Some domination problems on trees and on general graphs

E.M. Bakker, J. van Leeuwen

RUU-CS-91-22
June 1991

Utrecht University

Department of Computer Science

Padualaan 14, P.O. Box 80.089,
3508 TB Utrecht, The Netherlands,
Tel. : ... + 31 - 30 - 531454

Some domination problems on trees and on general graphs

E.M. Bakker, J. van Leeuwen

Technical Report RUU-CS-91-22
June 1991

Department of Computer Science
Utrecht University
P.O.Box 80.089
3508 TB Utrecht
The Netherlands

ISSN: 0924–3275

Some Domination Problems on Trees and on General Graphs *

Erwin M. Bakker, Jan van Leeuwen
Department of Computer Science, Utrecht University
P.O.Box 80.089, 3508 TB Utrecht, the Netherlands

Abstract

In this paper we study several variants of domination problems. Linear-time algorithms are given for path-domination, independent efficient domination and efficient total domination on trees. On the other hand, it is proven that these problems are NP-complete for general graphs. All domination problems studied in this paper derive from considerations about distributed datastructuring and resource allocation in communication networks.

1 Introduction

A major problem in computer networks is the problem of determining optimal locations of resources. The optimality of a location of a resource may depend on many different objectives. Often this is formulated as a *domination problem* of some kind. In its typical form, a domination problem asks for a set V' of nodes of size k or less in the network, such that every node is within a distance of 1 from some node of V'. (Here k is some well-chosen integer.) Different extra restrictions on V' then leads to different domination problems. Through the years a wide variety of domination problems has been distinguished. (See [C78], [HHL84] for a survey.) In general there is little hope for solving these problems efficiently for arbitrary networks, as most of them are known to be NP-complete.
In this paper we study three particular domination problems: path-domination, independent efficient domination and efficient total domination. The path-domination problem is roughly the problem of finding a minimal number of paths such that every node that does not lie on a path is adjacent to at least one path. The independent efficient domination problem is the problem of finding a minimal cardinality dominating set V' such that no two nodes in V' are adjacent and every node not in V'

*This work was partially supported by the ESPRIT Basic Research Actions of the EC under contract no. 3075 (project ALCOM).

is dominated by exactly one node in V'. The efficient total domination problem is the problem of finding a minimal cardinality dominating set V' such that every node in the network, i.e., also every node in V', is dominated by exactly one node in V'. The problems have a rather direct motivation from considerations in distributed datastructuring and resource allocation. We will prove that all these problems are NP-complete for general graphs. On the other hand, for certain special classes of graphs they are polynomial-time solvable. With techniques from [B87] one can devise polynomial-time algorithms based on dynamic programming that solve these problems on graphs with bounded tree-width. (For the definition of tree-width and a survey on the classes of graphs with bounded tree-width the reader is referred to [B86].) In particular, for trees these techniques yield linear-time algorithms. However, these techniques do not yield the most efficient algorithms. Moreover the generated algorithms do not give much extra insight in the structure of the problems. Therefore, it is useful to look for linear-time algorithms that are more efficient and direct (and hopefully give some extra insight in the problems as a by-product). The paper is organized as follows. In Section 2 path-domination is studied. Independent efficient domination, and efficient total domination are studied in Section 3, and 4, respectively. Finally, in Section 5 some conclusions are given. Throughout the paper it is assumed that the reader is familiar with the basic concepts from graph theory and graph algorithms (cf. [H69], [G85]) and with the theory of NP-completeness (cf. [GJ79]).

2 Path-Domination

Let $G = (V, E)$ be a network, i.e., a graph. A simple path P in G is a sequence $[v_1, ..., v_t]$ of distinct nodes $v_i \in V$ such that $(v_i, v_{i+1}) \in E$ for every $i \in \{1, ..., (t-1)\}$. v_1 and v_t are called the *end-nodes* of P. We say that $v \in V$ is adjacent to P if v does not lie on P and there exists a node $w \in P$ such that $(v, w) \in E$. Furthermore, two paths P_1 and P_2 in G are *node-disjoint* iff there does not exist a $v \in V$ such that $v \in P_1$ and $v \in P_2$.

Notation Let $G = (V, E)$ be a graph and $\mathcal{P}$ a set of paths in G. With $\mathcal{P}(V)$ we denote the set of nodes that lie on at least one path that is element of $\mathcal{P}$.

In [HHL84] the notion of *path-domination* is introduced. This notion is defined as follows.

Definition 2.1 *Let $G = (V, E)$ be a graph and $\mathcal{P}$ a set of node-disjoint simple paths in G. $\mathcal{P}$ is a* path-dominating set *of G iff for every $v \in V - \mathcal{P}(V)$ there exists a $P \in \mathcal{P}$ such that v is adjacent to P.*

Definition 2.2 *The* path-domination number $\gamma_p(G)$ *of a graph* G *is equal to the cardinality of the smallest path-dominating set of* G. *(Note that the cardinality of a path-domination set is equal to the number of paths in that set.)*

In this section we will first argue that determining path-domination numbers is computationally hard in general, in the sense that it is NP-complete for general graphs. The larger part of this section is devoted to a proof that for trees the path-domination number can be determined by a linear-time algorithm. The following simple lemma can be proven.

Lemma 2.3 *Let* $G = (V, E)$ *be a connected graph with* $|V| \geq 3$. *There exists a minimum path-dominating set* $\mathcal{P}$ *of* G *such that no node of degree 1 is element of* $\mathcal{P}(V)$.

Proof. Let $\mathcal{P}$ be a minimum path-dominating set of G. We will show that if there exists a node v of degree 1 that is element of $\mathcal{P}(V)$, then an alternative minimum path-dominating set $\mathcal{P}'$ can be constructed such that $v \notin \mathcal{P}'(V)$ without introducing any new nodes of degree 1 in $\mathcal{P}'(V)$.
Assume there exists a node $v \in \mathcal{P}(V)$ of degree 1. Let $P \in \mathcal{P}$ be such that $v \in P$ and let w be the neighbor of v in G. As $|V| \geq 3$ it follows that the degree of w is greater than 1. If $w \in P$, we can simply delete v from P. If $w \notin P$, then $P = [v]$ and $w \notin \mathcal{P}(V)$, for otherwise we can delete P from $\mathcal{P}$ and still have a path-dominating set, contradicting the minimality of $\mathcal{P}$. By replacing the path $P = [v]$ by the path $[w]$ the number of nodes of degree 1 that are element of $\mathcal{P}(V)$ decreases by 1. In both cases we obtain a minimum path-dominating set $\mathcal{P}'$ such that $v \notin \mathcal{P}'(V)$ and the number of degree 1 nodes in it is reduced. By repeating this procedure one obtains a minimum path-dominating set as described in the lemma. □

In the next theorem it is shown that the problem of finding the path-dominating number of an arbitrary graph is NP-hard even when restricted to planar graphs with no faces with fewer than 5 edges. The following problem is considered.

Problem: PATH-DOMINATING SET
Instance: A graph $G = (V, E)$, and a positive integer $k \leq |V|$.
Question: Is there a path-dominating set of size k or less for G?

Theorem 2.4 *PATH-DOMINATING SET is* NP*-complete even when restricted to planar graphs with no faces with fewer than 5 edges.*

Proof. PATH-DOMINATING SET is clearly in NP, since a nondeterministic algorithm can always guess a set of simple paths in the given graph and check whether this set is a path-dominating set with the desired cardinality constraint in polynomial time.

To prove the problem NP-complete we use a transformation from HAMILTONIAN PATH restricted to planar graphs with no faces with fewer than 5 edges, which is known to be NP-complete (see [GJT76]). Assume an instance of this problem is given. Let $G = (V, E)$ be the given planar graph. We transform the graph G to a new graph G' by connecting to every node in G a new node of degree 1. The transformation can only increase the number of edges of a face. Hence G' is also a planar graph with no face with fewer than 5 edges. This clearly is a polynomial-time computable transformation. Let W' denote the set of these new nodes. We claim that G has a Hamiltonian path iff $\gamma_p(G') = 1$. Assume $\gamma_p(G') = 1$. Then by Lemma 2.3 there exists a path-dominating set $\mathcal{P}$ of cardinality 1 such that $W' \cap \mathcal{P}(V) = \phi$. If there exists a $v \in V$ such that $v \notin \mathcal{P}(V)$, then there exists a $w' \in W'$ that is not dominated by the path in $\mathcal{P}$. We conclude that $\mathcal{P}(V) = V$, and hence the path $P \in \mathcal{P}$ is a Hamiltonian path in G. On the other hand, if G has a Hamiltonian path, then it is clear that $\gamma_p(G') = 1$. This proves the theorem. □

Note that if $\mathcal{C}$ is a class of graphs such that HAMILTONIAN PATH restricted to $\mathcal{C}$ is NP-complete and such that $\mathcal{C}$ is closed under the construction used in the previous theorem, then PATH-DOMINATING SET restricted to $\mathcal{C}$ is NP-complete.

2.1 Determining the Path-Domination Number for Trees

With techniques developed in [B87] we can show that there exist polynomial-time algorithms for determining γ_p when we restrict the problem to graphs with bounded tree-width. However, the algorithms obtained are not the most efficient algorithms and do not give extra insight in the combinatorics of the problem. Therefore it is useful to look for explicit algorithms to determine γ_p for this class of graphs. In the following we will restrict the problem to *trees*. An algorithm (*Algorithm PDS*) is presented that constructs a minimum path-dominating set for every given tree.
Let T be a tree. Algorithm PDS starts at an arbitrary node v of T (effectively considering T rooted at v). It recursively determines a "special" minimum path-dominating set for all subtrees rooted at the sons of this node. It will always try to construct a minimum path-dominating set such that a path of this set ends in the root of the subtree. If there are two subtrees in which a path ends in the root, then these paths are joined together with v to obtain one new path, thereby decreasing the number of paths with 1. If there is only one subtree in which a path ends in the root, then v is added to this path. Thus no new path is introduced. Finally, if there is not such a subtree and there is no path in one of the subtrees that dominates v, then a new path starting in v must be introduced. Observe that this path is not needed to dominate one of v's sons. In Algorithm PDS v then obtains the status *free-start-node*. It is clear that if a node with status *free-start-node* has a father that is element of a path, then it is no longer necessary for this node to be element of a path. Therefore, if v should become element of a path, then all its sons with status *free-start-node* obtain a new status *dominated* (by the path of which v now is an

element).
Let $T = (V, E)$ be a tree. Every node $v \in V$ has a variable *status*(v) which can take a value out of the following four possibilities:

- **free-start-node**, if a path $\in \mathcal{P}$ starts at v that only dominates sons that are already dominated by other paths.
- **end-node**, if v is an end-node of a path $\in \mathcal{P}$ and does not have the status *free-start-node.*
- **path-node**, if v is element of a path $\in \mathcal{P}$ and does not have the status *end-node* or *free-start-node.*
- **dominated**, if v is adjacent to a path $\in \mathcal{P}$ and $v \notin \mathcal{P}(V)$.

$\mathcal{P}$ is the set of node-disjoint simple paths which the algorithm builds in the subtree rooted at v.

Algorithm PDS:

```
procedure Path-Dominating Set (v :node)
begin
    forall sons w of v do
        Path-Dominating Set (w)
    enddo ;
    if v has no sons or all sons of v have status dominated
    then
        begin a new path at v ;
        {the path is not needed to dominate one of the sons of v}
        status(v) := free-start-node
    elsif there exist two sons of v that have status end-node
    then
        make one path by joining the paths of which the two sons are end-node with v ;
        status(v) := path-node
    elsif exactly one son of v has status end-node
        {and all the other sons have status free-start-node, path-node or dominated}
    then
        add v to the path of which the son is end-node ;
        {v becomes the new end-node of this path}
        status(v) := end-node
    elsif one or more sons have status free-start-node
        {and all the other sons have status path-node or dominated}
    then
        begin a new path at v ;
        {this path dominates all sons with status free-start-node}
```

```
        status(v) := end-node
    elsif one or more sons have status path-node
        {and all the other sons have status dominated}
    then
        status(v) := dominated ;
    endif ;
    if v belongs to a path, i.e., does not have status dominated
    then
        all sons of v with status free-start-node get the new status
        dominated and the paths they "started" are discarded
    endif
end {procedure Path-Dominating Set} ;

begin {Main}
    input a tree T ;
    let v be a node of T ;
    orient T as a tree rooted at v ;
    Path-Dominating Set(v)
    {the nodes with status free-start-node, path-node and end-node are
    the elements of the paths that form the path-dominating set P of T}
end {Main}
```

The figure below shows an example of a minimum path-dominating set calculated by Algorithm PDS. The given tree has a minimum path-dominating set consisting of four paths. In the figure the different statuses that a node can adopt in Algorithm PDS are shown.

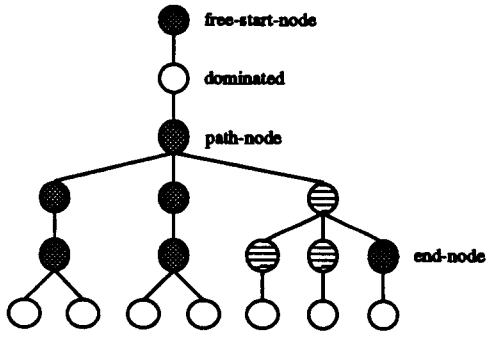

Figure 2.1

Note that we have only indicated how to construct the set of paths $\mathcal{P}$. For the sake of clarity of presentation we did not explicitly maintain the set of paths. But this can be done in a straightforward manner. In the following we will prove that $\mathcal{P}$ is a minimum path-dominating set of the tree T, i.e., $\gamma_p(T) = |\mathcal{P}|$.

2.2 Correctness and Complexity of Algorithm PDS

Let $T = (V, E)$ be a tree and $T' = (V', E')$ a subtree of T. Assume $\mathcal{P}$ is a path-dominating set of T. Then by $\mathcal{P}_{T'}$ ($\mathcal{P}$ restricted to T') we denote the set of paths in T' remaining after deleting all nodes $\in (V - V')$ from the paths in $\mathcal{P}$.

Lemma 2.5 *Let $T = (V, E)$ be a tree and $v, w \in V$ such that $(v, w) \in E$. Let T_v and T_w be the two components of $T - (v, w)$ rooted at v and w, respectively. Then $\gamma_p(T_v) + \gamma_p(T_w) - 1 \leq \gamma_p(T) \leq \gamma_p(T_v) + \gamma_p(T_w)$.*

Proof. Let $\mathcal{P}$, $\mathcal{P}_v$ and $\mathcal{P}_w$ be minimum path-dominating sets of T, T_v and T_w, respectively. It is clear that $\mathcal{P}_v \cup \mathcal{P}_w$ forms a path-dominating set of T. Hence $\gamma_p(T) \leq \gamma_p(T_v) + \gamma_p(T_w)$.
Consider $\mathcal{P}_{T_v}$ and $\mathcal{P}_{T_w}$. If (v, w) belongs to a path in $\mathcal{P}$ or if $v, w \notin \mathcal{P}(V)$, then these sets are clearly path-dominating sets of T_v and T_w, respectively. Thus $|\mathcal{P}_{T_v}| + |\mathcal{P}_{T_w}| \leq \gamma_p(T) + 1$ must hold. In the other case we may assume, w.l.o.g., that $w \notin \mathcal{P}(V)$ and $v \in \mathcal{P}(V)$. Clearly $\mathcal{P}_{T_v}$ is a path-dominating set of T_v and $\mathcal{P}_{T_w} \cup \{[w]\}$ is a path-dominating set of T_w. It follows that $|\mathcal{P}_v| + |\mathcal{P}_w| \leq |\mathcal{P}_{T_v}| + |\mathcal{P}_{T_w} \cup \{[w]\}| = \gamma_p(T) + 1$. Thus the lemma holds. □

Lemma 2.6 *Let $T = (V, E)$ be a tree and $v, w \in V$ such that $(v, w) \in E$. Let T_v and T_w be the two components of $T - (v, w)$ rooted at v and w, respectively. If every minimum path-dominating set $\mathcal{P}_w$ of T_w is such that w is dominated by one of its sons and $w \notin \mathcal{P}_w(V)$, then $\gamma_p(T) = \gamma_p(T_v) + \gamma_p(T_w)$.*

Proof. From the previous lemma we know that $\gamma_p(T_v) + \gamma_p(T_w) - 1 \leq \gamma_p(T) \leq \gamma_p(T_v) + \gamma_p(T_w)$. Suppose that $\gamma_p(T) = \gamma_p(T_v) + \gamma_p(T_w) - 1$. Let $\mathcal{P}$ be a minimum path-dominating set of T. It is clear that $|\mathcal{P}_{T_w}| \leq \gamma_p(T_w)$. If $w \in \mathcal{P}(V)$, then $\mathcal{P}_{T_w}$ is a minimum path-dominating set of T_w such that $w \in \mathcal{P}_w(V)$, a contradiction. If $w \notin \mathcal{P}(V)$, then $\mathcal{P}_{T_v}$ is a minimum path-dominating set of T_v. It follows that $|\mathcal{P}_{T_w}| = \gamma_p(T_w) - 1$. Hence $\mathcal{P}_{T_w} \cup \{[w]\}$ is a minimum path-dominating set, which again is a contradiction. □

Assume a tree is given with a certain path-dominating set. If we consider the tree as a rooted tree, then every node can be given a status according to the rules used in Algorithm PDS, i.e., the status of a node depends only on the role of its sons with respect to the given path-dominating set. In the following this status is intended when the notion "status of a node" is used.

Theorem 2.7 *Let $T = (V, E)$ be a tree and $v \in V$.* Path-Dominating Set*(v) constructs a minimum path-dominating set such that*

1. *if the status of v is equal to* dominated, *then there exists no alternative minimum path-dominating set such that v has another status.*

2. *if the status of v is equal to* path-node, *then there exists no alternative minimum path-dominating set such that the status of v is equal to* free-start-node *or* end-node.

3. *if the status of v is equal to* end-node, *then there exists no alternative minimum path-dominating set such that v has status* free-start-node.

Proof. By induction on the number of nodes of T. It can easily be verified that the theorem holds for trees with ≤ 3 nodes. Assume that the theorem holds for all trees with less than k nodes. Let $T = (V, E)$ be a tree with k nodes, and $v \in V$. Consider T as a rooted tree with root v. Let $w_1, ..., w_d \in V$ be the sons of v. With T_{w_i} we denote the subtree of T rooted at w_i. Consider Path-Dominating Set(v) (PDS(v) for short). First, we will prove that the obtained set of paths is a path-dominating set. Secondly, we will prove that the cardinality of this set is equal to $\gamma_p(T)$ and that the claims made concerning the status of v as stated in the theorem hold.
By induction PDS(w_i) constructs a path-dominating set of T_{w_i}. If v obtains the status *dominated*, then there are no further changes (i.e., all sons of v keep their original status) and v is clearly adjacent to a path. If v obtains the status *free-start-node*, then there are also no further changes and v is element of a path. If v gets the status *path-node* or *end-node*, then v itself is element of a path and only sons that had the status *free-start-node* obtain a new status *dominated*. This proves that in all cases the newly constructed set of paths $\mathcal{P}$ is a path-dominating set of T.
Now we will prove that the cardinality of $\mathcal{P}$ is equal to $\gamma_p(T)$ and that the claims made concerning the status of v as stated in the theorem hold. Assume that v has status *free-start-node*, in which case all sons have status *dominated*. By induction the constructed path-dominating sets of T_{w_i} are minimum. Then by Lemma 2.6 it follows that $\mathcal{P}$ must be minimum. Assume that v has a status other than *free-start-node*. Then suppose, to the contrary, that there exists a path-dominating set $\mathcal{P}'$ of T such that $|\mathcal{P}'| \leq |\mathcal{P}|$ and status(v) conflicts with the claims as stated in the theorem, or such that $|\mathcal{P}'| < |\mathcal{P}|$. We will show that neither of both holds. Let $\gamma_1 = \sum_{i=1}^{d} \gamma_p(T_{w_i})$. As v has a status other than *free-start-node* it is clear that $|\mathcal{P}| \leq \gamma_1$. If $\mathcal{P}'$ is a path-dominating set such that $|\mathcal{P}'| \leq |\mathcal{P}| \leq \gamma_1$ and v has status *free-start-node* in $\mathcal{P}'$, then by the previous lemma this would contradict the minimality of γ_1. Furthermore, if v has status *dominated* in $\mathcal{P}'$, then $\mathcal{P}'$ is a path-dominating set of the subtrees rooted by v's sons. If $|\mathcal{P}'| < \gamma_1$, this is contradicting the minimality of γ_1. Thus in the following we only have to consider a path-dominating set $\mathcal{P}'$ in which v has a status equal to *end-node* or *path-node*.
Assume that v has status *dominated* in $\mathcal{P}$. Hence $|\mathcal{P}| = \gamma_1$ and every son has status *path-node* or *dominated*. Consider $\mathcal{P}'$. By induction, if a subtree has a path-dominating set such that its root has status *free-start-node* or *end-node*, then this set cannot be minimum. It follows that if v has status *end-node* or *path-node* in $\mathcal{P}'$, then $|\mathcal{P}'| > \gamma_1$, i.e., there cannot exist a minimum path dominating set in which v has another status. Hence claim 1) holds.

Assume that v has status *path-node* in $\mathcal{P}$. Hence there are at least two sons with status *end-node*. Assume that there are k sons with status *free-start-node*. Then $|\mathcal{P}| = \gamma_1 - k - 1$. Consider $\mathcal{P}'$. By induction, every subtree rooted by a node with status *end-node, path-node* or *dominated* in $\mathcal{P}$ cannot have a minimum path-dominating set in which the status of the root is equal to *free-start-node*. Hence it is not possible to introduce new *free-start-nodes* without extra costs. Therefore it is now easy to verify that $|\mathcal{P}'| \geq \gamma_1 - k - 1$, if v has status *path-node* in $\mathcal{P}'$. If v has status *end-node* in $\mathcal{P}'$, then the number of paths is equal to the number of paths in the subtrees minus the number of *free-start-nodes*. By induction it is not possible to introduce new *free-start-nodes* without extra costs. Hence $|\mathcal{P}'| > \gamma_1 - k - 1$. This proves that there cannot exist an alternative minimum path dominating set in which v has status *end-node*. Hence claim 2) holds.
Similarly, it can be proven that if v has status *end-node* in $\mathcal{P}$, then $|\mathcal{P}'| \geq |\mathcal{P}|$ if v has status *path-node* or *end-node* in $\mathcal{P}'$. Hence claim 3) holds. □

The algorithm clearly requires only $O(|V|)$ steps.

3 Independent Efficient Domination

Definition 3.1 *Let $G = (V, E)$ be a graph, and D a subset of V. D is a* dominating set *of G iff for every $v \in V - D$ there exists a $w \in D$ such that $(v, w) \in E$.*

Dominating sets are well studied. An overview of complexity results for dominating set problems can be found in [J85]. An interesting extra restriction on dominating sets is *efficiency.*

Definition 3.2 *A dominating set D of a graph $G = (V, E)$ is* efficient *iff $|N(v) \cap D| = 1$ for every $v \in V - D$. (Where $N(v)$ is equal to the set of neighbours of v in G.)*

In [YL90] the problem of finding a minimum efficient dominating set is called PERFECT DOMINATION. In that paper it is shown that PERFECT DOMINATION is NP-complete for bipartite and chordal graphs. Furthermore, a linear-time algorithm is given for a weighted version of the problem for trees.
A trivial efficient dominating set of an arbitrary graph $G = (V, E)$ is always at hand, namely $D = V$. It is not true however that there always exists an *independent* efficient dominating set of G. Recall that an independent set of G is a subset of V such that no two nodes of this subset are adjacent. For example, the tree depicted in Figure 3.1 obviously has no independent efficient dominating set.

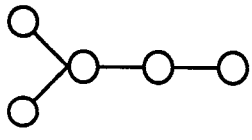

Figure 3.1

In the next theorem we will prove that the following problem is NP-complete.

Problem: INDEPENDENT EFFICIENT DOMINATING SET
Instance: A graph $G = (V, E)$.
Question: Does G have an independent efficient dominating set?

Theorem 3.3 *INDEPENDENT EFFICIENT DOMINATING SET is NP-complete.*

Proof. INDEPENDENT EFFICIENT DOMINATING SET (IEDS, for short) is clearly in NP, since a nondeterministic algorithm can always guess a subset of the nodes of the given graph and check whether this subset is an independent efficient dominating set in polynomial time. To prove the problem NP-complete we use a transformation from 3-SAT.

Problem: 3-SAT.
Instance: A set $X = \{x_1, ..., x_n\}$ of variables. A collection $C = \{c_1, ..., c_m\}$ of clauses over X such that each clause $c \in C$ has $|c| = 3$.
Question: Is there a satisfying truth assignment for C ?

Assume an instance of 3-SAT is given. We show that the instance of 3-SAT can be transformed to a graph $G = (V, E)$ such that: G has an independent efficient dominating set iff there is a satisfying truth assignment for the set of clauses of the given instance of 3-SAT. The transformation is easily seen to be polynomial-time computable.

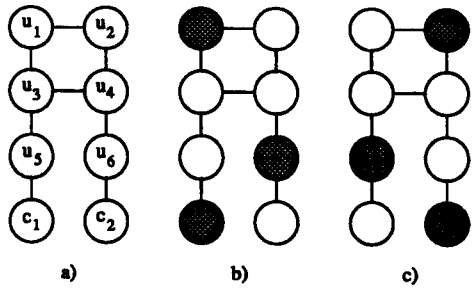

Figure 3.2

First, we discuss the transformation. Every variable x_i of the instance of 3-SAT corresponds to a subgraph as depicted in Figure 3.2a. These subgraphs are connected with edges incident to nodes c_1 and c_2 to the rest of the graph. Every clause C_j also corresponds to a subgraph, say G_C. (We will discuss this later.) c_1 is connected to G_C if $x_i \in C_j$, and c_2 is connected to G_C if $\overline{x_i} \in C_j$. It is clear that if G has an independent efficient dominating set D, then node u_3 cannot be an element of D because, if not, then u_2 cannot be element of D and cannot be dominated by one

of its neighbors. Similarly, u_4 cannot be an element of D. It follows that either u_1 or u_2 is element of D. If $u_1 \in D$ this corresponds with x_i *true* and if $u_2 \in D$ this corresponds with x_i *false*. See Figure 3.2b and 3.2c, respectively. The shaded nodes in these pictures represent the nodes that must be element of D.

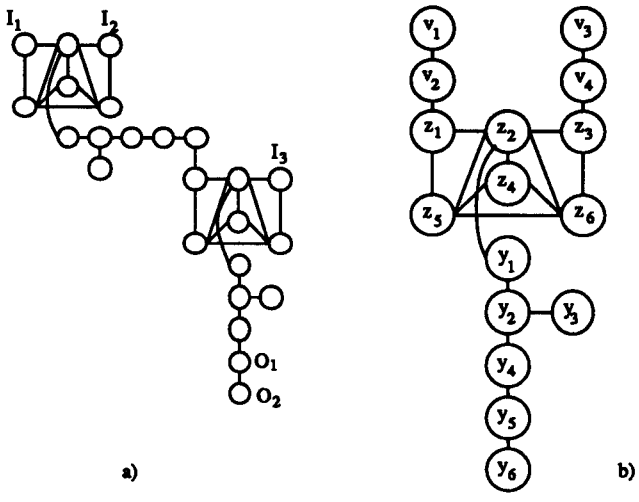

Figure 3.3

Every clause $C_j = (x_{i_1}, x_{i_2}, x_{i_3})$ corresponds to a subgraph G_C as given in Figure 3.3a. Nodes I_1, I_2 and I_3 are connected to the c_1 nodes of the subgraphs associated with x_{i_1}, x_{i_2} and x_{i_3}, respectively. Nodes I_1, I_2 or I_3 in G_C are connected to a c_2-node instead, if the negotiation of the corresponding variable is element of the clause. G_C consists of two "or-structures" (see Figure 3.3b). An "or-structure" will take two inputs (the v-nodes) and produce one output (the last two y-nodes). From the preceding argument it is clear that either v_1 or v_2 must be element of D as these nodes represent the nodes u_5 and c_1 or u_6 and c_2 in the subgraph associated with a variable that is element of the clause. The same holds for v_3 and v_4. If $v_1 \in D$, then the input value is equal to *false*. If $v_2 \in D$, then the input value is equal to *true*. In the following we will show that the "or-structure" indeed acts as an or-gate, i.e. on "logical" input values y_1, y_2 it produces an output value $y_1 \vee y_2$. There are four cases to consider. 1) If $v_2, v_4 \in D$, i.e., both input values are equal to *true*, then of all z-nodes only z_4 can be element of D. It folows that z_4 necessarily must be element of D as none of the y-nodes is able to dominate z_5 and z_6. Now it is easy to verify that y_6 must be element of D also. Hence the output value is equal to *true*. 2) If $v_1, v_3 \in D$, i.e., both input values are equal to *false*, then z_1 and z_2 must be dominated by one of their neighbors. Either z_2 or z_5 or z_6 can be element of D. It follows that z_2 must be element of D. Now it is easy to verify that y_5 must be element of D. Hence the output value is equal to *false*. 3) If $v_1, v_4 \in D$, i.e., the input values are equal to *false*, *true*, respectively, then of all z-nodes only z_4 or z_5 can be element of D. It folows that z_5 necessarily must be element of D as z_1 must be dominated by one of the z-nodes. Now it is easy to verify that y_6 must be

element of D also. Hence the output value is equal to *true*. 4) If $v_2, v_3 \in D$, the proof is similar to 3).
G_C consists of two "or-structures". From the preceding argument it is clear that either O_1 or O_2 must be element of D, representing either *false* or *true*, respectively. In G the O_1 node of the subgraph associated with C_j is connected to the O_1 node of the subgraph associated with C_{j+1}, where $j \in \{1, ..., (m-1)\}$. In each subgraph either O_1 or O_2 must be element of D, if an independent efficient dominating set of G exists. As the O_1-nodes are connected to each other, it follows that all O_2-nodes must be element of D. Thus all outputs of the "or-structures" must be equal to *true*. Hence G has an independent efficient dominating set iff there is a satisfying truth assignment for the given set of clauses C. □

3.1 Determining an Efficient Independent Dominating Set for Trees

Again with the techniques developed in [B87] it can be shown that the IEDS problem is solvable in polynomial time for graphs with bounded tree-width. In this section we will present and prove an explicit linear time algorithm that solves the problem for trees. This solves one of the open problems stated in [HHL84].
Let $T = (V, E)$ be a tree and $D \subseteq V$ an independent efficient dominating set of T. Let $v \in V$. Orient T as a tree rooted at v. Every node is either element of D or dominated by exactly one neighbor $\in D$. If a node is dominated, then it is either dominated by its father or it is dominated by exactly one son. In the next algorithm each of the previous three cases will correspond to status info kept at each node of the tree. The algorithm starts at a node v and recursively computes all possible statuses that the sons of v can adopt in an independent efficient dominating set of T, if such a set exists. With this information either all possible statuses that v can adopt are determined or it is noticed that the sons can only adopt incompatible statuses, for instance if two sons are bound to be element of the dominating set. In the later case it is concluded that T does not have an independent efficient dominating set.
Let $T = (V, E)$ be a tree. In the algorithm below every node $v \in V$ has a variable *status*(v) which can take a value out of the following five possibilities:

- **dominator**, if v is an element of the independent efficient dominating set.
- **to-be-dominated**, if v must be dominated by its father.
- **dominated**, if v is dominated by exactly one son.
- a pair (**status1, status2**), if it is possible to give either **status1** or **status2** to v without getting conflicts with the statuses of its sons. (In the following we say that "v can adopt **status1** (**status2**)".)
- **conflict**, if it is not possible to construct an independent efficient dominating set for this tree.

Algorithm IEDS

procedure Independent Efficient Dominating Set(v :node)
forall sons w of v **do**
Independent Efficient Dominating Set(w)
enddo ;
if {condition 0}
v has no sons
then {v is a leaf, i.e., either v is dominated by its father or
it is element of the independent efficient dominating set}
status(v) := (**to-be-dominated , dominator**)
elsif there exists a son of v with status **conflict**
then status(v) := **conflict**
elsif {condition 1}
(one or more sons have status **to-be-dominated** and
all the other sons can adopt the status **to-be-dominated**) **or**
(more than one son has status (**to-be-dominated, dominator**) and
all the other sons can adopt the status **to-be-dominated**)
then status(v) := **dominator**
elsif {condition 2}
(exactly one son has status **dominator** and
all the other sons can adopt the status **dominated**) **or**
(one or more sons have status **dominated** and
one son has status (**to-be-dominated, dominator**) and
all the other sons can adopt the status **dominated**) **or**
(one son has status (**to-be-dominated, dominator**) and
one or more sons have status (**dominated, dominator**) and
all the other sons can adopt the status **dominated**)
then status(v) := **dominated**
elsif {condition 3}
one or more sons have status **dominated** and
all the other sons have status (**to-be-dominated, dominated**)
then status(v) := **to-be-dominated**
elsif {condition 4}
all sons have status (**dominated, to-be-dominated**)
then status(v) := (**to-be-dominated, dominator**)
elsif {condition 5}
exactly one son has status (**to-be-dominated, dominator**) and
all the other sons have status (**to-be-dominated, dominated**)
then status(v) := (**dominated, dominator**)
elsif {condition 6}
one or more sons have status (**dominated, dominator**) and
all the other sons can adopt the status **dominated**

then status(v) := (**to-be-dominated, dominated**)
else status(v) := **conflict**
endif
end {procedure Independent Efficient Dominating Set} ;

begin {Main}
input a tree T ;
let v be a node of T ;
orient T as a tree rooted at v ;
Independent Efficient Dominating Set(v) ;
if status(v) is equal to **conflict** or **to-be-dominated**
then there does not exist an independent efficient dominating set of T
else T has an independent efficient dominating set
endif
end {Main}

Note that all possible status pairs occur in Algorithm IEDS. It is easy to prove (by induction) that the statuses which occur in Algorithm IEDS are the only possible statuses.
If the algorithm concludes that an independent efficient dominating set of T exists, then it is possible that some nodes have a status pair. By traversing the tree T starting at v and choosing one status of each status pair, every node obtains a status that is either **dominated**, **to-be-dominated** or **dominator**. Note that such a choice may have implications on the status of a son that has a status pair. The set of nodes with status **dominator** forms an independent efficient dominating set. It is possible to obtain all possible different independent efficient dominating sets of T in this way. We will not study this further.

3.2 Correctness and Complexity of Algorithm IEDS

Theorem 3.4 *Let $T = (V, E)$ be a tree and $v \in V$.* Independent Efficient Dominating Set (v) *determines all possible statuses of v in an independent efficient dominating set of T. If status(v) is equal to* **conflict**, *then T has no efficient dominating set and if status(v) is equal to* **to-be-dominated**, *then $T \cup \{(v, w)\}$ has an independent set, where w is a new node that must be element of the dominating set.*

Proof. By induction on the number of nodes of T. It can easily be verified that the theorem holds for trees with ≤ 3 nodes. Assume the theorem holds for trees with $< k$ nodes. Let $T = (V, E)$ be a tree with k nodes, and $v \in V$. Consider T as a rooted tree with root v. Let $w_1, ..., w_d \in V$ be the sons of v. With T_{w_i} we denote the subtree of T rooted by w_i. Consider Independent Efficient Dominating Set(v) (IEDS(v) for short). By induction IEDS(w_i) determines all possible statuses

of w_i. Observe that if one son of v has status **to-be-dominated**, then necessarily all sons must have status **to-be-dominated** and v must be assigned the status **dominator**. If one of the sons cannot adopt the status **to-be-dominated**, then it is not possible to construct an independent efficient dominating set of T and v gets the status **conflict**. It is straightforward to verify that condition 1 (see procedure IEDS) describes all the cases in which the previous is fullfilled. Furthermore, if one son has status **dominator**, then all other sons must have status **dominated** and v must be assigned the status **dominated**. Condition 2 describes all the cases in which one son indeed must have status **dominator** while the other sons do not have a conflicting status. Finally, if all sons have status **dominated**, then v must be assigned the status **to-be-dominated**. Condition 3 describes this case. The three previous cases also indicate the conditions to be met for the following three cases. If a node v has no sons, i.e., it is a leaf, then the node is either an element of the dominating set or it is dominated by its father. Thus the status (**to-be-dominated, dominator**) must be assigned to the leaves of the tree. Furthermore, this status can only be assigned if all sons of v can adopt the status **dominated** *and* **to-be-dominated**. Hence condition 0 and condition 1 precisely cover these cases. A node will be assigned the status (**dominated, dominator**) if all sons can adopt the status **to-be-dominated** *and* there exists exactly one son that can also adopt the status **dominator** while all the other sons can also adopt the status **dominated**. This is reflected in condition 5. Finally, a node will be assigned the status (**to-be-dominated, dominated**) if all sons can adopt the status **dominated** while there also exists a son that can adopt the status **dominator**. It is easy to verify that this is formulated in condition 6. □

The algorithm clearly requires only $O(|V|)$ steps.

4 Efficient Total Domination

Definition 4.1 *Let $G = (V, E)$ be a graph, $D \subseteq V$. D is a* total dominating set *of G iff for every $v \in V$ there exists a $w \in D$ such that $(v, w) \in E$.*

Note that the nodes of a dominating set do not have to be dominated, whereas every node of a total dominating set has to be dominated by some other node in the set. Total domination was first studied in [CDH80]. In [LPHH84] a linear-time algorithm is presented that computes the minimum cardinality of a total dominating set for trees. The problem becomes NP-complete for bipartite and chordal graphs (see [B84], [CN84], [LPHH84]). In [HHL84] it is remarked that a notion of efficiency can also be introduced for total dominating sets.

Definition 4.2 *A total dominating set D of a graph G is* efficient *iff $|N(v) \cap D| = 1$ holds for every $v \in V$.*

Let $G = (V, E)$ be a graph with an efficient total dominating set $D \subseteq V$. Then, by definition, for every $v \in D$ there exists exactly one neighbor $w \in V$ that is also element of D. Furthermore, all neighbors ($\neq w$) of v and all neighbors ($\neq v$) of w must be element of $V - D$. Hence G can be partitioned in components as depicted in Figure 4.1 by deleting all edges between nodes in $V - D$.

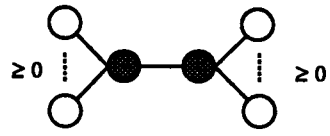

Figure 4.1

Observe that these components can easily be used to construct a graph for which an efficient total dominating set exists. We will not study this any further. However, these observations are very helpful in the rest of this section.
In the next theorem we will prove that the following problem is NP-complete.

Problem: EFFICIENT TOTAL DOMINATING SET
Instance: A graph $G = (V, E)$.
Question: Has G an efficient total dominating set?

Theorem 4.3 *EFFICIENT TOTAL DOMINATING SET is NP-complete.*

Proof. EFFICIENT TOTAL DOMINATING SET (ETDS, for short) is clearly in NP, since a nondeterministic algorithm can always guess a subset of the nodes of the given graph and check whether this subset is an efficient total dominating set in polynomial time. To prove the problem NP-complete we use a transformation from 3-SAT. Assume an instance of 3-SAT is given. Let $X = \{x_1, ..., x_n\}$ be the set of variables and $C = \{C_1, ..., C_m\}$ the collection of clauses over X. We show that the instance of 3-SAT can be transformed to a graph $G = (V, E)$ such that: G has an efficient total dominating set iff there is a satisfying truth assignment for the set of clauses of the given instance of 3-SAT. The transformation will be seen to be polynomial-time computable.

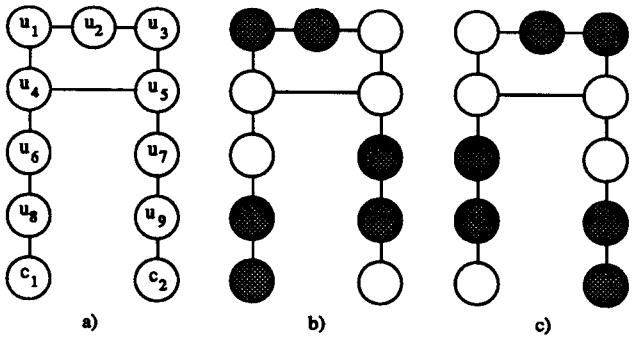

Figure 4.2

First, we discuss the transformation. Every variable x_i of the instance of 3-SAT corresponds to a subgraph as depicted in Figure 4.2a. These subgraphs are connected with edges incident to nodes c_1 and c_2 to the rest of the graph. Every clause C_j also corresponds to a subgraph, say G_C. (We will discuss this later.) c_1 is connected to G_C if $x_i \in G_C$, and c_2 is connected to G_C if $\overline{x_i} \in G_C$. Assume that D is an efficient total dominating set of G. If u_1 and u_4 are element of D then u_3 cannot be element of of D and cannot be dominated by one of its neighbors. Hence u_1 and u_4 cannot be element of D at the same time. Similarly, this must hold for u_3 and u_5. It follows that either $u_1, u_2 \in D$ or $u_2, u_3 \in D$. If $u_1, u_2 \in D$, this corresponds with x_i *true* and if $u_2, u_3 \in D$, this corresponds with x_i *false*. See Figure 4.2b and 4.2c, respectively.

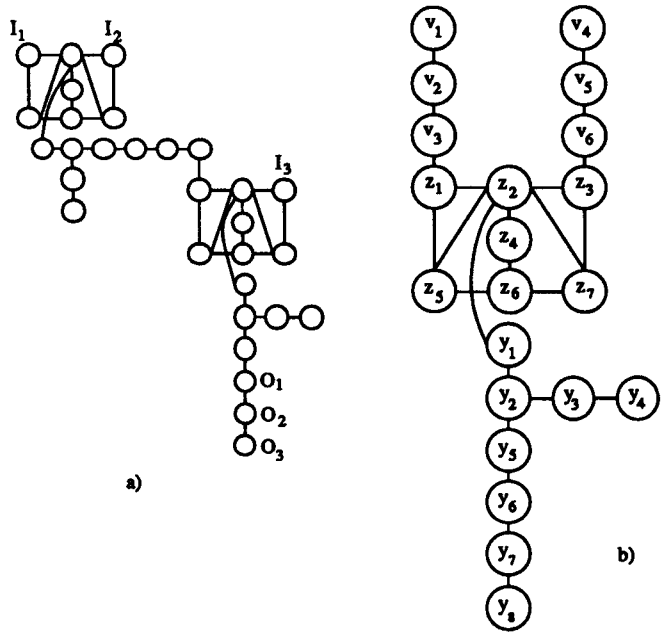

Figure 4.3

Every clause $C_j = (x_{i_1}, x_{i_2}, x_{i_3})$ corresponds to a subgraph G_C as given in Figure 4.3a. Nodes I_1, I_2 and I_3 are connected to the c_1 nodes of the subgraphs associated with x_{i_1}, x_{i_2} and x_{i_3}), respectively. Nodes I_1, I_2 or I_3 in G_C are connected to c_2 if the negotiation of the corresponding variable is element of the clause. G_C consists of two "or-structures" (see Figure 4.3b). An "or-structure" will take two inputs (the v-nodes) and produce one output (the last three y-nodes). From the preceding argument it is clear that either v_1 and v_2 or v_2 and v_3 must be element of D as these nodes represent the nodes u_6, u_8, c_1 or u_7, u_9, c_2 in the subgraph associated with a variable that is element of the clause. The same holds for v_4, v_5 and v_5, v_6. If $v_1, v_2 \in D$, then the input value is equal to *false*. If $v_2, v_3 \in D$, then the input value is equal to *true*. In the following we will show that the "or-structure" indeed acts as an or-gate, i.e., on "logical" input values y_1, y_2 it produces an output value $y_1 \vee y_2$. There are four cases to consider. 1) If $v_2, v_3, v_5, v_6 \in D$, i.e., both input values are

equal to *true*, then of all z-nodes only z_4 and z_6 can be element of D. It follows that z_4 and z_6 necessarily must be element of D as none of the y-nodes is able to dominate z_5 and z_7. Now it is easy to verify that y_7 and y_8 must be element of D also. Hence the output value is equal to *true*. 2) If $v_1, v_2, v_4, v_5 \in D$, i.e., both input values are equal to *false*, then z_1 and z_3 must be dominated by one of their neighbors z_2, z_5 or z_7. It is easy to verify that z_5 and z_7 cannot be elements of D. Thus z_2 must be element of D. As z_2 and z_6 must be dominated as well, it follows that z_4 must be element of D. Now it is easy to verify that y_6 and y_7 must be element of D. Hence the output value is equal to *false*. 3) If $v_1, v_2, v_5, v_6 \in D$, i.e., the input values are equal to *false*, *true*, respectively, then of all z-nodes only z_4, z_5 or z_6 can be element of D. It folows that z_6 necessarily must be element of D as z_4 and z_5 must be dominated by one of the z-nodes. Furthermore, z_5 must be element of D as z_2 must be dominated. Now it is easy to verify that y_7 and y_8 must be element of D also. Hence the output value is equal to *true*. 4) If $v_2, v_3, v_4, v_5 \in D$, the proof is similar to 3).
G_C consists of two "or-structures". From the previous it is clear that either O_1, O_2 or O_2, O_3 must be element of D, representing the "logical" value of the associated clause, i.e., either *false* or *true*, respectively. In G the O_1-node of the subgraph associated with C_j is connected to the O_1-node of the subgraph associated with $C_{(j+1)}$, where $(j \in \{1, ..., (m-1)\}$. From the preceding argument it is clear that in each subgraph either O_1, O_2 or O_2, O_3 must be element of D, if an efficient total dominating set of G exists. As the O_1-nodes are connected to each other, it follows that all O_2-, O_3-nodes must be elements of D. Thus all outputs of the "or-structures" must be equal to *true*. Now it is easy to verify that G has an efficient total dominating set iff there is a satisfying truth assignment for the given set of clauses C. □

4.1 Determining an Efficient Total Dominating Set for Trees

However, for trees the situation is much better. Let $T = (V, E)$ be a rooted tree with root v. Assume that T has an efficient total dominating set D. Then for every node $w \in V$ four cases can be distinguished. 1) $w \notin D$ is dominated by exactly one son. It follows that all the other neighbors of w are element of $V - D$. In the next algorithm (*Algorithm ETDS*) w will have status *dominated*. 2) $w \notin D$ is dominated by its father. It follows that all sons of w must have status *dominated*. In Algorithm ETDS w will have status *to-be-dominated*. 3) w is element of D and no son of w is element of D. It follows that all sons of w must have status *to-be-dominated* and w's father is an element of D. In Algorithm ETDS w will have status *dominator*. 4) $w \in D$ is dominated by exactly one son. It follows that all neighbors of w and this son are element of $V - D$. In Algorithm ETDS w will have status *d-dominator*.
Algorithm ETDS is very similar to the algorithm described in the previous section. It starts at a node v and recursively computes all statuses that its sons can adopt

in an efficient total dominating set of T. These statuses determine which statuses v can adopt. Again it is possible that the statuses of the sons are incompatible in which case T does not have an efficient total dominating set.
Let $T = (V, E)$ be a tree. In the algorithm below every node $v \in V$ has a variable *status*(v) which can take a value out of the following six possibilities:

- **dominator**, if v is an element of the efficient total dominating set but is not dominated by a son.
- **d-dominator**, if v is an element of the efficient total dominating set and is dominated by exactly one son.
- **to-be-dominated**, if v is not an element of the efficient total dominating set and is not dominated by any son.
- **dominated**, if v is not element of the efficient total dominating set and is dominated by exactly one son.
- a pair (**status1, status2**) if it is possible to give either **status1** or **status2** to v without getting a conflict with the statuses of its sons. (In the following we say that "v can adopt **status1** (**status2**)".)
- **conflict**, if it is not possible to construct an efficient total dominating set for this tree.

Algorithm ETDS

procedure Efficient Total Dominating Set (v :node) ;
begin
forall sons w of v **do**
Efficient Total Dominating Set (w)
enddo ;
if {condition 0}
v has no sons
then *status*(v) := (**to-be-dominated, dominator**)
elsif one son of v has status **conflict**
then *status*(v) := **conflict**
elsif {condition 1}
one or more sons have status **to-be-dominated** and
all the other sons have status (**to-be-dominated, dominated**)
then status(v) := **dominator**
elsif {condition 2}
(exactly one son has status **dominator** and
all the other sons can adopt the status **to-be-dominated**) **or**
(exactly one son has status (**dominator, d-dominator**) and

```
        one or more sons have status (to-be-dominated, dominator) or
        to-be-dominated and all the other sons can adopt the status
        to-be-dominated)
    then status(v) := d-dominator
    elsif {condition 3}
        one or more sons have status dominated and
        all the other sons have status (to-be-dominated, dominated)
    then status(v) := to-be-dominated
    elsif {condition 4}
        (exactly one son has status d-dominator and
        all the other sons can adopt the status dominated) or
        (exactly one son has status (dominator, d-dominator) and
        one or more sons have status (dominated, d-dominator) and
        all the other sons can adopt the status dominated)
    then status(v) := dominated
    elsif {condition 5}
        all the sons have status (to-be-dominated, dominated)
    then status(v) := (to-be-dominated, dominator)
    elsif {condition 6}
        one or more sons have status (to-be-dominated, dominator) and
        all the other sons can adopt the status to-be-dominated
    then status(v) := (dominator, d-dominator)
    elsif {condition 7}
        one or more sons have status (dominated, d-dominator) and
        all the other sons can adopt status dominated
    then status(v) := (to-be-dominated, dominated)
    elsif {condition 8}
        exactly one son has status (dominator, d-dominator) and
        all other sons have status (to-be-dominated, dominated)
    then status(v) := (d-dominator, dominated)
    else status(v) := conflict
    endif
end {procedure Efficient Total Dominating Set} ;

begin {Main}
    input a tree T ;
    let v be a node of T ;
    orient T as a tree rooted at v ;
    Efficient Total Dominating Set(v) ;
    if status(v) is equal to conflict, dominator or to-be-dominated
    then there does not exist an efficient total dominating set of T
    else T has an efficient total dominating set
    endif
```

end {Main}

Note that the status pairs (**to-be-dominated, d-dominator**) and (**dominated, dominator**) do not occur in the algorithm. By induction it can be proven that these statuses do not occur in an algorithm that computes all possible statuses of a node in an efficient total dominating set. For trees with one node this is true. Assume that it is true for trees with $< k$ nodes. A node can only obtain a status pair (**to-be-dominated, d-dominator**) if all sons can adopt the status **dominated** and there is exactly one son that can also adopt the status **dominator** (while the other sons can also adopt the status **to-be-dominated**). By induction this is impossible. A node can only obtain a status pair (**dominated, dominator**) if all sons can adopt the status **to-be-dominated** and there is exactly one son that can also adopt the status **d-dominator** (while the other sons can also adopt the status **dominated**). Again, by induction this is impossible.

4.2 Correctness and Complexity of the Algorithm

Theorem 4.4 *Let $T = (V, E)$ be a tree and $v \in V$.* Efficient Total Dominating Set (v) *determines all possible statuses of v in an efficient total dominating sets of T. If status(v) is equal to* **conflict**, *then T has no efficient total dominating set. If status(v) is equal to* **dominator**, *or* **to-be-dominated**, *then T itself has no efficient total dominating set, but $T \cup \{(v, w)\}$, $T \cup \{(v, w), (w, x)\}$, respectively, has ($w$ and x are new nodes).*

Proof. Parts of this proof are very similar to the proof of Theorem 3.4. Again we prove the theorem by induction on the number of nodes of T. It can easily be verified that the theorem holds for trees with ≤ 3 nodes. Assume the theorem holds for trees with $< k$ nodes. Let $T = (V, E)$ be a tree with k nodes, and $v \in V$. Consider T as a rooted tree with root v. Let $w_1, ..., w_d \in V$ be the sons of v. With T_{w_i} we denote the subtree of T rooted by w_i. Consider Efficient Total Dominating Set(v) (ETDS(v) for short). By induction ETDS(w_i) determines all possible statuses of w_i. By the last remark in the previous section it is clear that we do not have to consider the status pairs (**to-be-dominated, d-dominator**) and (**dominated, dominator**). Furthermore, note that if one son of v has status **to-be-dominated**, then necessarily all sons must have status **to-be-dominated**, and v must be assigned the status **dominator**. In condition 1 all occurences of this case are stated. If one son has status **dominator**, then all the other sons must be able to adopt the status **to-be-dominated**. The first part of condition 2 is clear. Consider the second part. If one son has status (**dominator, d-dominator**) while a second son has status (**to-be-dominated, dominator**), then the first son must adopt the status **dominator** because the status **d-dominator** is conflicting with the possible statuses of the second son. All the other sons must be able to adopt the status **to-be-dominated**. Condition 3 and condition 4 are similar to condition 1

and condition 2, respectively. Here the status **dominated**, and **d-dominator** play the role of **to-be-dominated**, and **dominator**, respectively. Again these four cases indicate the conditions to be met in the following four cases. If a node v has no sons, i.e., it is a leaf, then the node is either an element of the dominating set or it is dominated by its father. As the procedure starts assigning the possible statuses at the leaves of the tree it must assign the status (**to-be-dominated, dominator**) to v. Furthermore this status can only be assigned if all sons of v can adopt the status **dominated** *and* **to-be-dominated**. Hence condition 0 and condition 5 precisely cover these cases. A node will be assigned the status (**dominator, d-dominator**) if all sons can adopt the status **to-be-dominated** *and* there exists exactly one son that can also adopt the status **dominator**. This is reflected in condition 6. A node will be assigned the status (**to-be-dominated, dominated**) if all sons can adopt the status **dominated** while there also exists a son that can adopt the status **d-dominator**. It is easy to verify that this is formulated in condition 7. Finally, a node will be assigned the status (**d-dominator, dominated**) if it has exactly one son that has status (**dominator, d-dominator**) and all other sons have status (**to-bedominated, dominated**). It is straightforward to verify the other claims made in the theorem. □

Clearly the algorithm only requires $O(|V|)$ steps. Note that the information computed by the algorithm can be used to determine all possible different efficient total dominating sets of the given tree, if such sets exist. We will not study this further.

5 Conclusions

We have shown that there exists an efficient linear-time algorithm that computes a minimum path-dominating set for any given tree. Furthermore, we devised efficient linear-time algorithms that determine whether a given tree has an independent efficient dominating set or an efficient total dominating set. As expected the algorithms are more direct and efficient than the algorithms obtained by more general techniques as described in [B87]. Furthermore extra insight in the combinatorics of the problems is obtained. Another useful by-product of the last two algorithms is that the information produced by the algorithms can be used to determine all possible independent efficient dominating sets or all possible efficient total dominating sets, if such sets exist. This justifies our search for these explicit algorithms that solve the three problems described above.

PATH-DOMINATING SET is NP-complete even when restricted to planar graphs with no faces with fewer than 5 edges. INDEPENDENT EFFICIENT DOMINATING SET and EFFICIENT TOTAL DOMINATING SET are NP-complete. The complexity of the problems restricted to many other classes of graphs remains open.

6 Acknowledgements

We thank Hans Bodlaender and Goos Kant for useful suggestions.

References

[B84] A.A. Bertossi. *Dominating Sets for Split and Bipartite Graphs.* Information Processing Letters, Vol. 19, 1984, pp. 37-40.

[B86] H.L. Bodlaender. *Classes of Graphs with Bounded Tree-Width.* Technical Report RUU-CS-86-22, Dept. of Computer Science, Utrecht University, 1986.

[B87] H.L. Bodlaender. *Dynamic Programming on Graphs with Bounded Tree-Width.* Technical Report RUU-CS-87-22, Dept. of Computer Science, Utrecht University, 1987, also in: T. Lepistö and A. Salomaa (Eds.), Automata, Languages and Programming, 15th Int. Colloq., Proceedings, Lecture Notes in Computer Science, Vol. 317, Springer Verlag, Heidelberg, 1988, pp. 105-118.

[CN84] G.J. Chang, G.L. Nemhauser. *The k-Domination and the k-Stability Problems on Sun-Free Chordal Graphs.* SIAM J. Algebraic Discrete Methods, Vol. 5, 1984, pp. 332-345.

[C78] E.J. Cockayne. *Domination of Undirected Graphs - a Survey.* Lecture Notes in Mathematics, Vol. 642, Springer Verlag, Berlin, 1978, pp. 141-147.

[CDH80] E.J. Cockayne, R.M. Dawes, S.T. Hedetniemi. *Total Domination in Graphs.* Networks, Vol. 10, 1980, pp. 211-219.

[GJ79] M.R. Garey, D.S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness.* W.H. Freeman, San Francisco, California, 1979.

[GJT76] M.R. Garey, D.S. Johnson, R.E. Tarjan. *The Planar Hamiltonian Circuit Problem is NP-complete.* SIAM J. Comp., Vol. 5, 1976, pp. 704-714.

[G85] A. Gibbons, *Algorithmic Graph Theory.* Cambridge, University Press, Cambridge, 1985.

[H69] F. Harary. *Graph Theory.* Addison-Wesley Publ. Comp., Reading, Mass., 1969.

[HHL84] S. Hedetniemi, S. Hedetniemi, R. Laskar. *Domination in Trees: Models and Algorithms.* In, Y. Alavi, G. Chartrand, L. Lesniak, D.R. Lick, C.E. Wall (Eds.), Graph Theory with Applications to Algorithms and Computer Science, John Wiley & Sons Inc., 1984, pp. 423-442.

[J85] D.S. Johnson. *The NP-Completeness Column: An Ongoing Guide.* Journal of Algorithms, Vol. 6, 1985, pp. 434-451.

[LPHH84] R. Laskar, J. Pfaff, S.M. Hedetniemi, S.T. Hedetniemi. *On the Algorithmic Complexity of Total Domination.* SIAM J. Algebraic Discrete Methods, Vol. 5, 1984, pp. 420-425.

[YL90] C. Yen, R.C.T. Lee. *The Weighted Perfect Domination Problem.* Information Processing Letters, Vol. 35, 1990, pp. 295-299.