On the Complexity and Distributed Construction of Energy-Efficient Broadcast Trees in Static Ad Hoc Wireless Networks

Ashwinder Ahluwalia*and Eytan Modiano
MIT LIDS
Cambridge, MA 02139
ash@mit.edu, modiano@mit.edu

Li Shu
Draper Laboratory
Cambridge, MA 02139
lshu@draper.com

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.**

I. 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 pre-existing infrastructure. Common examples include networks for emergency response, sensor networks and various military applications. In this paper, we are interested in the construction of energy-efficient broadcast trees in ad hoc networks, as posed in [6]. 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.

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}. 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^α (typically $\alpha \geq 2$).

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 problems, in that cost is *node* weighted instead of edge

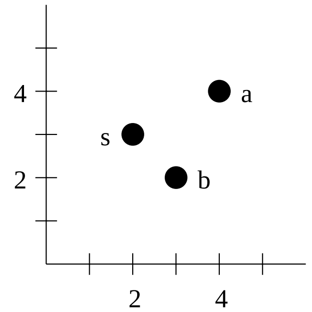

Figure 1: The Multicast Advantage

weighted. Consider the example shown in Figure 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 [6], 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 $max_{n \epsilon N} Power(s,n)$. A transmission at this power will be received by all nodes in N.

In the first part of this paper, we characterize the complexity of this problem, and show it is NP-complete. We then develop a distributed approximation algorithm, and show its performance is comparable to the best known centralized algorithm ([6]) in the average case.

II. Problem Formulation

The ad hoc wireless broadcasting problem can be stated as follows: We are given a set of nodes N and a function $\phi : N \to \mathbb{Z} \times \mathbb{Z}$, 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 $R \epsilon \mathbb{Z}^+$, 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 $(i,j) \epsilon E \iff d_{ij} \leq R$. Assuming this graph is connected, the wireless broadcasting problem can be stated as follows:

Given a source $s \epsilon N$ construct the minimum cost directed tree T (rooted at s) that connects s to

*The work of A. Ahluwalia was supported by a Fellowship from Draper Laboratory.

every node in $N - \{s\}$ via a directed path according to the following cost function. Define $f(x) = max\{d_{xj}^{\alpha} : (x,j) \epsilon T\}$. Then the cost of T is defined as $\sum_{n \epsilon N} f(n)$.

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 $l \epsilon \mathbb{Z}^+$, as $BCAST$. Naturally, one is interested in the complexity of the above problem. Unfortunately, the first contribution of this paper, stated in Theorem 1 below, tells us that this problem is computationally difficult; and a polynomial time solution is unlikely.

Theorem 1. *BCAST is NP-complete.*

Proof of this theorem is based on a reduction from the connected node cover problem in planar graphs, and is included in the appendix.

III. Broadcast Tree Algorithms

Work by Wieselthier, et. al. looked directly at the above broadcast problem ([6]), and proposed a centralized algorithm to construct energy efficient broadcast trees. They showed that their algorithm, the Broadcast Incremental Protocol (BIP), performed well as 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 paper, so it is particularly relevant. Additionally, to our knowledge, BIP is the best known algorithm for this problem ([7]). Consequently, we will use the performance of BIP as a measuring stick in judging our distributed algorithm.

BIP is a greedy algorithm, that mimics Prim's algorithm ([8]) for constructing MST's. Throughout its execution, BIP maintains a set of nodes T that denote the tree made so far (initially, $T = \{s\}$ and the power of s is set to 0). At each step, BIP attempts to increase the power of a node in $t \epsilon T$ to reach a node in $n \epsilon N - T$. Specifically, BIP increases the power of the node t that requires the least *additional* power to reach a node in $N - T$. The node n is then added to T, and the process is repeated until $T = N$. Once the tree is constructed, a sweep algorithm is run on the tree to reduce the power of nodes in specific cases where a node can be reached by multiple transmitters.

Although BIP has already been shown to construct low cost trees, 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 (alternatively, each node can gather the information and compute the tree independently). 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 efficiently. A localized algorithm is one in which nodes' decisions are based on network conditions within some limited distance.

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 for computing minimum spanning trees in directed graphs.

A Distributed Construction of Broadcast Trees

At the beginning of the algorithm 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. 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 that 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 (e.g., interference avoidance).

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.

A.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. Throughout the execution of the centralized 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,

$$\alpha_i(r) = \tfrac{P(r) - P(r_i)}{U_i(r_i, r)}$$
$$r_i < r \leq R$$

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 α 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 $rmin_j$ 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. 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 a particular node may be a member of more than one cluster).

Interestingly, the greedy behavior of the above algorithm can be implemented distributively, with one important difference. The distributed algorithm, for reasons stated earlier, does not attempt to find a global minimum, but instead attempts to find *local* minima. Although this may result in less power efficient clusterings, such inefficiencies are inherent to many localized algorithms.

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 node i is unmarked, it computes, for each neighbor j, the minimum value of $\alpha_j(r)$ for $r \geq distance(i, j)$. 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 α found through this computation as $rmin_{j \to i}$ and $amin_{j \to i}$, respectively.

Each node i then finds the neighbor node k with minimum value of $amin_{k \to i}$, and sends k a PREFERRED message containing range value $rmin_{k \to i}$. This message is sent at full power, so that it can be heard by all of i's neighbors.

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 it 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 α_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 $O(N^2)$ MARKED_STATUS messages, and $O(N^2)$ PREFERRED/RANGE_INCREASE messages.

A Clustering Sweep Procedure: Note that in the clustering produced by the algorithm above, 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 ([6]). 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.

Joining Clusters Together: After a clustering has been found, we use a well known distributed algorithm for constructing directed minimum spanning trees (DMST) [5] to join the clusters together. Specifically, we do the following:

1. Construct the directed graph $G' = (V', E')$ were $V' = V$, and the cost of each edge (i', j') is equal to $max(0, P(distance(i', j')) - P(r_i))$ 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 [5] 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.*

A.2 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 practice, 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. Although the presentation in [5] demonstrated that clusters can still be joined successfully under these conditions, the clustering phase of our algorithm is of concern.

To extend our clustering algorithm to work without global synchronization and in the presence of arbitrary message delays we can note the following. In each stage, three substages occur: 1') PREFERRED messages are sent, 2') some nodes increase their range and send a RANGE_INCREASE, and 3') MARKED states are propagated. If we ensure a node does not go onto substage $k + 1'$ without receiving the messages of substage k', the clustering developed will be identical to the the synchronous case. For a node to make the same decision as in the synchronous algorithm in a substage $k + 1'$, it must have received all messages destined to it, and sent off in any previous substage (all state must be up to date). Viewed more generally, we can tolerate reordering if the algorithm *requires nodes to wait* for all messages to arrive.

We can do this as follows. Assume that every node has up to date information from the previous stage In substage 1', each unmarked node sends a PREFFERED message as in the synchronous algorithm. Each node is not allowed to enter a substage 2' until it has received *all* PREFERRED messages from its unmarked neighbors. Once a node is allowed to enter substage 2', it sends off a RANGE_INCREASE message as in as in the synchronous algorithm. If it will not increase its range, the node sends off a NON_ACTION_RANGE_INCREASE message to indicate this. As with substage 2', no node is allowed to enter substage 3' until a substage 2' message is received from *all* neighbor nodes. Once a node enters substage 3', it sends off a STATUS message *only* if it was unmarked in the last stage. Once a node has received a STATUS message from every unmarked neighbor, it composes and sends a STATUS_SUMMARY message, which lists out all nodes that have been marked in this stage (this ensures that a MARKED update from a node travels two hops). A 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). This algorithm ensures that no two neighbor nodes are out of sync by more than one substage, regardless of message delay.

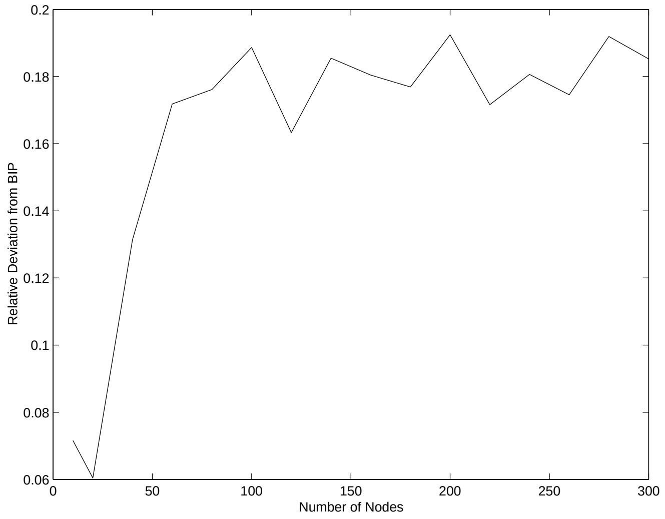

Figure 2: Average case performance as simulated against BIP. All nodes are restricted to the 1x1 unit square.

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.

B Simulation Results

To gauge the performance of our algorithm against BIP, we simulated the performance of BIP, and our distributed algorithm, 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 distributed coordinates)in the unit square, and one node was randomly chosen to be the source. Each algorithm was then executed on each of the 100 instances. After a tree was computed, the appropriate sweep procedure was executed on the tree. We compared the performance of our algorithm to BIP for networks with between 10 and 300 nodes. The results, (shown in Figure 2), display the relative performance of our algorithm as compared to BIP, averaged over the 100 instances. As can be seen from the figure, our distributed algorithm consumed, on average, 18% more energy than the centralized BIP.

IV. Conclusion

In this paper, we have shown that the problem of forming minimum energy broadcast trees is NP-complete. Additionally, we developed a distributed algorithm that computes sub-optimal broadcast trees using $O(N^2)$ message complexity. This algorithm computes all N possible broadcast trees (one per each on N possible source nodes), and only consumes 18% more power on average than trees produced by the centralized BIP algorithm.

References

[1] M. R. Garey, D. S. Johnson, "Computers and Intractability: A Guide to the Theory of NP-completeness", New York- Freeman, 1979.

[2] M. R. Garey, D. S. Johnson, "The Rectilinear Steiner Problem is NP-complete", SIAM Journal of Applied Math, 32 (4), June 1977.

[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] P. A. Humblet "A Distributed Algorithm for Minimum Weight Directed Spanning Trees", IEEE Transactions on Communications, 31 (6), June 1983.

[6] 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 Infocom 2000, New York, March 2000.

[7] 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.

[8] T.H. Cormen, C.E. Leiserson, and R.L. Rivest, "Introduction to algorithms", The MIT Press, 1991.

A. Proof of NP-completeness

Before presenting the proof of Theorem 1, we start by going over the theorems, algorithms and definitions that we use to prove $BCAST$'s NP-completeness.

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 $(i, j) \epsilon E$, $i \epsilon S$ or $j \epsilon S$.

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.

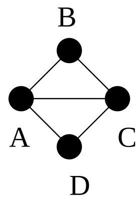

Figure 3: The set $\{B, D\}$ is a node cover of this graph G. $\{A, D\}$ is a connected node cover of G. G also has a connected dominating set $\{C\}$.

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 $\pi_1 : V \to \mathbb{R} \times \mathbb{R}$ such that if we draw a point at $\pi_1(v)$ for all $v \epsilon V$, and then draw a straight line segment from $\pi_1(i)$ to $\pi_1(j)$ in the plane for all $(i, j) \epsilon E$, no line segments will cross. The function π_1 is referred to as a **planar embedding** of the planar graph G.

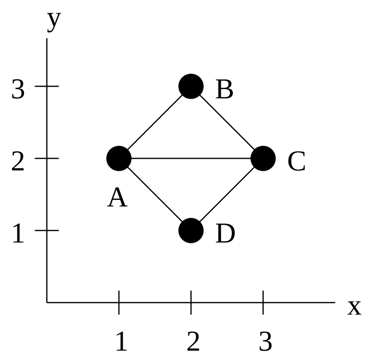

Figure 4: G is also a planar graph. This demonstrates a planar embedding for the graph G.

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 $\pi_2 : V \to \mathbb{Z} \times \mathbb{Z}$, and each edge is mapped to a sequence of horizontal and vertical grid segments, such that no two edges ever cross.

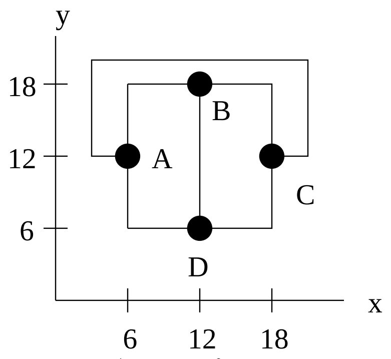

Figure 5: A POGD for the graph G.

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

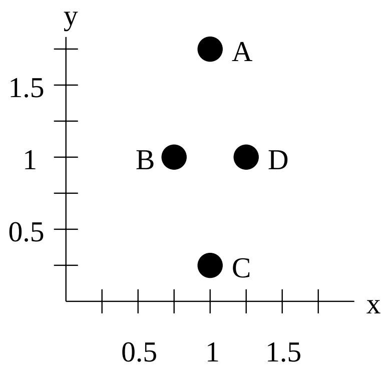

Figure 6: An example showing G is also a unit disk graph. This is a demonstration of π_3.

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*

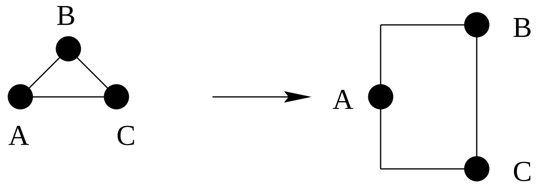

Figure 7: Constructing the POGD in Step 1 of the reduction.

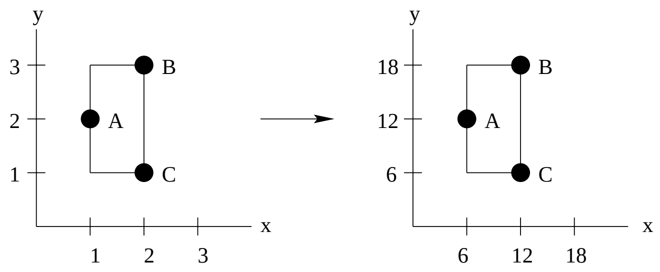

Figure 8: Multiplying the grid size in Step 2 of the reduction.

some given $k \in \mathbb{Z}^+$, is NP-complete. Proved in [2]. Hereafter, we refer to this decision problem as $PLANAR$.

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 that k?") as $CDSUDG$. We reproduce the reduction used in the proof of this theorem (from [4]) below.*

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 7).

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. At this point, we set $r = 1$ grid length (hereafter we also refer to a node transmitting at range r as using 1 unit of power). This illustrated in Figure 8.

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 π_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 π_2) is referred to as the **center node** of this node region. See Figure 9.

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

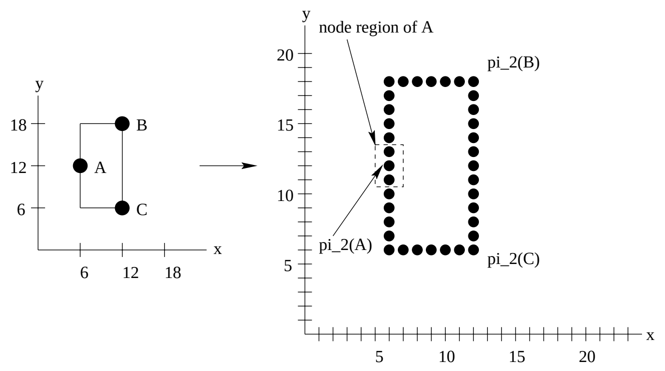

Figure 9: Step 3 of the reduction. P is the set of nodes in the graph on the right. The end nodes in A's node region are at (6,11) and (6,13).

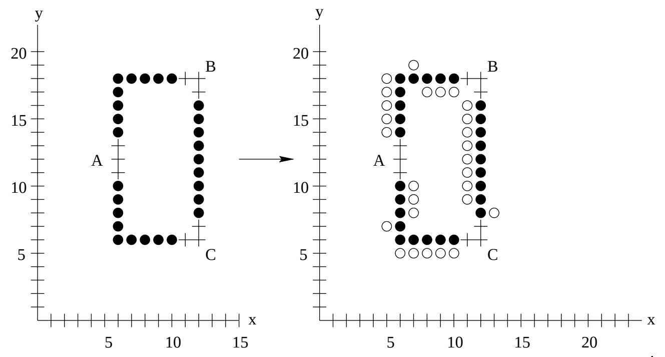

Figure 10: Step 4 of the reduction. Black nodes are nodes in P', and P_l nodes are white. Nodes denoted with a "+" are in node regions.

$P'' \subset P$ as the set of nodes in P that are not center nodes. P_l is then constructed such 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.

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 $|V| - |E| - 1 + k + |P''|$.*

A Proof of Theorem 1

We construct a reduction from $PLANAR$ to $BCAST$ inspired by the reduction used in [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

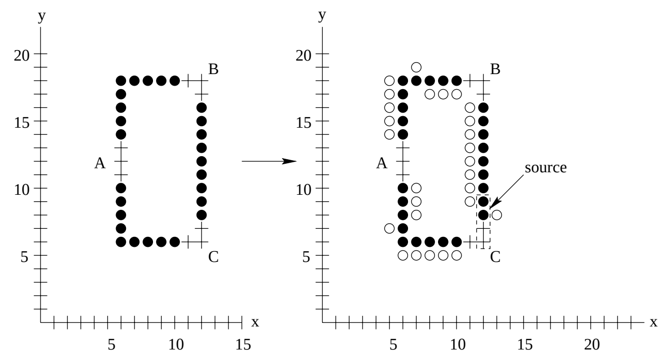

Figure 11: Step 4 of the reduction, with a box around the nodes n_1 through n_4. Black nodes are nodes in P', and P_l nodes are white. Nodes denoted with a "+" are in node regions.

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

1. The nodes of the $BCAST$ instance are the same as those in the generated $CDSUDG$ 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 11, 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 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 that node 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. For problems in which a value is a parameter (in this case the value is node coordinates), a strongly NP-complete problem is one that remains NP-complete even if the parameters are restricted to be polynomially bounded by the size of the problem instance ([1]).

In order to complete our proof, we must show that $BCAST \in NP$. This is clearly true because verifying the correctness of a broadcast tree can be done in polynomial time.