A New Algorithm for the Minimum Connected Dominating Set Problem on Ad Hoc Wireless Networks

Sergiy Butenko* Carlos A.S Oliveira*
Panos M. Pardalos *

Abstract

Given a graph $G = (V, E)$, a dominating set D is a subset of V such that any vertex not in D is adjacent to at least one vertex in D. Efficient algorithms for computing the minimum connected dominating set (MCDS) are essential for solving many practical problems, such as finding a minimum size backbone in Ad Hoc networks. Wireless Ad Hoc networks appear in a wide variety of applications, including mobile commerce, search and discovery, and military battlefield. In this paper we give a polynomial-time algorithm that finds an approximate solution to the minimum dominating set and minimum connected dominating set problems. The algorithm uses the idea of starting from a solution where all vertices of the graph are included. The it works by removing elements that are not necessary in the MCDS. We prove that this algorithm gives a constant performance guarantee. We also propose a distributed version of this algorithm, with similar properties. The results show that, despite its simplicity, the proposed algorithm gives very good solutions, when compared to other existing approaches.

1 Introduction

In some applications of wireless networks, such as mobile commerce, search and rescue, and military battlefield, one deals with communication systems having no fixed infrastructure, referred to as *Ad Hoc wireless networks*. An essential problem concerning Ad Hoc wireless networks is to design routing protocols allowing for communication between the hosts. The dynamic nature of Ad Hoc networks makes this problem especially challenging. However, in some cases the problem of computing an acceptable virtual backbone can be reduced to the well known minimum connected dominating set problem in unit-disk graphs [4].

Given a graph $G = (V, E)$ with the set of vertices V and the set of edges E, a *dominating set* (DS) is a set $D \subseteq V$ such that each vertex in $V \setminus D$ is adjacent

*303 Weil Hall, University of Florida, Gainesville, USA, 32611-6595. Phone: 1 (352) 392-9011. Fax: 1 (352) 392-3537. E-mail addresses: {butenko,oliveira,pardalos}@ise.ufl.edu

to at least one vertex in D. A connected dominating set (CDS) is a DS which is also a connected subgraph of G. We note that computing the minimum CDS (MCDS) is equivalent to finding a spanning tree with the maximum number of leaves in G. In a unit disk graph, two vertices are connected whenever the Euclidean distance between them is at most one.

Ad Hoc networks can be modeled using unit-disk graphs as follows. The hosts in a wireless network are represented by vertices in the corresponding unit-disk graph, where the unit distance corresponds to the transmission range of a wireless device. It is known that both CDS and MCDS problems are NP-hard [7]. This remains the case even when they are restricted to planar, unit disk graphs [3].

Following the increased interest in wireless Ad Hoc networks, many approaches have been proposed for the MCDS in the recent years [4, 6, 13, 1]. Most of the heuristics are based on the idea of creating a dominating set incrementally, using some greedy technique. Some other approaches try to construct a MCDS using an initial independent set [13]. An *independent set* (IS) in G is a set $I \subseteq V$ such that, for each pair of vertices $u, v \in I$, $(u, v) \notin E$. An independent set I is *maximal* if any vertex not in I has a neighbor in I. Obviously, any maximal independent set is also a dominating set.

There are several polynomial-time approximation algorithms for the dominating set problem, when the graph is restricted to be planar. For instance, in [8], Guha and Khuller propose an algorithm which gives a performance guarantee of $2H(\Delta)$, where Δ is the maximum degree of the graph and $H(n) = 1 + 1/2 + \cdots + 1/n$ is the harmonic function. Other approximation algorithms are given in [4, 11]. A Polynomial Time Approximation Scheme (PTAS) for MCDS in unit-disk graphs is also possible, as shown in [9] and more recently in [5].

A common feature of the current techniques for solving the MCDS problem is that the algorithms create the CDS from scratch, at each iteration adding some vertices according to a greedy criterion. For the general dominating set problem, the only exception known to the authors is briefly explained in [12], where a solution is created by sequentially removing vertices. These algorithm share the disadvantage that a setup time must be defined in order to construct a complete MCDS. This becomes a problem when, for example, a reorganization of the network requires a new execution of the algorithm. Other weakness of the current techniques is that frequently complicated strategies must be applied, in order to achieve a good performance guarantee.

In this paper, we propose a new algorithm for computing approximate solutions to the minimum connected dominating set problem. In particular, we discuss in detail the application of this algorithm to the CDS problem in unit graphs. The algorithm uses a new technique which starts from a feasible solution, and removes vertices from this initial solution, until a minimal connected dominating set is found. Using this technique, the proposed algorithm is able to work with feasible solutions since the beginning, and therefore there is no required set up time for the algorithm. The approach has also the advantage of being simple to implement, with experimental results comparable to the best

existing algorithms.

This paper uses standard graph theory notation. Given a graph $G = (V, E)$, a subgraph of G induced by the set of vertices S is represented by $G[S]$. The set of adjacent vertices (also called neighbors) of $v \in V$ is denoted by $N(v)$. Also, we use $\delta(v)$ to denote the number of vertices adjacent to v, i.e. $\delta(v) = |N(v)|$.

The paper is organized as follows. In Section 2 we present the main algorithm and prove some results about its time complexity and approximation guarantee. In Section 3, we discuss a distributed implementation for our algorithm. Computational results from our techniques are discussed in Section 4. Finally, in Section 5 we give some concluding remarks about this work.

2 Algorithm for the MCDS Problem

In this section we describe an algorithm for the minimum connected dominating set problem. Most heuristics for the MCDS work by selecting vertices to be a part of the dominating set and adding them to the final solution. We proceed using the inverse method: the algorithm starts with all vertices in the initial CDS. Then, at each step a number of vertices is removed from the current set using a greedy method. Algorithm 1 is a formal description of the resulting procedure.

The initial step consists of taking the set V of all vertices as the initial feasible solution. In the algorithm, we consider two types of vertices. A *fixed vertex* is one that cannot be removed from the CDS, since its removal would make the dominating set infeasible and/or disconnected. Fixed vertices represent decisions taken by the algorithm about which vertices will be in the final dominating set. *Non-fixed* vertices can be removed only if they do not disconnect the graph.

At each step of the algorithm, at least one vertex is selected to be part of the final CDS, while other vertices are removed, if possible.

In the **while** loop of Algorithm 1, a vertex u is selected to be removed from the current CDS. The algorithm always chooses vertices with the lowest degree to be removed. Note that if there exists some non-fixed vertex v with $\delta(v) = 1$, then we can just remove it from the connected dominating set. Note also that such a vertex cannot disconnect the remaining graph. This is true because, firstly, the vertex is non-fixed, thus no other vertex depends on it for dominance; and secondly, it is linked to just one other vertex and therefore cannot be an internal vertex in a path.

If removing u makes the graph disconnected, then we clearly need u in the final solution, and thus u must be fixed. Otherwise, we remove u from the current CDS D and select some neighbor $v \in N(u)$ to be fixed, in the case that no other neighbor of u has been fixed before. We select a vertex with the highest connectivity to be fixed, since we want to minimize the number of fixed vertices. These steps are repeated while there is a non-fixed vertex in D. In the following theorem we show that the algorithm outputs a CDS correctly.

Algorithm 1: Compute a CDS

```
/* all vertices are in the initial solution */
D ← V
Set all vertices v ∈ V to non-fixed
while there is v ∈ D s.t. v is non-fixed do
    F ← {v | v is not fixed}
    u ← v ∈ F s.t. δ(v) is minimum
    if δ(u) > 1 and D \ u is not connected then
        Set u to fixed
    else
        D ← D \ {u}
        for all s ∈ N(u) : δ(s) = δ(s) − 1
        if there is no (u, v) ∈ E s.t. v is fixed then
            w ← v ∈ D s.t. (u, v) ∈ E and δ(v) is maximum
            Set w to fixed
        end
    end
end
Return D
```

Theorem 1 *Algorithm 1 returns a connected dominating set, and has time complexity $O(nm)$.*

Proof: We show by induction on the number of iterations that the returned set is a connected dominating set. This is certainly true at the beginning, since the graph is connected, and therefore $D = V$ is a CDS. At each step we remove the vertex with minimum degree, only if the removal does not disconnect D. The algorithm also makes sure that for each removed vertex u there is a neighbor $v \in N(u)$ which is fixed. Thus, for each vertex not in D, there will be at least one adjacent vertex included in the final set D. This implies that D is a CDS.

To determine the time complexity of Algorithm 1, note that the **while** loop is executed at most $n-1$ times, since we either remove or fix at least one vertex at each iteration. During each step, the most expensive operation is to determine if removing a vertex disconnects the graph. To do this we need $O(m+n)$ time, which corresponds to the time needed to run the depth first search (or breadth first search) algorithm. Thus, the total time complexity of Algorithm 1 is given by $O(nm)$. □

The proposed algorithm can be considered a convenient alternative to existing methods. Some of the advantages can be seen not only in computational complexity but in other terms as well. First of all, it is the simplicity of the method. Most algorithms for MCDS start by creating a (not necessarily connected) dominating set, and subsequently they must go through an extra step to ensure that the resulting set is connected. In the case of Algorithm 1, no extra step is needed, since connectedness is guaranteed at each iteration. Another favorable consideration is that the algorithm always maintains a feasible

solution at any stage of its execution, thus providing a feasible backbone at any time during the computation.

Next, we determine the performance guarantee of Algorithm 1. We use the following result from [1] in our analysis.

Lemma 2 *The size of any maximal independent set of G is at most $4OPT+1$, where OPT is the size of a minimum CDS of G.*

In the following lemma, we show that there is an independent set with size at least 1/4 of the size of the generated CDS.

Lemma 3 *Given a graph $G(V, E)$ and the resulting connected dominating set D calculated by Algorithm 1, there is an independent set of G containing at least $|D|/4$ vertices.*

Proof: To prove this, we will construct an independent set which will have at least $|D|/4$ vertices. Let us initially consider the cycles in the induced graph $G[D]$. Each vertex v in such a cycle must have been fixed during the removal of another vertex, not in the cycle. This is true because otherwise v could be removed without disconnecting the rest of the vertices, and would still have some adjacent vertex in D, which contradicts the terminating condition of Algorithm 1. Thus, for each vertex v in the cycle there is a distinct vertex u in G which is adjacent to v.

We start by removing induced cycles in $G[D]$. For each such induced cycle $C = \{v_1, \ldots, v_k\}$, we will show that an independent set with cardinality at least half the size of C can be found. Initially, we have an independent set with $\lfloor |C|/2 \rfloor$ vertices, given by $I = \{v_i \mid i \text{ is odd}\}$. These vertices are independent by definition of induced cycle. Then, for each vertex $w_i \in C \setminus I$, consider the vertex u_i which was removed in the same iteration that w_i was fixed. Let K be the set defined by all such u_i, corresponding to $w_i \in C \setminus I$. Each vertex u in K must be independent of I, because otherwise the algorithm would not have chosen a new vertex to be fixed. Thus, we can take at least one of the vertices in K to extend the independent set I, which will therefore have cardinality $\lceil |C|/2 \rceil$.

After removing all induced cycles from D, the resulting set will be composed of independent trees, which we will call $\{T_1, \ldots, T_k\}$. Each tree T_i is a bipartite graph, and therefore its vertices can be divided into two independent sets, T_{i_1} and T_{i_2}. Then obviously at least one of these independent sets will have cardinality at least equal to $|T_i|/2$.

Finally, we denote by $I_1, \ldots, I_l$ the independent sets obtained in the previous constructions. We try to combine these independent sets to find an independent set I^*. However, when taking the union $\hat{I} = \bigcup_{j=1}^{l} I_j$, it is possible that some vertices in different independent sets can be neighbors in the union $\hat{I}$. We analyze this situation by first noting that two tree components T_i and T_j cannot be neighbors, according to our construction. Then it remains the case where an induced cycle C_i, or an induced tree T_i contains a neighbor of some cycle C_j.

We say that a vertex $v \in I_i$ is an *interfering* vertex if it is adjacent to some vertex $u \in I_j$, of another independent set I_j. The proposed solution consists of the following procedure. Order all interfering vertices according to the number of interferences caused by each vertex (in order words, by decreasing value of $|\{w : (v, w) \in E, w \in I_j, j \neq i\}|$, for each vertex v). Let the resulting sequence be $v_1, \ldots, v_k$. Then, sequentially remove the v_1 from their corresponding IS and repeat the procedure, until no interfering vertex remains.

Note that each removed element must have some corresponding vertex in the IS of another component in the rest of the graph. So the number of elements that have been removed is at most 1/2 of the original number of elements in the union of all IS's. This implies that the set I^* resulting from the union of all these modified independent sets has the cardinality of at least $1/2|\bigcup_i I_i|$. Thus, $|I^*|$ is at least $|D|/4$. □

The following theorem estimates the performance guarantee of the proposed algorithm.

Theorem 4 *The cardinality of the dominating set D computed by Algorithm 1 is at most* $16OPT + 4$*, where OPT is the size of a minimum CDS of G.*

Proof: From Lemma 3 we have:

$$|IS| \geq |I^*| \geq |D|/4$$

where $|IS|$ is the size of a maximal independent set, and I^* is the independent set found in Lemma 3. Then, using Lemma 2

$$|D| \leq 4|IS| \leq 4(4OPT + 1) = 16OPT + 4.$$

□

3 A Distributed Implementation

In this section we discuss a distributed algorithm for the MCDS problem, based on Algorithm 1. For Ad Hoc wireless network applications, algorithms implemented in a non-centralized, distributed environment have great importance, since this is the way that the algorithm must run in practice. Thus, we propose a distributed algorithm that uses a strategy similar to Algorithm 1. We describe bellow how the steps of the algorithm are defined in terms of a sequence of messages.

In the description of the distributed algorithm, we say that a link (v, u) is *active* for vertex v, if u was not previously removed from the connected set. The messages in the algorithm are sent through active links only, since all other links lead to vertices which cannot be part of the CDS. We assume, as usual, that there is a starting vertex, found by means of some *leader election algorithm* [10]. It is known [1] that this can be done in $O(n \log n)$ time. We also assume that

the leader vertex v_l is a vertex with the smallest number of neighbors. This feature is not difficult to add to the original leader election algorithm, so we will assume that this is the case.

The execution starts from the leader vertex, which runs the `self-removal` procedure. In the case of degree equal to one, this problem is simplified and the vertex just needs to send the message `DISCONNECTED` to indicate that it is not a part of the CDS anymore, and the message `SET-DOMINATOR`, to make its neighbor a fixed vertex in the CDS. If the degree is greater than one, it is necessary to verify if removing this vertex would disconnect the graph. If this is the case, we run the `Fix-vertex` procedure, since then the current vertex must be present in the final solution. Otherwise, the `Remove-vertex` procedure is executed.

The `Fix-vertex` procedure will execute the steps required to fix the current vertex in the CDS. Initially it sends the message `NEWDOM` to announce that it is becoming a dominator. Then, the current vertex looks for other vertices to be considered for removal. This is done based on the degree of each neighbor; therefore the neighbor vertex with the smallest degree will be chosen first, and will receive a `TRY-DISCONNECT` message.

The `Remove-vertex` procedure is executed only when it is known that the current vertex v can be removed. The first step is to send the message `DISCONNECTED` to all neighbors, and then select the vertex which will be the dominator for v. If there is some dominator in the neighborhood, it is used. Otherwise, a new dominator is chosen to be the vertex with the highest connectivity in $N(v)$. Finally, the message `SET-DOMINATOR` is sent to the chosen vertex.

3.1 Computational Complexity

Note that in the presented algorithm, the step with highest complexity consists in verifying connectedness for the resulting network. To do this, we run a distributed algorithm which verify if the graph is still connected when the current vertex is removed. An example of such algorithm is the distributed breadth first search (BFS), which is known to run in $O(D \log^3 n)$, where D is the diameter (length of the maximum shortest path) of the network, and sends at most $O(m + n \log^3 n)$ messages [2]. Thus, each step of our distributed algorithm has the same time complexity.

To speed up the process, we can change the requirements of the algorithm by asking the resulting graph to be connected in k steps for some constant k, instead of being completely connected. To do this, the algorithm for connectedness can be modified by sending a message with TTL (time to live) equal to a constant k. This means that after k retransmissions, if the packet does not reach the destination, then it is simply discarded. The added restriction implies in fact that, we require connectedness in at most k hops for each vertex. We think that this is not a very restrictive constraint, since it is also desirable that paths between vertices are not very long. With this additional requirement, the diameter of the graph becomes a constant, and therefore the resulting time complexity for each step becomes $O(\log^3 n)$. The time complexity of the whole

algorithm is $O(n \log^3 n) = \tilde{O}(n)$. We use the notation $\tilde{O}(f(n))$ to represent $O(f(n) \log^k n)$, for some constant k.

The number of messages sent while processing a vertex is also bounded from above by the number of messages used in the BFS algorithm. Thus, after running this in at most n vertices, we have an upper bound of $O(nm+n^2 \log^3 n)$ (which is $\tilde{O}(n^2)$ when the graph is sparse) for the total message complexity. These results are summarized in the following theorem.

Theorem 5 *The distributed algorithm with k-connectedness requirement runs in time $O(n \log^3 n) = \tilde{O}(n)$ and has message complexity equal to $O(nm + n^2 \log^3 n)$. For sparse graphs, the message complexity is $\tilde{O}(n^2)$.*

The details of the resulting algorithm are shown in Figure 1. We prove its correctness in the following theorem.

Theorem 6 *The distributed algorithm presented in Figure 1 finds a correct CDS.*

Proof: The basic difference in structure between the connected dominating sets created by the distributed algorithm and the centralized algorithm is that we now require connectedness in k steps. Of course, this implies that the resulting solution is connected.

To show that the result is a dominating set, we argue similarly to what was proved for Algorithm 1. At each iteration, a vertex will be either removed from the solution, or set to be in the final CDS. In the case that a vertex is removed, it must be dominated by a neighbor, otherwise it will send the message `SET-DOMINATOR` to one of its neighbors. Thus, each vertex not in the solution is directly connected to some other vertex which is in the solution. This shows that the resulting solution is a DS, and, therefore a CDS.

Now, we show that the algorithm terminates. First, every vertex in the network is reached, because the network is supposed to be connected, and messages are sent from the initial vertex to all other neighbors. After the initial decision (to become fixed or to be removed from the CDS), a vertex just propagates messages from other vertices, and do not ask further information. Since the number of vertices is finite, this implies that the flow of messages will finish after a finite number of steps. Thus, the algorithm terminates, and returns a correct connected dominating set. □

4 Experiments

Computational experiments were run to determine the quality of the results given by the algorithm proposed for the MCDS problem. We implemented both the centralized and distributed version of the algorithm using the C programming language. The computer used was a PC with Intel processor processor and enough memory to avoid disk swap. The C compiler used was the `gcc` from the

General actions:
  on `TRY-DISCONNECT`, do `Self-removal`
  on `SET-DOMINATOR`, do `Fix-vertex`
  on `NEWDOM`, do
    { dominator← source, `Fix-vertex`}

Self-removal:
  If $\delta(v) = 1$, **then**
    send message `DISCONNECTED` to neighbor
    send message `SET-DOMINATOR` to neighbor
  Else, run distributed BFS algorithm from this
    vertex.
    If some vertex is not reached **then**
      `Fix-vertex`
    Else
      `Remove-vertex`
    End-If
  End-If

Fix-vertex:
  If v is non-fixed, **then**
    set v to fixed
    send message `NEWDOM` to neighbors
    ask the degree of each non-fixed,
      non-removed neighbor
    send message `TRY-DISCONNECT` to
      neighbors, according to increasing degree order
  End-If

Remove-vertex:
  send to active neighbors the message
    `DISCONNECTED`
  If there is no dominator, **then**
    ask the degree of active neighbors
    set u to neighbor with highest degree
  Else
    set u to dominating neighbor
  End-If
  send message `SET-DOMINATOR` to vertex u

Figure 1: Actions for a vertex v in the distributed algorithm.

Size	Radius	Avg.deg	AWF	BCDP	Distr.	Non-distr.
100	20	10.22	28.21	20.11	20.68	19.18
	25	15.52	20.00	13.80	14.30	12.67
	30	21.21	15.07	10.07	10.17	9.00
	35	27.50	11.67	7.73	8.17	6.30
	40	34.28	9.27	6.47	6.53	4.93
	45	40.70	7.60	5.80	6.13	4.17
	50	47.72	6.53	4.40	4.77	3.70
120	20	7.45	38.62	27.71	28.38	27.52
	25	11.21	26.56	18.26	19.00	17.78
	30	15.56	20.67	13.40	14.63	12.40
	35	20.84	15.87	10.03	11.27	9.13
	40	25.09	12.67	8.33	9.00	7.00
	45	30.25	10.47	7.23	7.67	5.77
	50	36.20	8.87	6.00	6.57	4.70

Table 1: Results of computational experiments for instances with size 100×100 and 120×120. The values represent averages over 30 iterations.

Size	Radius	Avg.deg	AWF	BCDP	Distr.	Non-distr.
140	30	11.64	25.76	18.10	18.90	17.03
	35	15.51	20.00	13.33	14.37	12.73
	40	19.40	16.40	10.87	11.57	9.67
	45	23.48	13.33	9.03	9.40	7.77
	50	28.18	11.33	7.63	8.33	6.23
	55	33.05	9.80	6.57	7.17	5.30
	60	38.01	8.80	5.70	6.20	4.53
160	30	9.14	31.88	22.20	23.20	21.88
	35	12.21	24.50	17.07	17.36	16.18
	40	15.63	19.73	13.47	14.07	12.43
	45	19.05	16.33	11.07	11.40	9.93
	50	22.45	13.80	9.47	9.67	7.93
	55	26.72	12.20	7.80	8.33	6.87
	60	30.38	10.33	7.10	7.47	5.80

Table 2: Results of computational experiments for instances with size 140×140 and 160×160. The values represent averages over 30 iterations.

GNU project, without any optimization. The machine was running the Linux operating system.

In the computational experiments, the graphs tested were created randomly. The vertices of the graph are distributed randomly over an area, varying from 100×100 to 160×160 square units. The unit graph is determined by the random location of the vertices, and the size of the radius, whose value ranged from 20 to 60 units. The resulting instances were solved by an implementation of Algorithm 1, as well as by a distributed implementation, described in the previous section. The algorithms used for comparison are the ones proposed in [1] and [4]. They are refereed to in the results as AWF and BCDP, respectively.

The results show that the non-distributed version of Algorithm 1 gives consistently results which are not worse than any of the other algorithms. The distributed version the algorithm gives comparable results, although not as good as the non-distributed version. This can be explained by the fact that the distributed implementation lacks the benefit of global information, used by Algorithm 1 to, for example, finding alway the vertex with smallest degree. However, despite the restrictions on the distributed algorithm, it performs very well. It must be noted also that the resulting implementation is very simple compared to the other approaches, and therefore can be executed faster.

5 Concluding Remarks

In this paper we proposed a new approach to the minimum connected dominating set problem. The algorithm is applied to Ad Hoc Wireless networks, which can be modeled as unit disk graphs. We have shown that the proposed algorithm gives a constant approximation guarantee for the size of the CDS returned. The algorithm is useful mainly in situations where one wants to avoid a specific setup time, since it maintains a feasible solution at any time during the computation. A distributed version of the algorithm is also presented, which tries to adapt the basic algorithmic idea to a parallel setting.

The experimental algorithms show that both algorithms are able too find good quality solutions, with values compared to some of the best algorithms. Also, the technique allows such results without much complexity in terms of implementation. Also, the algorithm proposed can be run during the operation of the network, since it always maintains solution feasibility. Thus, the proposed algorithm can become more useful than current approaches, when deployment in dynamic environments.

References

[1] K. M. Alzoubi, P.-J. Wan, and O. Frieder. Distributed heuristics for connected dominating set in wireless ad hoc networks. *IEEE ComSoc/KICS Journal on Communication Networks*, 4(1):22–29, 2002.

[2] B. Awerbuch and D. Peleg. Network synchronization with polylogarithmic overhead. In *Proc. 31st Symp. Found. Computer Science*, pages 514–522, 1990.

[3] B. S. Baker. Approximation algorithms for NP-complete problems on planar graphs. *Journal of the ACM (JACM)*, 41(1):153–180, 1994.

[4] S. Butenko, X. Cheng, D.-Z. Du, and P. M. Pardalos. On the construction of virtual backbone for ad hoc wireless network. In S. Butenko, R. Murphey, and P. M. Pardalos, editors, *Cooperative Control: Models, Applications and Algorithms*, pages 43–54. Kluwer Academic Publishers, 2002.

[5] X. Cheng, X. Huang, D. Li, and D.-Z. Du. Polynomial-time approximation scheme for minimum connected dominating set in ad hoc wireless networks. submitted to Networks, 2003.

[6] B. Das and V. Bharghavan. Routing in ad-hoc networks using minimum connected dominating sets. In *International Conference on Communications*, pages 376–380, 1997.

[7] M. R. Garey and D. S. Johnson. *Computers and Intractability - A Guide to the Theory of NP-Completeness*. W. H. Freeman, San Francisco CA, 1979.

[8] S. Guha and S. Khuller. Approximation algorithms for connected dominating sets. *Algorithmica*, 20(4):374–387, 1998.

[9] H. B. Hunt III, M.V. Marathe, V. Radhakrishnan, S.S. Ravi, D.J. Rosenkrantz, and R.E. Stearns. NC-approximation schemes for NP- and PSPACE-hard problems for geometric graphs. *J. Algorithms*, 26:238–274, 1998.

[10] N. Malpani, J. Welch, and N. Vaidya. Leader election algorithms for mobile ad hoc networks. In *Proc. Fourth International Workshop on Discrete Algorithms and Methods for Mobile Computing and Communications*, pages 96–103, 2000.

[11] M. V. Marathe, H. Breu, H. B. Hunt III, S. S. Ravi, and D. J. Rosenkrantz. Simple heuristics for unit disk graphs. *Networks*, 25:59–68, 1995.

[12] L. A. Sanchis. Experimental analysis of heuristic algorithms for the dominating set problem. *Algorithmica*, 33:3–18, 2002.

[13] I. Stojmenovic, M. Seddigh, and J. Zunic. Dominating sets and neighbor elimination based broadcasting algorithms in wireless networks. In *Proc. IEEE Hawaii Int. Conf. on System Sciences*, 2001.