

# Distributed Construction of Energy-Efficient Ad Hoc Wireless Broadcast Trees

by

Ashwinder S. Ahluwalia

Submitted to the Department of Electrical Engineering and Computer  
Science

in partial fulfillment of the requirements for the degree of  
Master of Engineering in Computer Science and Engineering  
at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

Feb 2002

© Ashwinder S. Ahluwalia, MMII. All rights reserved.

The author hereby grants to MIT permission to reproduce and distribute  
publicly paper and electronic copies of this thesis document in whole or in  
part.

Author .....  
Department of Electrical Engineering and Computer Science  
Feb. 1, 2002

Certified by .....  
Eytan Modiano  
Assistant Professor, Department of Aeronautics and Astronautics  
Thesis Supervisor

Certified by .....  
Dr. Li Shu  
Communications and Network Engineer, The Charles Stark Draper  
Laboratory  
Thesis Supervisor

Accepted by .....  
Arthur C. Smith  
Chairman, Department Committee on Graduate Students



# Distributed Construction of Energy-Efficient Ad Hoc Wireless

## Broadcast Trees

by

Ashwinder S. Ahluwalia

Submitted to the Department of Electrical Engineering and Computer Science  
on Feb. 1, 2002, in partial fulfillment of the  
requirements for the degree of  
Master of Engineering in Computer Science and Engineering

### Abstract

We address the energy-efficient broadcasting problem in ad hoc wireless networks. First we show that finding the minimum energy broadcast tree is NP-complete and develop an approximation algorithm, which computes sub-optimal solutions in polynomial time. We present a distributed algorithm that computes all  $N$  possible broadcast trees simultaneously with  $O(N^2)$  message complexity. We compare our algorithm's performance to the best known centralized algorithm, and show that it constructs trees consuming, on average, only 18% more energy. Finally, we introduce the multiple source broadcasting problem, and explore algorithms that address this problem.

Thesis Supervisor: Eytan Modiano

Title: Assistant Professor, Department of Aeronautics and Astronautics

Thesis Supervisor: Dr. Li Shu

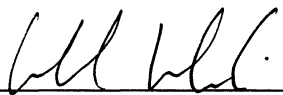
Title: Communications and Network Engineer, The Charles Stark Draper Laboratory



## Acknowledgment 2/1/02

This thesis was prepared at The Charles Stark Draper Laboratory, Inc., under IR&D, GCC group, charge #18542.

Publication of this thesis does not constitute approval by Draper or the sponsoring agency of the findings or conclusions contained herein. It is published for the exchange and stimulation of ideas.

  
\_\_\_\_\_  
(author's signature)

# Assignment

Draper Laboratory Report Number T-1420

In consideration for the research opportunity and permission to prepare my thesis by and at The Charles Stark Draper Laboratory, Inc., I hereby assign my copyright of the thesis to The Charles Stark Draper Laboratory, Inc., Cambridge, Massachusetts.

   
\_\_\_\_\_  
(author's signature) (date)

# Contents

<b>1</b>	<b>Introduction</b>	<b>11</b>
1.1	Problem Formulation . . . . .	14
1.2	Background . . . . .	15
1.2.1	Complexity . . . . .	15
1.2.2	Algorithms . . . . .	17
1.2.3	The need for a distributed algorithm . . . . .	20
<b>2</b>	<b>Complexity</b>	<b>23</b>
2.1	Relevant Background Work . . . . .	23
2.2	Proof of NP-completeness . . . . .	29
<b>3</b>	<b>Constructing Broadcast Trees</b>	<b>33</b>
3.1	The Broadcast Incremental Protocol (BIP) . . . . .	33
3.1.1	Improving BIP: Hybrid Algorithms . . . . .	37
3.2	Computing Broadcast Trees Distributively . . . . .	38
3.2.1	The formation of clusters . . . . .	40
3.2.2	Synchronous Distributed Clustering Algorithm . . . . .	42

3.2.3	Implementation Considerations . . . . .	45
3.2.4	An alternate clustering algorithm . . . . .	48
3.2.5	A Clustering Sweep Procedure . . . . .	50
3.2.6	Joining Clusters Together . . . . .	55
3.3	Simulation Results . . . . .	56
3.4	Multiple Source Broadcast . . . . .	63
<b>4</b>	<b>Conclusion</b>	<b>69</b>



# List of Figures

1-1	The Multicast Advantage . . . . .	13
2-1	Example of Node Cover, Connected Node Cover, and Connected Dominating Set . . . . .	24
2-2	Example of Planar Graph and Planar Embedding . . . . .	24
2-3	Example of Planar Orthogonal Grid Drawing . . . . .	25
2-4	Example of Unit Disk Graph . . . . .	25
2-5	Step 1 of the reduction used for CDSUDG . . . . .	26
2-6	Step 2 of the reduction used for CDSUDG . . . . .	27
2-7	Step 3 of the reduction used for CDSUDG . . . . .	27
2-8	Step 4 of the reduction used for CDSUDG . . . . .	28
2-9	Step 4 of the reduction used for CDSUDG, with a box around nodes $n_1$ through $n_4$ . . . . .	30
3-1	BIP example - starting configuration . . . . .	34
3-2	BIP example - Node 1 is added to $T$ . . . . .	34
3-3	BIP example - Node 3 is added to $T$ . . . . .	35

3-4	BIP example - Node 2 is added to $T$ to and the BIP computation is complete.	35
3-5	Performance of $BIPHybrid_1$ and $BIPHybrid_2$ as compared to BIP. . . . .	39
3-6	Simulation Results at Infinite Range . . . . .	59
3-7	Simulation Results at Limited range, 100 node networks . . . . .	61
3-8	Simulation Results at Limited range, 200 node networks . . . . .	62
3-9	Simulation Results of Multiple Source BIP . . . . .	67
3-10	Simulation Results of Multiple Source Distributed Algorithm . . . . .	68

# Chapter 1

## Introduction

Over the past decade, research interest in the area of ad hoc networks has increased dramatically. An ad hoc network can be described as a network that is built in the absence of preexisting infrastructure. Common examples include networks for emergency response, sensor networks and various military applications. In emergency communication networks, the following scenario is a good example: communication infrastructure has been destroyed by a natural disaster, so each rescuer is given a radio that can broadcast messages wirelessly (at some limited range), and we would like to route messages between radios that cannot directly transmit to one another. Because the infrastructure for this communication network was not built before the radios are used, this is considered an ad hoc network. Ad hoc networks are most useful in environments where the cost of constructing a fixed communication network is too high, and too time consuming compared to deploying an ad hoc network in its place. Consequently, research in this field focuses on constructing networks that can function with minimal resources (to lower per-node cost), without sacrificing usability. Examples of

ad hoc networking issues include energy usage, wireless interference avoidance, and routing robust to mobility.

In this thesis, we are interested in the construction of energy-efficient one-to-all trees in ad hoc networks, as posed in [10]. Starting with a given source node  $s$ , the problem is to find a broadcast tree that allows  $s$  to send a message to all other nodes, using the minimum amount of energy. Although it is tempting to do so, we do not simultaneously deal with other issues such as channel contention and mobility - we believe that a well-formed solution to this problem can serve as an intuitive start to algorithms that also include other issues.

We focus on a specific type of ad hoc network where all nodes are stationary (usually referred to as a “static” ad hoc network), and equipped with an omnidirectional transmitter. We assume that the transmission range of the transmitter can be adjusted from 0 up to a maximum range  $R_{max}$ . Although previous energy-efficient routing research has focused on environments where the transmitter can either be turned off, or transmitting at its maximum range ([11]), we adopt the model of more recent ad hoc networking research where transmitters are permitted to choose any range in between 0 and  $R_{max}$  ([13]-[17]). The transmitter is referred to as omnidirectional because it does not focus the message transmission in a particular direction. Therefore, when the omnidirectional transmitter sends a message at range  $r$ , all nodes within distance  $r$  can receive the message, regardless of their position. Naturally, when the node transmits a message at a higher range, it consumes more power. In analyzing the range-power tradeoff, we adopt a common communications model, where the power required to transmit the message is proportional to  $r^\alpha$  ( $\alpha \geq 2$  in all networks, and most typically is greater than 2).

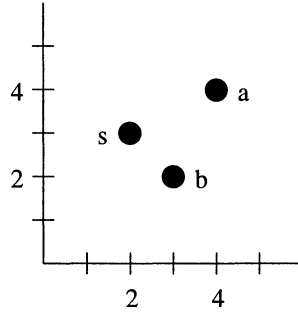


Figure 1-1. The Multicast Advantage

Note that because each node can only transmit up to distance  $R_{max}$ , it is possible that the source  $s$  cannot reach all nodes in the network directly. Therefore, some nodes will have the responsibility to forward messages on the behalf of  $s$ . We can then rephrase the one-to-all problem as one of assigning each node,  $n_i$ , a range  $r_i$  at which to forward received messages. The total cost of the broadcast tree can then be expressed as  $\sum r_i^\alpha$ .

This is a very atypical cost function for a graph connectivity problem, in that cost is *node* weighted instead of edge weighted. Consider the example shown in Figure 1-1. In this situation, we denote the cost incurred when node  $s$  transmits at a distance just large enough to reach  $b$  as  $Power(s, b)$ , and define  $Power(s, a)$  analogously. If  $s$  attempts to send a message to node  $b$ , this message will not be received by node  $a$  because  $a$ 's distance to  $s$  is larger than  $b$ 's distance to  $s$  ( $d_{as} > d_{bs}$ ). However,  $s$  could optionally transmit at  $Power(s, a)$ , in which case the message would be received by *both*  $a$  and  $b$  (because  $d_{bs} < d_{as}$ ). By transmitting to  $a$ ,  $s$  can get a transmission to  $b$  for “free” because of each transmission’s omnidirectionality. In [10], this was referred to as the “multicast advantage.” In general, the power required for a node  $s$  to transmit a message to a set of nodes  $N$  is  $\max_{n \in N} Power(s, n)$ . A transmission at this power will be received by all nodes in  $N$ .

The multicast advantage and node weighted cost function certainly differentiate this prob-

lem from other network connectivity problems. In the first part of this thesis, we characterize the complexity of this problem, and show it is NP-complete. Consequently, it is inefficient to find the optimal solution to this problem, as it may take exponential time. Hence, it is important to investigate the possibility of an approximation algorithm - a polynomial time algorithm that does not necessarily find the optimal solution, but constructs solutions that are not much more costly than optimal. We first look at a previously published global approximation scheme ([10]), and explore improvements to it. We then investigate a distributed approximation algorithm, and show its performance is comparable to a centralized approximation ([10]) in the average case. We conclude by introducing the multiple source broadcasting problem, and explore the extent to which multiple sources can be used to save energy.

## 1.1 Problem Formulation

The ad hoc wireless broadcasting problem can be stated as follows: We are given a set of nodes  $N$  and a function  $\phi : N \rightarrow \mathbb{Z} \times \mathbb{Z}$ , which gives us a set of coordinates for each node on the two dimensional plane. Each of these nodes represents a static (non-mobile) wireless-enabled device that is capable of both transmitting and receiving messages from its neighbors. Additionally, we are given a range  $R \in \mathbb{Z}^+$ , which represents the maximum distance any node can transmit a message, and a constant  $\alpha > 0$ . We construct the undirected graph  $G = (V, E)$  where  $V = N$  and  $(i, j) \in E \iff d_{ij} \leq R$ . Assuming this graph is connected, the wireless broadcasting problem can be stated as follows:

Given a source  $s \in N$  construct the minimum cost directed tree  $T$  (rooted at  $s$ )

that connects  $s$  to every node in  $N - \{s\}$  via a directed path such that  $d_{ij} \leq R \forall (i, j) \in T$ . Given that  $f(x) = \max\{d_{xj}^\alpha : (x, j) \in T\}$ , we define the cost of  $T$  as  $\sum_{n \in N} f(n)$ .

Hereafter, we refer to the decision version of this problem, where we are asked to determine whether there exists such a tree with cost less than  $l \in \mathbb{Z}^+$ , as *BCAST*.

## 1.2 Background

### 1.2.1 Complexity

In [18], it was shown that the graph version of the *BCAST* problem is NP-complete. In the graph version of *BCAST*, we are given a directed graph with an arbitrary, nonnegative cost on each edge. Given a source  $s$ , the graph version of the *BCAST* problem is to construct a subset of edges  $E'$  with minimum cost (where the cost is  $\sum_{n \in N} f(n)$ , and  $f(n)$  is the maximum cost over all  $E'$  edges exiting node  $n$ ) such that the subgraph induced by  $E'$  contains a path  $s$  to every node. We refer to the graph version of *BCAST* as *GENBCAST*. The proof of *GENBCAST*'s NP-completeness can be done via a simple reduction from the set cover problem ([18]).

Although the *BCAST* problem can be modeled as a *GENBCAST* problem, the NP-completeness of *GENBCAST* does not necessarily imply that *BCAST* is NP-complete. For this to be true, we would have to prove that the *GENBCAST* problem remains NP-complete for graphs that have a geometrically restricted cost function. This cost function must reflect positions of each node in the 2D plane, where the the cost of an edge  $(i, j)$  corresponds

to the power used in transmitting from  $i$ 's position to  $j$ 's position in the plane. To prove the problem remains NP-complete for graphs that are thus geometrically constricted is not trivial - as an exercise, consider the difficulty of extending *GENBCAST*'s reduction from set cover to work for this class of cost function.

In considering the NP-completeness of *BCAST*, some insight can be drawn from other connectivity problems with similar cost functions. In [5], it was shown that the problem of constructing strongly connected subgraphs of minimum cost (with the same cost function as in *GENBCAST*) is NP-complete. That is, the problem of constructing a subset  $E'$  of directed edges, with minimum cost, such that there is a directed path from each node to every other node via edges in  $E'$ , was proven NP-complete. Notably, the proof in [5] did not show that this problem, with geometrically restricted cost functions, was NP-complete. Rather, they proved only that the strongly connected version of *GENBCAST* was NP-complete.

Without further consideration, it is unclear whether or not the *BCAST* problem is NP-complete. On the one hand, one may argue that the geometric constraint makes the problem more tractable than the NP-complete *GENBCAST* - this additional constraint can be used to eliminate many potential solutions. Quite possibly, this additional constraint might result in the existence of at most a polynomial number of feasible solutions (which would imply that *BCAST* can be solved in polynomial time). On the other hand, one might argue that this additional constraint has no effect on the problem's difficulty, or makes the problem more difficult than *GENBCAST*. Certainly, *GENBCAST* contains some similarities to the Steiner problem, in that "Steiner" points in a Steiner tree somewhat correspond to nodes of large transmission "radius"; both serve as points where the tree branches out to several



nodes at once. Notably, Steiner problems, when restricted to graphs in 2D, are still NP-complete ([7]). It is quite possible that *GENBCAST* behaves similarly when geometrically restricted.

Some understanding of how this geometric constraint affects the complexity of this class of connectivity problems was addressed in [6]; it was shown that the strongly connected version of this problem, when restricted to cost functions that reflect node positions in the plane, still remains NP-complete. Naturally, the reduction in [6] is much more complicated than the one presented in [5]. Unlike the reduction in [5], the reduction in [6] has to map each instance of an NP-complete problem to a set of *node positions* in the plane, which adds considerable difficulty to the proof.

In addition to constructing this mapping to node positions in the plane, the proof in [6] underwent a lengthy case-by-case analysis to demonstrate that the number of nodes being generated by their reduction is polynomially related to the size of the input (if this were not the case, the reduction would not be polynomial). Although this is an easier method of proof given the considerable complexity of any placement algorithm, the reading of the case analysis in [6] made it more difficult to ascertain exactly how the reduction was being performed. This gave us an appreciation both for the complexity of the role of the geometric constraint in *BCAST*, and the need for simplicity in any geometric reduction.

## 1.2.2 Algorithms

Most previous research on energy efficient messaging in ad hoc networks has not focused directly on problems similar to *BCAST*. The differences seem to lie in the model of the

energy-limited wireless environment.

For example, Das and Bhargavan adopted a model in which all nodes share a common maximum range, and a node can either not be transmitting at this maximum range, or not be transmitting at all (no range control). In [11], they look at the problem of finding a minimum sized subset of nodes that is a connected dominating set - this subset obeys that property that every node in the graph is either in this subset or in the range of a node in this subset, and that the graph induced by this subset is connected (this can be used as a “backbone” for unicast routing). Other research also adopted this model, but extended it to allow nodes to roam between several power modes, indicating their level of participation in network routing (see [12] for an example).

Later papers have extended this model to one where each node is able to transmit at any range between 0 and the maximum range. As opposed to the above model, this environment allows minimization of energy consumption at a finer granularity. Additionally, a model assuming range control allows the network designer to make the decision of providing a maximum range high enough to ensure connectivity without having to worry simultaneously about the effects on energy efficiency (granularity effects). This model is also more reflective of current low-energy transmitter technology ([13]), and many other papers have shown this model is useful for purposes other than energy savings, such as collision avoidance ([14], [16]) and quality of service ([15], [17]). For these reasons, this is the model that we have adopted in our analysis.

Work by Wieselthier, et. al. looked directly at the above broadcast problem ([10]), and proposed a centralized algorithm to construct energy efficient broadcast trees. They showed

that their algorithm, the Broadcast Incremental Protocol (BIP), performed well in practice compared to minimum spanning trees and shortest path trees. The BIP algorithm assumes the same model for power-range tradeoff that we assume in this thesis (power proportional to  $r^\alpha$ ), so it is particularly relevant. Additionally, to our knowledge, BIP is the best known algorithm for this problem in the present literature ([18]). Consequently, we will use the performance of BIP as a measuring stick in judging our algorithms. As we will discuss below, BIP assumes that each node does not have a range limitation ( $R_{max} = \infty$ ). This is an important distinction from our problem, in which we assume that all nodes have some predefined range limit.

A year after BIP was introduced, [18] proved that the BIP algorithm has a constant approximation ratio of 12. That is, the power consumed by a broadcast tree generated by BIP is *at most* 12 times the power consumed by the minimum power tree. Although the value of the constant is not as important, the fact that this was a constant (i.e. not a function of the number of nodes) was significant. This differentiates the *BCAST* problem from other problems in wired networks that have similar characteristics, like the Directed Steiner Problem (in [8], it was proven that no polynomial time algorithm for this problem can achieve an approximation ratio better than  $O(\lg n)$ ). Furthermore, this gives hope that, although solving *BCAST* optimally is intractable, it can be approximated to within a small factor.

### 1.2.3 The need for a distributed algorithm

Although BIP has already been shown to construct energy efficient trees in practice ([18]), it's centralized nature requires one node to collect the position information of every node in the graph, compute the BIP tree, and distribute the solution to all other nodes in the network. This can result in considerable time, message complexity, and power consumption. Additionally, this requires that the node performing the computation also has considerable resources (energy, processor, and memory). In the low cost, resource limited environment that is typical in ad hoc networks, this may not always be feasible.

These reasons motivate a need for a localized, distributed algorithm that can compute broadcast trees without sacrificing performance. A localized algorithm is one in which nodes' decisions are based on network conditions within some limited distance. In this type of implementation, many nodes are simultaneously computing local parts of the tree, and use messages to coordinate activities with neighboring nodes. This results in considerably less computation time, message complexity, and power consumption as compared to a centralized algorithm. Such an algorithm would also use the *collective* resources of the network, avoiding the need to invest in costly high-resource nodes. For example, [19] presented a localized, distributed algorithm for the energy efficient unicast routing problem in networks with the same power-range tradeoff.

As part of our work, we have developed a localized, distributed algorithm that computes broadcast trees. In the first portion of the proposed distributed algorithm, nodes calculate a clustering on the graph. Clustering has been used as a strategy in many other ad hoc networking problems, including unicast routing, collision avoidance, and power control.

Clustering algorithms form groups of nodes, where each group contains one elected “clusterhead” node, responsible for coordinating activities on the behalf of the rest of the group. For example, clusterhead routing has been proposed to solve the unicast routing problem ([21], [11]) - packets are routed along a clusterhead “backbone” until reaching the clusterhead of the destination node’s cluster. Once there, the clusterhead sends the message directly to the destination node.

In our distributed algorithm, we attempt to generate a clustering that uses minimum energy while still assigning each node to cluster. Once this clustering has been computed, clusterheads are connected together to form a broadcast tree via an extension of a well known distributed algorithm ([9]) for computing directed MSTs.



# Chapter 2

## Complexity

We attempt to prove the following theorem.

**Theorem 1.** *BCAST is NP-complete.*

### 2.1 Relevant Background Work

Before presenting the theorems and algorithms (from other papers) that we used to prove *BCAST*'s NP-completeness, we go over some preliminary definitions for the purposes of clarity.

**Node Cover:** Given an undirected graph  $G = (V, E)$ , a **node cover** is a set of nodes

$S \subseteq V$  such that for every edge  $(i, j) \in E$ ,  $i \in S$  or  $j \in S$ .

**Connected Node Cover:** A **connected node cover** of a graph  $G = (V, E)$  is a node

cover  $S$  such that the graph induced by  $S$  on  $G$  is connected.

**Connected Dominating Set:** A **dominating set** of a graph  $G = (V, E)$  is a subset  $S \subseteq V$  such that every node in  $V$  is either in  $S$  or is a neighbor of a member of  $S$ . A **connected dominating set** is a set  $S$  such that the subgraph induced by  $S$  is connected, and  $S$  is a dominating set.

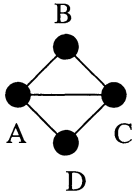


Figure 2-1. The set  $\{B, D\}$  is a node cover of this graph  $G$ .  $\{A, D\}$  is a connected node cover of  $G$ .  $G$  also has a connected dominating set  $\{C\}$ .

**Planar Graph:** A **planar graph**  $G = (V, E)$  is a graph that can be drawn in the plane without any edges overlapping. In other words, there exists a function  $\pi_1 : V \rightarrow \mathbb{R} \times \mathbb{R}$  such that if we draw a point at  $\pi_1(v)$  for all  $v \in V$ , and then draw a straight line segment from  $\pi_1(i)$  to  $\pi_1(j)$  in the plane for all  $(i, j) \in E$ , no line segments will cross. The function  $\pi_1$  is referred to as a **planar embedding** of the planar graph  $G$ .

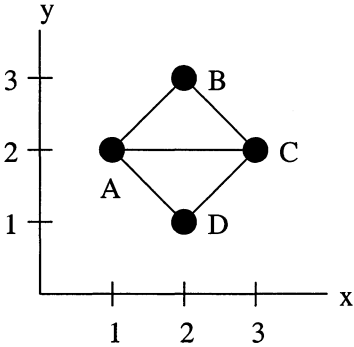


Figure 2-2.  $G$  is also a planar graph. This demonstrates a planar embedding for the graph  $G$ .

**Planar Orthogonal Grid Drawing:** Given a planar graph  $G = (V, E)$ , a **planar orthogonal grid drawing (POGD)** of  $G$  is a drawing on a grid such that each vertex is mapped to a grid point via some function  $\pi_2 : V \rightarrow \mathbb{Z} \times \mathbb{Z}$ , and each edge is mapped



to a sequence of horizontal and vertical grid segments, such that no two edges ever cross.

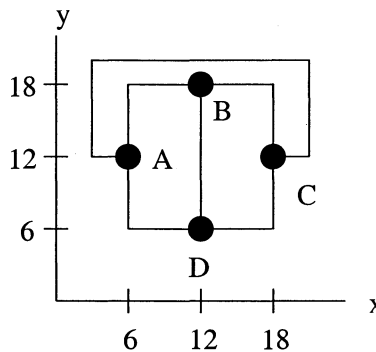


Figure 2-3. A POGD for the graph  $G$ .

**Unit Disk Graph:** A graph  $G = (V, E)$  is considered a **unit disk graph** if there exists a mapping  $\pi_3 : V \rightarrow \mathbb{Q} \times \mathbb{Q}$  to points on the two dimensional grid such that  $(i, j) \in E$   $\pi_3(i)$  and  $\pi_3(j)$  are less than distance 1 apart.

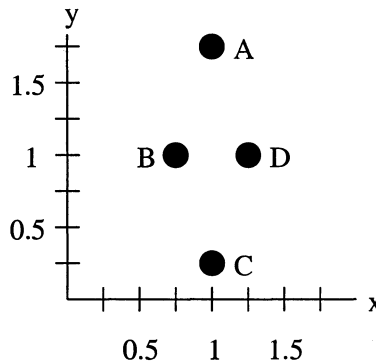


Figure 2-4. An example showing  $G$  is also a unit disk graph. This is a demonstration of  $\pi_3$ .

Having defined the terms above, we are now in a position to present the results of other papers used in our proof:

**Theorem 2. - NP completeness of Planar Connected Node Cover** *Given a planar graph  $G = (V, E)$  of maximum degree less than or equal to 4, determining the existence of a connected node cover  $V^* \subseteq V$  of  $G$  such that  $|V^*| \leq k$ , for some given  $k \in \mathbb{Z}^+$ , is NP-complete. Proved in [1]. Hereafter, we refer to this decision problem as PLANAR.*

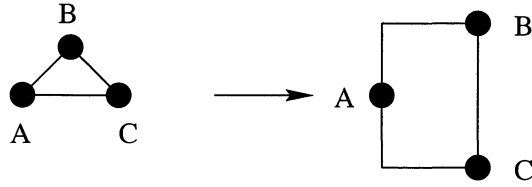


Figure 2-5. Constructing the POGD in Step 1 of the reduction.

**Theorem 3. - Orthogonal Grid Drawings of Planar Graphs** *Given a planar graph  $G = (V, E)$  of maximum degree less than or equal to 4, an orthogonal grid drawing of this graph can be drawn in polynomial time, such that the size of the grid is polynomial in  $|V|$ . Proved in [3].*

**Theorem 4. - Connected Domination in Unit Disk Graphs** *Finding a minimum sized connected dominating set of a unit disk graph is NP-complete. We refer to the decision version of the connected dominating set problem (i.e. “does there exist a connected dominating set of size no more than  $k$ ?”) as CDSUDG. We reproduce the reduction used in the proof of this theorem (from [4]) below.*

**Proof of Theorem 4:** We now describe the reduction used in Clark, et al. in [4] to prove Theorem 4.

Given an instance of *PLANAR*, with graph  $G = (V, E)$ , maximum node cover size  $k \in \mathbb{Z}^+$ , we convert it to an instance of the *CDSUDG* problem as follows.

1. We first construct the POGD of  $G$  using the algorithm mentioned in Theorem 3 (this is done on an example graph in Figure 2-5).

2. We then multiply the size of the grid by 6 so that each line segment of length one is mapped to a segment of length 6. This illustrated in Figure 2-6.

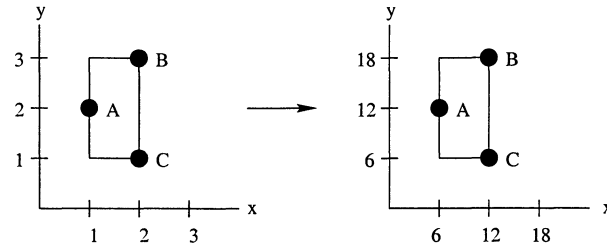


Figure 2-6. Multiplying the grid size in Step 2 of the reduction.

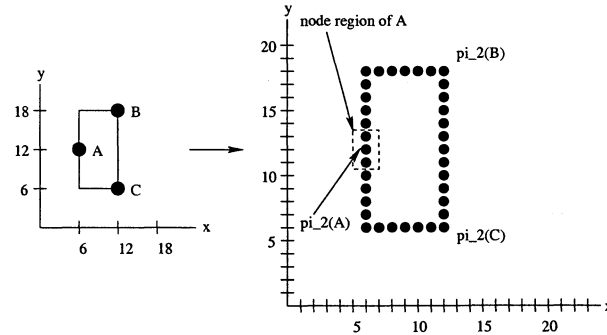


Figure 2-7. Step 3 of the reduction.  $P$  is the set of nodes in the graph on the right. The end nodes in  $A$ 's node region are at  $(6,11)$  and  $(6,13)$ .

3. Place a node at every grid point in the POGD, and denote this set of nodes as  $P$  (For example, if the line segment from  $(0,0)$  to  $(0,2)$  is in the orthogonal grid drawing,  $P$  contains nodes at positions  $(0,0)$ ,  $(0,1)$ , and  $(0,2)$ ). Note that each vertex  $v \in V$  in the instance of *PLANAR* maps to a node in  $p \in P$  such that  $\pi_2(v) = p$  (where  $\pi_2$  is the function in the definition of a POGD). For each  $p \in P$  such that  $\pi_2(v) = p$  for some  $v \in V$ , we refer to  $p$  and all nodes in  $P$  that are within 1 grid length of  $p$  as the **node region** of  $v$ . Those nodes that are in the node region by virtue of being within 1 grid length of  $p$  are called the **end nodes** of that node region. The other node (the one that is mapped to from  $V$  via  $\pi_2$ ) is referred to as the **center node** of this node region. See Figure 2-7.

4. We then construct the set  $P_l$  as follows. Construct the subset  $P' \subset P$ , which contains all nodes in  $P$  that are not in node regions. Also, for future reference, we denote the set  $P'' \subset P$  as the set of nodes in  $P$  that are not center nodes.  $P_l$  is then constructed such

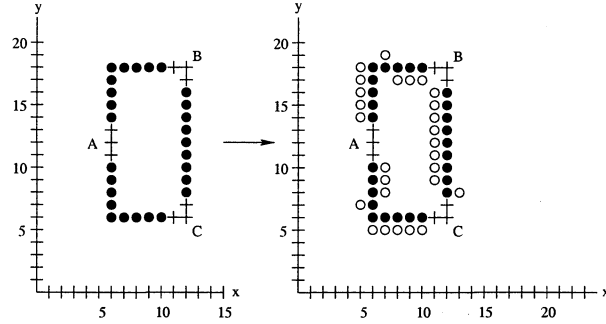


Figure 2-8. Step 4 of the reduction. Black nodes are nodes in  $P'$ , and  $P_l$  nodes are white. Nodes denoted with a “+” are in node regions.

that 1) each  $P_l$  node is placed at a grid point, 2) for each node in  $P'$  there is exactly one node in  $P_l$  located one grid length away, 3) for each node in  $P_l$  there is exactly one node in  $P'$  located one grid length away, and 4) no node in  $P_l$  is within one grid length of any node in  $P - P'$ . This operation effectively creates a “layer” of nodes around the original POGD’s edges, which is why we use the subscript  $l$ .

To complete the reduction, we construct a unit disk graph so that every node in  $P \cup P_l$  corresponds to a node in the unit disk graph, and edge  $(i, j)$  exists in the unit disk graph iff  $i$  and  $j$ ’s corresponding nodes are within distance 1 of each other. This completes the reduction used in Theorem 4.

Denote  $|V|$  as the total number of nodes in the original *PLANAR* instance, and  $|E|$  as the total number of edges. In the last step of the reduction in [4], the following lemma was proved:

**Lemma 1.** *There is a vertex cover of size no more than  $k$  in the original *PLANAR* instance iff there is a connected dominating set in the corresponding unit disk graph of size no more than  $|V| - |E| - 1 + k + |P''|$ .*

## 2.2 Proof of NP-completeness

We construct a reduction from *PLANAR* to *BCAST* inspired by the reduction used in Theorem 4 ([4]), showing that we can convert any instance of *PLANAR* into an appropriate instance of *BCAST* in polynomial time. We confirm the correctness of our transformation by showing that every positive instance of *PLANAR* maps to a positive instance of *BCAST*, and that every negative instance of *PLANAR* maps to a negative instance of *BCAST*. This demonstrates that *BCAST* is NP-hard. We go on to prove that it is NP-complete (thereby proving Theorem 1) by showing  $BCAST \in NP$ .

**The reduction** In proving the NP-hardness of *BCAST*, we can extend the reduction in [4] and use some of the properties derived there to prove the correctness of our reduction.

To extend the reduction in [4], we construct the *BCAST* instance from *PLANAR* instance as follows. First, we perform the reduction in [4] to an instance of *CDSUDG*. We then modify this reduction as follows:

Choose an arbitrary magnified POGD edge segment such that one end of the segment corresponds to a center node position (note that the POGD is magnified six times, so it must be 6 grid units long, and contain 6 nodes). Denote the first four nodes in  $P$  along this POGD segment (starting from the center node) as  $n_1, n_2, n_3$  and  $n_4$ . Hence,  $n_1$  corresponds to a center node, and  $n_2$  to an end node. Adjust the  $P_l$  nodes corresponding to  $n_3$  and  $n_4$  so that they are not within one grid length of each other. Note that this adjustment to the *CDSUDG* instance can be done for any  $P_l$ , while still satisfying the other conditions required of nodes in  $P_l$ . Therefore, the *CDSUDG* instance is still valid after this adjustment has been made,

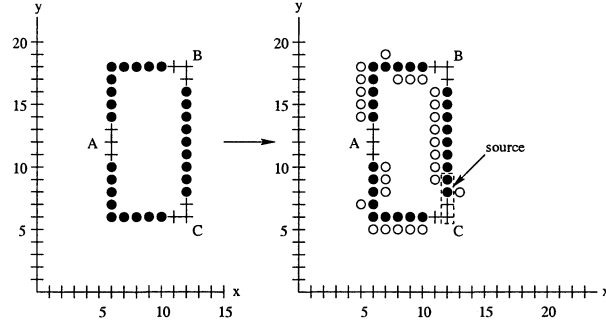


Figure 2-9. Step 4 of the reduction, with a box around the nodes  $n_1$  through  $n_4$ . Black nodes are nodes in  $P'$ , and  $P_i$  nodes are white. Nodes denoted with a “+” are in node regions.

and all proofs concerning the *CDSUDG* instance ([4]) still hold for this modified reduction.

We then extend the instance of *CDSUDG* to a corresponding *BCAST* instance as follows:

1. The nodes of the *BCAST* instance are the same as those in the modified *CDSUDG* instance (note that this is valid because each node in the generated *CDSUDG* instance is located at integer coordinates).
2. Set the source node of the *BCAST* instance,  $s$ , to be  $n_3$  from above. This is demonstrated in Figure 2-9, where the source is chosen to be at  $(12, 8)$ .
3. The range of each *BCAST* node is set to 1 grid length.

Note that even with the addition of these steps, the total time for the reduction is still polynomial.

**Proving NP-hardness from this reduction** Assume that we have taken a *PLANAR* instance and converted it to an instance of *CDSUDG*, and then extended the *CDSUDG* instance as noted above to construct an instance of *BCAST*. Then the following lemma holds:

**Lemma 2.** *There exists a BCAST tree of power no more than  $|V| - |E| - 1 + k + |P''|$*

*iff there exists a connected node cover of size no more than  $k$  in the original instance of PLANAR.*

**Proof:** Note the following observations about the *BCAST* instance constructed:

**Observation 1:** All neighbors of a node in the instance of *CDSUDG* are at distance *exactly* 1. Because the range of each node in the *BCAST* instance is 1, this implies that in any *BCAST* tree, a given node is either using 1 unit of power or 0 units of power. Therefore, we can consider a node in the *BCAST* instance as either being “on” or “off”.

**Observation 2:** The source node must be included in any connected dominating set in the generated instance of *CDSUDG* (because it is the sole node within one grid length of its corresponding  $P_l$  node).

**Observation 3:** Any connected dominating set *CDS* for the instance of *CDSUDG* can be mapped to a valid tree in the matching *BCAST* problem. To do so, turn on only those nodes in the *BCAST* instance that are in *CDS*. This is a valid *BCAST* tree because it includes the source  $s$  as turned “on” (by Observation 2), and for a given node,  $n$ , in the *BCAST* instance there is a path from  $s$  to  $n$  via “on” nodes (by virtue of *CDS* being a connected dominating set). Additionally, the number of elements in *CDS* is equal to the power used in the *BCAST* instance (by Observation 1). Therefore, every solution to the generated *CDSUDG* instance maps to a corresponding *BCAST* solution. We can also prove the converse statement. To prove this, note that the “on” nodes in a *BCAST* solution must constitute a dominating set (otherwise, there is a node which cannot be reached by the source in the *BCAST* solution, implying it is invalid). Additionally, in any valid *BCAST* tree, there is a path from the source to every “on” node. This implies the set of “on” nodes

is also connected. Therefore, we can map a *BCAST* solution to a *CDS* in the matching *CDSUDG* problem by selecting the set of “on” nodes. Note that the power used in the *BCAST* solution is exactly equal to the cardinality of the *CDS* that it maps to.

Observation 3 implies that there exists a connected dominating set of size no more than  $J$  in the *CDSUDG* instance iff the corresponding instance of *BCAST* contains a broadcast tree of power no more than  $J$ . This statement, taken together with Lemma 1, implies Lemma 2. ■

Lemmas 1 and 2 imply that we can map every instance of *PLANAR* to an instance of *BCAST* in polynomial time, proving that *BCAST* is indeed NP-hard. Also, note that the coordinates of each *BCAST* node generated this way have size that is at most a polynomial function in the number of nodes (because in [3], the size of the POGD is polynomial in the number of nodes). This fact further implies that *BCAST* is *strongly* NP-hard (see [2] for a definition of strong NP-hardness).

We show *BCAST* is strongly NP-complete, by demonstrating that  $BCAST \in NP$ . We show this by demonstrating that there is a polynomial time verifier for *BCAST*. Given a *BCAST* tree rooted at source  $s$ , with range  $R$ , and total power allotment  $l$ , we can compute whether this is a valid tree (no node transmitting beyond  $R$  distance, and all nodes reached via a directed path from  $s$ ), and also compute its total cost, in polynomial time. This is all that we must do to verify that a given *BCAST* tree is a valid solution for a specific instance of *BCAST*. This completes the proof of Theorem 1. ■



# Chapter 3

## Constructing Broadcast Trees

### 3.1 The Broadcast Incremental Protocol (BIP)

In light of the NP-completeness of *BCAST*, the fact that BIP achieves a constant approximation ratio is remarkable. It is even more impressive given the algorithm's simplicity. BIP ([10]) performs much like Prim's algorithm for constructing MST's ([20]). BIP grows a tree from the source, and augments the tree with the node that can be added with the *least additional cost*. It continues to add one node at a time until all nodes have been added to the tree. Once the tree is complete, a very simple "sweep" procedure traverses the tree and lowers power in cases of overlap, while still ensuring there is a path from the source to every node.

Throughout its execution, BIP maintains a set of nodes  $T$  that denote the tree made so far. Additionally, it maintains a power level  $p_i$  for each node in  $T$  (initially,  $T = \{s\}$  and  $p_s$  is set to 0). At each step, BIP attempts to increase the power of a node  $t \in T$  to reach a

node  $n \in N - T$ . Specifically, BIP increases the power of the node  $t$  that requires the least *additional* power to reach a node in  $N - T$  (the additional power for  $t$  to reach node  $n$  is  $Power(t, n) - p_t$ ). Once the node pair  $(n, t)$  that requires the least additional power has been identified,  $n$  is added to  $T$  with  $p_n = 0$ , and  $p_t$  is increased to  $Power(t, n)$ . This process continues until  $T = N$ .

Consider the following example, where we assume the power to transmit at distance  $d$  is exactly  $d^2$ :

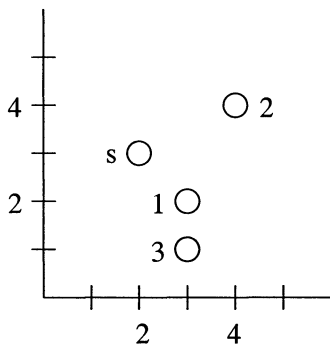


Figure 3-1. BIP example - the starting configuration.

Initially,  $T = \{s\}$  and  $p_s = 0$ , so the additional costs to add nodes 1, 2, and 3 from  $s$  are 2, 5, and 5 respectively. For this reason, node 1 is added to  $T$  and  $p_s$  is set to 2 (as shown in Figure 3-2).

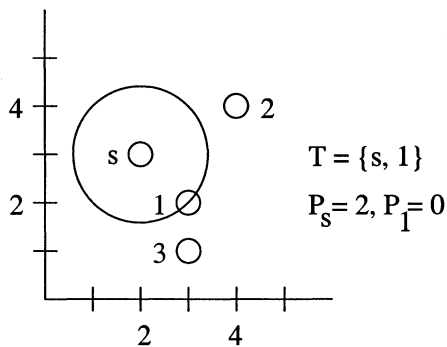


Figure 3-2. BIP example - Node 1 is added to  $T$ .

In the next iteration, the additional costs to add nodes 2 and 3 from  $s$  are  $5 - p_s = 3$ ,

and  $5 - p_s = 3$  respectively. Similarly, the additional cost to add nodes 2 and 3 from node 1 are 5 and 1 respectively. The minimum of these four values is the additional cost to add 3 via a power increase at node 1. Therefore, this power increase is chosen (see Figure 3-3).

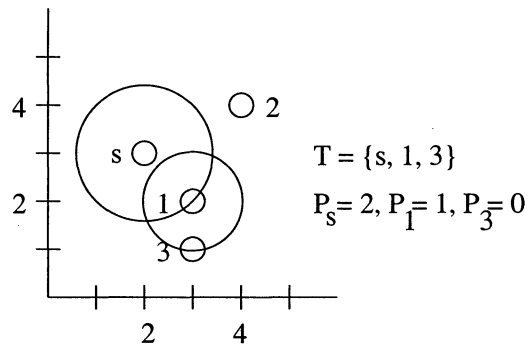


Figure 3-3. BIP example - Node 3 is added to  $T$ .

At this point, the additional cost to add node 2 from nodes  $s$ , 1, and 3 are 3, are  $5 - p_s = 3$ ,  $5 - p_1 = 4$  and  $10 - p_3 = 10$  respectively. Therefore node 2 is added by increasing the power of node  $s$  to 5, and the final tree is as represented in Figure 3-4.

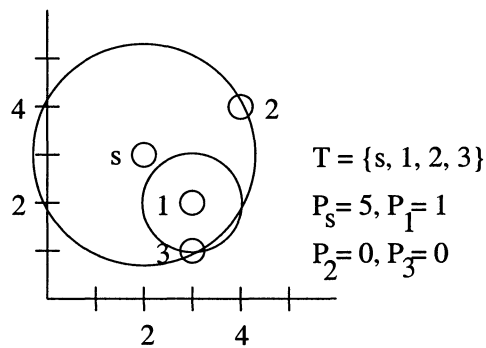


Figure 3-4. BIP example - Node 2 is added to  $T$  to and the BIP computation is complete.

Note that the tree we have made is a valid *BCAST* tree - there is a path from  $s$  to all nodes in the graph. Additionally, note that we never specified the range  $R_{max}$  - BIP assumes that nodes do not have a range limitation. That is, each node can always transmit to any other node as long as it uses enough power. As we look at improving BIP, we will focus on algorithms where we assume the BIP environment (no range limitation). However, when de-

veloping a distributed algorithm for the purposes of feasibility in real-world implementation, we will include the range restriction, as this is more representative of real-world transmitter limitations.

In our example, notice that the cost of the BIP tree can be improved, by observing that the node 1 need not be transmitting. This is because node  $s$  is already able to reach node 3 and node 1. Therefore we can turn off node 1 and still have a path to node 3 from  $s$ . This “overlap” situation commonly occurs in trees constructed via BIP. Consequently, the authors of BIP introduced a “sweep” algorithm ([10]). After BIP has constructed a tree, the sweep algorithm looks for situations similar to the one in our example, and reduces the power of certain nodes in the tree without making the tree invalid. In particular, sweep looks for the following general condition: Let  $h_i$  be the minimum number of hops needed to receive a message from  $s$  at node  $i$  in the given tree, and refer to the condition  $p_i \geq \text{Power}(i, j)$  as  $i$  covering  $j$ . Then, if there are nodes  $i, j$  and  $k$  such that 1)  $h_i > h_j$ , 2)  $i$  covers  $k$  and 3)  $k$  is the farthest node from  $j$  such that  $j$  covers  $k$ , then reduce the power of  $j$  so it no longer covers  $k$ . In other words, if there is an upstream node  $i$  and a downstream node  $j$  both covering the same node, then sweep checks to see if  $j$ 's power can be reduced (note that the upstream/downstream requirements ensure that  $s$  will still have a path to this “doubly covered” node after the power decrease). This algorithm significantly improves the average cost of trees generated by BIP.

### 3.1.1 Improving BIP: Hybrid Algorithms

As said previously, BIP is an astonishingly simple greedy algorithm. Especially given the NP-completeness of the *BCAST* problem, it is surprising that BIP does not have to exhaustively try out a series of choices at each step to determine which will be most beneficial at later iterations. Even without such behavior it still performs well. Consider what would happen if we performed a limited exhaustive search in the earlier stages of the algorithm, and then allowed BIP to continue with greedy choices after some point. This is the sort of algorithm we had in mind when constructing a BIP hybrid.

In this algorithm, which we call *BIPHybrid<sub>1</sub>*, an initial power assignment and partial tree  $T$  is constructed, which is then “completed” by running BIP. To construct a tree, we assign the source a power level. Once this power level has been chosen,  $p_s$  is set to this power, and  $T$  is set to include the source and all nodes reached by the source at this power (all non-source nodes that are in  $T$  are set to have zero power initially). Then, BIP is run with this starting  $T$  and power assignment. Once the BIP algorithm terminates, we have constructed a tree. In *BIPHybrid<sub>1</sub>*, this procedure is repeated for all  $N - 1$  possible power levels that the source may be set to. Then the minimum power tree (among the  $N - 1$  trees thus generated) is returned.

We also looked at an extension of *BIPHybrid<sub>1</sub>*, which we refer to as *BIPHybrid<sub>2</sub>*. In this algorithm, a similar procedure is performed, except the initial tree  $T$  is made by setting the power of the source, and then setting the power of a node that is reachable from the source. Then BIP is run to “complete” this initial tree. This procedure is then repeated for all possible initial trees (in which two nodes are on). The lowest energy tree is then

returned. Note that this means that the cost of the *BIPHybrid<sub>2</sub>* tree is never more than the cost of the *BIPHybrid<sub>1</sub>* tree (for the same problem instance).

By looking at these two algorithms, we can get a sense of how far BIP is from the optimal solution. If we notice that the cost savings (compared to BIP) grows very quickly going from *BIPHybrid<sub>1</sub>* to *BIPHybrid<sub>2</sub>*, we have reason to believe that BIP constructs trees with cost much higher than the optimal solution. Likewise, if the cost savings does not grow quickly, we have a rough “sense” that BIP performs close to optimal on average. We simulated the performance of BIP, *BIPHybrid<sub>1</sub>* and *BIPHybrid<sub>2</sub>* for various size networks confined to a 1x1 grid. After each algorithm was run, the BIP “sweep” algorithm was run on the tree.

As we can see from Figure 3-5, *BIPHybrid<sub>1</sub>* is a 5% improvement on BIP, and this does not change as a function of instance size. This indicates that the power of the source’s transmission can have a significant impact on power efficiency. Note that *BIPHybrid<sub>2</sub>* doesn’t add much additional improvement - it averages to about another 2% savings over *BIPHybrid<sub>1</sub>*. Unfortunately, due to the computational resources required, we couldn’t measure the performance of *BIPHybrid<sub>2</sub>* for larger instances. However, the lack of additional savings going from *BIPHybrid<sub>1</sub>* to *BIPHybrid<sub>2</sub>* seems to indicate that BIP does not significantly deviate from optimal in the average case.

## 3.2 Computing Broadcast Trees Distributively

In this section, we describe a localized, distributed algorithm that computes broadcast trees. In the first portion of the proposed distributed algorithm, nodes calculate a clustering on the graph. Then, the clusters are joined together using a well known distributed algorithm

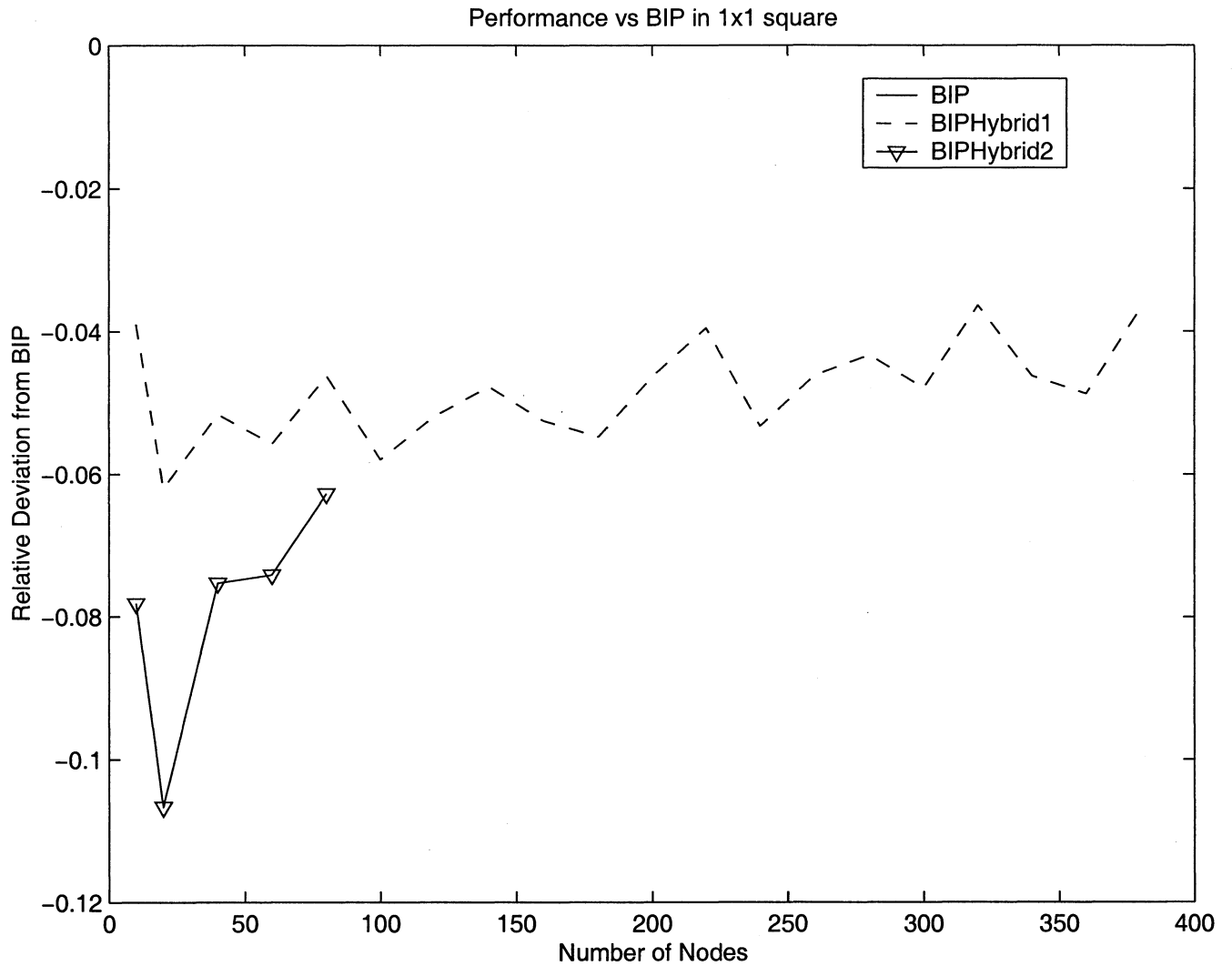


Figure 3-5. Performance of  $BIPHybrid_1$  and  $BIPHybrid_2$  as compared to BIP.

for computing directed minimum spanning trees.

At the beginning of the algorithm we assume each node has the following information:

1. Each node  $i$  knows the distance to every node in  $i$ 's neighborhood. A node's **neighborhood** is defined as the set of nodes that are within distance  $R$  (the maximum distance that a node can transmit a message). Nodes that are in  $i$ 's neighborhood are referred to as **neighbors** of  $i$ .
2. Each node  $i$  also knows the distance of each neighbor to every node in the neighbor's

neighborhood. We refer to the set of  $i$ 's neighbors, and  $i$ 's neighbors' neighbors as  $i$ 's **two-hop neighborhood**.

As an example, this information could be gathered by determining pairwise node delay via timestamps. Notice that each node only requires *localized* information about some small portion of the network (the two hop neighborhood). This is a key difference from previous algorithms like BIP, which require each node to have global network information. Also, note that this is only meaningful in networks with limited range - if each node had unlimited range, having two-hop neighborhood information would be equivalent to having global information. In networks with limited range however, the two-hop neighborhood may constitute a small fraction of the graph. Of course, this also holds for networks in which the range of nodes is limited for reasons other than inherent transmitter limitations (like avoiding interference).

In our initial development of a distributed algorithm, we assume that the network is synchronized via a global clock, no messages are lost, and that there is no interference. We then extend the algorithm to work without the benefit of a global clock in networks where interference and packet loss is possible.

### 3.2.1 The formation of clusters

In the first phase of the algorithm, a clustering is constructed on the nodes using the aforementioned distance information. Once this phase is complete, each node will be assigned to at least one cluster, and each cluster will have one "clusterhead" node. We define the cost of a particular cluster as the power required for the clusterhead node to transmit to all other nodes in the cluster (in one transmission). In addition, we consider the cost of a particular



clustering to be the sum of the costs of its clusters. Given this cost function, we attempt to develop a minimum cost clustering.

Before describing our distributed clustering algorithm, we first describe a centralized clustering scheme. A simple centralized algorithm can be described as follows: Throughout the execution of the algorithm, each node  $i$ 's range is referred to as  $r_i$ , and each node is either unmarked or marked, reflecting its membership in a cluster. The algorithm begins with  $r_i = 0$  for all  $i$ , and all nodes unmarked, and proceeds as follows:

1. For each node  $i$ , compute the function  $\alpha_i(r)$ . If  $i$  was to increase its range to  $r$ , this function represents the average cost induced per unmarked node within distance  $r$  of  $i$ . More precisely,

$$\alpha_i(r) = \frac{P(r) - P(r_i)}{U_i(r_i, r)}$$

$$r_i < r \leq R$$

where  $P(x)$  = power to transmit at range  $x$

$U_i(x_1, x_2)$  = number of unmarked nodes in between distances  $x_1$  and  $x_2$  of  $i$

$r_i$  = node  $i$ 's present transmission range

2. For each node  $i$ , compute the range at which  $\alpha_i(r)$  is minimized. This is the most cost efficient range increase (in a greedy sense) for node  $i$ . Denote this range as  $rmin_i$ , and the value of  $\alpha$  at this range as  $amin_i$ .
3. Find the node  $j$  that has the smallest value of  $amin$  - this is the node that (globally) has the most cost efficient range increase. Increase  $r_j$  to  $rmin_j$ , and mark nodes  $j$  and

all nodes within distance  $r_{min_j}$  of  $j$ .

4. Repeat steps 1-3 until all nodes are marked.

Once the above algorithm terminates, the final  $r_i$  values specify a clustering (because nodes are only marked if they belong to a cluster). Each node with nonzero  $r_i$  is considered a clusterhead, and all nodes within distance  $r_i$  of  $i$  are considered members of  $i$ 's cluster (note that, in our algorithm, a particular node may be a member of more than one cluster). Because we greedily choose the range increase that minimizes the *average* additional cost induced per node marked, we are hopeful that the clustering produced is cost-efficient.

Interestingly, the greedy behavior of the above algorithm can be implemented distributively, with one important difference. In a distributed implementation it is very costly (in message complexity) to compute a global minimum - therefore, the distributed algorithm (described below) attempts to find *local* minima instead. Although this may result in less power efficient clusterings, such inefficiencies are inherent to many localized algorithms.

### 3.2.2 Synchronous Distributed Clustering Algorithm

As in the global algorithm, each node  $i$  maintains a range value  $r_i$  initially set to 0, and is initially unmarked. Additionally, we ensure that each node  $i$  maintains up-to-date values of: 1)  $r_j$  for all neighbors  $j$ , and 2) MARKED status of all nodes in the two-hop neighborhood. The algorithm we propose operates in stages. During each stage, local minima are computed, and the ranges of some nodes are consequently increased. Information about the range increases are then propagated. Once this has been completed, each node has updated its state information to reflect the last stage's changes, and the next stage begins.

Note that because we are assuming that all nodes are synchronized, we describe the algorithm in stages and substages where each stage and substage begin on predefined clock boundaries. In each stage, each node executes the following substages:

**Substage 1.** If  $i$  is unmarked, node  $i$  computes, for each neighbor  $j$ , the minimum value of  $\alpha_j(r)$  for  $r \geq \text{distance}(i, j)$ . That is,  $i$  finds the most cost efficient range increase for  $j$ , looking only at those ranges that would allow  $i$  to be a member of  $j$ 's cluster. Denote the value of the range and  $\alpha$  found through this computation as  $rmin_{j \rightarrow i}$  and  $amin_{j \rightarrow i}$ , respectively.

Each node  $i$  then finds the neighbor node  $k$  with minimum value of  $amin_{k \rightarrow i}$ , and sends  $k$  a PREFERRED message containing range value  $rmin_{k \rightarrow i}$ . This message is sent at maximum power.

If  $i$  is marked already, it does not participate in this substage.

In addition each node  $i$  (marked or unmarked) executes the following steps:

**Substage 2.** At this substage,  $i$  has received all PREFERRED messages from its unmarked neighbors. If  $i$  receives a PREFERRED message with range value  $r'$  from all unmarked nodes within distance  $r'$  (indicating that  $i$  is a local minima),  $i$  increases  $r_i$  to the value  $r'$ . Upon increase, it transmits a RANGE\_INCREASE message at maximum power, telling all neighbor nodes that  $i$  has increased its range to  $r'$ . If  $i$  is not already a member of a cluster, it also broadcasts a MARKED\_STATUS message to its two-hop neighborhood, indicating that  $i$  has been marked (by virtue of becoming a clusterhead).

**Substage 3.** If  $i$  receives a RANGE\_INCREASE message from a neighbor  $j$  such that

the distance from  $i$  to  $j$  is less than the new value of  $r_j$ ,  $i$  is a member of  $j$ 's cluster. Consequently, if  $i$  is not already a member of another cluster, it broadcasts MARKED\_STATUS message to its two-hop neighborhood, indicating that  $i$  has been newly marked. Additionally, at this substage,  $i$  may receive a MARKED\_STATUS message from a neighbor that has just become a clusterhead (in the previous substage). It forwards this message at maximum power (to ensure that it goes to all nodes two hops away from the clusterhead).

**Substage 4.** At this substage, each node  $i$  may receive a MARKED\_STATUS message from a newly marked neighbor  $j$ . It retransmits this message at maximum power, to ensure that it reaches  $j$ 's two hop neighborhood. At the next substage, all messages will have reached their intended receivers. Therefore, in the next substage, each node  $i$  will have up to date information on  $r_j$  for all neighbors  $j$ , and the MARKED status of every node in the two hop neighborhood. After this substage, a new stage begins.

The algorithm terminates once all nodes have been marked (and hence no PREFERRED messages are being generated). Because nodes only mark themselves when they have become a member of a cluster, this also means that, upon termination, the final values of  $r_i$  produce a clustering. Note the following properties of this algorithm:

- A. **The algorithm terminates in a linear number of stages.** Consider any stage of the algorithm where not all nodes have been marked. We show that at least one new node will be marked in this stage. Let  $amin_i$  be the minimum value of  $\alpha_i$  for node  $i$ . There must exist some node  $j$  for which  $amin_j$  is minimum over all nodes. Because this is the *global* minimum, in the next stage, all unmarked nodes within distance  $rmin_j$

of  $j$  will send  $j$  a PREFERRED message with range value  $rmin_j$  (if not,  $amin_j$  would not have been the global minimum). Therefore, in the next stage,  $j$  will increase its range, and some set of previously unmarked nodes will be newly marked. This further implies that in every stage, at least one node is marked, completing the proof.

**B. The algorithm uses  $O(N^2)$  messages.** In each stage, there is at most 1 PREFERRED message and 1 RANGE\_INCREASE message per unmarked node, resulting in  $O(N)$  messages per node per stage, and  $O(N^2)$  messages total. Additionally, each node transmits a MARKED\_STATUS message when it has been marked (this occurs once per node throughout the algorithm). Because each node has two-hop neighborhood information, it can compute a spanning tree upon which to forward this MARKED\_STATUS message. Therefore, it takes at most  $O(N)$  messages to forward the MARKED\_STATUS message to the two-hop neighborhood. Hence, we have at most  $O(N^2)$  total MARKED\_STATUS messages, and  $O(N^2)$  total PREFERRED/RANGE\_INCREASE messages.

### 3.2.3 Implementation Considerations

Although this synchronous algorithm works fine when there is a global clock and we assume no messages are lost or reordered, this is not at all a reasonable expectation of real-world environments. In the real world, keeping global clocks up to date requires considerable message complexity and node coordination. Additionally, messages can be lost in wireless communication, requiring retransmissions that cause arbitrary message delays.

To extend our clustering algorithm to work without a global clock in the presence of

arbitrary message delays (but not message loss) we can note the following. In the absence of a global clock, the sole difference from the synchronous case is that different nodes might simultaneously be at different stages of the algorithm (one node might be in stage 4, substage 2, while another is at stage 7, substage 1). Because nodes are “out of sync”, this could lead to node decisions being made with incomplete or conflicting state information, resulting in a different tree (different from the synchronous algorithm). We can remedy this situation by using the receipt of a message as a *virtual* clock signal, implemented with the following rule:

Each node  $n$  is not allowed to enter a particular substage until *all messages that could conceivably be destined for  $n$*  (and sent off by other nodes as the result of a previous substage) have been received.

This ensures: 1) each node doesn't make a decision until all state information from previous stages has been received, and 2) no node can be more than one stage ahead of its neighbors.

Note that in a network where all messages are guaranteed to arrive at their destination within  $\delta$  seconds, this rule would be equivalent to requiring each node to wait  $\delta$  seconds before proceeding to the next substage. In such a network, this behavior would ensure the algorithm constructs the same tree as in the synchronous case, solely because the message schedules are identical (and consequently, nodes are not out of sync). Furthermore, note that the implementation of this rule in *any* network (with any message delivery guarantee), would result in an identical message schedule, and consequently result in the same tree produced by the synchronous algorithm.

To extend our algorithm to cases where messages are arbitrarily delayed, we develop an

algorithm that obeys this rule. We can do this as follows. Assume that every node has up to date information from the previous stage (values of  $r_i$  for each neighbor, and MARKED status of each node in the two hop neighborhood). Then, each node does the following:

1. In the beginning of the stage, each unmarked node sends a PREFERRED message as in the synchronous algorithm. Marked nodes do not perform any operation. Each node is then not allowed to continue until it has received *all* PREFERRED messages from its unmarked neighbors.
2. Once this has occurred, if this node is going to increase its range, it sends off a RANGE\_INCREASE message as in the synchronous algorithm. If it will not increase its range, the node sends a message to indicate this. As before, this node is not allowed to continue to the next step until a message is received from *every* neighbor node pertaining to the neighbor's range decision.
3. Once this node is allowed to continue, it sends off a MARKED\_STATUS message *only* if it was unmarked in the last stage. This message will indicate if this node was marked in this stage. The node then suspends its operation until it has received a MARKED\_STATUS message from every unmarked neighbor.
4. Once its operation is allowed to continue, this node composes and sends a STATUS\_SUMMARY message, which lists all nodes that have been marked in this stage (this ensures that a MARKED\_STATUS update from a node travels two hops). Note that if we immediately forwarded the MARKED\_STATUS message without composing a summary, this could potentially mean that each node sends off  $O(n)$  messages in

this step (increasing the overall message complexity to  $O(n^3)$ ). However, if we compose a summary, this is reduced to 1 message sent per node, and an overall message complexity of  $O(n^2)$ .

5. This node is then not allowed to enter the next stage until a STATUS\_SUMMARY message has been received from all neighbors (note that once this is done, the node has received all messages destined for it in this stage, and therefore has up to date information and is prepared for the next stage).

This algorithm ensures that no two neighbor nodes are out of sync by more than one substage, regardless of message delay. Note that the algorithm obeys the above rule, and therefore constructs the same tree as the synchronous algorithm.

Since the above algorithm tolerates messages delivered with arbitrary delay, it can further be extended to tolerate networks where messages can be lost through the use of a link-layer retransmission protocol (ARQ). Such a protocol guarantees the eventual delivery of packets, although the delivery time of the packet may vary based on the need for retransmission. However, since the above algorithm can tolerate packets that are arbitrarily delayed, we are assured that it will terminate successfully.

### 3.2.4 An alternate clustering algorithm

Although the above algorithm can tolerate interference and message loss, it may do so at the cost of many ARQ retransmits. This is especially of concern since most messages are being sent within each node's two hop neighborhood, increasing the likelihood for message interference and subsequent message loss. In addition to wasting energy resources via retransmits,



frequent message loss/interference would also increase the time required to compute a clustering. For these reasons, we investigated the possibility of an algorithm that makes decisions based on potentially incomplete or out-of-date local information (without having to wait for messages to be retransmitted). Although this algorithm doesn't compute the same clustering as the synchronous clustering algorithm above, it computes a clustering much faster in the average case. Each node has the following behavior:

1. Regardless of its status, node  $i$  continually broadcasts a STATE message containing its marked status, the marked status of every neighbor, and its current range  $r_i$ , every  $\delta$  seconds. If  $i$  is unmarked, this message also contains  $i$ 's PREFERRED neighbor information.
2. If  $i$  ever receives a PREFERRED message with range value  $r'$  from all unmarked nodes within distance  $r'$  (this is a local minima),  $i$  increases  $r_i$  to the value  $r'$ , and updates its STATE message accordingly.
3. If  $i$  notes that a neighbor has been marked, or that a neighbor has increased its range so that  $i$  has been marked,  $i$  updates its STATE message (this is a way to propagate news of a marked node two hops).
4. Once every node in  $i$ 's two hop neighborhood is marked,  $i$  is no longer needed to forward changes in marked state, and is not a candidate to increase its range. Therefore, it stops sending STATE messages.

Note that once all nodes have stopped sending STATE messages, every node is marked. Additionally, in step 2, each node doesn't worry about receiving messages that may corre-

spond to different stages - instead, it makes a “dirty”, quick decision based on the entire history of received messages. Therefore, it is very likely that the clustering made by this algorithm will be different than that of the algorithm presented in section 3.2.2.

This algorithm is guaranteed to terminate even in the presence of losses (as long as each message has a nonzero probability of being transmitted successfully). We can sketch a proof of this statement as follows. Note that, just as in the proof of convergence for the algorithm in section 3.2.2, whenever all nodes are not marked, there must always exist a range increase that has the globally minimum average cost. Therefore, if no node in the network is prepared to increase its range, it can only be because STATUS updates have not propagated because of message losses. However, these updates are repeated periodically, so eventually the updated status will propagate, nodes will update their status, and the global minimum will be realized (i.e. nodes that had not sent a PREFERRED message corresponding to the global minimum range increase will eventually receive status updates, which lead them to send this PREFERRED message). Hence, some node will eventually increase its range, and at least one node will be marked. This means that eventually, all nodes must be marked and the algorithm will terminate.

### **3.2.5 A Clustering Sweep Procedure**

Note that in the clustering produced by any clustering algorithm, it is quite possible that a node is simultaneously a member of more than one cluster. That is, clusters may overlap. As in the BIP procedure, there is a similar opportunity to implement a “sweep”-like algorithm ([10]). This “sweep” goes through the clustering in a distributed manner, and finds nodes

whose range can be reduced, while still making sure every node is still a member of at least one cluster. The ranges of these nodes are then reduced to produce a lower power clustering. We implemented a very simple cluster sweep procedure which performs this operation. For simplicity purposes, we present the algorithm in a network where packets are not lost, but may arrive out of order (in which case, a link-level ARQ protocol can be used to deal with losses as mentioned above).

Before the running of the “sweep” algorithm, each clusterhead node (node with non-zero clustering range assignment) computes a list of clusterhead “neighbors”. Clusterheads are considered neighbors if one node is in both of their clusters. This can be accomplished by initially requiring each node to broadcast (at full range) a list of the clusters that it is a member of. Any two neighboring clusterhead nodes will receive a message from their shared node, and each will be aware that the other clusterhead is a neighbor.

Throughout the execution of the algorithm, each node  $i$  maintains a status of marked or unmarked (indicating whether or not this node has reached its final range assignment), and a current range referred to as  $r_i$ . If we refer to  $c_i$  as the number of clusters of which  $i$  is a member (where a clusterhead is counted as a member of its own cluster), each clusterhead node  $i$  also maintains up to date values of  $c_j$  for all members  $j$  of  $i$ 's cluster. The algorithm works very similarly to the algorithm in section 3.2.2 - in each stage, local extrema are computed (in this case we are comparing the local maxima - the clusterhead node that can save the most power), ranges are lowered, and status changes are propagated before proceeding to the next stage. During the algorithm, ranges are only *lowered* (this will be important in proving convergence). At each stage of the algorithm, each clusterhead node

goes through the following steps:

1. Each clusterhead  $i$  determines the node in its cluster that is furthest away from it but not further than  $r_i$ . Refer to this node as  $j$ . If  $c_j = 1$ ,  $i$  is the only cluster head covering  $j$ . Therefore,  $i$ 's range cannot be lowered further (because ranges are only lowered, no other clusterhead will be able to cover  $j$ ). In this case,  $i$  marks itself, and propagates a MARKED\_STATUS message to each clusterhead neighbor.

If  $c_j > 1$ ,  $i$  may potentially be able to lower its power while still allowing  $j$  to be a member of a cluster. To determine how much power can be saved,  $i$  looks at its fellow cluster members in order of decreasing distance from  $i$ , and proceeds until reaching a node  $k$ , where  $c_k = 1$ . Note that node  $i$  cannot reduce its range any lower than  $d_{i \rightarrow k}$ , the distance from  $i$  to  $k$ . Node  $i$  sends a POWER\_SAVED message to each unmarked clusterhead neighbor indicating the power saved is  $r_i^\alpha - d_{i \rightarrow k}^\alpha$ .

2. Before entering this step, each clusterhead node  $i$  waits until it has received a POWER\_SAVED or MARKED\_STATUS message from *every* unmarked clusterhead neighbor. Once it has done so, if  $i$ 's POWER\_SAVED value is the maximum among its clusterhead neighbors (a local maxima),  $i$  reduces its range value to  $d_{i \rightarrow k}$ . Node  $i$  then sends a RANGE\_UPDATE message to each unmarked clusterhead neighbor, indicating its new range value (it does this even if its range value has *not* changed). Note that if a node ever decreases its range it is also marked (it has reduced its range as much as possible), so this message contains two pieces of information in the event of a range decrease.

3. Each clusterhead node  $i$  waits until receiving a RANGE\_UPDATE message from *every*

unmarked clusterhead neighbor before entering this step. Because it has two hop distance information,  $i$  can then compute updated values of  $c_j$  for all members  $j$  of  $i$ 's cluster (note that if a neighboring clusterhead node decreases its power, it may be possible that  $c_j$  values change). Once it has done this, node  $i$  now has up to date information about what occurred in this stage, and is ready to proceed to the next stage.

Once all clusterhead nodes are marked, there is no more potential for ranges to be decreased, and the algorithm terminates. Note that the  $r_i$  values at the end of this algorithm denote a clustering - because we start out with a clustering, for every node  $i$ ,  $c_i \geq 1$  at the start of the algorithm. Additionally, throughout the algorithm, we make sure that  $c_i$  never falls below 1, meaning that every node must be in a cluster once the algorithm is completed. Note the following interesting properties of this algorithm.

- A. **This algorithm converges in  $O(n)$  stages.** Assume that in the present stage, not all clusterhead nodes have been marked. Then there must exist some clusterhead node such that its POWER\_SAVED value is the maximum POWER\_SAVED value among all nodes in the graph. This node must mark itself in this stage (once it receives the POWER\_SAVED values of its clusterhead neighbors). Therefore, at least one clusterhead node must be marked per stage, and the proof is complete.
- B. **Each stage requires  $O(n^2)$  messages.** Note that each clusterhead node can transmit to a neighboring clusterhead node in at most two hops. We could naively implement this by sending a message from the clusterhead (at maximum power), which is imme-

diately forwarded by all who receive the message (at maximum power). This would require  $O(n)$  messages per clusterhead per stage, and  $O(n^2)$  messages per stage total.

C. **We can improve this to  $O(n)$  messages per stage.** This can be done by splitting each individual step where a clusterhead sends a message to its clusterhead neighbors into two separate steps. In the first step, each clusterhead node transmits the message at maximum power. All nodes (including those that are not clusterheads) are required to wait for the receipt of this message from *each* of its neighboring clusterheads before going to the next step. Once a node can enter the next step, it sends a summary message (summarizing all the messages it received in the last step) at maximum power. Each of the two steps consumes  $O(n)$  messages, implying each of steps 1-3 above can be done with  $O(n)$  messages total per stage.

D. **The algorithm requires a total of  $O(n^2)$  messages.** A and C together imply that the algorithm takes  $O(n^2)$  messages. Additionally, the preliminary step (where clusterhead neighbors are determined), requires  $O(n)$  total messages because each node transmits one message.

E. **We can adjust the algorithm to allow clusterhead nodes to turn off entirely.** We adjust step 1 to check for the situation where a clusterhead can turn off entirely - in the case that  $k$  is the closest node to clusterhead node  $i$ ,  $i$  can be turned off if  $c_k \geq 2$  and  $c_i \geq 2$ . If  $i$  can be turned off, its POWER\_SAVED is  $r_i^\alpha$ . The behavior of steps 2 and 3 remain the same.

Note that in the above algorithm, each clusterhead node either reduces its power by a

large amount or doesn't change its power at all. This is directly a consequence of step 1, where we find the node  $k$  farthest from the cluster such that  $c_k = 1$ . If instead, in each stage, a clusterhead node decreases its power by a small increment (reduce power to reach only the second farthest covered node) until no node can reduce power any further, this may result in an improved cluster sweep (at the expense of more computation). Instead of the "all or none" behavior, this variant would allow the power decrease to be spread out among clusterhead nodes. We investigated this possibility, and found that this additional flexibility gave no distinguishable improvement. For this reason, we rely solely on the more efficient "all or none" clusterhead sweep algorithm.

### 3.2.6 Joining Clusters Together

After a clustering has been made, we use a well known distributed algorithm for constructing directed minimum spanning trees [9] to join the clusters together. Specifically, we do the following:

1. Construct the directed graph  $G' = (V', E')$  where  $V' = V$ , and the cost of each edge  $(i', j')$  is equal to  $\max(0, P(\text{distance}(i', j')) - P(r_i))$  where  $r_i < R$ , and  $P(x)$  denotes the power to transmit at a range  $x$ . This represents the incremental power required to establish a link from  $i'$  to  $j'$  after the clustering has been performed.
2. Once the cost of each edge has been computed, we run the algorithm for computing a directed minimum spanning tree on  $G'$  for source  $s$ . By definition of the directed spanning tree, we will have constructed a broadcast tree rooted at  $s$ .

Our distributed algorithm first computes a clustering, sweeps the clustering, and then runs the DMST algorithm to join the clusters together. Note that the algorithm in [9] computes the DMST rooted at *every* node with  $O(n^2)$  message complexity. Therefore, our algorithm computes the broadcast tree rooted at every node *simultaneously*.

### 3.3 Simulation Results

To gauge the performance of our algorithm against BIP, we simulated the performance of BIP, our distributed algorithm, and several other candidate algorithms in networks restricted to the 1 by 1 unit square. In simulating these algorithms at a particular network size, we first constructed a set of 100 instances, each having the same number of nodes. For each instance, nodes were randomly placed with uniformly probability in the unit square, and one node was randomly chosen to be the source. Each algorithm was then run on each of the 100 instances. After a tree was computed from any algorithm, the sweep procedure from [10] was run on the tree before comparison.

**OptPartition** One of the candidate algorithms we looked at is called OptPartition. In this algorithm, the 1 by 1 square is recursively partitioned using the procedure SqPart. Given a square, SqPart does nothing if the square has at most 10 nodes. Otherwise, SqPart splits the square into 4 smaller, equal-sized squares, and attempts to partition each of them recursively. This algorithm can be used to generate a partitioning of the unit square such that each partition contains no more than 10 nodes.

An exhaustive search algorithm is then run to find the optimal tree within each partition



(looking at every node as a possible source). This means that if a partition contains 5 nodes, 5 optimal trees are computed, and each tree contains only the 5 nodes in the partition. We refer to the optimal tree rooted at node  $n$  and covering the nodes in the  $n$ 's partition as  $Opt(n)$ . OptPartition then runs an algorithm very similar to BIP, except instead of adding one node at a time, it adds one partition at a time.

Initially a tree  $T = \{s\}$  and  $p_s = 0$ . We add all nodes in  $s$ 's partition to  $T$ . Additionally, each  $p_i$  is set to correspond to  $i$ 's power in  $Opt(s)$ . Nodes are continually added to  $T$  as follows:

1. The additional cost to reach a node  $n$  in  $N - T$  from a node  $t$  in  $T$  is set to be  $Power(t, n) - p_t + Power[Opt(n)]$ , where  $Power[Opt(n)]$  is the cost of  $Opt(n)$ .
2. The node of least additional cost is chosen. The value of  $p_t$  is set to  $Power(t, n)$ . The value of  $p_i$  for all nodes in the  $n$ 's partition are set to their values in  $Opt(n)$ . All nodes in  $n$ 's partition are added to  $T$ .
3. If  $N = T$ , the algorithm stops. Otherwise, steps 1 and 2 are repeated.

**DistClusterBIPJoin** In this algorithm, the algorithm from section 3.2.2 is used to generate a clustering. Then BIP is run, except when a node  $t$  is first added to  $T$ , and it has a power  $cl_t$  assigned to it from the clustering algorithm,  $p_t$  is set to  $cl_t$  instead of 0. Intuitively, in DistClusterBIPJoin, the clustering algorithm develops a clustered framework which BIP “joins” together during its execution.

**BLAIP** This algorithm is referred to as the Broadcast Least Average Incremental Protocol, and is a variant of BIP. Instead of adding just one node to  $T$  at a time, where the additional cost is minimum, BLAIP adds several nodes to  $T$  at each step. At each iteration, the power of one node in  $T$  is increased to cover several nodes, where the *average* additional cost of adding the nodes is minimized. For this purpose, the average additional cost of adding a set of nodes  $G \subseteq N - T$  from a node  $t$  is computed as  $\frac{\max_{g \in G} \{Power(t,g)\} - p_t}{|G|}$ . This is an attempt to preserve the greedy behavior of BIP, but assert an altered greedy choice that reflects the “multicast advantage.”

The results of our simulation (shown in Figure 3-6) display how our distributed algorithm (referred to as DistClusterDMSTJoin in the figure) compared relative to the cost of the BIP tree, as averaged over the 100 instances. This graph also includes the performance of the candidate algorithms described above.

As the graph above indicates, most of the algorithms presented perform similarly to BIP. Notably, OptPartition performs 10% better than BIP for instances of size 10. At size 10, OptPartition is computing the globally optimal tree, and so it is natural to assume that its cost will be less than BIP. The magnitude of BIP’s deviation from optimal agrees roughly with the results presented in [10]. Of the algorithms above, only DistClusterDMSTJoin is fully distributed in nature. The performance of DistClusterDMSTJoin seems to oscillate at about 18%, and does not significantly change as a function of instance size. Because we have analyzed the performance of the algorithm for network sizes spanning two orders of magnitude, we believe that this indicates our distributed algorithm will remain at about 18% worse than BIP in performance for networks of more than 300 nodes.

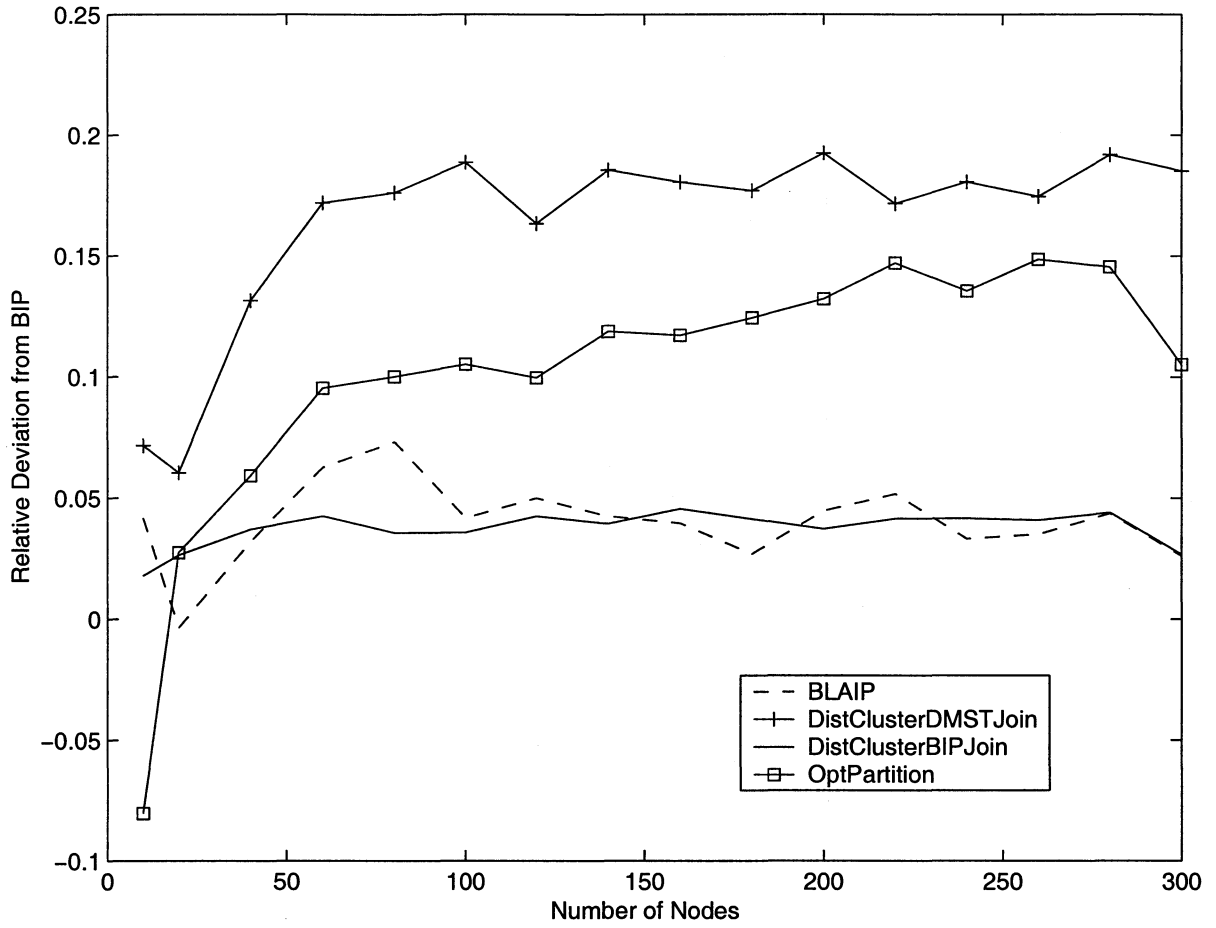


Figure 3-6. Average case performance as simulated against BIP. All nodes are restricted to the 1x1 unit square.

Additionally, note that DistClusterBIPJoin seems to behave, on average, almost exactly halfway in between DistClusterDMSTJoin and BIP. Because DistClusterBIPJoin starts with the same basic clustering as DistClusterDMSTJoin, this indicates that BIP “joins” together the clusters at lower additional cost than the DMST algorithm. This, taken with the poor performance of OptPartition, suggests that the dominating factor in energy-efficient tree construction is not the cost of a locally confined subtrees, but the cost of joining clusters/trees together.

We also analyzed the performance of our algorithm in networks with limited range. To do this, we used the same network instances as used above. For each instance, we measured the performance of our algorithm and BIP for several values of  $R_{max}$ . If for a specific range and instance, some node was unreachable from the source (i.e.  $R_{max}$  was too small), no algorithm was run on that instance. The results of our simulation for instances with 100 and 200 nodes can be seen in Figures 3-7 and 3-8. Note that all graphs were fully connected at ranges 0.3 and 0.2 for instances of size 100 and 200, respectively.

In Figures 3-7 and 3-8, we have presented the cost of both algorithms at limited range, represented as relative deviation from the cost of the BIP tree (when each node has infinite range). Therefore, we see that the BIP algorithm converges to zero deviation as range increases. Additionally, these graphs show that, as range increases, the deviation of the distributed algorithm converges to its value as shown in Figure 3-6 (for example, at 100 nodes, the deviation at infinite range is 18%; therefore the graph of Figure 3-7 should show the distributed algorithm converging to 18%). Note also that both algorithms do not oscillate significantly as a function of range (especially for ranges above the minimum range required

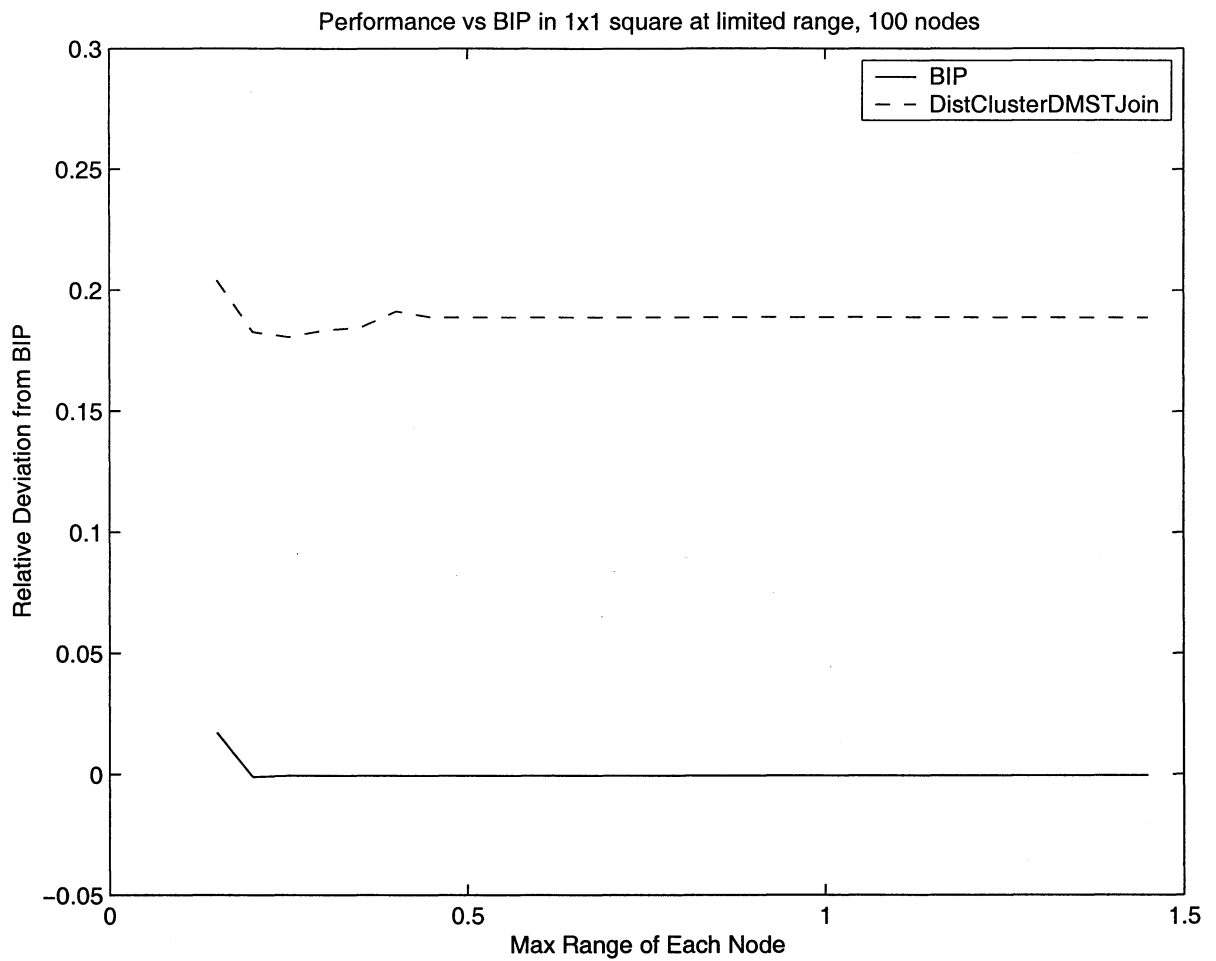


Figure 3-7. Average case performance for different ranges in networks with 100 nodes. All nodes are restricted to the 1x1 unit square.

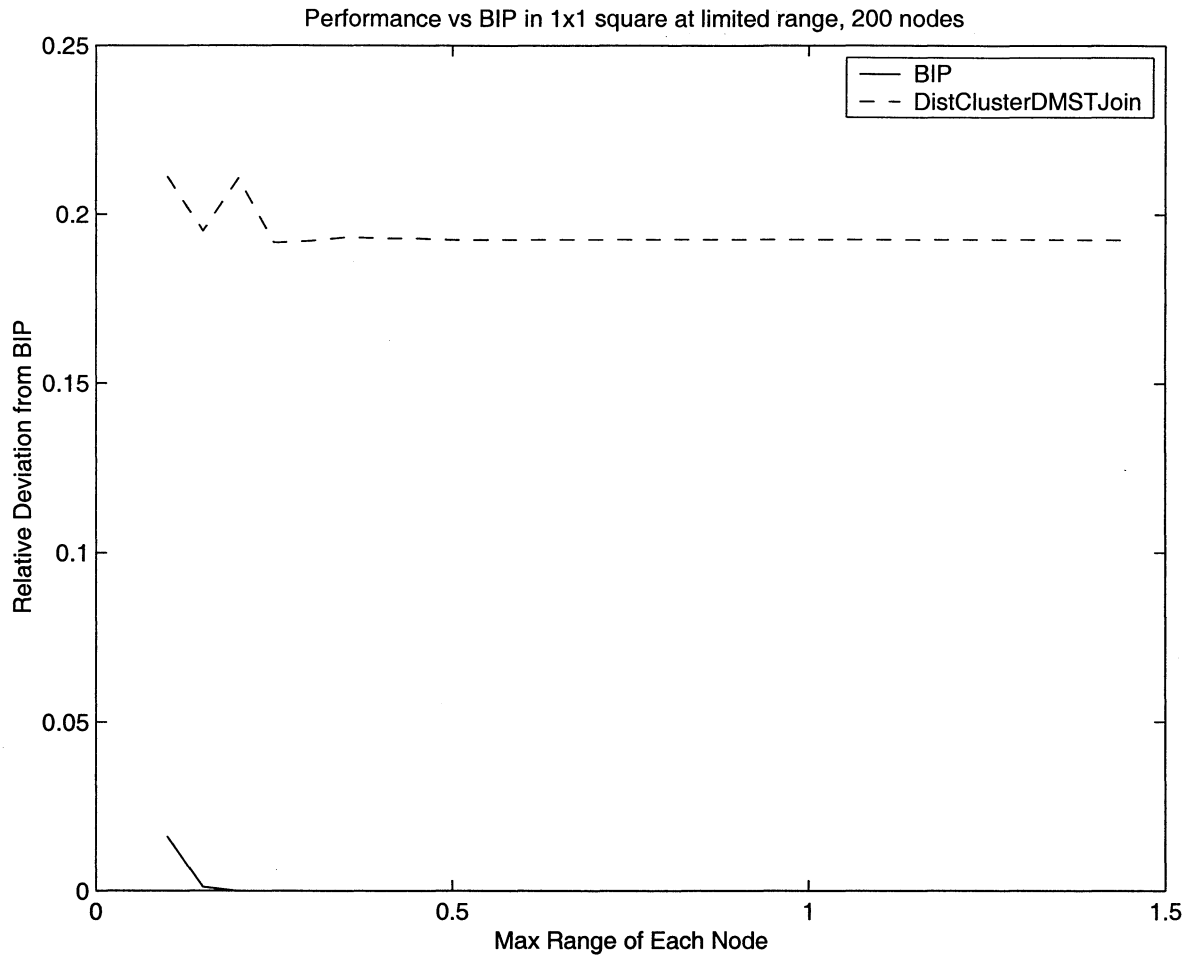


Figure 3-8. Average case performance for different ranges in networks with 200 nodes. All nodes are restricted to the 1x1 unit square.

for connectivity). This indicates that a small range can be chosen (which ensures connectivity), without sacrificing the quality of the solution produced by our distributed algorithm. Additionally, at smaller ranges, the distributed algorithm will perform better in terms of time complexity, message complexity, and energy consumed, since the size of each two hop neighborhood is smaller.

### 3.4 Multiple Source Broadcast

In our analysis so far, we have assumed there is only one source from which one message is sent to all other nodes in the network. However, in many networks, it may be useful to have several sources send the same message to every node in the network. For example, in a sensor network, certain special “supervisor” sensors may have satellite links to a central communication center. If a control command is to be issued from the communication center, destined to reach all sensors, it can be sent to all supervisor nodes via satellite, and then forwarded to the entire sensor network. In such a scenario, significant energy could be saved by broadcasting the message from *many* supervisor nodes instead of just one. We investigate this possibility, and constructed variants of our distributed algorithm and BIP to work in this case.

The multiple source problem can be formally presented as follows. We are given a set of nodes  $N$  and a function  $\phi : N \rightarrow \mathbb{Z} \times \mathbb{Z}$ , which gives us a set of coordinates for each node on the two dimensional plane. Additionally, we are given a range  $R \in \mathbb{Z}^+$ , which represents the maximum distance any node can transmit a message, and a constant  $\alpha > 0$ . We construct the undirected graph  $G = (V, E)$  where  $V = N$  and  $(i, j) \in E \iff d_{ij} \leq R$ . Assuming this

graph is connected, the problem can be stated formally as:

Given a set of sources  $S \subseteq N$  construct the minimum cost directed forest  $F$  (where each tree in this forest is rooted at some  $s \in S$ ) such that,  $\forall n \in N - S$ ,  $\exists s \in S$  such that there is a directed path from  $s$  to  $n$  in  $F$ , and  $d_{ij} \leq R \quad \forall (i, j) \in T$ .  
 Given that  $f(x) = \max\{d_{xj}^\alpha : (x, j) \in F\}$ , we define the cost of  $F$  as  $\sum_{n \in N} f(n)$ .

Note that because this is a generalization of *BCAST*, the decision version of this problem is also NP-complete.

**Multiple Source BIP** To modify BIP to work for multiple sources, we can note the following: the proof of BIP's correctness relies on the fact that at each iteration, the tree being constructed,  $T$ , contains a path from the single source  $s$  to every node in  $T$ . Each time a node is added to  $T$  this property is maintained. Therefore, the tree at the end of the algorithm contains a path from the source to every node (and for this reason is a valid tree). Analogously, in the multiple source problem, we can maintain a forest  $F$  such that there is a path to every node in the forest from at least one source node. When adding a node from  $N - F$  to  $F$  we can use the same criteria as in the original BIP (least additional power). If we continue to add nodes in this way, the forest constructed at the end of the algorithm will ensure a path to every node from some source node. This algorithm can be described more completely as follows:

Throughout its execution, maintain a set of nodes  $F$  that denote the forest made so far. Additionally, maintain a power  $p_i$  for each node in  $F$  (initially,  $F = S$  and  $p_s$  is set to 0 for each node  $s \in S$ ). At each step, increase the power of a node  $f \in F$  to reach a node in  $n \in N - F$ .



Specifically, increase the power of the node  $f$  that requires the least *additional* power to reach a node in  $N - F$  (the additional power for  $f$  to reach node  $n$  is  $Power(f, n) - p_f$ ). Once a node  $n$  has been chosen to be added by some node  $f$ ,  $n$  is added to  $F$  with  $p_n = 0$ , and  $p_f$  is increased to  $Power(f, n)$ . This process continues until  $F = N$ .

We consider the above algorithm as an extension of BIP to the multiple source problem.

**Distributed Algorithm with Multiple Sources** To understand how we might extend the distributed algorithm, we must generalize each step that it takes. This would include the clustering algorithm, the clustering sweep, and the DMST algorithm used to join clusters together. Note that the distributed clustering algorithm presented above generates a clustering that is *source independent*. That is, the clustering developed is not affected by the choice of source node. Therefore, this algorithm can still be used in the multiple source case. A similar argument can be applied to the clustering sweep.

Peculiarly, note also that if we did not modify the DMST algorithm we would still construct a valid tree. However, this tree would only utilize one source, and may consequently miss an opportunity for cost savings. To fix this, we first look at how to remedy the multiple source problem in another type of graph, where all edge weights are non-negative (with no geometric restriction), and the cost of a forest is the *sum* of the edge weights. In this environment, add a directed edge to the original graph between each pair of sources in  $S$  with cost 0 (if the edge already exists, we modify its cost to be 0). Note that if we now compute the minimum cost, single source directed spanning tree rooted at any node  $s$  in  $S$ , the resulting tree will specify the minimum cost *forest*.

In the multiple source *BCAST* environment, where the cost function and geometric

restrictions are different, we employ a similar idea. We modify the use of the DMST algorithm as follows. Note that originally the cost of each directed edge  $(i', j')$  in our graph  $G'$  is set to  $\max(0, P(\text{distance}(i', j')) - P(r_i))$ , where  $r_i$  is the range assigned from the clustering sweep algorithm, and  $P(x)$  denotes the power to transmit at a range  $x$ . We add a directed edge to  $G'$  between each pair of sources in  $S$  with cost 0. We then run the DMST algorithm of [9] on this modified graph. Note that although the DMST algorithm calculates the broadcast tree rooted at *every* node, our modified algorithm is based on a DMST algorithm that computes a forest with only source nodes as roots.

**Performance** We simulated the efficiency of the modified BIP algorithm and our distributed algorithm on graphs with multiple sources to gauge their performance. The results of our simulation are shown in Figures 3-9 and 3-10. As done previously, for simulation at a particular network size, we first constructed a set of 100 instances, each having the same number of nodes. For each instance, nodes were randomly placed with uniformly probability in the unit square, and a random set of sources was chosen. Each algorithm was then run on each of the 100 instances. After a tree was computed from any algorithm, the sweep procedure from [10] was run on the tree before comparison. The results were then averaged over the 100 instances.

Figure 3-9 demonstrates that the power of the BIP tree seems to decrease about 4% for every additional 2% of the graph that is assigned to source nodes. Additionally, the cost savings begins to diminish as the percentage of source nodes increases. The behavior of our distributed algorithm, shown in 3-10, seems to be very similar. In this case, 5% cost is saved per additional 2% sources, which diminishes as the percentage of source nodes is increased.

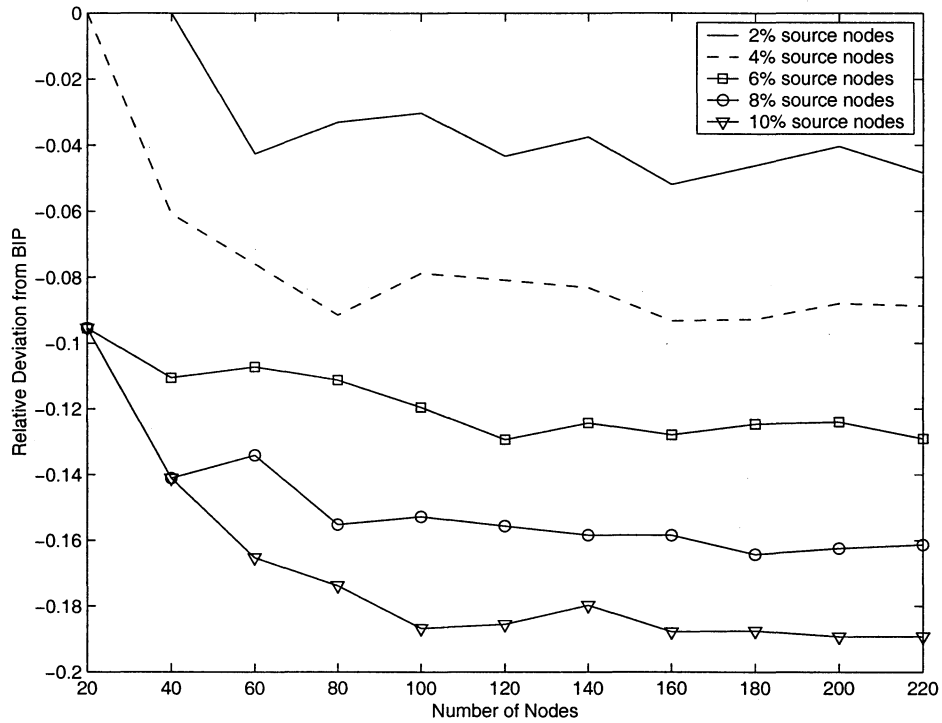


Figure 3-9. Performance of the multiple source variant of BIP relative to BIP with one source. All nodes are restricted to the 1x1 unit square.

From both figures, it seems that the most additional savings (with 10% source nodes) is about 20% for BIP and 25% for our distributed algorithm. This seems to indicate that the use of additional sources is only marginally useful for the purposes of minimizing energy consumption.

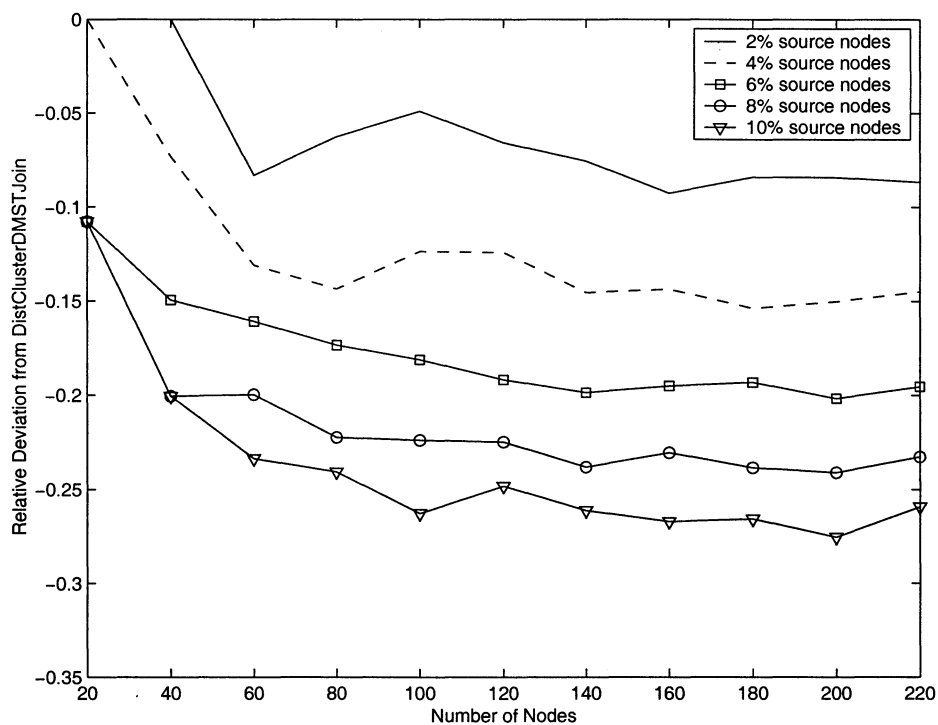


Figure 3-10. Performance of the multiple source variant of our distributed algorithm relative to our distributed algorithm with one source. All nodes are restricted to the 1x1 unit square.

# Chapter 4

## Conclusion

In this thesis, we have shown that the problem of forming minimum energy broadcast trees is NP-complete. Additionally, we have developed a distributed, scalable algorithm that computes sub-optimal broadcast trees using  $O(N^2)$  message complexity. This algorithm computes all  $N$  possible broadcast trees (one for each of  $N$  possible source nodes), and only consumes 18% more power on average than trees produced by the centralized BIP algorithm. We have also introduced the multiple source broadcasting problem, and demonstrated that multiple sources can be used to achieve marginal cost savings.

We believe that there is significant future work to be done in this area. For one, the NP-completeness result, although reflecting the real-world limitation of transmitters, assumes that each node has a limited range. It would be interesting to know how relaxing this restriction effects the complexity of this problem. On the approximation side, the bound in [18] doesn't tell us how BIP performs in the average case. Once this is known, one can determine whether better approximations are needed, and if distributed algorithms, like the

one presented in this thesis, need to be improved. Additionally, the BIP algorithm, and our distributed algorithm, do not take advantage of any geometric properties of trees in the plane; it is quite possible that a geometrically inspired algorithm could outperform both of these algorithms significantly, while simultaneously providing a tighter bound on performance.

In our analysis, we did not attempt to specifically ensure that the distributed algorithm consumed very little energy through its execution. Some work still needs to be done to construct and bound the energy-efficiency of routing algorithms - without this analysis, one cannot tradeoff energy use to compute routes versus the energy saved by using these routes. Additionally, more work has to be done in range controlled and mobile environments to determine how algorithms with more than one goal (energy, contention, QoS) can be implemented in a localized, distributed fashion. Lastly, although using multiple sources has not proven to be effective in saving energy, it may improve other aspects of the broadcast tree (like delay, state storage, etc). Some exploration of these possibilities may provide an incentive to use multiple sources for the purposes of broadcast traffic.

# Bibliography

- [1] M. R. Garey, D. S. Johnson, "The Rectilinear Steiner Problem is NP-complete", SIAM Journal of Applied Math, 32 (4), June 1977.
- [2] M. R. Garey, D. S. Johnson, "Computers and Intractability: A Guide to the Theory of NP-completeness", New York- Freeman, 1979.
- [3] R. Tamassia, I. G. Tollis, "Planar Grid Embedding in Linear Time", IEEE Transactions on Circuits and Systems, 36 (9), September 1989.
- [4] B. N. Clark, C. J. Colbourn, D. S. Johnson, "Unit Disk Graphs", Discrete Mathematics, 86, pgs. 165-177, 1990.
- [5] W. T. Chen and N. F. Huang, "The Strongly Connecting Problem on Multihop Packet Radio Networks," IEEE Transactions on Communications, Vol. 37, No. 3, pp. 293-295, Oct. 1989
- [6] A. E. F. Clementi, P. Penna and R. Silvestri, "On the Power Assignment Problem in Radio Networks", ECCO Electronic Colloquium on Computational Complexity (ECCC), Report No. 054, 2000.
- [7] M. Karpinski and A. Zelikovsky. "New approximation algorithms for the Steiner tree problems," J. Combinatorial Optimization 1(1):47-66, 1997 Tech. report TR-95-036, Univ. of California, Berkeley, International Computer Science Inst., 1995.
- [8] M. Charikar, C. Chekuri, T. Cheung, Z. Dai, A. Goel, S. Guha and M. Li, "Approximation algorithms for directed Steiner problems", Proceedings of the ninth annual ACM-SIAM symposium on Discrete algorithms, p.192-200, San Francisco, Jan. 1998.
- [9] P. A. Humblet "A Distributed Algorithm for Minimum Weight Directed Spanning Trees", IEEE Transactions on Communications, 31 (6), June 1983.
- [10] J.E. Wieselthier, G.D. Nguyen, A. Ephremides. "On the Construction of Energy-Efficient Broadcast and Multicast Trees in Wireless Networks", In Proceedings of IEEE Conference on Computer Communications (IEEE Infocom 2000), New York, March 2000.
- [11] B. Das and V. Bharghavan. "Routing in ad-hoc networks using minimum connected dominating sets", In IEEE International Conference on Communications (ICC'97), 1997.

- [12] Y. Xu, J. Heidemann, and D. Estrin. "Adaptive Energy-Conserving Routing for Multihop Ad Hoc Networks", Research Report 527, USC/Information Sciences Institute, October, 2000.
- [13] B. Daneshrad. "Variable Rate Low power FHSS Baseband Processor", <http://www.ee.ucla.edu/~babak/thss/thss.html>.
- [14] M. B. Pursley, H. B. Russell, and J. S. Wysocarski. "Energy-efficient transmission and routing protocols for wireless multiple-hop networks and spread-spectrum radios," AFCEA/IEEE EuroComm 2000 Conference, pp. 1-5, Munich, Germany, 17-19 May 2000.
- [15] N. Jindal and A. Goldsmith. "Capacity and Optimal Power Allocation for Fading Broadcast Channels with Minimum Rates", To be presented at IEEE Globecom 2001.
- [16] S. Kandukuri and S. Boyd. "Optimal power control in interference limited fading wireless channels with outage probability specifications", IEEE J. on Selected Areas in Communications: Wireless Communications Series, 2001.
- [17] K. Leung. "Integrated Link Adaptation and Power Control for Wireless IP Networks," Proc. IEEE VTC2000, Tokyo, Japan, May 2000, pp. 2086-2092.
- [18] P.-J. Wan, G. Calinescu, X.-Y. Li., O. Frieder. "Minimum-energy broadcast routing in static ad hoc wireless networks" , INFOCOM 2001. Proceedings. IEEE , Volume: 2 , 2001, Page(s): 1162 -1171.
- [19] V. Rodoplu and T. H. Meng, "Minimum energy mobile wireless networks", IEEE Jour. Selected Areas Comm., vol. 17, no. 8, pp. 1333-1344, Aug. 1999.
- [20] T.H. Cormen, C.E. Leiserson, and R.L. Rivest, "Introduction to algorithms", The MIT Press, 1991.
- [21] P. Krishna, N. H. Vaidya, M. Chatterjee, and D. K. Pradhan, "A cluster-based approach for routing in dynamic networks", In Proc. of ACM SIGCOMM Computer Communication Review, pages 372-378, 1997.