Chapter 4

# A NEW HEURISTIC FOR THE MINIMUM CONNECTED DOMINATING SET PROBLEM ON AD HOC WIRELESS NETWORKS

Sergiy Butenko
*Department of Industrial Engineering*
*Texas A& M University*
*College Station, TX 77843-3131*
butenko@tamu.edu

Xiuzhen Cheng
*Department of Computer Science*
*The George Washington University*
*Washington DC 20052*
cheng@gwu.edu

Carlos A.S Oliveira*
*Department of Industrial and Systems Engineering*
*University of Florida*
*Gainesville, FL 32611-6595*
oliveira@ufl.edu

P. M. Pardalos
*Center for Applied Optimization*
*Department of Industrial and Systems Engineering*
*University of Florida*
*Gainesville, FL 32611-6595*
pardalos@ufl.edu

*Corresponding author

**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 chapter we propose a new efficient heuristic algorithm for the minimum connected dominating set problem. The algorithm starts with a feasible solution containing all vertices of the graph. Then it reduces the size of the CDS by excluding some vertices using a greedy criterion. We also discuss a distributed version of this algorithm. The results of numerical testing show that, despite its simplicity, the proposed algorithm is competitive with other existing approaches.

## 1. Introduction

In many 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 simple undirected 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 to at least one vertex in $D$. If the graph is connected, 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 unit.

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 (see Figure 4.1). 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 problem in the recent years [1, 4, 6, 13]. Most of the heuristics are based on the idea of creating a dominating set incrementally, using some greedy technique. Some approaches try to construct a MCDS by finding a maximal independent set, which is then expanded

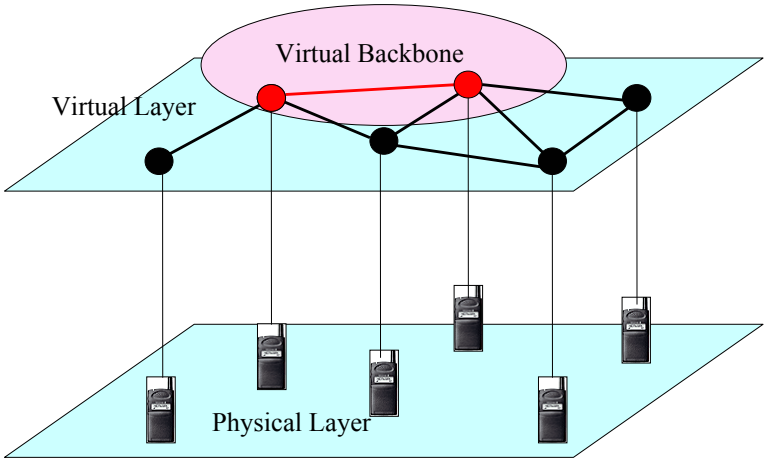


*Figure 4.1.* Approximating the virtual backbone with a connected dominating set in a unit-disk graph

to a CDS by adding "connecting" vertices [4, 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 MCDS problem. For instance, Guha and Khuller [8] propose an algorithm with approximation factor of $H(\Delta)+2$, where $\Delta$ 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 currently available techniques for solving the MCDS problem is that the algorithms create the CDS from scratch, adding at each iteration 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. A shared disadvantage of such algorithms is that they may require additional setup time, which is needed to construct a CDS from scratch. Another weakness of the existing approaches is that frequently they use complicated strategies in order to achieve a good performance guarantee.

In this chapter, we propose a new heuristic 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 MCDS problem in unit-disk graphs. The algorithm starts with a feasible solution, and recursively removes vertices from this solution, until a minimal CDS is found (here, by a *minimal* CDS we mean a connected dominating set, in which removing any vertex would result in a disconnected induced subgraph). Using this technique, the proposed algorithm maintains a feasible solution at any stage of its execution; therefore, there are no setup time requirements. The approach also has the advantage of being simple to implement, with experimental results comparable to the best existing algorithms.

This chapter uses standard graph-theoretical notations. 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 chapter is organized as follows. In Section 2 we present the algorithm and prove some results about its time complexity. In Section 3, we discuss a distributed implementation of this algorithm. Results of computational experiments with the proposed approach are presented in Section 4. Finally, in Section 5 we give some concluding remarks.

## 2. Algorithm for the MCDS Problem

In this section, we describe our algorithm for the minimum connected dominating set problem. As we already mentioned, most existing heuristics for the MCDS problem 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 we select a vertex using a greedy method and either remove it from the current set or include it in the final solution. Algorithm 1 is a formal description of the proposed procedure.

At the initialization stage, we take the set $V$ of all vertices as the starting CDS (recall that we deal only with connected graphs). In the algorithm, we consider two types of vertices. A *fixed vertex* is a vertex that cannot be removed from the CDS, since its removal would result in an infeasible solution. Fixing a vertex means that this vertex will be a part of the final dominating set constructed by the algorithm. *Non-fixed* vertices can be removed only if their removal does not disconnect the subgraph induced by the current solution. At each step of the algorithm, at least one vertex is either fixed, or removed from the current feasible solution.

In Algorithm 1, $D$ is the current CDS; $F$ is the set of fixed vertices. In the beginning, $D = V$ and $F = \emptyset$. At each iteration of the **while** loop of Algorithm 1, we select select a non-fixed vertex $u$, which has the minimum

**Algorithm 1:** Compute a CDS

```
/* D is the current CDS; F is the set of fixed vertices */
D ← V
F ← ∅
while D \ F ≠ ∅ do
    u ← argmin{δ(v) | v ∈ D \ F}
    if G[D \ {u}] is not connected then
        F ← F ∪ {u}
    else
        D ← D \ {u}
        forall s ∈ D ∩ N(u) do
            δ(s) ← δ(s) − 1
        end forall
        if N(u) ∩ F = ∅ then
            w ← argmax{δ(v) | v ∈ N(u)}
            F ← F ∪ w
        end
    end
end
Return D
```

degree in $G[D]$. 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 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.

**Theorem 2.1.** *Algorithm 1 returns a connected dominating set, and has the time complexity of* $O(nm)$.

*Proof.* We show by induction on the number of iterations that the returned set $D$ 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. At 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 virtual backbone at any time during the computation.

## 3. A Distributed Implementation

In this section, we discuss a distributed heuristic 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 below 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 CDS. The messages in the algorithm are sent through active links only, since all other links lead to vertices which cannot be a 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, which runs the `self-removal` procedure. First, we verify if removing this vertex would disconnect the subgraph induced by the resulting set of vertices. 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 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 can be thought of as 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 3.1.** *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 4.2. We prove its correctness in the following theorem.

**Theorem 3.2.** *The distributed algorithm presented in Figure 4.2 finds a correct CDS.*

*Proof.* One of the basic differences between the structure of connected dominating sets created by the distributed algorithm and the centralized algorithm is that we now require connectedness in $k$ steps, i.e., the diameter of the subgraph induced by the resulting CDS is at most $k$. Of course, this implies that the final 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 does 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. Numerical Experiments

Computational experiments were run to determine the quality of the solutions obtained by the heuristic proposed for the MCDS problem. We implemented both the centralized and distributed versions of the algorithm using the C programming language. The computer used was a PC with Intel processor and enough memory to avoid disk swap. The C compiler used was the `gcc` from the GNU project, without any optimization. The machine was running the Linux operating system.

In the computational experiments, the testing instances were created randomly. Each graph has 100 or 150 vertices distributed randomly over an area, varying from $100 \times 100$ to $180 \times 180$ square units. The edges of a unit-disk graph are determined by 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. For the distributed algorithm, we used the additional requirement of $k$-connectedness with $k = 20$. The algorithms used for comparison are 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 4.2.* Actions for a vertex $v$ in the distributed algorithm.

*Table 4.1.* Results of computational experiments for instances with 100 vertices, randomly distributed in square planar areas of size $100 \times 100$ and $120 \times 120$, $140 \times 140$, and $160 \times 160$. The average solutions are taken over 30 iterations.

| Size | Radius | Average degree | AWF | BCDP | Distr. | Non-distr. |
|---|---|---|---|---|---|---|
| $100 \times 100$ | 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 |
| $120 \times 120$ | 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 |
| $140 \times 140$ | 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 |
| $160 \times 160$ | 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 |

ones proposed in [1] and [4]. They are referred to in the results (Tables 4.1 and 4.2) as AWF and BCDP, respectively.

The results show that the non-distributed version of Algorithm 1 consistently gives results which are not worse than any of the other algorithms. The distributed version of 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 for, *e.g.*, always finding the vertex with smallest degree. However, despite the restrictions on the distributed algorithm, it performs very well. It

*Table 4.2.* Results of computational experiments for instances with 150 vertices, randomly distributed in square planar areas of size $120 \times 120$, $140 \times 140$, $160 \times 160$, and $180 \times 180$. The average solutions are taken over 30 iterations.

| Size | Radius | Average degree | AWF | BCDP | Distr. | Non-distr. |
|---|---|---|---|---|---|---|
| $120 \times 120$ | 50 | 54.51 | 9.47 | 6.30 | 6.70 | 4.63 |
| | 55 | 63.02 | 7.73 | 5.70 | 6.40 | 4.20 |
| | 60 | 71.12 | 6.67 | 4.83 | 5.53 | 4.00 |
| | 65 | 81.08 | 6.00 | 3.73 | 4.30 | 3.53 |
| | 70 | 89.22 | 5.00 | 3.23 | 3.67 | 3.10 |
| | 75 | 98.04 | 4.79 | 3.04 | 2.82 | 2.54 |
| | 80 | 104.64 | 4.64 | 2.82 | 2.09 | 2.00 |
| $140 \times 140$ | 50 | 42.91 | 11.60 | 7.60 | 8.33 | 6.13 |
| | 55 | 49.75 | 10.07 | 6.87 | 7.40 | 5.20 |
| | 60 | 57.74 | 8.33 | 5.87 | 6.80 | 4.47 |
| | 65 | 64.70 | 7.60 | 5.27 | 6.20 | 4.17 |
| | 70 | 72.04 | 6.93 | 4.83 | 5.50 | 4.00 |
| | 75 | 78.82 | 5.87 | 3.87 | 4.53 | 3.60 |
| | 80 | 86.55 | 5.47 | 3.33 | 3.93 | 3.33 |
| $160 \times 160$ | 50 | 33.94 | 14.00 | 9.63 | 10.07 | 8.43 |
| | 55 | 40.31 | 12.27 | 8.47 | 8.97 | 6.73 |
| | 60 | 45.89 | 10.73 | 7.30 | 8.13 | 5.73 |
| | 65 | 53.36 | 9.60 | 6.27 | 6.77 | 4.83 |
| | 70 | 58.75 | 8.67 | 6.10 | 6.60 | 4.43 |
| | 75 | 64.39 | 7.80 | 5.40 | 6.13 | 4.37 |
| | 80 | 72.05 | 6.60 | 4.63 | 5.80 | 4.00 |
| $180 \times 180$ | 50 | 27.92 | 17.60 | 11.57 | 12.37 | 10.37 |
| | 55 | 33.05 | 15.13 | 10.13 | 10.43 | 8.47 |
| | 60 | 38.09 | 12.40 | 8.50 | 9.23 | 7.27 |
| | 65 | 43.95 | 11.53 | 7.70 | 8.53 | 6.10 |
| | 70 | 48.75 | 10.13 | 7.17 | 7.43 | 5.33 |
| | 75 | 55.08 | 9.33 | 6.30 | 6.87 | 4.53 |
| | 80 | 60.73 | 8.33 | 5.70 | 6.47 | 4.33 |

must also be noted that the resulting implementation is very simple compared to the other approaches, and therefore can be executed faster.

## 5. Concluding Remarks

In this chapter, we proposed a new approach to the minimum connected dominating set problem. The proposed heuristic algorithm is applied to ad hoc wireless networks, which are modeled as unit disk graphs. The algorithm is especially valuable in situations where setup time is costly, since it maintains a feasible solution at any time during the computation and thus can be exe-

cuted without interrupting the network operation. A distributed version of the algorithm is also presented, which tries to adapt the basic algorithmic idea to a distributed setting. The experimental results show that both algorithms are able to find good quality solutions, with values compared to some of the best algorithms. The above mentioned advantages and the simplicity of the proposed algorithm make it an attractive alternative when solving the MCDS problem 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, W. Wu, and D.-Z. Du. Polynomial-time approximation scheme for minimum connected dominating set in ad hoc wireless networks. To appear in *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.