A Linear Algorithm for Universal Minimal Dominating Functions in Trees

E.J. Cockayne, University of Victoria, B.C., Canada
G. MacGillivray, University of Regina, Sask., Canada
C.M. Mynhardt, University of South Africa, Pretoria, South Africa

Abstract. A *dominating function* is a feasible solution to the LP relaxation of the minimum dominating set 0–1 integer program. A minimal dominating function (MDF) $\mathbf{g}$ is called *universal* if every convex combination of $\mathbf{g}$ and any other MDF is also a MDF. The problem of finding a universal MDF in a tree T can also be described by a linear program. This paper describes a linear time algorithm that finds a universal MDF in T, if one exists.

1. Introduction.

Let $G = (V, E)$ be a graph. The problem of finding a minimum dominating set in G can be described by the following 0–1 integer program P.

P: $\min \sum_{v \in V} f_v$
subject to
$$(A + I)\mathbf{f} \geq \mathbf{1}$$
$$f_v \in \{0, 1\} \qquad \text{for all } v \in V,$$
where A is the adjacency matrix of G, $\mathbf{f}$ is indexed by the elements of V, and $\mathbf{1}$ is the all-ones vector.

The question of when an optimal solution to the linear programming (LP) relaxation of P yields an optimal solution to P was first studied by Farber [7].

We define a *dominating function* of a graph G to be a feasible solution $\mathbf{f}$ to the LP relaxation of P, where $f_v \geq 0$ for all $v \in V$. This generalises the notion of a dominating set in a graph. (If $f_v \in \{0, 1\}$ for all $v \in V$, then $\{v: f_v = 1\}$ is a dominating set of G.) Equivalently, a dominating function of G can be defined to be a function

$$f: V \to [0, 1]$$
such that
$$\sum_{u \in N[v]} f(u) \geq 1, \text{ for all } v \in V,$$
where $N[v]$ is the closed neighbourhood of the vertex v.

Let $\mathbf{f}$ and $\mathbf{g}$ be dominating functions of a graph G. We write $\mathbf{f} \leq \mathbf{g}$ if $f_v \leq g_v$ for all $v \in V$. If $\mathbf{f} \leq \mathbf{g}$ and there is some $v \in V$ for which $f_v < g_v$ we write $\mathbf{f} < \mathbf{g}$. A minimal dominating function (MDF) is a dominating function $\mathbf{f}$ such that there is no dominating function $\mathbf{g}$ with $\mathbf{g} < \mathbf{f}$.

Although any convex combination of dominating functions is also a dominating function, it is not true that every convex combination of MDFs is again an MDF. It was proved in [4] that either every convex combination of two MDFs $\mathbf{f}$ and $\mathbf{g}$ is an MDF, or no convex combination of $\mathbf{f}$ and $\mathbf{g}$ is an MDF. We are thus led to study the binary relation R on the set of MDFs of a graph G, defined by $\mathbf{f}\, R\, \mathbf{g}$ just if every convex combination of $\mathbf{f}$ and $\mathbf{g}$ is an MDF of G. We say that an MDF $\mathbf{f}$ is *universal* if $\mathbf{f}\, R\, \mathbf{g}$ for every MDF $\mathbf{g}$ of G.

The study of universal MDFs was initiated in [4], where various conditions for the existence and non-existence of universal MDFs of graphs were presented. Universal MDFs of trees were investigated in [5].

In this paper we describe a linear time algorithm that finds, for a given tree T, a universal MDF all of whose values are zero or one (a 0–1 universal MDF), if one exists. It is proved in [6] that a tree has a universal MDF if and only if it has a 0–1 universal MDF.

Linear algorithms for related domination problems on trees are presented in [1–3,7,8].

2. The Algorithm.

Let T be a tree. We use L to denote the set of *leaves* of T, that is

$$L = \{v \in V : d(v) = 1\}.$$

A vertex is called *remote* if it is adjacent to one or more leaves. We use R to denote the set of remote vertices of T,

$$R = \{v \in V : N(v) \cap L \neq \emptyset\},$$

(where $N(v)$ is the open neighbourhood of v). Unless $T = K_2$, $L \cap R = \emptyset$.

Let G be a graph, and let $\mathbf{f}$ be a dominating function of G. The *boundary* of $\mathbf{f}$ is defined to be the set

$$B_{\mathbf{f}} = \{v : \sum_{u \in N[v]} \mathbf{f}_u = 1\}.$$

We now define a special type of vertex which plays a central role in the existence of universal MDFs of trees. Let $\mathbf{f}$ be a MDF of a tree T. A vertex v is called *f-cool* if $B_{\mathbf{f}} \cap N[v]$ is contained R. We say that v is *cool* if v is $\mathbf{f}$-cool for some MDF $\mathbf{f}$ of T, and use C to denote the set of cool vertices of T. Since, by minimality, each leaf belongs to the boundary of every MDF, $R \cap C = \emptyset$. The location and arrangement of cool vertices to a large extent determines whether a tree has a universal MDF (this is explained in [5], also see Theorem 2.2 below).

2.1. Lemma. [5]. *The vertex v is a cool vertex of a tree T if and only if*

(i) *$d(u) \geq 3$ for each $u \in N(v) - R$ and*

(ii) *$N(v)$ contains at least two vertices, each of which is adjacent to at least two vertices of $V - R$.*

2.2. Theorem. [5]. *The MDF $\mathbf{g}$ of a tree T is a universal MDF if and only if*

(i) *$\mathbf{g}_v = 0$ for all cool vertices v and*

(ii) *$B_{\mathbf{g}} \supseteq V - R$.*

It follows from the definition of a convex set that the set of universal MDFs of a graph is convex (see [7]). For trees it follows from the above results that the existence of a universal MDF is equivalent to the existence of a feasible solution to the following linear program Q.

$$Q: \quad \min \sum_{v \in V} x_v$$
subject to
$$\sum_{u \in (N[v]-C)} x_u = 1 \qquad \text{for all } v \notin R$$
$$x_v \geq 0 \qquad \text{for all } v \in V.$$

(The choice of the objective function is not important.)

We now describe a linear time algorithm for the 0–1 integer program corresponding to Q (i.e., $x_v \in \{0, 1\}$, for all $v \in V$).

Let T be a tree rooted at r and $s \in V(T)$. We denote by T_s the subtree of T, rooted at s, induced by s and all descendants of s.

The existence of a 0–1 universal MDF of a tree T is clearly equivalent to the existence of a subset S of V for which

(i) $S \cap C = \emptyset$, and
(ii) $|S \cap N[v]| \geq 1$ for each vertex v, with equality if $v \notin R$.

We call such a set S a *universal minimal dominating set* (universal MDS). (We note that condition (ii) assures that S is a minimal dominating set because every vertex contains in its closed neighbourhood at least one non-remote vertex.) Let $v \in V(T)$. A subset S_v of $V(T_v)$ which has properties (i) and (ii) above, where C and R are taken with respect to T, is called a *u-set for* T_v.

The algorithm is based on the following observation.

2.3. Lemma. *Let T be a tree rooted at r. Then T admits a universal MDS (i.e., T has a 0–1 universal MDF) if and only if for each vertex $v \neq r$ at least one of the following three conditions holds, and (i) or (ii) holds for $v = r$:*

(i) *T_v has a u-set S_v with $v \in S_v$,*
(ii) *T_v has a u-set S_v with $v \notin S_v$, or*
(iii) *there is a subset X of $V(T_v)$ such that*
  (a) $v \in X$, and
  (b) for every $u \in N_{T_v}(u)$, $S_u = X \cap V(T_u)$ is a u-set for T_u with $u \notin S_u$.

Proof: The proof is easy, and is omitted. ∎

We first give an overview of the algorithm. Suppose T is rooted at r. We work inwards towards r from the leaves of T. A vertex v is processed after all vertices $u \in N_{T_v}(v)$ have been processed (thus r will be the last vertex to be considered). Each vertex can be assigned the labels I, N, F, according to which of conditions (i), (ii), (iii) of Lemma 2.3 holds, respectively (also see Table 2.1). It is possible for a vertex to recieve more than one of these labels, or none of them. The procedure for labelling a vertex is outlined in Lemmas 2.4, 2.5 and 2.6. If some vertex v can not be assigned at least one of the labels I, N, F, then by Lemma 2.3 the tree T has no universal MDS. If every vertex can be labelled, Lemma 2.3 states that T admits a universal MDS if and only if its root r recieves at least one of the labels I or N (if the only label assigned to r is F, then the corresponding set dominates no vertex in $N[r]$).

A careful choice of the root can simplify the labelling rules (cf. the proof of Lemma 2.4). Suppose T is rooted at $r \notin L$. Let v be a remote vertex adjacent to the leaf l. The unique (r, l)-path in T contains v. Therefore, for any $w \in V(T) - \{l\}$, whenever v is in $V(T_w)$, it is adjacent to a member of L (namely l). This choice of r reduces the number of cases which must be considered because it makes it impossible for a remote vertex to be labelled F.

We want to produce the universal MDS when it exists. This requires some minor additions to the procedure described above. With each vertex v we associate three sets: I_v, N_v and F_v. If v is labelled I, N, F, then I_v, N_v, F_v are, respectively, a u-set S_v with $v \in S_v$ a u-set S_v with $v \notin S_v$ and a subset X of $V(T_v)$ such that $v \notin X$ and for every $u \in N_{T_v}(v)$, $S_u = F_v \cap V(T_u)$ is a u-set for T_u with $u \notin S_u$. Each of these sets can be constructed from the collection $\{I_u, N_u, F_u : u \in N_{T_v}(v)\}$ according to rules easily derived from the labelling procedure (eg. see the proof of Lemma 2.4 below).

Label of vertex v	**Meaning**
I	T_v has a u-set S_v with v In S_v
N	T_v has a u-set S_v with v Not in S_v
F	There is a subset X of $V(T_v)$ such that v Not in X, and for every u In $N_{T_v}(v)$, v Not in X, v Not in X, u-set for T_u with u Not in S_u. If such a set X is the intersection of a universal MDS S and $V(T_u)$, then v is Forced to be in S.

Table 2.1 The vertex labels and their meanings

The correctness of the algorithm follows from Lemma 2.3 and following Lemmas, from which the labelling rules are derived.

2.4. Lemma. *Let T be a tree rooted at $r \notin L$, and $v \in V$. Then T_v has a u-set S_v with $v \in S_v$ (i.e., v can be labelled I) if and only if $v \notin C$, every $u \in N_{T_v}(v) - R$ is labelled F and,*

(i) *if $v \notin R$, every $w \in N_{T_v}(v) \cap R$ is labelled N, or*
(ii) *if $v \in R$, every $w \in N_{T_v}(v) \cap R$ is labelled I or N.*

Proof: ($\Rightarrow$) Suppose T_v has a u-set S_v with $v \in S_v$. Since a u-set for T_v contains no cool vertices, $v \notin C$. Consider $u \in N_{T_v}(v) - R$. Since $|S_v \cap N[u]| = 1$, the vertex u is not dominated by any vertex of S_v except v. Thus the subset $S = S_v \cap V(T_u)$ of $V(T_u)$ has properties (a) and (b) of Lemma 2.3 (iii). That is, u is labelled F.

Suppose $v \notin R$. Consider $w \in N_{T_v}(v) \cap R$. An argument similar to the above shows $w \notin S_v$ By the choice of r, the vertex w is adjacent in T_w to a member of L. Therefore each leaf of T_w adjacent to w is in S_v (otherwise it would be undominated by S_v). Hence the set $S = S_v \cap V(T_w)$ is a u-set for T_w with $w \notin S$. That is, w is labelled N.

Finally, suppose $v \in R$. Consider $w \in N_{T_v}(v) \cap R$. Since a u-set S_v for T_v can dominate w (and v) any positive number of times, w may or may not belong to S_v. If $w \notin S_v$ then, as above, w is labelled N (recall that w cannot be labelled F). Suppose $w \in S_v$. Then $S = S_v \cap V(T_w)$ is a u-set for T_w with $w \in S$. That is, w is labelled I.

($\Leftarrow$) Suppose $v \notin C$, and every $u \in N_{T_v}(v) - R$ is labelled F. Then for every $u \in N_{T_v}(v) - R$ there is a subset X_u of $V(T_u)$ such that $S_x = X_u \cap V(T_x)$, where $x \in N(u)$, is a u-set for T_x with $x \notin S_x$. Suppose condition (i) holds, that is, $v \notin R$ and every $w \in N_{T_v}(v) \cap R$ is labelled N. Then each tree T_w has a u-set S_w with $w \notin S_w$. Therefore

$$S_v = \left(\bigcup_{u \in N_{T_v}(v) - r} X_u \right) \cup \left(\bigcup_{w \in N_{T_v}(v) \cap R} S_w \right) \cup \{v\}$$

is a u-set for T_v with $v \in S_v$.

Now suppose condition (ii) holds, that is, $v \in R$ and every $w \in N_{T_v}(v) \cap R$ has a u-set S_w. Depending on the labels assigned to w, the set S_w may or may not contain w. In either case the set S_v constructed as above is a u-set for T_v with $v \in S_v$. ∎

A similar argument can be used to establish the following two Lemmas.

2.5. Lemma. *Let T be a tree rooted at $r \notin L$, and $v \in V$. Then T_v has a u-set S_v with $v \notin S_v$ (i.e., v can be labelled N) if and only if $v \notin L$ and either*

(i) *$v \in R$ and every $u \in N_{T_v}(v)$ is labelled I or N, or*

(ii) *$v \notin R$, some $u \in N_{T_v}(v)$ is labelled I and all vertices $w \in N_{T_v}(v) - u$ are labelled N.*

Proof: The proof is similar to the proof of Lemma 2.4 and is omitted. ∎

2.6. Lemma. *Let T be a tree rooted at $r \notin L$, and $v \in V$. Then $V(T_v)$ has a subset X such that*

(a) *$v \notin X$, and*

(b) *for every $u \in N_{T_v}(v)$, $S_u = S \cap V(T_u)$ is a u-set for T_u with $u \notin S_u$ (i.e., v can be labelled F) if and only if $v \notin R$ and every $u \in N_{T_v}(v)$ is labelled N.*

Proof: The proof is similar to the proof of Lemma 2.4 and is omitted. ∎

A pseudo-code description of the algorithm is shown in Algorithm 2.1.

Algorithm 2.1. Pseudo-code implementation of the algorithm.

```
var
   T:          tree;                                {adjacency list of T}
   I, N, F:    array of boolean;                    {labels}
   C, L, R:    set of vertex;                       {cools, leaves, remotes}
   parent:     array of vertex;


procedure I_label (v: vertex);                      {try to label v with I}
begin                                               {using Lemma 2.4}
   I[v] :=  not (v in C)                            {v not cool is needed}
   If I[v] then begin
      Iv := {v};
      for all u adjacent to v, except parent[v] do
         if not (v in R) then begin                 {condition (i)}
            I[v] := I[v] and F[u];
            Iv := Iv ∪ Fu
         end else begin                             {condition (ii)}
            I[v] := I[v] and (F[u] or N[u]);
            if N[u] then
               I[v] := I[v] ∪ Nu
            else
               I[v] := I[v] ∪ Iu
         end
   end
end;


procedure F_label (v: vertex);                      {try to label v with F}
begin                                               {using Lemma 2.6}
   F[v] :=  not (v in R);
   if F[v] then
      for all u adjacent to v, except parent[v] do begin
         F[v] := F[v] and N[u];
         Fv := Fv ∪ Nu;
      end
end;


procedure N_label (v: vertex);                      {try to label v with N}
var                                                 {using Lemma 2.5}
   I_count:    integer;                             {# of child'n lablled I}
   I_not_N:    integer;                             {# labelled I and not N}
begin
   N[v] := not (v in L);                            {v not a leaf is needed}
   if N[v] then
      if v in R then
{
      When v is remote, condition (i) is used.
}
         for all u adjacent to v, except parent[v] do begin
            N[v] := N[v] and (I[u] or N[u]);
            if N[v] then
               if N[u] then
                  Nv := Nv ∪ Nu
               else
                  Nv := Nv ∪ Iu;
         end
```

```
      else begin
{
         When v is not remote, condition (ii) is used. First check that
         the label N can be given.  This requires at least one child
         labelled I, and there can be at most one child labelled I and
         not labelled N (this is forced to be the I used).
}
         I_not_N := 0;
         I_count := 0;
         for all u adjacent to v, except parent[v] do
            if I[u] then begin
               I_count := I_count + 1;
               if not N[u] then
                  I_not_N := I_not_N + 1
            end;
         N[v] := N[v] and (I_count > 0) and (I_not_N ≤ 1);
         if N[v] then begin
{
            If v can be labelled N, build the set Nv
}
            for all u adjacent to v, except parent[v] do
               if I[u] then
                  if (not N[u]) or (I_count > 0) then begin
                     Nv := Nv ∪ Iu;
                     I_count := 0
                  end
               end else
                  Nv := Nv ∪ Nu
         end
      end
end;


procedure label (v: vertex);                          {label v with I, N, F}
var                                                   {if possible}
   labelled: boolean;
begin
   labelled := TRUE;
   for all u adjacent to v, except parent[v] do
      if labelled then begin                          {if possible}
         parent[u] := v;
         label(u);
         labelled := labelled  and  (I[u] or N[u] or F[u])
      end;
   if labelled then begin                             {if all children}
      I_label(v);                                     {were labelled, then}
      F_label(v);                                     {try to label v}
      N_label(v)
   end else begin
      I[v] := FALSE;
      F[v] := FALSE;
      N[v] := FALSE
   end
end;
```

```
{ M A I N   P R O G R A M }
begin
   compute L, R, C;
   choose r not in L;
   label(r);
   if I[r] or N[r] then
      actions if T has a universal MDF
   else
      actions if T has no universal MDF
end.
```

Before we can justify our claim of linear time, we must discuss the implementation of the sets I_v, N_v and F_v ($v \in V$). The only operation to be performed is the union of two of these sets. Further, once the parent of vertex v has been processed, the sets corresponding to v are never used again. Furthermore, if u and w are siblings, $X_u \cap Y_v = \emptyset$, where $X, Y \in \{F, I, N\}$. It therefore makes sense to implement each set as a linked list with a pointer to the first and last element. The assignment $A := A \cup B$ can then be carried out in constant time by

B.last^.next := A.first;
A.first := B.first;

that is, by transferring the contents of the linked list B into the linked list A.

We now show that the algorithm is linear. Procedure label is called once for each vertex v and it, in turn, calls each of procedures I_label, F_label, and N_label once. Each neighbour of v is examined once in procedure I_label, and once in procedure F_label. Procedure N_label examines a vertex once or twice depending on whether v is or is not remote. Each edge incident with v is examined at most four times, thus each edge of T is examined at most eight times. If T is stored as an adjacency list, the neighbours of v can be examined in time $O(d(v))$, so the total amount of time required by procedure label is

$$O(8 \cdot |E|) = O(|V|).$$

The sets L, R, and C are also required. The leaves of T can be found by examining the neighbourhood of each vertex, and the degree sequence of T can be computed at the same time. This causes each edge to be examined twice. Once L is known, R can easily be computed in $O(|V|)$ steps. When both the degree sequence of T and the set R are known, the set C can be computed via Lemma 2.1. (One way to do this is as follows: first count the number of non-remote neighbours of each vertex (in conjunction with the degree sequence of T, this makes conditions (i) and (ii) of Lemma 2.1 easy to check). Then, scan the vertices of T again, adding to C any vertex for which both conditions of Lemma 2.1 hold.) At worst, each edge must be examined four times. Hence the sets L, R, and C can be computed in time

$$O(6 \cdot |E|) = O(|V|).$$

The above argument shows that a universal MDS of T can be computed in linear time, if it exists.

We note that the pseudo-code in Algorithm 2.1 does not describe the most efficient implementation of the algorithm. One improvement that could be made is to terminate the "for" loops in procedures I_label, F_label, and N_label once it is clear that the label cannot be assigned to the vertex in question. This does not effect the worst case time complexity of the algorithm.

3. Acknowledgements.

The first author is grateful for the support of the Natural Sciences and Endneering Research Council of Canada, under grant A7544. The third author acknowledges the support of the South African Federation for Research Development. The second author thanks Denis Hanson for his help and encouragement.

References

1. Bange, D.W., A.E. Barkauskas, and P.J. Slater, *Efficient dominating sets in graphs*, in "Applications of Discrete Mathematics", R.D. Ringeisen and F.S. Roberts (eds.), SIAM, Philadelphia, 1988.
2. Beyer, T., A. Proskorowski, S.T. Hedetniemi, and S. Mitchell, *Independent domination in trees*, Proc. 8th Southeast Conference on Combinatorics, Graph Theory and Computing, Utilitas Mathematica (1977), 321–328.
3. Cockayne, E.J., S. Goodman, and S.T. Hedetniemi, *A linear algorithm for the domination number of a tree*, Inf. Proc. Lett. 4 (1975), 41–44.
4. Cockayne, E.J, G. Fricke, S.T. Hedetniemi and C.M. Mynhardt, *Properties of minimal dominating functions of graphs.* (submitted).
5. Cockayne, E.J., G. MacGillivray and C.M. Mynhardt, *Convexity of minimal dominating functions of trees.* (submitted).
6. Cockayne, E.J., G. MacGillivray and C.M. Mynhardt, *Convexity of minimal dominating functions of trees II.* (submitted).
7. Farber, M., *Domination, independent domination and duality in strongly chordal graphs*, Discrete Applied Math. 7 (1984), 115–130.
8. Yen, C.C., and R.C.T. Lee, *The weighted perfect domination problem*, Inf. Proc. Lett. 35 (1990), 295–299.