, $\propto$, on $S(A)$, such that $s_1 \propto s_2$ if for every possible message M, each node upon receiving M sends exactly the same set of messages, in both states s_1, and s_2. Obviously, $\propto$ divides $S(A)$ into equivalence classes. During a run of a distributed protocol at node v, we define the *relevant state* of v to be the equivalence class to which the internal state of v belongs, under the relation $\propto$.

Definition 3 *The* state complexity *of an asynchronous algorithm A, is the maximum, over all the nodes, of the number of times a node changes its relevant state (i.e. changes the equivalence class) during the execution of the algorithm, in the worst case.*

In many distributed algorithms the following pattern of communication, called *local-polls*, that consists of three phases occurs several times: first a node sends the same message to a subset of its neighbors, then, in the second phase, each neighbor that receives the message responds with a reply, and in the third phase, the originating node collects all the replies and continues with its computation. Usually the replies are called acknowledgments.

Informally, we define the *weak-cycle* complexity of a distributed algorithm to be the cycle complexity of the algorithm discounting the cycles in which a node sends only a response message as part of a local poll pattern of communication. The motivation for this definition is that in many algorithms the cycle complexity is high due to acknowledgment type of messages. Applying Lemma 1 to these algorithms would result in no savings in the message complexity.

Again, let us define these notions more formally. Let A be a distributed protocol, and let v be a node in the network.

Definition 4 *We say that a message M sent from node u to v is an* acknowledgment*, if the cycle in which M has been sent was initiated by a message received from v, and M was the only message sent during this cycle.*

We say that a cycle is a *real* cycle if the messages that were sent during the cycle were not acknowledgments and were sent to more than one neighbor.

Definition 5 *The* weak cycle complexity *of an asynchronous algorithm A, is the maximum, over all the nodes, of the number of real cycles a node goes through during the execution of the algorithm, in the worst case.*

Definition 6 *The cycles in which a node sends a message to only one neighbor are called* trivial cycles*, and the* trivial cycle complexity *is the total number of such cycles in the network during the execution of the algorithm.*

In order to transform algorithm A into an α type, apply to it the following three changes (assuming FIFO discipline on the links):

1. Any change in the relevant state of a node in step 2 of the computation cycle is broadcast by the node to all its neighbors, before step 3 is performed. Thus, whenever a node u receives a message M from node v, the relevant state of v at the time M was sent is available at u.

2. Replace each response message (acknowledgment) by a predetermined fixed-size standard response message. (The standard response is a fixed message with $O(1)$ bits that is recognized as such). Upon receiving the standard response, each node can compute the real response from the relevant-state information of the neighbor, that it has last received.

3. Any message that is sent by a node v in A to a subset of the neighbors is sent to all the neighbors of v. Since the neighbors know the relevant state of v at the time that the message was sent, each will be able to determine whether the message was addressed to it, or not.

Lemma 3 *Given a network with an (r,d)-sparser and a distributed algorithm A with K weak cycle complexity, and t trivial cycle complexity which was modified as discussed above, then A can be run in the network in $O(mr + (K+s)dr + tr + yrn)$ messages where $y \log n$ is the size of the output of the algorithm at each node in bits, and s is the state complexity of A (assuming that the value of the relevant state of a node can be transmitted with $\log n$ bits).*

Proof: Apply Lemma 1 to the modified algorithm A. In any local-poll of node v the responses from all the neighbors of v that are in one cell are sent as one message from the center of the cell to the corresponding center of the neighboring cell that contains v. That is, when a center of a cell receives a message which was sent to nodes in its cell, it checks to see whether any of these nodes have to respond with an acknowledgment. If yes, it sends back only one copy of the standard response message. For each real cycle at a node a message is sent to all its neighbors, thus incurring Kdr messages for all the nodes in the simulation. Similarly, a message is sent to all the neighbors each time a node changes is state, incurring a total of srd messages.

Recall that in a trivial cycle at node v exactly one message is sent from node v. Since such a message could in general go over a path of length r in the simulation, the total complexity due to the trivial cycles is tr. ∎

Example 2 *Consider the Depth First Search algorithm presented in [Awe85b]. The trivial cycle complexity t, of that algorithm is n. The weak cycle complexity s, of that algorithm is 2, and its state complexity is also 2. Thus, if a $(\log n, n)$-sparser already exists in the network the DFS algorithm can be run in $O(n \log n)$ messages.*

Once a sparser is given in the network, the message complexity of any algorithm is determined by its three parameteres: trivial cycle complexity, real cycle complexity, and state complexity. E.g. the MST algorithm of [GHS83] has a trivial cycle complexity $O(n \log n)$, real cycle complexity $O(\log n)$ and state complexity $O(\log n)$.

Acknowledgments: We thank Baruch Awerbuch and David Peleg for helpful discussions. In particular, David brought to our attention [Pel89] in early 1989, and Baruch pointed out that the standard shortest paths algorithm works as well in the weighted case. We would also like to thank Mike Merritt and Mike Saks for helpful discussions.

References

[AAG87] Y. Afek, B. Awerbuch, and E. Gafni. Applying static network protocols to dynamic networks. In *Proc. of the 28th IEEE Annual Symp. on Foundation of Computer Science*, pages 358–370, October 1987.

[AAPS87] Y. Afek, B. Awerbuch, S. Plotkin, and M. Saks. Local management of a global resource in a communication network. In *Proc. of the 28th IEEE Annual Symp. on Foundation of Computer Science*, pages 347–357, October 1987.

[ABCP91] B. Awerbuch, B. Berger, L. Cowen, and D. Peleg. Fast constructions of sparse neighborhood covers. Extended abstract, November 1991.

[AG85] B. Awerbuch and R. Gallager. Distributed bfs algorithms. In *Proc. of the 26th IEEE Annual Symp. on Foundation of Computer Science*, pages 250–256, October 1985.

[AG87] B. Awerbuch and R. Gallager. A new distributed algorithm to find breadth first search trees. *IEEE Trans. Info. Theory*, pages 315–322, May 1987.

[AGLP89] B. Awerbuch, A. Goldberg, M. Luby, and S. Plotkin. Network decomposition and locality in distributed computation. In *Proc. of the 30th IEEE Annual Symp. on Foundation of Computer Science*, pages 364–369, October 1989.

[AP90a] B. Awerbuch and D. Peleg. Efficient distributed construction of sparse covers. Technical Report CS90-17, Weizman Institute of Science, Dep. of Computer Science, July 1990.

[AP90b] B. Awerbuch and D. Peleg. Network synchronization with polylogarithmic overhead. In *Proc. of the 31st IEEE Annual Symp. on Foundation of Computer Science*, pages 514–522, October 1990.

[AP90c] B. Awerbuch and D. Peleg. Sparse partitions. In *Proc. of the 31st IEEE Annual Symp. on Foundation of Computer Science*, pages 503–513, October 1990.

[AR90] Y. Afek and M. Ricklin. Sparser: A paradigm for running distributed algorithms. Extended abstract, April 1990.

[Awe85a] B. Awerbuch. Complexity of network synchronization. *Journal of the ACM*, 32(4):804–823, October 1985.

[Awe85b] B. Awerbuch. A new distributed depth-first-search algorithm. *Information Processing Letters*, 20(3):147–150, April 1985.

[BG87] D. P. Bertsekas and R. G. Gallager. *Data Networks*. Prantice Hall, 1987.

[CL85] K. M. Chandy and L. Lamport. Distributed snapshots: Determining global states of distributed systems. *ACM Trans. on Computer Systems*, 3(1):63–75, January 1985.

[DS80] W. Dijkstra and C. S. Scholten. Termination detection for diffusing computations. *Information Processing Letters*, 11-1:1–4, August 1980.

[Eph86] A. Ephremides. *The routing problem in Computer Networks*. Springer Verlag, 1986.

[Gal76] R. G. Gallager. A shortest path routing algorithm with automatic resynch. Unpublished note, March 1976.

[Gal82] R. G. Gallager. Distributed minimum hop algorithms. Technical Report LIDS-P-1175, M.I.T. Lab for Information and Decision Systems, January 1982.

[GHS83] R. G. Gallager, P. A. Humblet, and P. M. Spira. A distributed algorithm for minimum weight spanning trees. *ACM Trans. Program. Lang. Syst.*, 5:66–77, January 1983.

[JM82] J. Jaffe and F. Moss. A responsive distributed routing protocol. *IEEE Trans. on Communication*, COM-30(7, part II):1758–1762, July 1982.

[LS91] N. Linial and M. Saks. Decomposing graphs into regions of small diameter. In *Proc. of 2nd ACM-SIAM Symp. on Discrete Algorithms*, pages 320–330, January 1991.

[MRR80] J. M. McQuillan, I. Richer, and E. C. Rosen. The new routing algorithm for the arpanet. *IEEE Trans. on Communication*, COM-28(5), May 1980.

[Pap74] U. Pape. Implementation and efficiency of moor-algorithms for the shortest route problem. *Mathematical Programming*, 7:212–222, 1974.

[Pel89] D. Peleg. Sparse graph partitions. Technical Report CS89-01, Dep. of Applied Math. The Weizmann Institute, Rehovot, Israel, February 1989.

[Seg83] A. Segall. Distributed network protocols. *IEEE Trans. on Information Theory*, IT-29(1):23–35, January 1983.

[Spi86] J. M. Spinelli. Broadcasting topology and routing information in computer networks. Master's thesis, MIT, March 1986.