Title

A Solution For the Location Problem in Arbitrary Computer Networks Using Generic Dominating Sets

Authors

Garcia-Luna-Aceves, J.J.
Spohn, M.A.

Publication Date

2005-03-13

DOI

doi:10.1145/1066677.1066844

Peer reviewed

A Solution For the Location Problem in Arbitrary Computer Networks Using Generic Dominating Sets*

Marco Aurélio Spohn
Computer Science Department
University of California at Santa Cruz
Santa Cruz, CA 95064
maspohn@cse.ucsc.edu

J.J. Garcia-Luna-Aceves
Computer Engineering Department
University of California at Santa Cruz
Santa Cruz, CA 95064
jj@cse.ucsc.edu

ABSTRACT

Many problems exist related to the location problems of resources in a computer network to accommodate client demands subject to constraints imposed on the clients and the servers. For example, one classical location problem consists of computing the *dominating sets* (DS) of a network. A DS is a set of nodes in the network (called dominating nodes) such that the remaining nodes in the network are adjacent to at least one dominating node. The problem of finding a DS of minimum cardinality is known to be NP-complete. A variety of conditions may be imposed on the dominating set D in a graph $G = (V, E)$. Among them, we have *multiple domination* and *distance domination*. *Multiple domination* requires that each vertex in $V - D$ be dominated by at least k vertices in D for a fixed positive integer k. *Distance domination* requires that each vertex in $V - D$ be within distance r of at least one vertex in D for a fixed positive integer r. We refer to the problem of computing DS when these two conditions are taken into account as the *Generic Dominating Sets* (GDS) problem. Prior work on solving the GDS problem focuses on *interval graphs* (IG), which can represent only a few network topologies. We present the first solutions to the GDS problem for arbitrary graphs. Simulation results regarding several configurations are presented.

Keywords

Computer networks, location, domination in graphs.

1. INTRODUCTION

Location problems on computer networks deal with the location of services (or facilities) to accommodate client demands [5] subject to some constraints. For example, given a computer network, servers could be deployed such that every client has a server within a given distance (in terms of hops). One classical location problem is the *domination problem*. The domination problem seeks to determine a minimum number of nodes D such that every other node in the network is adjacent to a node in D. The problem of finding a DS of minimum cardinality is known to be NP-complete [4].

Domination in graphs has many applications in computer networks. Many broadcasting and topology control techniques that make use of dominating sets (DS) have been reported in the literature [1, 3]. In some scenarios (e.g., in wireless ad-hoc networks) the algorithm must compute the DS having only partial knowledge of the network topology (e.g., only the two-hop neighborhood). In wired networks, because the topology does not change, or it changes not very often, we make the assumption that the whole network topology is known.

From graph theory, we know that a variety of conditions may be imposed on the dominating set D in a graph $G = (V, E)$. Among them, we have *multiple domination*, and *distance domination* [6]. *Multiple domination* requires that each vertex in $V - D$ be dominated by at least k vertices in D for a fixed positive integer k. *Distance domination* requires that each vertex in $V - D$ be within distance r of at least one vertex in D for a fixed positive integer r. We refer to the problem of computing a DS when these two conditions are taken into account as the *Generic Dominating Sets* (GDS) problem. The problem of computing a GDS of minimum cardinality for arbitrary graphs is also NP-complete.

Joshi et al. [8] have provided solutions for solving the GDS problem for *interval graphs* (IG). A graph G is said to be an interval graph if there is a one-to-one correspondence between a finite set of closed intervals of the real line and the vertex set V, and two vertices u and v are connected if and only if their corresponding intervals have a nonempty intersection. Even though the solutions presented by Joshi et.al. [8] are optimal, IGs are limited to very simple network topologies.

We adopt the nomenclature presented in [8] to classify GDS problems. The (k, r) dominating problem is defined as the problem of selecting a minimum cardinality vertex set D of a graph $G = (V, E)$ such that every vertex u not in D is at a distance r or less from at least k vertices in D. If we impose the condition that there exist a vertex in D at a distance of at most r for each vertex in D, then the set D is a $total(k, r)$ dominating set, and if there exist a vertex in D adjacent to each vertex in D, then the set D is a $reliable(k, r)$ dominating set.

When the dominating set D is seen as a set of nodes providing some services to other nodes, then redundancy is achieved by choosing k greater than one. The distance parameter r allows us to increase local availability by reducing the distance to the servers.

Many practical networking problems can be solved by setting these two parameters to values such that the requirements are satisfied. More servers in the vicinity incurs smaller average delays. If the servers should be backed up by other servers, then a *reliable*

*This work was supported in part by CNPq (Brazil), and the Baskin Chair of Computer Engineering.

SAC'05 March 13-17, 2005, Santa Fe, New Mexico, USA

Table 1: Notation

N_r^i	The set of r-hop neighbors of node i
$\mathfrak{N}_{r,i}$	The r-hop neighborhood of node i; i.e., $\bigcup_{k=1}^{r} N_k^i$
D	The dominating set
$i.Domin$	The $Domin$ value of node i is k minus the number of nodes in D within distance r of i
I	The set of nodes not dominated yet (initially $I = V$)
$I^{'}$	the set of nodes $(k,r)-$dominated

dominating set is desirable. For example, we can specify the number of servers that should respond to any given client, and also the maximum distance to the servers. Alternatively, we can also enforce that every server must be within a maximum distance from any other server, or that for every server at least one other server should be in the same segment of the network.

The minimum cardinality of the dominating set D is denoted by $\gamma(G)$ and is called the *domination number*. For multiple domination, this parameter is called the *k-domination number* and is denoted by $\gamma_k(G)$. For distance domination, it is called the *distance-r domination number* and is denoted by $\gamma_{\leq r}(G)$.

Henning et al have presented some bounds on the *distance-r domination number* [7]. They show that, for an integer $r \geq 1$, if graph G is a connected graph of order $n \geq r+1$, then $\gamma_{\leq r}(G) \leq \frac{n}{r+1}$. An algorithm that computes a *distance-r dominating set* within the established bounds is also presented.

The rest of the paper is organized as follows. Section 2 presents solutions for the GDS problem for arbitrary networks. Because the GDS problem in arbitrary graphs is NP-complete, the proposed algorithms seek only approximations to the optimal solution. Section 3 presents simulation results comparing our algorithm to the algorithm presented in [7] for computing $(1,r)$-dominating sets, and also simulation results for our algorithm under a variety of configurations. Section 4 concludes this work.

2. GENERIC DOMINATING SETS

2.1 (k,r)-dominating set

A node is said to be $(k,r)-$dominated (or simply dominated) if there are at least k neighbors within distance r in D (refer to Table 1 for notation). The $Domin$ value of a node i (referenced as $i.Domin$) is defined as k minus the number of nodes in D within distance r of i.

Consider an arbitrary graph $G = (V, E)$. Initially all nodes in the graph are in the set I (i.e., $I = V$), and the $Domin$ value of each node is set to k, i.e., $\forall\ v \in V : v.Domin = k$ (see Algorithm 1). This means that all nodes are in need of at least k dominating nodes within distance r. Whenever a node gets covered (i.e., $Domin$ value is zero), the node is moved to the set $I^{'}$. That is the same as to say that nodes in the set $I^{'}$ are $(k,r)-$dominated. At the beginning of the main iteration of Algorithm 1, a new candidate c is chosen among the nodes in the set I, and the selection criteria is that c must be the node with the largest $Domin$ value among the nodes remaining in the set I. Because more dominating nodes are selected, they impact a larger number of nodes. At the very first iteration, c is any node from I, and D is empty. For every new candidate, a set of nodes is selected as potential dominating nodes for c. This set is called P, and it has cardinality $c.Domin$ (i.e., $|P| = c.Domin$). The nodes selected for P must be within distance r from c, and farther away from c. By choosing the nodes farther away from c, they can potentially dominate a larger set of nodes. If such nodes are not available (i.e., there is not even enough nodes in the one-hop neighborhood), then node c is said to be unsatisfiable, and it is placed in the set D. To check coverage, a tree rooted at node c is built, and only nodes still in set I and in c's r-hop neighborhood are taken into account. Nodes no longer in the set I are not considered because their status is already (k,r)-covered, hence they do not participate in the selection of dominating nodes for the remaining nodes not covered yet. In this case, node c is said to be *satisfiable*, because there are at least $c.Domin$ nodes in the tree (excluding node c).

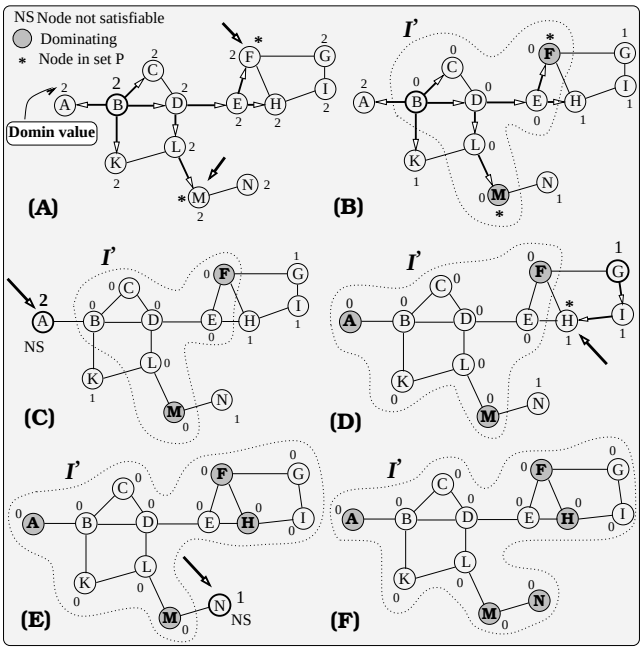

Figure 1: Computing a $(2,3)-$Dominating Set using the KR Algorithm: $Domin$ value is shown by each node; set of (k,r)-dominated nodes (i.e., set $I^{'}$) is shown as a circled area.

If c is satisfiable, the algorithm proceeds with the following computation. Start processing the neighbors of node c in their order of distance until either c is dominated, or a node in P is reached. In this process, for each node n encountered, if n is satisfiable then select the next node (if there are no node left at the same distance, choose the first one from the next level), else place node n in the dominating set D and update the $Domin$ value for each node within distance r from node n. During this process of adding unsatisfiable nodes, if node c is satisfied, then choose the next candidate (again, a node with the largest $Domin$ value among the nodes in the set I), and repeat the main iteration. If c is not satisfied and a node in P is reached, then choose $c.Domin$ nodes from the set P such that they are farther away (number of hops) from node c. The procedure ends when the set I is empty (i.e., all nodes in V are $(k,r)-$dominated).

Figure 1 depicts an example of applying Algorithm 1 to compute a $(2,3)-$dominating set of the network, having node B as the starting node. The tree built using procedure *r-Tree* is indicated by the arrows starting at node B (Figure 1(A)). Node B selects nodes F and M for its set P (note that both are three-hops away from B). All nodes in B's three-hop neighborhood are satisfiable. Because nodes are processed according to their distance, eventually a node in set P is reached. When this happens, $B.Domin$ (i.e.,

2) nodes are selected from set P. In this case, both F and M are selected (Figure 1(B)). Figure 1(C) shows the network after this selection. All nodes within three-hops from node F and M have their $Domin$ value updated accordingly. The next candidate with largest $Domin$ value among the nodes not covered yet is node A. The nodes in the set $I^{'}$ fragment the network in several disjoint components, and node A is by itself in its fragment. Hence, node A does not have any potential dominating nodes, and because it is not satisfiable it is chosen for the set of dominating nodes. After the selection of node A, all the remaining candidates have the same $Domin$ value (i.e., 1) (Figure 1(D)). Lets suppose that node G is selected as the next candidate. Node H is selected for G's P set, because node H is the node farther away from G in the connected component in which G takes part. Eventually node E is reached because the node at the lower level (i.e., nodes I) is satisfiable. After node H is selected as a dominating node (Figure 1(E)), only node N is left out, and it is clearly not satisfiable, because it still needs one dominating node and all the remaining nodes are already (k, r)-dominated (i.e., they are in set $I^{'}$). After node N is selected for the dominating set, all nodes in the network are covered (i.e., $Domin = 0$). That is, every node in the network that is not a dominating node have at least 2 dominating nodes within distance 3.

DEFINITION 2.1. *A node c is a candidate if it is among the nodes in the set I with the largest $Domin$ value.*

Let R_i be the set of nodes within distance r from node i but still in the set I (i.e., $R_i = \mathfrak{N}_{r,i} \cap I$). Let T_i be the tree rooted at node i, such that $height(T_i) \leq r$, and formed only by nodes in R_i.

REMARK 2.1. *For candidate node c, all nodes in $I \cap \mathfrak{N}_{r,c}$ have a $Domin$ value smaller than or equal to $c.Domin$.*

By definition 2.1, node c is a candidate because it is among the nodes with the largest $Domin$ value in set I. Hence, all nodes in R_c have a $Domin$ smaller than or equal to $c.Domin$.

Let P_c be a set of nodes with cardinality $c.Domin$ formed by the deepest nodes in T_c. Let $M = R_c \setminus P_c$. In other words, $P_c \subseteq R_c$ such that $|P_c| = c.Domin$ and for all $i \in P_c$, for all $j \in M$, $height(i) \geq height(j)$. The existence of set P_c with respect to the candidate c implies that c is satisfiable.

Let $S_i \subseteq \mathfrak{N}_{r,i}$ such that for all $k \in S_i$, $k.Domin = 1$; that is, $S_i = \{k \in \mathfrak{N}_{r,i} \mid k.Domin = 1\}$.

If c is satisfiable, nodes from set R_c are selected to dominate c while $c.Domin > 0$. A node m is selected to dominate c if m is unsatisfiable (i.e., $|T_m| < m.Domin$), or m belongs to set P_c.

LEMMA 2.1. *$\{m \cup D\}$ is a (k, r)-dominating set for $\{D \cup I^{'} \cup S_m\}$.*

A node with $Domin$ value one (i.e., the node needs just one more dominating node) and that is in set S_m, becomes (k, r)-covered by node m. Consequently, all nodes in set S_m become (k, r)-covered by node m, and are inserted in set $I^{'}$.

LEMMA 2.2. *If P_c does not exist then $\{D \cup c\}$ is a (k, r)-dominating set for $\{D \cup I^{'} \cup S_c\}$*

If node c is not satisfiable, then node c is added to the set D, and consequently to the set of nodes already covered (i.e., $\{D \cup I^{'}\}$).

THEOREM 2.1. *Algorithm 1 correctly computes a (k, r)-dominating set for any connected graph $G = (V, N)$.*

PROOF. As Algorithm 1 follows Lemmas 2.1 and 2.2 it is clear that it correctly computes a (k, r)-dominating set for a given graph $G = (V, N)$. The selection process continues until no nodes are left in set I. Nodes are inserted in set $I^{'}$ only when they have at least k dominating nodes within r hops, or when they are selected as dominating nodes. Candidate c is either satisfiable, or it is chosen as a dominating node. At the end, all nodes in V are in the set $I^{'}$, and every node is either dominated or a dominating node. □

Algorithm 1: KR

```
Data   : G = (V, E), k, r
Result : D, the (k, r)-dominating set
begin
    foreach i ∈ V do
        i.Domin = k;
    I' ←− ∅;  I ←− V;  D ←− ∅;
    while I ≠ ∅ do
        /* return next node from set I with largest Domin */
        c = Next(I);
        /* create a hash table (hash = distance), and each list sorted by Domin
           in decreasing order */
        HT = NewHashTable(r);
        r-Tree(G, r, c, HT, I);
        if |HT| < c.Domin then
            D ←− D ∪ {c};  c.Domin = 0;
            foreach j ∈ N_{r,c} do
                if j.Domin > 0 then
                    −− j.Domin;
        else
            /* c.Domin more distant nodes from node c */
            P ←− LastN(HT, c.Domin);
            while c.Domin > 0 do
                /* next from HT with lowest index (i.e., closer nodes first),
                   and largest Domin ≠ 0 */
                n = Next(HT);
                if n ∈ P then
                    /* get the c.Domin most distant nodes from HT
                       (i.e., set P_c) */
                    Q ←− LastN(HT, c.Domin);
                    foreach x ∈ Q do
                        D ←− D ∪ {x};  x.Domin = 0;
                        foreach j ∈ N_{r,x} do
                            if j.Domin > 0 then
                                −− j.Domin;
                    Break;
                else
                    /* check if node n is satisfiable */
                    isKCovered(G, r, n, K, I);
                    if K < n.Domin then
                        D ←− D ∪ {n};  n.Domin = 0;
                        foreach j ∈ N_{r,n} do
                            if j.Domin > 0 then
                                −− j.Domin;
        I' ←− I' ∪ {c};  I ←− I \ {c};
        /* remove from set I all neighbors in the r-hop neighborhood that are
           already kr-dominated */
        foreach j ∈ N_{r,c} do
            if j.Domin == 0 AND j ∈ I then
                I' ←− I' ∪ {j};  I ←− I \ {j};
end
```

THEOREM 2.2. *For any graph $G(V, E)$, Algorithm 1 computes a $(1, r)$-dominating set with cardinality within bounds established in [7].*

PROOF. The solution presented in [7] (referenced here as OneR) selects dominating nodes during the traversal of the spanning tree T of graph G. Let D_{OneR} be the DS computed using algorithm OneR, and D_{KR} be the DS computed using algorithm 1. In OneR, for every node $i \in T$ selected for set D, a subset $D_i^{OneR} \subset \mathfrak{N}_{r,i}$ with cardinality $|D_i^{OneR}| = r$ is removed from T (except for the last dominating node, when $|D_i^{OneR}| \leq 2r$). In algorithm 1, for every node j selected as a dominating node (and $j \in T$ because T is a spanning tree of G), a subset $D_j^{KR} \subseteq \mathfrak{N}_{r,j}$ with cardinality $|D_j^{KR}| \leq |\mathfrak{N}_{r,j}|$ is dominated. That is the same as to say that for every node i selected as dominating node, while the KR algorithm removes $|\mathfrak{N}_{r,i}| \geq r$ nodes from graph G (and $i \in T$), algorithm $OneR$ removes always r nodes from T. Hence, we have that $|D_{KR}| \leq |D_{OneR}|$. And because algorithm OneR computes a dominating set within bounds established in [7], so does algorithm 1 when computing $(1, r)$-dominating sets. □

2.1.1 Running time

Lets consider an arbitrary graph $G = (V, E)$ of order $n = |V|$, represented using adjacency-lists. Algorithm 2 implements the procedure *r-Tree*, which is called from Algorithm 1. *r-Tree* builds a tree rooted at node n using *breadth-first search* (BDF) [2]. In [2] it is shown that the total running time of BDF is $O(V + E)$. The main difference between algorithm *r-Tree* and standard BDF is that the former is restricted to the r-hop neighborhood, and the latter runs over the whole graph (network). Algorithm 2 also includes insertion in a hash table (we assume hashing with chaining). Because we assume the hash value to be the distance of a node, there is no cost regarding computing the hash value. In our case the cost of inserting a node in the hash table is constant (i.e., $O(1)$). Thus, Algorithm 2 runs in linear time in the size of the adjacency-list representation of G.

Procedure *isKCovered* (Algorithm 3) also mimics BFS, with the only addition of searching the hash table HT (what can be accomplished in $O(1)$ time). In the main algorithm (i.e., Algorithm 1), the overhead for initialization is $O(n)$. The main *while* loop is executed at most n times. The most costly operation internal to the while loop is performed in the second *while* loop, in which the procedure *isKCovered* is executed at most $O(|\mathfrak{N}_{r,*}|)$ times (where $|\mathfrak{N}_{r,*}|$ is the average size of the r-hop neighborhood). Assuming $E \ll n^2$, and $|\mathfrak{N}_{r,*}| < n$, Algorithm 1 runs in $O(n^2)$ time.

2.2 Total (k,r)-dominating set

A *total (k,r)-dominating* set for a graph $G = (V, E)$ is a set $D \subset V$ such that every node v not in D is at a distance r or less from at least k nodes in D, and every vertex $p \in D$ is at a distance r or less from at least one other node in D.

In the *(k,r)-dominating* set problem we do not find another member within distance r to dominate a dominating node of D. But, to compute the *total (k,r)-dominating* set, when adding a new member n to the set D, if node n is not dominated by any other node in D (i.e., $n.Domin = k$), then the $Domin$ value of node n is set to 1 instead of 0 (as in the *(k,r)-dominating* problem). Therefore, a dominating node is removed from the set I only after it is dominated by at least one dominating node.

When computing the set R_c, nodes that are already members of D, but are not dominated yet (i.e., $Domin = 1$) and still members of set I, are not taken into account. Otherwise, eventually a node could be chosen as a dominating node more than once; what is not acceptable.

LEMMA 2.3. *Let $\mathbb{D}$ be the set of nodes to be added to D in order to dominate candidate node c as determined by Algorithm 1. For all $d \in D$ such that $d.Domin = 1$, if $|\mathbb{D}| > 1$, then $\forall\, (d' \neq d) \in \mathbb{D}$ we have that $dist(d, d') > r$.*

(a) *$\forall\, d \in \mathbb{D}$, if $|T_d| = 0$, then select a node $n \in N_1^d$ to dominate d. Hence, because n is a neighbor, both d and n has $Domin$ set to 0 and are moved to set I.*

Algorithm 2: r-Tree

```
Data  : G = (V, E), r, n, HT (hash table), I
begin
    R = 𝔑_{r,n} ∩ I ;
    for j ∈ R do
        j.color = White;  j.dist = ∞;  j.parent = ∅;
    n.color = Gray;  n.dist = 0;  n.parent = ∅;  Q = {n};
    while Q ≠ ∅ do
        h = Head[Q];  O = N_1^h ∩ R;
        foreach j ∈ O do
            if j.color == White then
                j.color = Gray;  j.dist = dist.h + 1;
                j.parent = h;  Enqueue(Q, j);
        /* do not insert in HT the starting node */
        if h ≠ n then
            InsertHashTable(HT, h, h.dist);
        Dequeue(Q);  h.color = Black;
end
```

Algorithm 3: isKCovered

```
Data  : G = (V, E), r, n, K, I
begin
    R = 𝔑_{r,n} ∩ I;
    for j ∈ R do
        j.color = White;
    n.color = Gray;  Q = {n};
    while Q ≠ ∅ do
        h = Head[Q];  O = N_1^h ∩ R;
        foreach j ∈ O do
            if j.color == White AND j ∈ I then
                j.color = Gray;  K = K + 1;
                if K ≥ n.Domin then
                    return ;
                Enqueue(Q, j);
        Dequeue(Q);  h.color = Black;
end
```

Lemma 2.3(a) implies that a node from set I, but not in set D, may change status from dominated node to dominating node. If $c \in D$ (i.e., if a candidate is from D, then $c.Domin = 1$), then Lemma 2.3(a) also applies.

THEOREM 2.3. *Algorithm 1 with the proposed modifications correctly computes a* total (k, r)-dominating set *for any arbitrary connected graph $G = (V, N)$.*

PROOF. The theorem follows from Lemma 2.3 and Theorem 2.1. □

2.3 Reliable (k,r)-dominating set

A *reliable (k,r)-dominating set* for a graph $G = (V, E)$ is a set $D \subset V$ such that every node v not in D is within distance r from at least k nodes in D, and every node r in D is adjacent to at least one node in D.

Whenever a node n is added to the dominating set and the node is not adjacent to any other node in D, a node from N_1^n is also

Table 2: Network Diameter and Average Node Degree

# of nodes	Network Diameter	Average Node Degree
100	10.41 ± 0.12	7.85 ± 0.06
150	12.77 ± 0.11	8.04 ± 0.05
200	14.69 ± 0.15	8.19 ± 0.04
250	16.69 ± 0.13	8.28 ± 0.05
300	18.24 ± 0.13	8.4 ± 0.05
350	19.73 ± 0.13	8.44 ± 0.05
400	20.75 ± 0.13	8.53 ± 0.05
450	22.27 ± 0.13	8.61 ± 0.05
500	23.05 ± 0.12	8.76 ± 0.04

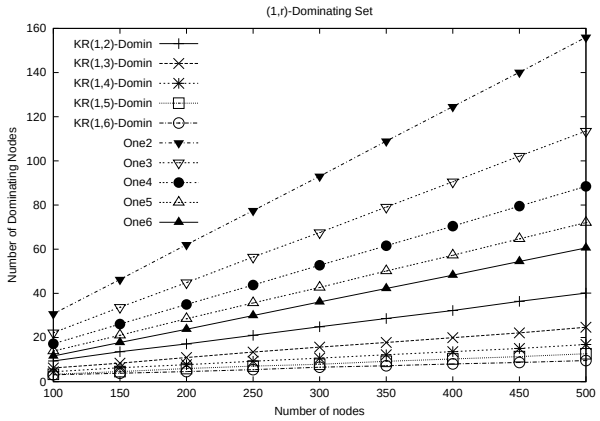

Figure 2: Computing $(1, r)-$dominating set: *OneR* versus *KR*

selected for the set D. If there is a neighbor j of node n (i.e., $j \in N_1^n$) that cannot be satisfied then select node j. If all one-hop neighbors can be satisfied, then select the one-hop neighbor with the largest one-hop neighborhood set (excluding those neighbors shared with node n).

LEMMA 2.4. *For every node n selected for D, if no other node in N_1^n is a dominating node, then one node from N_1^n is selected to dominate n.*

THEOREM 2.4. *Algorithm 1 with the proposed modifications correctly computes a* reliable (k,r)-dominating set *for any arbitrary connected graph $G = (V, E)$.*

PROOF. The theorem follows from Lemma 2.4 and Theorem 2.1. □

3. SIMULATION RESULTS

For the simulations, we vary the network size (i.e., number of nodes) and measure the total number of dominating nodes. For each configuration (i.e., number of nodes) we obtain the value for the metric for 100 arbitrary networks (nodes are randomly placed over the terrain, and connectivity is tested to ensure that the network is connected). Results represent the average over the 100 different networks . The network size is varied from 100 nodes to 500 nodes. Table 2 presents the values for the network diameter, and the average node degree for all network sizes. These results show that as the network size increases so does the network diameter. But it also shows that we try to keep the same average node degree for all network sizes.

Figure 2 shows the results comparing the algorithm presented in [7] (here referenced as One*R*, with R being the distance parameter) with the (k,r)-dominating set algorithm (referenced as KR). Because the One*R* algorithm strictly follows the paths on the spanning tree, it does not take advantage of the particularities of the topology. For the configurations under consideration, the KR algorithm always produces smaller dominating sets. We can also observe that both algorithms have results within the bounds established in [7].

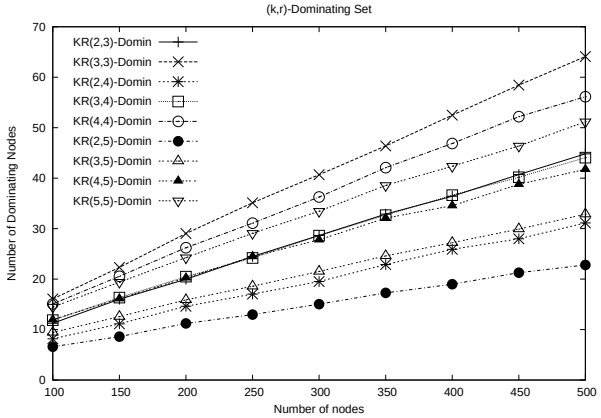

Figure 3: Computing $(k, r)-$dominating sets with *KR*

Figure 3 presents the results using the KR algorithm for several configurations. We can see that as we relax the number of dominating nodes and fix the distance parameter, the total number of dominating nodes decrease. It is noteworthy the closeness among the results for $KR(2, 3)$, $KR(3, 4)$, and $KR(4, 5)$. It suggests that if we increase both the distance and the required number of dominating nodes per node, it does not impact much the total number of dominating nodes.

4. CONCLUSIONS

The location problem in computer networks is similar to computing *Generic Dominating Sets* (GDS), also called (k, r)-dominating sets, in graphs. Prior work solves the GDS problem for *interval graphs* (IG), which are limited to very simple topologies. The GDS problem for arbitrary graphs is NP-Complete. To date, there is just one proposed algorithm for computing $(1, r)$-dominating sets for arbitrary graphs [7] (referenced in the text as One*R*, with R as the distance parameter). We presented the first known approach for computing a GDS of an arbitrary topology. Simulation results compare our solution to the algorithm One*R*, and we also present results for several other configurations.

5. REFERENCES

[1] B. Chen, K. Jamieson, H. Balakrishnan, and R. Morris. Span. In *Mobile Computing and Networking*, pages 85–96, 2001.

[2] T. H. Cormen, C. E. Leiserson, and R. L. Rivest. *Introduction to Algorithms*. MIT Press, 1990.

[3] F. Dai and J. Wu. Distributed dominant pruning in ad hoc wireless ad hoc networks, Feb 2002.

[4] M. R. Garey and D. S. Johnson. *Computers and Intractability*. Freeman, San Francisco,, 1978.

[5] G. Y. Handler and P. B. Mirchandani. *Location on Networks: Theory and Algorithms*. The MIT Press, 1979.

[6] T. W. Haynes, S. T. Hedetniemi, and P. J. Slater, editors. *Fundamentals of Domination in Graphs*. Marcel Dekker, Inc., 1998.

[7] M. A. Henning, O. R. Oellermann, and H. C. Swart. The diversity of domination. *Discrete Mathematics*, 161(1-3):161–173, December 1996.

[8] D. Joshi, S. Radhakrishnan, and C. Narayanan. A fast algorithm for generalized network location problems. In *ACM-SAC*, pages 701–8, 1993.