Two Algorithms for Secure Graph Domination

AP Burger, AP de Villiers & JH van Vuuren
Department of Logistics, Stellenbosch University,
Private Bag X1, Matieland, 7602, South Africa
apburger@sun.ac.za, 14812673@sun.ac.za
and vuuren@sun.ac.za

Dedication to Professor Christina Mynhardt

Kieka Mynhardt has, over the years, been a shining example of what it means to be a scholar at the forefront of her field, what it means to contribute to the discipline of graph theory on a fundamental level and what it means to pursue excellence. She has been and remains a prolific author, particularly in the area of graph domination, and her energy, enthusiasm and almost dogged pursuit of the truth has been an inspiration to many a graph theorist in South Africa and elsewhere. In short, she has set an example and a standard that are indeed difficult to follow! It is therefore our honour to dedicate this paper, with deep appreciation, respect and gratitude, to our friend and colleague, Kieka Mynhardt, on the occasion of her birthday. Happy birthday, Kieka, may there be many more!

Abstract

A subset X of the vertex set of a graph G is a *secure dominating set* of G if X is a dominating set of G and if, for each vertex u not in X, there is a neighbouring vertex v of u in X such that the swap set $(X - \{v\}) \cup \{u\}$ is again a dominating set of G. The *secure domination number* of G, denoted by $\gamma_s(G)$, is the cardinality of a smallest secure dominating set of G. In this paper we present two algorithms (a branch-and-reduce algorithm as well as a branch-and-bound algorithm) for determining the secure domination number of a general graph G of order n. The worst-case time complexities of both algorithms are $\mathcal{O}(2^{n-s-\sum_{i=1}^{k}(|\mathcal{R}_i|-1)})$, where s is the number of support vertices in G and $\mathcal{R}_i, \ldots, \mathcal{R}_k$ are the redundancy classes of G (two vertices are in the same redundancy class if they are adjacent and share the same closed neighbourhood which forms a clique in G).

JCMCC **85** (2013), pp. 321-339

1 Introduction

Let $G = (V, E)$ be a simple graph of order n and denote the open neighbourhood of a vertex $u \in V$ by $N(u)$ and its closed neighbourhood by $N[v]$. A set $D \subseteq V$ is a *dominating set* of G if every vertex in $V - D$ is adjacent to some vertex in D. The minimum cardinality of a dominating set of G is called the *domination number* of G and is denoted by $\gamma(G)$. A set $X \subseteq V$ is a *secure dominating set* of G if it is a dominating set of G and if, for each $u \in V - X$, there exists a $v \in N(u) \cap X$ such that the *swap set* $(X - \{v\}) \cup \{u\}$ is again a dominating set of G, in which case v is said to *defend* u [8]. The minimum cardinality of a secure dominating set of G is called the *secure domination number* of G and is denoted by $\gamma_s(G)$. In [8] it was noted that

$$\gamma(G) \leq \gamma_s(G) \tag{1}$$

for any graph G, where $\gamma(G)$ denotes the domination number of G. A number of general bounds were also established for the parameter $\gamma_s(G)$ in [8], and values of $\gamma_s(G)$ were additionally established for various graph classes, such as paths, cycles, complete multipartite graphs and products of paths and cycles. Various aspects of secure domination and properties of the secure domination number of a graph have been studied in the literature [2, 3, 4, 5, 6, 10, 12].

After reviewing a number of preliminary results in §2, we present a framework for testing whether a given subset $X \subseteq V$ is a secure dominating set of G in §3. A branch-and-reduce approach and a branch-and-bound approach for determining the secure domination number of a general graph are presented in §4 and §5, respectively. The worst-case time complexities of these algorithms are noted in §6. Numerical results obtained by means of the algorithms are presented for a number of graph classes in §7. The paper closes in §8.

2 Preliminary results

Recall that for any $v \in X$, a vertex $u \in V - X$ is an X-external private neighbour of v if $N(u) \cap X = \{v\}$. We denote the set of all X-external private neighbours of v by $\mathrm{Epn}(v, X)$. Furthermore, if $V' \subseteq V$, then we denote the subgraph of G induced by V' by $G[V']$. Both our algorithms for determining the secure domination number of a graph G are based on the following characterisation of minimum secure dominating sets, which is due to Cockayne *et al.* [8, Proposition 2] and dates from 2005.

Theorem 1 ([8]) *Let X be a dominating set of G. Then a vertex $v \in X$ defends a vertex $u \in V - X$ if and only if $G[\mathrm{Epn}(v, X) \cup \{u, v\}]$ is complete.*

An *end-vertex*[1] of a graph is a vertex of degree 1 and a *support vertex* is a vertex of degree at least 2 that is adjacent to an end-vertex. Moreover, a set $R \subseteq V$ of at least two vertices is called a *redundant set* of G if any two distinct vertices $u, v \in R$ share a common closed neighbourhood (*i.e.* $N[u] = N[v]$) which forms a clique in G. A maximal redundant set of G is called a *redundancy class* of G. It follows by the next result that we may, without loss of generality, include all support vertices and exclude all but one vertices of each redundancy class of a graph G when seeking a minimum secure dominating set of G. It is easy to prove the following result.

Proposition 1 *Let s be a support vertex of G and let u and v be two distinct vertices in the same redundancy class of G. Then*
(a) there is a minimum secure dominating set of G containing s.
(b) no minimum secure dominating set of G contains both u and v.

We call a connected subgraph of a tree induced by a support vertex and its adjacent leaves an *end-cluster* of the tree. The simple heuristic presented in pseudo-code form as Algorithm 1 may be used to find an upper bound on the secure domination number of a connected graph. The heuristic is based on the fact that an end-cluster of a tree is securely dominated by its leaves. Therefore a secure dominating set X of a graph G may be obtained by computing a spanning tree of G and then including all the leaves of this tree in X, after which all end-clusters of the tree may be pruned away to form a smaller tree. The same pruning procedure may be applied to this smaller tree, after having inserted the newly formed leaves of this smaller tree into X, and so on, until all the vertices of the original spanning tree have been pruned away.

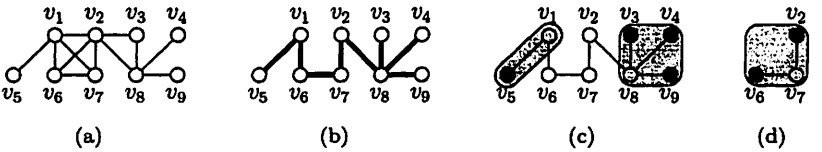

Figure 1: (a) A graph G_1 for which $\gamma_s(G_1) = 4$. (b) A spanning tree of G_1. (c)–(d) The result of identifying and pruning away end-clusters of the spanning tree in (b) according to Algorithm 1 to arrive at the secure dominating set $\{v_2, v_3, v_4, v_5, v_6, v_9\}$ of cardinality 6 for G_1.

[1] An end-vertex of a tree is often called a *leaf* of the tree.

Algorithm 1: Heuristic

Input : A connected graph G.
Output: A secure dominating set of G.

1 $X \leftarrow \emptyset$;
2 $T \leftarrow$ A spanning tree of G;
3 **while** $V(T) \neq \emptyset$ **do**
4 Insert all leaves of T into X;
5 Update T by removing all its end-clusters;
6 **end**
7 **return** $[X]$;

Consider, as an example, the graph G_1 in Figure 1(a). A spanning tree of G_1 is shown in Figure 1(b). During the first iteration of the while-loop spanning Steps 3–6 of Algorithm 1, the four leaves (indicated as solid vertices) in Figure 1(c) are inserted into the set X. Thereupon the two end-clusters (highlighted in grey in the figure) are removed from the tree to obtain the smaller tree in Figure 1(d). The two leaves of this smaller tree are also inserted into X after which the entire tree in the figure is pruned away (because the tree consists of a single end-cluster). This process results in the secure dominating set $X = \{v_2, v_3, v_4, v_5, v_6, v_9\}$ of cardinality 6 for the graph G_1.

3 Testing for secure domination

Let X be any subset of V. We partition the vertex set V into five subsets. Denote by X_P the set of vertices in X which have private neighbours external to X, and let $P = \bigcup_{v \in X_P} \mathrm{Epn}(v, X)$ be the set of these external private neighbours of the vertices in X_P. Furthermore, define $X_D = X - X_P$ and let D be the set of vertices in $V - (X \cup P)$ that are dominated by X_D. Finally, let U be the set of vertices not in X, P or D. Then

$$V = \underbrace{X_P \cup X_D}_{X} \cup P \cup U \cup D, \tag{2}$$

as illustrated in Figure 2. It follows by Theorem 1 that each vertex in D is defended by at least one vertex in X_D. Whilst every vertex in P is further *dominated* by exactly one vertex in X_P, some of the vertices in P may not be *defended* by any vertices in X_P. Even worse, some vertices in U may not even be dominated by any vertices in X, let alone be defended. For example, in Figure 2 the vertices $u_1, u_2 \in P$ are not

defended (by v_1), because $G[\{v_1, u_1, u_2\}]$ is not complete. The vertices $u_3, u_4 \in P$ are, however, defended (by v_2), because $G[\{v_2, u_3, u_4\}]$ is complete. The vertices $u_5, u_6, u_7 \in P$ are similarly defended (by v_3, v_4 and v_4, respectively). Furthermore, the vertex $u_8 \in U$ is defended (by v_4) since $G[\{v_4, u_6, u_7, u_8\}]$ is complete. However, $u_9 \in U$ is defended by neither v_3 nor v_4, because both $G[\{v_3, u_5, u_9\}]$ and $G[\{v_4, u_6, u_7, u_9\}]$ are incomplete. Finally, $u_{10}, u_{11}, u_{12} \in U$ are not even dominated by X, let alone defended.

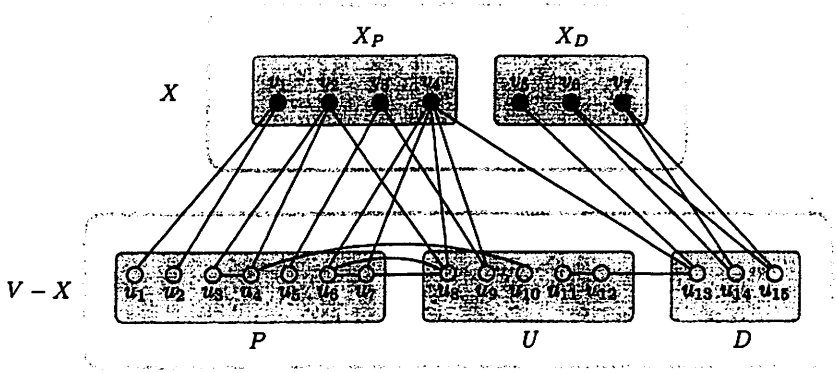

Figure 2: An example of the vertex set partition (2).

In order to verify whether X is a secure dominating set of G, it suffices, by Theorem 1, to verify that:

I every vertex in U is dominated by at least one vertex in X,

II the private neighbours in P of each vertex in X_P form a clique in G, and

III there exists, for each $u \in U$, a vertex $v \in X$ such that $G[\text{Epn}(v, X) \cup \{u, v\}]$ is complete.

If at least one of the above criteria I–III does not hold, then X is not a secure dominating set of G. These criteria have been ordered in increasing order of computational complexity above so as to represent an efficient set of secure domination testing criteria for our algorithms.

4 A branch-and-reduce algorithm

Our first (recursive) algorithm is a so-called *branch-and-reduce algorithm* and traverses a depth-first search tree to construct a secure dominating set

X of G such that $X \subseteq V - F$, where F denotes a set of specified, forbidden vertices (*i.e.* vertices that may not be included in X). The algorithm takes as input the quadruple (G, X_P, D, F), as described in §3 above. The sets P and U, also described in §3, may easily be calculated from this quadruple. Initially, X_P is taken as the set S of all support vertices of G while F is taken as the set R of all redundant vertices, both without loss of generality, in view of Proposition 1, while the set D is initially empty. A standard branch-and-reduce approach is employed at each call of the algorithm in the sense that each vertex $v \in V - S - R$ is considered in turn to be included either in X or F, respectively:

- Inclusion of a vertex v in X in the former case may annihilate external private neighbours of vertices already in X_P, resulting in a need to repartition the extended set $X' = X_P \cup X_D \cup \{v\}$ into a new subset X'_P of vertices with private neighbours external to X' and a subset X'_D with no such external private neighbours, in turn resulting in a new set D' of defended vertices. The graph may be reduced to $G - X'_D$, since the vertices in X'_D cannot annihilate external private neighbours of vertices in X'_P or any vertices that may be added to X'_P during a later algorithmic call. Furthermore, all the vertices in D' are defended. A smaller problem may then be solved by calling the algorithm with the reduced quadruple $(G - X'_D, X'_P, D', F)$ as input.
- In the latter case a smaller problem may be solved by simply calling the algorithm with input quadruple $(G, X_P, D, F \cup \{v\})$.

The algorithm is presented in pseudo-code form as Algorithm 2. Initially, the algorithm tests whether X_P is a secure dominating set of $G - D$; this occurs in the three nested if-tests spanning Steps 1–8, which corresponds to testing the validity of criteria I–III in §3. If all three tests succeed (*i.e.* if X_P securely dominates $G - D$), a secure dominating set has been found and the algorithm returns an empty set in Step 5, thereby terminating the current branch of the search tree. If no secure dominating set is found, the algorithm prepares for two further calls to itself by selecting another branching vertex $v \in V - X_P - F$ with largest closed neighbourhood to include in X (at this stage $X_D = \emptyset$). If all the vertices of G are either in X_P or in F (Step 9 of Algorithm 2), then no vertex can be considered as a branching vertex for which the branch of the search tree will yield a secure dominating set. Otherwise, two reduction rules are considered. The first reduction rule considers whether the closed neighbourhood of v is a singleton. If so, all the vertices in G are isolated, in which case G can only be securely dominated by adding all the vertices in $V - X_P - F$ to X, as performed in Step 11 of Algorithm 2.

Algorithm 2: BR

Input : A graph G, the set X_P, the set D and a forbidden set F.
Output: A minimum secure dominating set of G.

1 **if** X_P *is a dominating set of* $G - D$ **then**
2 **if** *for each* $s \in X_P$, $G[\{s\} \cup \mathrm{Epn}(s, X_P)]$ *is complete* **then**
3 $P \leftarrow \bigcup_{s \in X_P} \mathrm{Epn}(s, X_P)$; $U \leftarrow V - X_P - P - D$;
4 **if** *for each* $u \in U$ *there exists a* $s \in X_P$ *such that* u *is adjacent to all the vertices* $\mathrm{Epn}(s, X_P)$ **then**
5 **return** $\emptyset$;
6 **end**
7 **end**
8 **end**
9 **if** $V - X_P - F = \emptyset$ **then** disqualify branch;
10 Select any $v \in V - X_P - F$ with largest closed neighbourhood;
11 **if** $N[v] = 1$ **then return** $V - X_P - F$;
12 **else if** $N[v] = 2$ **then return** a single vertex from each component;
13 $X' \leftarrow \{v\} \cup X_P$; $X'_P \leftarrow \bigcup_{x \in X'} \mathrm{Epn}(x, X')$; $X'_D \leftarrow X' - X'_P$;
$D' \leftarrow N(X'_D) \cap (V - X')$;
14 **return** $\min\{\{v\} \cup \mathrm{BR}(G - X'_D, X'_P, D', F), \mathrm{BR}(G, X_P, D, F \cup \{v\})\}$;

Algorithm 3: FindBR

Input : A connected graph G.
Output: Secure dominating set of minimum cardinality for G.

1 $S \leftarrow$ set of support vertices of G;
2 $R \leftarrow$ all but one vertices from each redundancy class of G;
3 **return** $[\mathrm{BR}(G, S, \emptyset, R)]$;

If the closed neighbourhood of v is not a singleton, the second reduction rule is invoked, testing whether the closed neighbourhood of v is a 2-set. If so, the graph G consists of the vertex disjoint union of paths of orders 1 and 2. A secure dominating set of G may be formed by selecting a single vertex from each of its components, which occurs in Step 12 of Algorithm 2. If neither reduction rule applies, the branching rule is employed to generate two further, smaller problem instances when the algorithm calls itself. The repartition of X' into two new sets X'_P and X'_D, as described above, occurs in Step 13, while the two branching calls to Algorithm 2 occur Step 14, which returns a minimum secure dominating set of G.

A search tree is constructed from the recursive calls to Algorithm 2. The root of this search tree is generated by the initial call $\mathrm{BR}(G, S, \emptyset, R)$ to

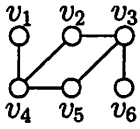

Figure 3: A graph G_2 for which $\gamma_s(G_2) = 4$. G_2 contains two support vertices, namely v_3 and v_4, and no redundancy classes.

Algorithm 2, which occurs in Step 3 of an external initialisation procedure for which a pseudo-code listing is presented in Algorithm 3. This latter algorithm takes as input a connected graph G and determines the set of support vertices S and the set of redundant vertices R of G. Algorithm 2 is called in Step 3 where S and R are used as input parameters to reduce the size of the search tree.

The search tree constructed by means of Algorithms 2 and 3 for the graph G_2 in Figure 3 is shown in Figure 4. Since G_2 has the set of support vertices $S = \{v_3, v_4\}$, the set X is initialised as $\{v_3, v_4\}$ in the root of the tree, labelled node 0. Although X is a dominating set of G_2, it is not a secure dominating set of G_2. The vertices of $G_2 - S$ are branched in order of decreasing size of their respective closed neighbourhoods. In the case of a tie, vertices are selected in the order in which they are labelled. The tree is traversed in a depth-first fashion, and branching on the inclusion of v_2 into X_P or F produces the sets shown in nodes 1 and 2 of the search tree in Figure 4, respectively. The vertex v_2 is included in X in node 1 of the search tree for which the union of the sets $X_P = \{v_3, v_4\}$ and $X_D = \{v_2\}$ is not a secure dominating set of G_2. When branching on node 1, the vertex v_2 may be removed from G_2, since v_2 can defend itself and cannot annihilate external private neighbours of any vertices that may later enter X_P. The vertex v_5 is chosen next to branch upon. This time the union of the sets $X_P = \{v_3, v_4\}$ and $X_D = \{v_2, v_5\}$ is a secure dominating set of cardinality 4 for G_2. The tree is therefore bounded in Step 5 of Algorithm 2, denoted by bounding reason "[a]" in the figure. Backtracking to node 3, the set of forbidden vertices is updated to $F = \{v_5\}$ and the vertex v_1 is chosen next to branch upon, leading to nodes 4 and 5 of the search tree. Node 4 is bounded since $X = \{v_1, v_2, v_3, v_4\}$ is a secure dominating set of cardinality 4 for G_2. In node 5 the set of forbidden vertices is updated to $F = \{v_1, v_5\}$ and the vertex v_6 is chosen next to branch upon in order to form nodes 6 and 7 of the search tree. Node 6 is bounded since $X = \{v_2, v_3, v_4, v_6\}$ is a secure dominating set of cardinality 4 for G_2. Node 7 is bounded since the vertices in $X = \{v_2, v_3, v_4\}$ do not form a secure dominating set of G_2 and the remaining vertices of G are already in F; hence a secure dominating set

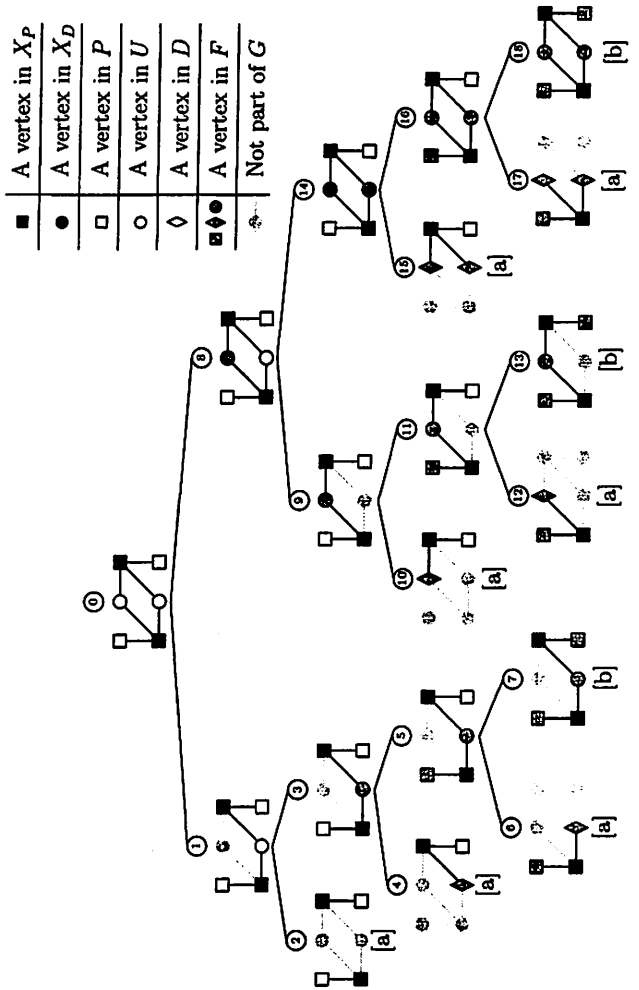

Figure 4: The secure dominating set depth-first search tree for the graph G_2 in Figure 3 obtained by means of Algorithms 2 and 3. The order of traversing the tree is shown by means of circled node numbers. Bounding the search tree is motivated as follows: [a] The tree is bounded in Step 5 of Algorithm 2, since a secure dominating set has been found. [b] The tree is bounded in Step 9 of Algorithm 2, because the vertices in X do not form a secure dominating set, while the remaining vertices are in F.

cannot be found. The tree is therefore bounded in Step 9 of Algorithm 2, denoted by the bounding reason "[b]" in the figure. Thereafter the search

backtracks to node 2 and the same procedure is followed in the remaining ten nodes of the search tree. Minimum cardinality secure dominating sets of size 3 are found in node 15 with $X = \{v_1, v_3, v_4\}$ and node 17 with $X = \{v_3, v_4, v_6\}$.

5 A branch-and-bound algorithm

Our second algorithm also constructs a secure dominating set X of G (recursively) from among the vertices in $V - F$, where $F \subset V$ again denotes a set of forbidden vertices, but this time the algorithm adopts a classical branch-and-bound approach. The algorithm takes as input the triple of sets (X_P, X_D, F), as described in §3. From this triple, the sets P, U and D, also described in §3, may readily be determined. Initially X_P is again taken as the set S of all support vertices of G and F is again taken as the set R of all redundant vertices of G, without loss of generality in view of Proposition 1, while the set X_D is initially empty. A branching decision is taken with respect to each vertex v in $V - S - R$ in turn, namely either to include it in $X = X_P \cup X_D$, or to include it in F:

- As in the branch-and-reduce approach in §4, the inclusion of a vertex v in X in the former case may annihilate external private neighbours of vertices already in X_P. The resulting set $X' = X_P \cup X_D \cup \{v\}$ must therefore again be repartitioned into a new subset X'_P of vertices with private neighbours external to X' and a subset X'_D with no such external private neighbours. A smaller problem may then be solved by calling the algorithm with the triple (X'_P, X'_D, F) as input.
- In the latter case a smaller problem is solved by calling the algorithm with input triple $(X_P, X_D, F \cup \{v\})$.

The algorithm is presented in pseudo-code form as Algorithm 4. The difference between this algorithmic approach and the one in §4 is that three variables, LowerBound, SmallestSetSoFar and UpperBound, are defined and maintained globally (*i.e.* external to Algorithm 4), but Algorithm 4 is able to update the values of the latter two of these variables as smaller and smaller secure dominating sets are uncovered during the search. When the search is complete, the variable SmallestSetSoFar contains a minimum secure dominating set of cardinality UpperBound for the graph G. However, if at some point during execution of the algorithm, the set X has fewer than LowerBound elements, where LowerBound is a theoretical lower bound on $\gamma_s(G)$, then X cannot possibly be a secure dominating set of G and hence there is no need for testing the validity of any of the criteria I–III in §3.

This observation is used to speed up the secure dominating set construction process in Step 1 of Algorithm 4. Furthermore, if so many vertices have been classified as forbidden (*i.e.* have been included in the set F and hence cannot be included in X) that $V - F$ does not even dominate G, then no completion of X can be a secure dominating set of G. This is the reason for the bounding test in Step 2 of the algorithm. Finally, if the set X is so large that inclusion of one more vertex into the set will yield a set larger than SmallestSetSoFar, then the set X cannot be completed to a *smaller* secure dominating set than SmallestSetSoFar, thus giving rise to the bounding test in Step 3 of Algorithm 4.

Algorithm 4: BB

Input : The sets X_P and X_D in (2), and a set F of forbidden vertices.

Output: A minimum secure dominating set of G from amoungst the vertices $V - F$.

1 **if** $|X_P \cup X_D| <$ LowerBound *then* go to Step 15;
2 **else if** $V - F$ does not dominate G **then return** [ ];
3 **else if** $|X_P \cup X_D| \geq$ UpperBound **then return** [ ];
4 $X \leftarrow X_P \cup X_D$;
5 **if** X *is a dominating set of* G **then**
6 **if** *for each* $s \in X_P$, $G[\{s\} \cup \mathrm{Epn}(s, X)]$ *is complete* **then**
7 $P \leftarrow \bigcup_{s \in X_P} \mathrm{Epn}(s, X)$; $D \leftarrow N(X_D) \cap (V - X)$;
$U \leftarrow V - X - D - P$;
8 **if** *for each* $u \in U$ *there exists a* $s \in X_P$ *such that* u *is adjacent to all the vertices* $\mathrm{Epn}(s, X)$ **then**
9 UpperBound $\leftarrow |X|$;
10 SmallestSetSoFar $\leftarrow X$;
11 **return** [ ];
12 **end**
13 **end**
14 **end**
15 Select any $v \in V - X - F$;
16 $X' \leftarrow \{v\} \cup X_P \cup X_D$; $X'_P \leftarrow \bigcup_{x \in X'} \mathrm{Epn}(x, X')$; $X'_D \leftarrow X' - X'_P$;
17 BB(X'_P, X'_D, F);
18 BB$(X_P, X_D, F \cup v)$;

If all three of the above tests fail, then it must be determined whether X is a secure dominating set of G; this occurs in the three nested if-tests spanning Steps 5–14 of the algorithm, which correspond to criteria I–III of §3. If all three of these tests succeed (*i.e.* if X securely dominates G), then a smaller secure dominating set than SmallestSetSoFar has been discovered,

resulting in an update of the global variables SmallestSetSoFar and UpperBound in Steps 9 and 10 of Algorithm 4, after which the algorithm returns to the previous level of recursion in Step 11. If no such smaller secure dominating set has been found, the algorithm prepares for two further calls to itself by selecting another branching vertex v to include in X so as to form the larger set $X' = X_P \cup X_D \cup \{v\}$ or alternatively to include in F. The repartition of X' into two sets X'_P and X'_D, as described in §4, occurs in Step 16, while the two branching calls to Algorithm 4 occur in Steps 17 and 18.

Algorithm 5: FindBB

Input : A connected graph G of order n.
Output: Secure dominating set of minimum cardinality for G.

```
1  LowerBound ← max{⌈n/1 + Δ⌉, ⌈diam(G) + 1/3⌉};
2  SmallestSetSoFar ← A valid secure dominating set of G;
3  UpperBound ← |SmallestSetSoFar|;
4  if LowerBound = UpperBound then return [SmallestSetSoFar];
5  else
6      S ← set of support vertices of G;
7      R ← all but one vertices of each redundancy class of G;
8      BB(S, ∅, R);
9  end
10 return [SmallestSetSoFar];
```

A search tree may again be constructed from the recursive calls to Algorithm 4. The root of this search tree is generated by the initial call $\mathrm{SDS}(S, \emptyset, R)$ to Algorithm 4, which occurs in Step 8 of an external initialisation procedure for which a pseudo-code listing is given in Algorithm 5. This latter algorithm takes as input a connected graph G of order n and determines the lower bound[2] LowerBound on $\gamma_s(G)$ in Step 1. Any valid (not necessarily minimum, but easily computable) secure dominating set SmallestSetSoFar of G is determined[3] in Step 2 of Algorithm 5, and the cardinality of this set is taken as the upper bound UpperBound on $\gamma_s(G)$ in Step 3 of the algorithm. If the bounds in Steps 1 and 3 coincide, then the secure dominating set computed in Step 2 is produced as output in Step 4; otherwise Algorithm 4 is called in Step 8.

[2]The lower bounds currently shown in Step 1 of the algorithm are well-known lower bounds on the domination number of a connected graph G, and hence also on $\gamma_s(G)$ by virtue of (1); see, for example, [11, Theorems 2.11 and 2.34].

[3]Such a set may be determined in a variety of ways. Algorithm 1 may, for example, be used.

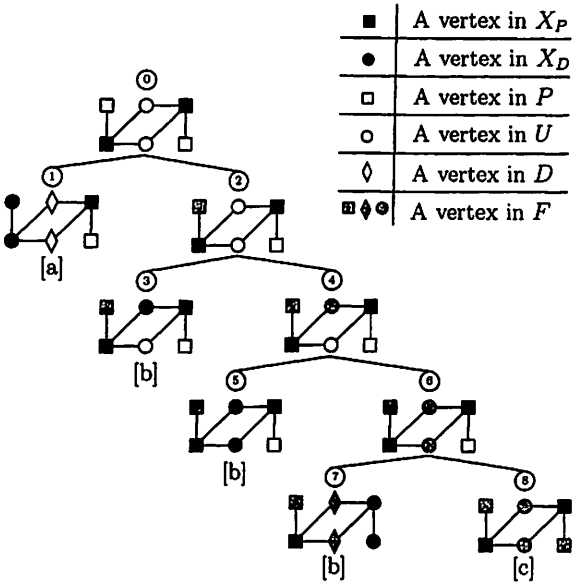

Figure 5: The secure dominating set depth-first search tree for the graph G_2 in Figure 1(a) obtained by means of Algorithms 4 and 5. The order of traversing the tree is shown by means of circled node numbers. Bounding of the search tree is motivated as follows: [a] The tree is bounded in Step 11 of Algorithm 4, because a smaller secure dominating set is found than SmallestSetSoFar. [b] The tree is bounded in Step 3 of Algorithm 4, because the current set X cannot be completed to a smaller secure dominating set than SmallestSetSoFar. [c] The tree is bounded in Step 2 of Algorithm 4, because $V - F$ does not securely dominate the graph.

Construction of the search tree by means of Algorithms 4 and 5 for the graph G_2 in Figure 1(a) is shown in Figure 5. The global variables are initialised as

$$\begin{aligned} \mathsf{LowerBound} &= \max\left\{\left\lceil \frac{n}{1+\Delta(G_2)} \right\rceil, \left\lceil \frac{\mathrm{diam}(G_2)+1}{3} \right\rceil\right\} \\ &= \max\left\{\left\lceil \frac{6}{1+3} \right\rceil, \left\lceil \frac{4+1}{3} \right\rceil\right\} = 2 \end{aligned}$$

and $\mathsf{SmallestSetSoFar} = \{v_1, v_2, v_5, v_6\}$ of cardinality $\mathsf{UpperBound} = 4$ in Steps 1–3 of Algorithm 5. Since G_2 has the set of support vertices $S = \{v_3, v_4\}$, the set X is initialised as $\{v_3, v_4\}$ in the root of the search tree, labelled node 0. Although X is a dominating set of G_2, it is not a secure dominating set of G_2. The vertices of $G_2 - S$ are branched upon

in the order in which they are labelled. Branching on the inclusion of v_1 into X or F produces the sets shown in nodes 1 and 2 of the search tree in Figure 5, respectively. The tree is again traversed in a depth-first fashion, and the vertex v_1 is chosen next in node 1 of the search tree for which the set $X = \{v_1, v_3, v_4\}$ is a secure dominating set of cardinality 3 for G_2. The global variables SmallestSetSoFar and UpperBound are updated to reflect this discovery at node 1. The tree is therefore bounded in Step 11 of Algorithm 4, denoted by the bounding reason "[a]" in the figure. Backtracking to node 2, the set of forbidden vertices is updated from the empty set to $F = \{v_1\}$ and the vertex v_2 is chosen next to branch upon, leading to nodes 3 and 4 of the search tree. At node 3 the tree is immediately bounded in Step 3 of Algorithm 4, denoted by the bounding reason "[b]" in the figure, because at this node the set $X = \{v_2, v_3, v_4\}$ cannot be completed to a *smaller* secure dominating set of G_2 than SmallestSetSoFar. Thereafter the search backtracks to node 4 and the same procedure is followed in the remaining four nodes of the search tree, never improving on SmallestSetSoFar $= \{v_1, v_3, v_4\}$ and finally terminating at node 8 where the tree is immediately bounded in Step 2 of Algorithm 4, denoted by the bounding reason "[c]" in the figure, because at this node the set $V - F$ does not even dominate G_2. The set SmallestSetSoFar $= \{v_1, v_3, v_4\}$ is therefore smallest possible, leading to the result $\gamma_s(G_2) =$ UpperBound $= 3$.

6 Worst-case complexity analysis

Since redundant vertices and support vertices may be included in X and F, respectively, before construction of the search trees starts, the worst-case time complexities of both algorithms are $\mathcal{O}(2^{n-s-\sum_{i=1}^{k}(|\mathcal{R}_i|-1)})$ for a graph of order n with s support vertices and $\mathcal{R}_1, \ldots, \mathcal{R}_k$ redundancy classes. This exponential worst-case time complexity is to be expected, since the decision problem associated with finding a dominating set of minimum cardinality is **NP**-complete [9, pp. 75]. Therefore, the problem of finding a secure dominating set of minimum cardinality is **NP**-hard.

There is, however, a practical run-time trade-off between the number of recursive algorithmic calls and the time associated with each such call when one switches from the branch-and-bound paradigm to the branch-and-reduce paradigm. In particular, there are typically fewer algorithmic calls in the branch-and-bound algorithm of §5 than in the branch-and-reduce algorithm of §4 due to the incorporation of the global lower and upper bounds which are utilised to bound the search tree in the former case. Whereas the input graph is globally defined and hence of the same order for all of the algorithmic calls in the former case, the input graph

becomes smaller for recursive calls in the latter case as the search progresses deeper into the search tree. Therefore it is not easy to make a general pronouncement as to which algorithm is expected to be faster in terms of actual computation time. For small graphs, however, the classical branch-and-bound algorithm of §5 seems to be faster, as we demonstrate in the next section.

7 Numerical results

Exact values of and bounds on the secure dominating numbers of members of specific graph classes were established by Cockayne *et al.* [8]. In particular, general bounds were presented for grid graphs in the plane ($\mathcal{P}_m \,\square\, \mathcal{P}_k$) and for grid graphs on the torus ($\mathcal{C}_m \,\square\, \mathcal{C}_k$). Some numerical results obtained via the algorithms in §4 and §5 for these graph classes are presented in Table 1.

In the table, the column labelled Calls contains the number of times each algorithm calls itself. All times are measured in seconds. It was verified that the values of γ_s determined via the algorithms in §4 and §5 are indeed within the theoretical bounds established by Cockayne *et al.* [8]. (No support vertices or redundancy classes are present for the examples in Table 1.) Both the algorithms were coded in Mathematica 8.0 [13] and executed on an Intel(R) Core(TM)2 Duo 3GHz processor with 3.7 GB RAM running in Linux Ubuntu [1].

Similar results are presented in Table 2 for all non-isomorphic, connected circulants of orders $n \leq 11$.

As a simple verification mechanism it was confirmed that both algorithms produced the same value of γ_s for all the above-mentioned test graphs. Since the numerical results reported in this section were obtained via computer implementations in the high-level programming language Mathematica 8.0 [13], the run-times in Tables 1–2 are relatively long. Experience has, however, shown that speed-ups of an order of magnitude of 10^2 are typically possible with implementations in low-level languages such as C or C++.

8 Conclusion

Two algorithms were presented in this paper for computing the secure domination number of a general graph. After introducing a number of preliminaries in §2, a general paradigm was established in §3 for deciding efficiently

	Graph	n	γ_s	Branch-and-reduce (§4) Calls	Time	Branch-and-bound (§5) LB	UB	Calls	Time
Grids in the plane	$\mathcal{P}_2 \,\square\, \mathcal{P}_2$	4	2	21	0.01	2	2	21	0.01
	$\mathcal{P}_2 \,\square\, \mathcal{P}_3$	6	3	87	0.04	2	3	89	0.04
	$\mathcal{P}_2 \,\square\, \mathcal{P}_4$	8	4	359	0.17	2	4	251	0.15
	$\mathcal{P}_2 \,\square\, \mathcal{P}_5$	10	4	1 593	0.85	3	4	819	0.64
	$\mathcal{P}_2 \,\square\, \mathcal{P}_6$	12	5	6 479	3.89	3	5	2 049	2.26
	$\mathcal{P}_2 \,\square\, \mathcal{P}_7$	14	6	26 111	17.50	4	6	7 539	11.63
	$\mathcal{P}_2 \,\square\, \mathcal{P}_8$	16	7	107 467	79.38	4	7	19 867	43.61
	$\mathcal{P}_2 \,\square\, \mathcal{P}_9$	18	7	438 805	353.69	5	7	58 849	153.63
	$\mathcal{P}_2 \,\square\, \mathcal{P}_{10}$	20	8	1 783 065	1 570.81	5	8	141 673	501.85
	$\mathcal{P}_2 \,\square\, \mathcal{P}_{11}$	22	9	7 250 823	7 015.41	6	9	507 499	2 147.22
	$\mathcal{P}_2 \,\square\, \mathcal{P}_{12}$	24	10	29 443 157	30 764.65	6	10	1 291 811	7 437.22
	$\mathcal{P}_3 \,\square\, \mathcal{P}_3$	9	4	695	0.37	2	4	383	0.27
	$\mathcal{P}_3 \,\square\, \mathcal{P}_4$	12	5	5 971	3.74	3	5	1 939	2.05
	$\mathcal{P}_3 \,\square\, \mathcal{P}_5$	15	6	49 471	35.53	3	6	8 987	17.88
	$\mathcal{P}_3 \,\square\, \mathcal{P}_6$	18	7	414 835	341.40	4	7	39 279	109.07
	$\mathcal{P}_3 \,\square\, \mathcal{P}_7$	21	8	3 420 847	3 164.34	5	8	267 851	1 154.19
	$\mathcal{P}_3 \,\square\, \mathcal{P}_8$	24	10	28 175 363	29 321.29	5	10	1 209 227	7 990.45
	$\mathcal{P}_4 \,\square\, \mathcal{P}_4$	16	7	101 207	76.93	4	7	20 933	48.61
	$\mathcal{P}_4 \,\square\, \mathcal{P}_5$	20	8	1 684 617	1 525.13	4	8	118 315	451.01
	$\mathcal{P}_4 \,\square\, \mathcal{P}_6$	24	9	27 889 381	29 366.20	5	9	848 199	4 989.06
	$\mathcal{P}_5 \,\square\, \mathcal{P}_5$	25	9	56 630 783	61 212.33	5	9	1 511 009	10 101.53
Grids on the torus	$\mathcal{C}_2 \,\square\, \mathcal{C}_3$	6	2	57	0.03	2	2	43	0.01
	$\mathcal{C}_2 \,\square\, \mathcal{C}_4$	8	4	343	0.18	2	4	289	0.19
	$\mathcal{C}_2 \,\square\, \mathcal{C}_5$	10	4	1 373	0.81	3	4	717	0.53
	$\mathcal{C}_2 \,\square\, \mathcal{C}_6$	12	5	5 805	3.91	3	5	2 497	3.16
	$\mathcal{C}_2 \,\square\, \mathcal{C}_7$	14	6	23 817	17.57	4	6	7 947	12.25
	$\mathcal{C}_2 \,\square\, \mathcal{C}_8$	16	6	98 325	78.79	4	6	13 475	23.24
	$\mathcal{C}_2 \,\square\, \mathcal{C}_9$	18	7	405 101	352.78	5	7	58 979	147.63
	$\mathcal{C}_2 \,\square\, \mathcal{C}_{10}$	20	8	1 665 483	1 577.66	5	8	141 643	468.39
	$\mathcal{C}_2 \,\square\, \mathcal{C}_{11}$	22	9	6 833 333	7 020.25	6	9	551 461	2 294.11
	$\mathcal{C}_2 \,\square\, \mathcal{C}_{12}$	24	10	27 954 993	31 173.99	6	10	1 531 511	9 057.62
	$\mathcal{C}_3 \,\square\, \mathcal{C}_3$	9	3	417	0.25	2	3	231	0.11
	$\mathcal{C}_3 \,\square\, \mathcal{C}_4$	12	4	4 131	2.70	3	4	1 687	1.56
	$\mathcal{C}_3 \,\square\, \mathcal{C}_5$	15	5	33 189	25.17	3	5	7 275	11.10
	$\mathcal{C}_3 \,\square\, \mathcal{C}_6$	18	6	277 793	239.87	4	6	36 361	81.38
	$\mathcal{C}_3 \,\square\, \mathcal{C}_7$	21	7	2 334 103	2 238.20	5	7	184 881	573.56
	$\mathcal{C}_3 \,\square\, \mathcal{C}_8$	24	8	19 536 193	20 724.33	5	8	719 527	3 326.37
	$\mathcal{C}_4 \,\square\, \mathcal{C}_4$	16	6	81 127	68.51	4	6	19 121	35.87
	$\mathcal{C}_4 \,\square\, \mathcal{C}_5$	20	7	1 370 737	1 390.53	4	7	114 751	362.28
	$\mathcal{C}_4 \,\square\, \mathcal{C}_6$	24	8	23 233 767	26 887.09	5	8	715 993	3 349.03
	$\mathcal{C}_5 \,\square\, \mathcal{C}_5$	25	9	46 105 853	52 294.60	5	9	1 940 223	11 167.51

Table 1: Results obtained by means of the algorithms in §4 and §5 for all grid graphs in the plane and grid graphs on the torus of orders not exceeding $n = 25$. The columns labelled LB and UB contain the initial values of the global variables LowerBound and UpperBound (as described in §5) for the branch-and-bound algorithm.

whether or not a given subset of the vertex set of a graph securely dominates the graph. Our first algorithm follows a branch-and-reduce approach and was presented and illustrated by means of an example in §4. Our second algorithm follows a classical branch-and-bound approach and was presented and illustrated by means of the same example in §5. The worst-case time complexities of the algorithms were noted in §6 and both algorithms were

Graph	n	$\|R\|$	γ_s	BR (§4) Calls	BR (§4) Time	BB (§5) LB	BB (§5) UB	BB (§5) Calls	BB (§5) Time
$C_3\langle 1\rangle$	3	1	1	3	0.01	1	1	3	0.01
$C_4\langle 1\rangle$	4	0	2	21	0.01	2	2	21	0.01
$C_4\langle 1,2\rangle$	4	1	1	3	0.01	1	2	3	0.01
$C_5\langle 1\rangle$	5	0	3	51	0.03	2	3	51	0.02
$C_5\langle 1,2\rangle$	5	1	1	3	0.01	1	1	3	0.01
$C_6\langle 1\rangle$	6	0	3	95	0.04	2	3	83	0.04
$C_6\langle 1,2\rangle$	6	0	2	43	0.02	2	2	43	0.02
$C_6\langle 1,3\rangle$	6	0	3	85	0.04	2	3	83	0.04
$C_6\langle 2,3\rangle$	6	0	2	57	0.03	2	2	43	0.02
$C_6\langle 1,2,3\rangle$	6	1	1	3	0.01	1	1	3	0.01
$C_7\langle 1\rangle$	7	0	3	199	0.09	3	3	149	0.07
$C_7\langle 1,2\rangle$	7	0	2	109	0.07	2	3	75	0.03
$C_7\langle 1,2,3\rangle$	7	1	1	3	0.01	1	1	3	0.01
$C_8\langle 1\rangle$	8	0	4	411	0.21	3	4	285	0.17
$C_8\langle 1,2\rangle$	8	0	3	217	0.14	2	3	189	0.11
$C_8\langle 1,3\rangle$	8	0	4	345	0.16	2	4	319	0.24
$C_8\langle 1,4\rangle$	8	0	3	317	0.18	2	3	199	0.11
$C_8\langle 1,2,3\rangle$	8	0	2	73	0.05	2	2	73	0.03
$C_8\langle 1,2,4\rangle$	8	0	2	117	0.08	2	3	73	0.03
$C_8\langle 1,3,4\rangle$	8	0	2	87	0.06	2	2	73	0.03
$C_8\langle 1,2,3,4\rangle$	8	1	1	3	0.00	1	1	3	0.02
$C_9\langle 1\rangle$	9	0	4	851	0.47	3	4	459	0.29
$C_9\langle 1,2\rangle$	9	0	3	367	0.21	2	3	265	0.15
$C_9\langle 1,3\rangle$	9	0	3	465	0.26	2	3	233	0.11
$C_9\langle 1,2,3\rangle$	9	0	2	191	0.15	2	3	117	0.06
$C_9\langle 1,2,4\rangle$	9	0	3	291	0.15	2	3	259	0.17
$C_9\langle 1,2,3,4\rangle$	9	1	1	3	0.01	1	1	3	0.01
$C_{10}\langle 1\rangle$	10	0	5	1 731	1.00	4	5	1 133	0.91
$C_{10}\langle 1,2\rangle$	10	0	3	757	0.45	2	3	385	0.24
$C_{10}\langle 1,3\rangle$	10	0	4	1 233	0.76	2	4	687	0.54
$C_{10}\langle 1,4\rangle$	10	0	4	1 169	0.63	2	4	673	0.55
$C_{10}\langle 1,5\rangle$	10	0	4	1 343	0.81	3	4	715	0.51
$C_{10}\langle 2,5\rangle$	10	0	4	1 323	0.78	3	4	727	0.54
$C_{10}\langle 1,2,3\rangle$	10	0	2	413	0.31	2	3	175	0.10
$C_{10}\langle 1,2,4\rangle$	10	0	2	249	0.17	2	3	111	0.05
$C_{10}\langle 1,2,5\rangle$	10	0	3	779	0.46	2	3	399	0.26
$C_{10}\langle 1,3,5\rangle$	10	0	4	1 253	0.64	2	4	751	0.67
$C_{10}\langle 1,4,5\rangle$	10	0	3	409	0.31	2	3	337	0.25
$C_{10}\langle 2,4,5\rangle$	10	0	2	289	0.18	2	2	111	0.04
$C_{10}\langle 1,2,3,4\rangle$	10	0	2	111	0.10	2	2	111	0.05
$C_{10}\langle 1,2,3,5\rangle$	10	0	2	211	0.18	2	3	111	0.05
$C_{10}\langle 1,2,4,5\rangle$	10	0	2	211	0.19	2	3	111	0.05
$C_{10}\langle 1,2,3,4,5\rangle$	10	1	1	3	0.01	1	1	3	0.01
$C_{11}\langle 1\rangle$	11	0	5	3 529	2.14	4	5	1 627	1.35
$C_{11}\langle 1,2\rangle$	11	0	3	1 537	1.01	3	3	669	0.47
$C_{11}\langle 1,3\rangle$	11	0	4	2 301	1.45	3	4	1 107	0.93
$C_{11}\langle 1,2,3\rangle$	11	0	3	711	0.53	2	3	481	0.35
$C_{11}\langle 1,2,4\rangle$	11	0	3	1 129	0.70	2	3	467	0.30
$C_{11}\langle 1,2,3,4\rangle$	11	0	2	297	0.27	2	3	167	0.09
$C_{11}\langle 1,2,3,4,5\rangle$	11	1	1	3	0.01	1	1	3	0.02

Table 2: Results obtained by means of the algorithms in §4 (BR) and §5 (BB) for all non-isomorphic, connected circulants of orders not exceeding $n = 11$. The columns labelled LB and UB contain the initial values of the global variables LowerBound and UpperBound (as described in §5) for the branch-and-bound algorithm. The column labelled $|R|$ contains the number of redundancy classes in each graph.

applied to three classes of graphs in §7: small grid graphs in the plane, small grid graphs on the torus, and small connected circulant graphs.

Further, related work, may include constructing a linear algorithm for the secure domination number of a tree, similar to that by Cockayne *et al.* [7] for the domination number of a tree.

Acknowledgements

The research towards this paper was supported financially by the South African National Research Council under grant numbers 77248 (first author), 81558 (second author) and 70593 (last author).

References

[1] Canonial, *Ubuntu Edition: Version 10.04* (2004), [Online], [Cited July 20th, 2012], Available from `http://www.canonical.com/`.

[2] A.P. Burger, E.J. Cockayne, W.R. Gründlingh, C.M. Mynhardt, J.H. van Vuuren & W. Winterbach, *Finite order domination in graphs.* Journal of Combinatorial Mathematics and Combinatorial Computing, 49(2004), 159–175.

[3] A.P. Burger, E.J. Cockayne, W.R. Gründlingh, C.M. Mynhardt, J.H. van Vuuren & W. Winterbach, *Infinite order domination in graphs.* Journal of Combinatorial Mathematics and Combinatorial Computing, 50(2004), 179–194.

[4] A.P. Burger, M.A. Henning & J.H. van Vuuren, *Vertex covers and secure domination in graphs.* Quaestiones Mathematicae, 31(2008), 163–171.

[5] E.J. Cockayne, *Irredundance, secure domination and maximum degree in trees.* Discrete Mathematics, 307(2007), 12–17.

[6] E.J. Cockayne, O. Favaron & C.M. Mynhardt, *Secure domination, weak roman domination and forbidden subgraphs.* Bulletin of the Institute of Combinatorics and its Applications, 39(2003), 87–100.

[7] E.J. Cockayne, S. Goodman & S. Hedetniemi, *A linear algorithm for the domination number of a tree.* Information Processing Letters, 4(1975), 41–44.

[8] E.J. Cockayne, P.J.P. Grobler, W.R. Gründlingh, J. Munganga & J.H. van Vuuren, *Protection of a graph.* Utilitas Mathematica, 67(2005), 19–32.

[9] M.R. Garey & D.S. Johnson, *Computers and intractability: A guide to the theory of NP-completeness* (Freeman, New York 1979).

[10] P.J.P. Grobler & C.M. Mynhardt, *Secure domination critical graphs.* Discrete Mathematics, 309(2009), 5820–5827.

[11] T.W. Haynes, S.T. Hedetniemi & P.J. Slater, *Fundamentals of domination in graphs* (Marcel Dekker, New York, 1998).

[12] C.M. Mynhardt, H.C. Swart & E. Ungerer, *Excellent trees and secure domination.* Utilitas Mathematica, 67(2005), 255–267.

[13] Wolfram Research, *Mathematica Edition: Version 8.0* (2005), [Online], [Cited May 30th 2012], Available from `http://www.wolfram.com/mathematica/`.