CIS-TR-80-14

OPTIMUM DOMINATION IN UNICYCLIC GRAPHS*

by

Sandra Mitchell Hedetniemi**

Abstract

A dominating set is a set of vertices of a graph such that every vertex not in the set is adjacent to a vertex in the set. The problem of determining a minimum dominating set in an arbitrary graph is known to be NP-complete. An algorithm which is linear in the number of vertices is presented for unicyclic graphs. This result extends the classes of graphs for which linear algorithms exist, i.e. trees and interval graphs.

* Research supported in part by the National Science Foundation under Grant MCS-79-03913.

** Department of Computer and Information Science, University of Oregon, Eugene, OR 97403.

1. Introduction

Let $G = (V,E)$ be an undirected graph. A vertex $v \in V(G)$ is said to <u>dominate</u>, or cover, all the vertices adjacent to it. A <u>dominating set</u> $D \subseteq V(G)$ is a set of vertices such that every vertex not in D is adjacent to a vertex in D. The <u>domination number</u> $d(G)$ is the minimum number of vertices in a dominating set. If a graph represents a network of housing neighborhoods, then a dominating set of vertices represents the minimum number of neighborhoods which must have extra facilities such as parks, elementary schools or fire stations. The problem of finding a minimum dominating set in an arbitrary graph is known to be NP-complete (cf. Garey and Johnson [3]). Furthermore, Booth [1] has shown that the problem is still NP-complete for the class of chordal graphs. However, Cockayne, Goodman and Hedetniemi [2] have shown that when chordal graphs are restricted to trees, a solution can be found in time linear in the number of vertices. Natajaran and White [5] present a linear solution to the generalized problem for trees where the vertices and edges are weighted and the dominating set has minimum weight among all dominating sets. Slater [6] generalized the domination to r-domination. The r-domination problem requires that each vertex either be in the set or within distance r to a vertex in the set. Kariv and Hakimi [4] extended Slater's linear solution to a linear solution on trees with both vertex and edge weights.

Booth [1] also has a linear solution to the domination problem on a subclass of planar graphs, interval graphs. In this paper, we present a linear solution for another subclass of planar graphs, unicyclic graphs.

2. Unicyclic graphs

A <u>unicyclic graph</u> is a connected graph with exactly one cycle. Trees and unicyclic graphs have similar recursive definitions. A graph G has a recursive definition if G can be constructed by a finite number of applications of operations chosen from a set O of operations, beginning with a graph g_0 chosen from an allowed set G_0 of initial graphs. For the class of trees, G_0 = {single edge} and O = {join a new vertex to an existing vertex}. The class of unicyclic graphs is obtained using the same set of operations but changing G_0 = {cycles}.

We will refer to those vertices in g_0 of a unicyclic graph, i.e. those vertices on the cycle, as <u>cycle vertices</u>. All other vertices in the unicyclic graph will be called <u>tree vertices</u>.

The similarities in the recursive definitions of trees and unicyclic graphs suggest that the procedure for finding a dominating set in a unicyclic graph should be similar as well.

3. Mixed domination of unicyclic graphs

Cockayne, Goodman and Hedetniemi [2] solve the mixed domination problem on trees, a generalization of domination. Let the vertices of a graph G be partitioned into three subsets, V_1, V_2, and V_3, where V_1 contains <u>free</u> vertices, V_2 contains <u>bound</u> vertices, and V_3 contains <u>required</u> vertices A <u>mixed dominating set</u> in G is a set S of vertices which contains all required vertices, i.e. $V_3 \subseteq S$, and which dominates all bound vertices, i.e. every $v \in V_2$ is either in S or adjacent to a vertex in S. Free vertices need not be dominated by S; however, they may be used to dominate bound vertices. The <u>mixed dominating number</u> md(G) is the minimum order of a mixed dominating set in G; such a set is called an <u>md-set</u>. If all the vertices are bound, then md(G) = d(G).

The correctness and construction of their algorithm is based on a theorem which allows them to prune an endvertex from a tree. A vertex in a graph is an <u>endvertex</u> if it has degree one. However, the proof of the theorem makes use of no other property of the tree; hence, the theorem holds for any graph with an endvertex. The theorem which follows is the theorem of Cockayne, Goodman and Hedetniemi with the word tree replaced by the word graph.

Theorem 1. (Cockayne, Goodman and Hedetniemi). Let G be a graph having free, bound and required vertices, V_1, V_2, and V_3, respectively. $V \geq 3$. Let v be an endvertex of G which is adjacent to u.

(i) if $v \varepsilon V_1$, then md(G) = md(G-u),

(ii) if $v \varepsilon V_2$ and G' is the graph which results by deleting v and relabeling u as "required", then md(G) = md(G'),

(iii) if $v \varepsilon V_3$ and $u \varepsilon V_3$, then md(G) = md(G-v) + 1,

(iv) if $v \varepsilon V_3$ and $u \varepsilon V_3$, and if G' is the graph which results from deleting v and relabeling u as "free", then md(G) = md(G') + 1.

The construction and correctness of the mixed domination algorithm for a unicyclic graph with free, bound and required vertices is based upon the pruning as defined in Theorem 1. After all endvertices of a unicyclic graph have been iteratively removed, then the intial graph g_0 (which is a cycle) will remain. The following three results specify the correct processing of this cycle. The proof of the last result is obvious and is therefore omitted.

Lemma 2. Let C = [1..k] be a cycle having k vertices; which are free, bound and required. Let v be required and adjacent to u and w. Then md(C) = md(C') + 1, where C' = C - v with u and w relabeled free if they are not already required.

Proof. Since v is required, it must be included in the dominating set. Any vertices adjacent to v, i.e. w or u are dominated by v; hence, they will only be included if also required. ∎

Lemma 3. Let C = [1..k] be a cycle having k vertices; vertices are only free and bound. Let v be a free vertex adjacent to vertices u and w.

(i) If either u or w is free, then md(C) = md(C-v).

(ii) If neither u nor w is free, then md(C) = min {md(C-v), md(C')+1}, where C' = C - v with u and w relabeled free.

Proof. Let v be a free vertex in a cycle with only free and bound vertices; let u and w be adjacent to v.

(i) Without loss of generality, assume u is free. If w is also free, md(C) = md(C-v). Therefore, assume w is bound. Let x be the other vertex adjacent to w. At most one of x, w, or v must be taken since v is free; and at least one of these vertices must be included in the dominating set since w is bound. Hence, md(C) = md(C-v) since omitting v will not result in a larger dominating set.

(ii) Both u and w are bound and must be dominated. Either they are dominated by including v in the dominating set or they are not. If they are dominated by v, then this is equivalent to adding 1 to the domination number, removing v and relabeling u and w as required. If they are not, then this is equivalent to just removing v. ∎

Lemma 4. Let C = [1..k] be a cycle having k bound vertices. Let $v \in C$. Then md(C) = md(C'), where C' = C - v and the two vertices adjacent to v have been relabeled free.

Algorithm DOMINATION is immediately suggested by Theorem 1 and Lemmas 2, 3 and 4. For completeness, we first present a shortened version of the tree domination algorithm of Cockayne, Goodman and Hedetniemi as a subalgorithm which prunes an endvertex v and updates the label of u adjacent to v in G.

```
Algorithm PRUNE (V, U, G, DOM)
    case label of v:
        free: G ← G - V
        bound:  u is relabeled required
                G ← G - V
        required:  if u is bound
                       then relabel u as required
                       fi
                   DOM ← DOM ∪ {V}
                   G ← G - V
        endcase.                                                    ■
```

We next present a subalgorithm which processes the cycle after a vertex has been removed. The resulting graph is a path which is processed by repeated calls to PRUNE.

```
Algorithm PROCESS_PATH (G, DOMG)
    while there exists an endvertex V adjacent to U in G do
        PRUNE (V, U, G, DOMG)
        od
    [let V be the vertex which remains]
    if V is bound or V is required
        then DOMG ← DOMG ∪ {V}
        fi.                                                         ■
```

Algorithm DOMINATION. Given a unicyclic graph G with free, bound and required vertices, find a mixed domination set DOMG with minimum cardinality.

```
0. [Prune endvertices]
   REQUIRED_PTR ← Ø
   FREE_PTR ← Ø
   DOMG ← Ø
   while there exists an endvertex V adjacent to U in G do
        if V is required
            then REQUIRED_PTR ← V
            else  if V is free
                        then FREE_PTR ← V
                        fi
            fi
        PRUNE (V,U,G,DOMG)
        od
1.  [Process the cycle vertices]
   if REQUIRED_PTR ≠ Θ
        then [let U and W be adjacent to V = REQUIRED_PTR and apply Lemma 2]
             DOMG ← DOMG ∪ {V}
             label u and w as free
             G  ← G - v
             PROCESS_PATH (G,DOMG)
        else if FREE_PTR ≠ Ø
                  then [let U and W be adjacent to V = FREE_PTR; apply Lemma 3]
                       if U is free or W is free
                           then G ← G - V
                                PROCESS_PATH (G,DOMG)
                           else  H ← G - V with U and W relabel  free
                                 DOMH ← DOMG
                                 PROCESS_PATH (H,DOMH)
                                 G ← G  - V
```

```
                    PROCESS_PATH (G,DOMG)
                    DOMG ← smaller {DOMG, DOMH}
                fi
            else [G contains only bound vertices; apply Lemma 4 with
                U and W adjacent to arbitrarily chosen V]
                G ← G - V with U and W relabeled free
                PROCESS_PATH (DOMG,G)
            fi
        fi.
```

Algorithm DOMINATION is easily shown to be linear in the number of vertices. Each tree vertex is processed exactly once, using the constant time execution of Algorithm PRUNE. The choice of the first cycle vertex to be processed can be made in constant time and its subsequent processing requires constant time. If this first cycle vertex chosen is free, it requires the remaining cycle vertices to be processed twice rather than once. In either of the three cases the processing of each of the remaining cycle vertices requires constant time. Hence, since each vertex is processed in constant time, Algorithm DOMINATION is linear in the number of vertices.

Acknowledgement. The author would like to thank Steve Hedetniemi for his numerous useful discussions about the correctness of the algorithm.

4. References

[1] Booth, K. Private communication, 1980.

[2] Cockayne, E., Goodman, S. and Hedetniemi, S. T., A linear algorithm for the domination number of a tree, Information Processing Lett., 4(1975), 41-44.

[3] Garey, M. and Johnson, S. Computers and Intractability: a guide to NP-completeness theory, (W. H. Freeman and Co.: San Francisco), 1979.

[4] Kariv, O. and Hakimi, S. An algorithmic approach to network problems I: the p-centers, SIAM J. Appl. Math., 37(1979), 513-538.

[5] Natajaran, K. and White, L. Optimum domination in weighted trees, Information Processing Lett., 7(1978), 261 - 265.

[6] Slater, P. R-domination in graphs, J. Assoc. Comput. Mach., 23(1976), 446-450.