Efficient Dominating Sets in Spanning Trees

Anthony E. Barkauskas

Mathematics Department
University of Wisconsin – La Crosse
La Crosse, Wisconsin 54601

Abstract. An efficient dominating set S for a graph G is a set of vertices such that every vertex in G is in the closed neighborhood of exactly one vertex in S. If G is connected and has no vertices of degree one, then G has a spanning tree which has an efficient dominating set. An $O(n)$ algorithm for finding such a spanning tree and its efficient dominating set is given.

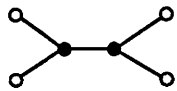
1. Introduction.

A set S of vertices in a graph $G = (V, E)$ is a *dominating set* for G if every vertex in V is either in S or adjacent to a vertex in S. We will assume $|V| = n$. (Terminology not explicitly defined in this paper will be consistent with that of [4].) If we think of a graph as a communication network, then a dominating set could be a set of broadcasting stations, each able to send a message to itself and all its neighbors, and collectively able to broadcast to all vertices of the graph. Special requirements are usually placed on a dominating set S, such as asking for a *minimum dominating set* (a dominating set S of smallest possible cardinality) in order to minimize the number of broadcasting stations which need to be constructed, or an *independent dominating set* (a dominating set S in which no two vertices of S are adjacent) in order to prevent interference from broadcasting stations which are adjacent. Hedetniemi and Laskar have written an excellent survey paper on domination and its variations [5] and have compiled a bibliography containing several hundred references [6].

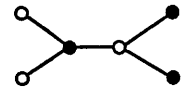
2. Efficient dominating sets.

In [2], Bange, Barkauskas and Slater introduced the concept of efficient domination. A set S of vertices is an *effcient dominating set* for G if S is a dominating set in which every vertex of G is dominated by exactly one member of S. If we call $N[u] = \{v \in V \mid v = u \text{ or } v \text{ is adjacent to } u\}$ the *closed neighborhood of* u, then S is an efficient dominating set for G if and only if every vertex in V is in the closed neighborhood of exactly one vertex in S. An efficient dominating set is both a minimum dominating set and an independent dominating set. If a communication network can be represented by a graph which has an efficient dominating set, it is possible to select a set of stations, all broadcasting on the same channel, so that the broadcasts reach every vertex with no interference caused by overlapping broadcasts. It is not always possible to achieve this type of domination, because not all graphs have efficient dominating sets. Figure 1 shows simple examples of

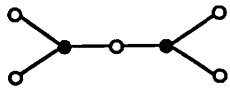
all three of these types of domination. Note that only the tree in Figure 1.d can be efficiently dominated. For two further simple examples, the path P_n always has an efficient dominating set containing $\lceil \frac{n}{3} \rceil$ vertices, while the cycle C_n has an efficient dominating set if and only if $n \equiv 0 \mod 3$.



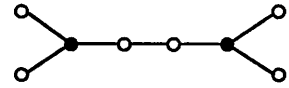
1.a. A minimum dominating set.



1.b. An independent dominating set.



1.c. A minimum and independent dominating set.



1.d. An efficient dominating set.

Figure 1. Three types of dominating sets.
Solid dots indicate vertices in the dominating set.

There is little hope that a polynomial time algorithm to find efficient dominating sets exists, since it has been shown in [3] that the problem of determining whether or not an arbitrary graph G has an efficient dominating set is NP-complete. As is usually the case in domination theory, if the graph is a tree it is much easier to find an efficient dominating set if one exists. In [3] an $O(n)$ algorithm is given for determining the maximum number of vertices of a tree which can be efficiently dominated and a constructive characterization is given for trees which have efficient dominating sets.

It is clear that both the complete graph K_n and the empty graph $\overline{K}_n$ have efficient dominating sets. Thus, given any graph G it is possible to obtain a graph G' which can be efficiently dominated either by adding edges to G or by deleting edges from G. One possible question is: given a graph G, what is the minimum number of edges which must be added to G or deleted from G in order to form such a graph G'? A different approach is to ask if there exists a subgraph G' having specified properties (for example, minimally connected) which can be efficiently dominated.

3. Efficient domination of spanning trees.

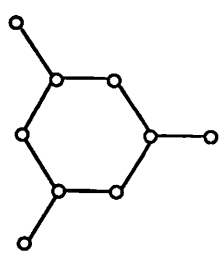
Now consider a graph as a model for a communication network in which edges represent where it is possible to construct lines of communication between vertices. We wish to decide which lines of communication need to be constructed

so that the resulting graph will be minimally connected and it will be possible to find a set of broadcasting stations which reach every vertex exactly once. Thus, we wish to find a spanning tree T for a graph G such that T has an efficient dominating set. Obviously if G is disconnected there is no spanning tree, and since not every tree has an efficient dominating set, there may be no such spanning tree even in a connected graph. The goal in this paper is to give sufficient conditions on a connected graph G to guarantee that G has a spanning tree T which has an efficient dominating set. We offer the first two observations.

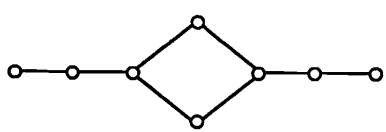
Observation 1. *If a connected graph G has an efficient dominating set, then G has a spanning tree which has an efficient dominating set.*

Proof: Let S be an efficient dominating set for G. For each vertex v in S, there is a corresponding star K_{1,n_v} consisting of v and its n_v neighbors in G. Since S is an efficient dominating set, every vertex of G is in exactly one such star. We can build a spanning tree T for G by starting with one such star as an initial tree T'. If T' does not span G, then since G is connected there exists a vertex y not in T' which is adjacent to some vertex x in T'. Extend T' to a larger subtree of G by adding edge xy and also the unique star containing y from the efficient dominating set for G. Repeat this process until T' spans G. Then T' will be a spanning tree of G which is efficiently dominated by the same set S which dominates G. ∎

If G does not have an efficient dominating set and is not a tree, G may or may not have a spanning tree which has an efficient dominating set. For example, every spanning tree of a cycle is a path, and as we have already noted, all paths can be efficiently dominated. The graph in Figure 2.a has a spanning tree which can be efficiently dominated, while the graph in Figure 2.b has no spanning tree which can be efficiently dominated.



2.a. A graph with a (unique) spanning tree which can be efficiently dominated.



2.b. A graph having no spanning tree which can be efficiently dominated.

Figure 2. Two graphs which cannot be efficiently dominated.

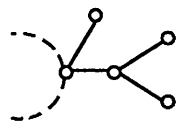
Note that the graph in Figure 2.b is just one example of an infinite class of such graphs since the two paths of length two may be replaced by any paths with length $\equiv 2 \pmod 3$.

Observation 2. *If a graph G has a Hamiltonian path, then G has a spanning tree which can be efficiently dominated.*

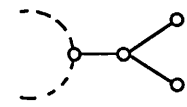
Proof: A Hamiltonian path for G is a spanning tree, and all paths can be efficiently dominated. ∎

One obvious way to attempt to construct a general algorithm for finding a spanning tree with an efficient dominating set is to proceed as follows. Pick an arbitrary vertex to be in the efficient dominating set, and let it and all its neighbors form a subtree T' of G which can be efficiently dominated. If T' does not span G and G is connected, find a vertex v not in T' adjacent to some vertex u in T'. If v has a neighbor x in G which is not in T', T' can be extended by adding edge uv together with the star formed by x and all its neighbors not in T'.

This method breaks down because it is not always possible to find such a vertex x. Vertex v might be a vertex of degree one in G, or all neighbors of v in G may already be vertices in T', none of which are leaves or vertices in the dominating set. In this case the existing T' cannot be extended to include v. Having started in this way, there is no way to overcome the problem of vertices of degree one. As can be seen in [3], some structures of the leaves in a tree determine immediately that the tree cannot have an efficient dominating set, while other structures allow the possibility, contingent upon the remaining structure of the tree. Figure 3.a shows an example of a structure which automatically precludes efficient domination of any spanning tree of G, while Figure 3.b shows a structure for which efficient domination depends on the remainder of the spanning tree.



3.a. A forbidden structure for efficient domination.



3.b. A conditional structure for efficient domination.

Figure 3. Two examples of structures which affect the possibility of an efficient dominating set.

If the vertex v is not of degree one in G, the difficulty created by having all neighbors of v in T' can be avoided by a more careful algorithm. In particular, we will see that if G is connected and has no vertices of degree one, there is always a spanning tree of G which can be efficiently dominated.

3. Main theorem

The first theorem gives a sufficient condition for a graph G to have a spanning tree which can be efficiently dominated.

Theorem 1. *If a graph G is connected and has no vertices of degree one, then G has a spanning tree which can be efficiently dominated.*

Proof: We will proceed inductively to build a rooted subtree T' of G which can be efficiently dominated. At each step we will show how to add at least one vertex to T' in such a way that the resulting larger subtree can be efficiently dominated. Since G is finite, the algorithm will terminate when T' has the same number of vertices as G. This construction will be done in such a way that at any stage, the following conditions will hold:

CONDITIONS. There will be a rooted spanning tree T of G in which each vertex v (except the root) will have a unique parent $p(v)$ and a set (possibly empty) of children which will be labeled $c_1, \ldots, c_{n_v}$. T' will be an efficiently dominated subtree of T, rooted at the same vertex. At each stage of the algorithm, the vertices of G are vertices in T and classified as follows:

I. Vertices in T'. Since any vertex v in the rooted tree T' must be dominated exactly once, exactly one of $v, p(v)$, or one child of v must be in the efficient dominating set. Thus, the vertices of T' can be labeled in the following three ways:
   1. D (for dominating). These vertices are in the efficient dominating set.
   2. DA (for dominated above). These are children (in T') of vertices labeled with D.
   3. DB (for dominated below). These are parents of vertices labeled with D.

II Vertices not in T'. These vertices can also be labeled in three ways.
   1. Vertices whose parents are in T'. There are two types.
      a. TDA (for temporarily dominated above). These are children of vertices labeled with D.
      b. TDB (for temporarily dominated below). These are children of vertices labeled with DA or DB. The labeling will be done so that all vertices with label TDB will have at least one child.
   2. U (for unused). These are vertices whose parents are not in T'.

As the algorithm proceeds, The tree T' will grow larger at each step but will not otherwise change. Thus, the labels D, DA, and DB are permanent labels. At each iteration, we will consider a vertex labeled with TDA or TDB and show how to assign it a permanent label and add it to T'. In the process it may be necessary to revise the portion of the spanning tree T which does not contain T', but such revision will satisfy the conditions above. The process will terminate when there are no more vertices labeled with TDA or TDB, because then T' will equal T and T spans G.

INITIAL STEP. Since G is connected it has a spanning tree T. We will consider T to be rooted at some vertex r. Let T' consist of the single vertex r. The single vertex r efficiently dominates T', so label it with D. Label all the children in T of r with TDA. Label all other vertices of T with the label U. Then T' and T satisfy the conditions above. (Note that there are not yet any vertices labeled with TDB, so condition II.1.b is automatically true.)

ITERATIVE STEP. Assume we have a spanning tree T for G and an efficiently dominated subtree T' containing k vertices labeled to satisfy the conditions above. If there are no vertices labeled with TDA or TDB, then $T' = T$ and we are done, otherwise we consider the two cases separately:

Case 1. Suppose v is a vertex labeled with TDA. Let the children in T of v, if any, be labeled $c_1, \ldots, c_n$. There are two subcases:

Case 1.a. No child in T of v is a leaf in T. In this case $p(v)$ is in T' and has label D, so we expand T' by adding vertex v and edge $vp(v)$, labeling v with DA, and we label the children of v with TDB. The specified conditions are satisfied, in particular, the vertices labeled with TDB have at least one child in T, and we have increased T' to $k + 1$ vertices.

Case 1.b. At least one child in T c_i of v is a leaf in T. Vertex c_i cannot be a vertex of degree one in G by hypothesis, hence, c_i must be adjacent (in G) to at least one vertex w_i different from v. Since T spans G, w_i must also be a vertex in T, and again we will consider subcases, depending on the label of w_i.

Case 1.b.i. For some leaf c_i, vertex w_i has label DA or DB. In this case we revise the tree T by deleting edge $vp(v)$ and adding edge $c_i\ w_i$ so that c_i is now the unique parent of v in T. Vertex w_i is already in T', so extend T' by adding vertex c_i, edge $c_i\ w_i$ vertex v and edge vc_i. Label vertex v with D, and label vertex c_i with DB. Label any remaining children of v with TDA. Note that vertex c_i has no children in T other than v, and v is already in T'. Note also that T is still a spanning tree for G, and that T' is a subtree of T which is efficiently dominated, the labelings satisfy the conditions given above, and T' has been increased to $k + 2$ vertices.

Case 1.b.ii. For each leaf c_i, vertex w_i has label D, or else is not in T'. In this case we revise the tree T by deleting edge vc_i and adding edge $c_i\ w_i$. If w_i is labeled with D then w_i is in T', so we can extend T' immediately by adding edge c_iw_i and vertex c_i, labeling c_i with DA. (Note that c_i has no children in T.) If w_i is not in T', then we revise T by deleting edge vc_i and adding edge c_iw_i. Note the conditions are still satisfied, in particular, any vertex labeled with TDB in the revised T must still have at least one child. Repeating this process for each leaf c_i, we eliminate all children of v which are leaves, and we can apply Case 1.a. Thus, T' has been increased to at least $k + 1$ vertices.

Case 2. Suppose v is a vertex with label TDB. Then by hypothesis v must have at least one child, call it c_0. Label the remaining children (if any) $c_1, \ldots, c_n$ as in Case 1. Again there are two subcases.

Case 2.a. Suppose none of $c_1, \ldots, c_n$ are leaves in T. Then extend T' by adding vertex v, edge $vp(v)$, vertex c_0, and edge vc_0, labeling c_0 with D and v with DB. Label the remaining children of v with TDB and the children of c_0 with TDA. The conditions are satisfied, in particular, each vertex with label TDB has at least one child, and T' has been increased to $k+2$ vertices.

Case 2.b. Suppose at least one c_i, $1 \leq i \leq n$, is a leaf in T. As in Case 1.b., c_i must be adjacent in G to at least one vertex w_i different from v in G, and there are two subcases, depending on the label for w_i.

Case 2.b.i. For some leaf c_i, vertex w_i has label DA or DB. This case is identical to Case 1.b.i, except that we must include c_0 when labeling the children of v with TDA. Thus, T' has been increased to $k+1$ vertices.

Case 2.b.ii. For each leaf c_i, vertex w_i has label D or else is not in T'. In this case we revise T exactly as in Case 1.b.ii, so that none of $c_1, \ldots, c_n$ is a leaf in T, and apply Case 2.a. Thus, T' has been increased to at least $k+2$ vertices.

Since T' is a subtree of T having the same root, and all children of vertices in T' which are not themselves in T' must be labeled with either TDA or TDB, if we repeat the iterative step until there are no vertices labeled with TDA or TDB, T' will equal T and we will have the desired efficiently dominated spanning tree for G. ∎

4. Graphs with vertice of degree one.

The conditions of Theorem 1 are by no means necessary. Observation 1 and Observation 2 have already shown that some graphs with vertices of degree one have spanning trees which can be efficiently dominated. It is also possible to extend Theorem 1 slightly.

Observation 3. *If G is a connected graph and all vertices of degree one in G are adjacent to a single vertex r, then G has a spanning tree which can be efficiently dominated.*

Proof: Let T be a spanning tree for G rooted at r. Let r be labeled with D, each child of r of degree one be labeled with DA, and the other vertices be labeled according to the conditions of Theorem 1. The conditions of Theorem 1 are satisfied, and the iterative step may be applied repeatedly to obtain a spanning tree which can be efficiently dominated. ∎

Figure 2.a gives an example of a graph G which satisfies none of the conditions listed so far, yet G has a spanning tree which can be efficiently dominated. It is clear that if G is to have a spanning tree which can be efficiently dominated, it must have a subtree T' which contains all the vertices of degree one of G, and

T' can be efficiently dominated. The graph in Figure 2.b shows that this is not enough, because although a path containing the two endpoints can be efficiently dominated, the graph has no spanning tree which can be efficiently dominated. In this case the iterative step of Theorem 1 cannot be applied because any spanning tree T which contains the path T' would have a leaf with label TDB, a violation of the conditions on T. This is inevitable because after constructing T', the remaining vertex of G is adjacent only to vertices with labels DA or DB. This observation leads to the following theorem.

Theorem 2. *Let G be a connected graph. Then G has a spanning tree which can be efficiently dominated if and only if there exists a subtree T' of G satisfying the following three conditions:*

1. *T' contains all the vertices of degree one of G;*
2. *T' can be efficiently dominated;*
3. *Every vertex in G which is not in T' is adjacent either to one of the vertices in the dominating set for T' or to some vertex in G which is not in T'.*

Proof: If G has a spanning tree T which can be efficiently dominated, then the three conditions are trivially satisfied by T.

Conversely, if the three conditions are satisfied, then root T' at some vertex. Take an efficient dominating set for T' which satisfies condition 3, and label the vertices of T' appropriately as D, DA, or DB. Since G is connected, T' can be extended to a spanning tree T for G. Consider each vertex v of T which is not in T'. If $p(v)$ has label D, label v with TDA. If $p(v)$ has label DA or DB, then if v is not a leaf of T, label v with TDB, otherwise, if v is a leaf of T, then by hypothesis v is adjacent to some vertex w which is either in D or is not in T'. If w is in D, revise T by deleting edge $vp(v)$ and adding edge vw, labeling v with TDA. If w is not in T', revise T by deleting edge $vp(v)$ and adding edge vw, labeling v with U. Label all vertices of T whose parents are not in T' with label U. Then T and T' satisfy the conditions of Theorem 1, and the iterative step may be applied to extend T' to an efficiently dominated spanning tree for G. ∎

5. Algorithm.

Theorem 1 and Observation 3 justify the following algorithm.

INPUT: A connected graph G and a vertex r. If G has any vertices of degree one, they must either be the vertex r or be adjacent to r.

OUTPUT: A spanning tree T rooted at r and a labeling of the vertices of T which shows the efficient dominating set for T.

```
PROGRAM EfficientTree;
  TYPE:
    labels: (D, DA, DB, TDA, TDB, U);
  VAR:
    G: graph;
    T: tree;
    v, c, w, r: vertex;
    Q: Queue_of_vertices;
    done: boolean;
    value: labels;
PROCEDURE LabelandEnter (Q:Queue_of_vertices; v: vertex; value: labels);
  BEGIN
  label(v):= value;
  Enter v in Q;
  END;
PROCEDURE Processchildren (T: tree; v, c: vertex; done: Boolean);
      (*Comment: Vertex v in tree T has an ordered set of children. Starting
      with either the first or second child of v, this procedure either
      assigns a permanent label to v, making done = TRUE, or else transfers
      each child of v of degree one to another vertex in T.*)
    BEGIN
    WHILE ((c is a child of v) and (not done)) DO
        BEGIN
        IF (c is not a leaf) THEN c:= nextchild of v
        ELSE
          BEGIN
          w:= any vertex which is adjacent to c in G but is not equal to v;
            (*Comment: c cannot be r or a child of r, hence c
            is not of degree one in G and such a vertex w must exist.*)
          CASE (label (w)) OF
              DA, DB:
                BEGIN
                Revise T by making w the parent of c, and c the parent of v;
                  (*Comment: the new tree still spans G. No vertices
                  labeled D, DA or DB have been affected.*)
                label (c):= DB;
                label (v):= D;
                FOR (each child c of v) DO LabelandEnter (Q, c, TDA);
                done:= TRUE;
                END
              D, TDA, TDB, U:
                BEGIN
```

```
                    Revise T by making w the parent of c;
                      (*Comment: T still spans G.*)
                    IF (label (w) = D) THEN label (c):= DA;
                    c:= nextchild of v;
                      END
                  ENDCASE;
                ENDELSE;
        ENDWHILE;
END;

BEGIN (*EfficientTree*)
INPUT (G);
FOR (each vertex v of G) DO label (v):= U;
INPUT (r); (*Comment: Any vertices of G of degree one
              must be adjacent to r or equal r.*)
SpanningTree (G,T,r); (*Comment: Find a spanning tree of G rooted at r.
                         G must be connected.*)
label (r):= D;
FOR each child c of r DO LabelandEnter (Q, c, TDA);
WHILE (not EmptyQueue (Q)) DO
  BEGIN
  v:= LeaveQueue (Q);
      (*Comment: Note that all vertices in Q are labeled TDA or TDB*)
  done := FALSE;
  IF (label (v) = TDA) THEN
    BEGIN
    Processchildren (T, v, firstchild (v), done);
      (*Comment: Start with the first child of v.*)
    IF (not done) THEN (*Comment: v has no children which are leaves.*)
        BEGIN
        label (v):= DA;
        FOR each child c of v DO LabelandEnter (Q, c, TDB);
        ENDIF;
    END
  ELSE (*Comment: label (v) equals TDB and v will have at least one child.*)
    BEGIN
    Processchildren (T, v, secondchild (v), done);
        (*Comment: Start with the second child of v.*)
    IF (not done) THEN
        (*Comment: None of the children of v after firstchild will be a leaf.*)
      BEGIN
      label (v):= DB;
```

```
        label (firstchild (v)):= D;
        FOR (each child c of firstchild (v)) DO LabelandEnter (Q, c, TDA);
        FOR (each child c of v other than firstchild) DO
          LabelandEnter (Q, c, TDB);
        ENDIF;
      ENDELSE;
    ENDWHILE;
OUTPUT (T with vertex labels);
END.
```

Each vertex v of the graph appears at most once in the queue. Processchildren inspects each child c of vertex v in the spanning tree T, possibly moving c to be inspected again as the child of a different vertex, but no child is inspected more than twice. If T has n vertices, $n-1$ of them are children, so the main loop of EfficientTree is $O(n)$. SpanningTree is $O(n)$ if implemented properly [9, p. 17]. Thus, the time complexity of algorithm EfficientTree for finding a spanning tree of G which has an efficient dominating set is $O(n)$.

6. Conclusion.

Theorem 1 gives a sufficient condition for a graph G to have a spanning tree which can be efficiently dominated, and algorithm EfficientTree finds a spanning tree and its efficient dominating set in linear time for such a graph G. Theorem 2 gives a necessary and sufficient condition for an arbitrary graph G to have a spanning tree which can be efficiently dominated, but does not suggest how to find such a tree. One method would be to find all the spanning trees of G, and check each one for an efficient dominating set using the $O(n)$ algorithm in [3]. This method is not efficient because the number of spanning trees grows exponentially with the order of the graph for dense graphs. In particular, if t denotes the number of spanning trees of a graph, then $t = n^{n-2}$ for the complete graph K_n. An $O(n + e + e \cdot t)$ algorithm for finding all spanning trees of G is discussed in [8, pp. 325-326], and a slightly improved algorithm can be found in [7]. It may be possible to use the structure of trees with efficient dominating sets to modify these algorithms to consider only a small portion of the spanning trees of G, and thus develop an efficient algorithm to apply Theorem 2, but this remains an open problem.

References

1. D. Bange, A. Barkauskas, L. Host, and P. Slater, *Efficient near-domination of grid graphs*, Congressus Numerantium 58 (1987), 83–92.
2. D. Bange, A. Barkauskas, and P. Slater, *Disjoint dominating sets in trees*, Sandia Laboratories Report, SAND 78-1087-J (1978).
3. D. Bange, A. Barkauskas, and P. Slater, *Efficient dominating sets in graphs*, in "Applications of Discrete Mathematics", R. Ringeisen, F. Roberts, eds., SIAM, 1988, pp. 189–199.
4. G. Chartrand and L. Lesniak, "Graphs and Digraphs", 2nd ed., Wadsworth, Belmont CA, 1986.
5. S. Hedetniemi and R. Laskar, *Recent results and open problems in domination theory*, in "Applications of Discrete Mathematics", R. Ringeisen, F. Roberts, eds., SIAM, 1988, pp. 205–218.
6. S. Hedetniemi and R. Laskar, *Bibliography on domination in graphs and some basic definitions*, Discrete Mathematics 86 (1990), 257–277.
7. R. Jayakumar, K. Thulasiraman, and M.N.S. Swamy, *MOD-CHAR: An implementation of Char's spanning tree enumeration algorithm and its complexity analysis*, IEEE Trans. Circuits Syst. 36 (2), (February, 1989), 219–228.
8. E.M. Reingold, J. Nievergelt, N. Deo, "Combinatorial Algorithms, Theory and Practice", Prentice-Hall, Englewood Cliffs, NJ, 1977.
9. R. Tarjan, *Data structures and network algorithms*, SIAM Regional Conferences Series 44 (1983).