AKCE J. Graphs. Combin., 6, No. 3 (2009), pp. 373-392

Rooted Secure Sets of Trees

Yiu Yu Ho and Ronald Dutton
Department of Computer Science
University of Central Florida
Orlando, FL 32816, USA
e-mail: *yiuyuho@gmail.com, dutton@cs.ucf.edu*

Communicated by: T.W. Haynes
Received 19 May 2009; accepted 14 September 2009

Abstract

Let $G = (V, E)$ be a graph. A set $S \subseteq V$ is a *defensive alliance* if for all $x \in S$, $|N[x] \cap S| \geq |N[x] - S|$. Thus, each vertex of a defensive alliance can, with the aid of its neighbors in S, be defended from attacks by its neighbors outside of S. A set S is a secure set if every subset $X \subseteq S$ can be defended from attacks by vertices outside of S, under an appropriate definition of such attacks and defenses. Given G and $x \in V$, a secure set *rooted* at x is a secure set that contains x. Polynomial algorithms for computing the cardinality of a minimum rooted secure set of trees are presented.

Keywords: graph algorithms, defensive alliance, secure set, tree.

2000 Mathematics Subject Classification: 05C02, 68R10.

1. Introduction

Secure sets in graphs is a concept introduced by Brigham et al. [4] as a generalization of defensive alliances [3, 12, 13, 14]. Let $G = (V, E)$ be a graph. If $x \in V$, then $N(x) = \{y : xy \in E\}$ and $N[x] = N(x) \cup \{x\}$ are the *open* and *closed neighborhood* of x, respectively. For $S \subseteq V$, $N(S) = \bigcup_{v \in S} N(v)$ and $N[S] = N(S) \cup S$ are the open and closed neighborhood of S, respectively. The set $N[S] - S$ is the *boundary* of S. The maximum degree among the vertices of G is denoted by $\Delta(G)$. For simplicity, we write Δ when the graph G in question is clear. The subgraph of G induced by S is the graph with vertex set S and edge set $E \cap (S \times S)$, and is denoted by $G[S]$. Terms and notations not introduced in this paper follow [10] or [16].

A *defensive alliance* is a subset S of V such that for all $x \in S$, $|N[x] \cap S| \geq |N[x] - S|$. One can think of the vertices in $N[x] - S$ as *attackers* of x and those in $N[x] \cap S$ as *defenders* of x. Thus, any vertex $x \in S$ has as many defenders as attackers; and any attack on x can be defended. In a secure set, multiple attacks on vertices in S can be simultaneously defended. There are several ways one might define what this means. The definition introduced below follows from [4]. At any moment, each vertex $y \in N[S] - S$ can

choose to attack one vertex in $N[y] \cap S$. Given the attack choices for every $y \in N[S] - S$, each vertex $x \in S$ may choose to defend one vertex in $N[x] \cap S$, in a way such that there are at least as many defenders as attackers at each $v \in S$. The formal definition follows.

Definition 1.

1. *Let $G = (V, E)$ be a graph. For any $S = \{s_1, s_2, \ldots, s_k\} \subseteq V$, an* attack *on S is any k mutually disjoint sets $A = \{A_1, A_2, \ldots, A_k\}$ for which $A_i \subseteq N[s_i] - S$, $1 \leq i \leq k$. If $y \in A_i$, then y* attacks *s_i. A* defense *of S is any k mutually disjoint sets $D = \{D_1, D_2, \ldots, D_k\}$ for which $D_i \subseteq N[s_i] \cap S$, $1 \leq i \leq k$. If $x \in D_i$, then x* defends *s_i.*

2. *An attack A is* defendable *if there exists a defense D such that $|D_i| \geq |A_i|$ for all $1 \leq i \leq k$.*

3. *A set S is* secure *if every attack on S is defendable.*

The cardinality of a minimum secure set of G is the *security number* of G and is denoted $s(G)$. We are interested in the following problems.

Problem 2. Secure Set

***Given**: A graph $G = (V, E)$ and an integer k.*

***Question**: Does there exist a secure set $S \subseteq V$ of size k or less?*

In [4], [7] and [8], the value of $s(G)$ was provided for several families of graphs. Bounds on $s(G)$ were given for general graphs based on other invariants of G such as the degree sequence, minimum degree, connectivity and domination number. In [6], exhaustive search algorithms and heuristics for finding $s(G)$ are presented, but no polynomial algorithm is known for computing $s(G)$ for general graphs. In fact, there is no known polynomial algorithm for determining whether a given set is secure.

Problem 3. Is Secure

***Given**: A graph $G = (V, E)$ and a subset $S \subseteq V$.*

***Question**: Is S a secure set of G?*

It is possible, with a transformation from Dominating Set (cf.[9], [GT2], p.190), to show that the complement of the Is Secure problem is NP-complete. Therefore, the Secure Set problem may not be in NP, if $P \neq NP$. For clarification of the theory of NP-completeness and related concepts, refer to the classic work of Garey and Johnson [9].

Many NP-complete problems are known to have linear time solutions when restricted to special classes of graphs, such as trees, partial k-trees and series parallel graphs ([1, 2, 11, 15, 18, 17]). A natural question is then whether one can compute the value of $s(G)$ in polynomial time on these graphs. This is interesting because the Secure Set problem is *probably not* in the class NP.

In this paper, we address the problem of computing secure sets of trees. In particular, we develop an $O(n \lg(\Delta))$ algorithm for the Rooted Secure Set problem, defined below.

2. Rooted Secure Sets in Trees

A degree one vertex, that is, a *leaf* in a graph G always forms a secure set. This makes the computation of $s(G)$ on a tree trivial, since $s(T) = 1$ for any tree T. If the vertices of a graph represent a set of entities, such as countries or companies, it is interesting to ask for a smallest secure set that contains a given country or company. In graph theoretical terms, what is a smallest secure set S_r that contains a particular vertex $r \in V(G)$?

Problem 4. Rooted Secure Set

***Given**: A graph $G = (V, E)$, a vertex $r \in V$ and an integer k.*

***Question**: Is there a secure set S_r containing r of size k or less?*

We call r the *root* of S_r. If a polynomial algorithm exists for the Rooted Secure Set problem, then one exists for the Secure Set problem. One such algorithm takes the minimum over secure sets rooted at each vertex. Note that the cardinality of a rooted secure set S_r is equal to one only if r is a leaf.

In order to show that a minimum cardinality rooted secure set S_r can be found in polynomial time, let us first examine some of the properties of such a set. Let r be a vertex in a tree T that is to be the root of a secure set S_r, and let T_r denote the tree T rooted at vertex r. For any other vertex $v \in V(T)$, let T_v be the subtree of T_r rooted at v and let p_v denote the parent of v in T_r. Let c_v denote the number of children of v in T_v. Notice that any minimum cardinality secure set S_r must induce a subtree $T[S_r]$ of T_r. A minimum secure set of T that contains r is connected, for otherwise the maximal connected subset that contains r forms a smaller secure set.

Definition 5. *Let $G = (V, E)$ be a graph and $S \subseteq V$. An attack A is* maximum *if $\bigcup_{A_i \in A} A_i = N[S] - S$. That is, an attack is maximum when every vertex in $N[S] - S$ attacks some vertex in S.*

Clearly, a set S is secure if and only if every maximum attack on S is defendable.

Lemma 6. *Let T be a tree and $S \subseteq V(T)$ such that $T[S]$ is a subtree of T. Then, $|N[y] \cap S| = 1$ for all $y \in N[S] - S$.*

Proof. Let $y \in N[S] - S$ and assume that $|N[y] \cap S| > 1$. Let u and v be vertices in $N[y] \cap S$. Since $T[S]$ is a subtree of T, there exists a path from u to v in $T[S]$ and a second path u, y, v. Thus T contains a cycle, contradicting the assumption that T is a tree. □

Lemma 6 shows that every vertex in $N[S] - S$ has exactly one neighbor in S. Thus, in a maximum attack on S, every vertex $y \in N[S] - S$ can only attack the one neighbor it

has in S. This implies that there is a *unique* maximum attack on S. Therefore, a set of vertices S of a subtree $T[S]$ is secure if and only if the unique maximum attack on S is defendable. In [4] it was shown that the problem of deciding if a given attack A on a set S is defendable can be decided in polynomial time. As a result, the Is Secure problem can be solved in polynomial time when restricted to trees. Therefore, the Rooted Secure Set problem when restricted to trees is in the class NP.

Definition 7. *A secure set S is* critical *if no proper subset of S is secure.*

Lemma 8. *Let T_r be a rooted tree and let S_r be a rooted secure set of T_r. Let A be the maximum attack on S_r and consider any vertex $v \in S_r - \{r\}$. If there is a defense D of A such that p_v defends v, then S_r is not critical.*

Proof. We show that S_r is not critical by showing that $S'_r = S_r - V(T_v)$ is a rooted secure set containing r. Let D be a defense of A in which p_v defends v. Let A' be the unique maximum attack on S'_r. Note that $A'_x = A_x$ for $x \in S'_r - \{p_v\}$ and $A'_{p_v} = A_{p_v} \cup \{v\}$. Consider a defense D' where $D'_x = D_x$ for $x \in S'_r - \{p_v\}$ and $D'_{p_v} = D_{p_v} \cup \{p_v\}$ ($p_v \in D_v$, which no longer requires defending since $v \notin S'_r$). D' is a defense of A', so S'_r is secure and S_r is not critical. □

By Lemma 8, if S is a minimum secure set, then S is critical, and in every defense D of A, p_v does not defend v for all $v \in S - \{r\}$. In other words, *a vertex in a minimum rooted secure set either defends itself or its parent, but never any of its children.*

An algorithm to calculate the cardinality of a minimum rooted secure set of a tree is presented next. We will employ the Wimer Method ([18, 17]). The following is a brief overview of tree compositions as used in the Wimer Method.

Definition 9. Rooted Trees

1. *The triple $(\{x\}, \emptyset, x)$ is a rooted tree with root x.*
2. *If $T_1 = (V_1, E_1, r_1)$ and $T_2 = (V_2, E_2, r_2)$ are rooted trees with roots r_1 and r_2 respectively, then $T = T_1 \circ T_2 = (V_1 \cup V_2, E_1 \cup E_2 \cup \{r_1 r_2\}, r_1)$ is a rooted tree with root r_1.*
3. *Nothing is a rooted tree unless it is obtained by a finite number of applications of 1 or 2.*

Rule 1 states that a vertex is a rooted tree. This is the smallest rooted tree. Rule 2 describes a tree composition, whereby a new rooted tree is constructed from two rooted trees T_1, r_1 and T_2, r_2 by adding an edge between r_1 and r_2 and selecting r_1 as the root of the resulting tree. This operation is a binary operation and is denoted by the $\circ$ operator. Note that $\circ$ is not a commutative operation.

In the algorithm that follows, let S be a minimum rooted secure set containing the root r of the rooted tree T_r. Let $v \in V(T)$ be arbitrary, let $S_v = S \cap V(T_v)$ and $U = \{u_1, u_2, \ldots, u_k\}$ be the children of v. Recall that c_v denotes the number of children of v (i.e., $c_v = k$, we use them interchangeably for convenience). Consider the subtree T_v, and associate with it an array of integer values $\alpha(T_v) = \{\alpha_j(T_v) : -\lfloor \frac{c_v}{2} \rfloor - 1 \leq j \leq \lceil \frac{c_v}{2} \rceil + 1\}$. Recall from Definition 1 that for $x \in S$, A_x denotes the set of attackers of x, and D_x denotes the set of defenders of x, when such an attack or defense is given. The entry $\alpha_j(T_v)$ is an integer representing the cardinality of a minimum set S_v such that

(a) For $x \in S_v - \{v\}$, $|A_x| \leq |D_x|$, and

(b) $|A_v \cap U| \leq |D_v \cap U| + j$. That is, among the children of v, the difference between the number of attackers of v and the number of defenders of v is at most j, where j ranges from $-\lfloor \frac{c_v}{2} \rfloor - 1$ to $\lceil \frac{c_v}{2} \rceil + 1$, and

(c) A is the unique maximum attack on S and D is a defense of A.

Note that j may be negative. To compute $\alpha(T_v)$, initialize $T_1 = (\{v\}, \emptyset, v)$, with associated values of $\alpha(T_1)$. Then, let T_2 be $T_{u_1}, T_{u_2}, \ldots, T_{u_k}$ in sequence and compute $\alpha(T_2)$ (or $\alpha(T_{u_i}), 1 \leq i \leq k$) recursively. Next, apply tree composition on T_1 and T_2, and let this result be the new T_1 ($T_1 \leftarrow T_1 \circ T_2$). Then, compute entries for the new $\alpha(T_1)$. One may consider each tree composition as attaching a child subtree T_{u_i} of v onto T_1, which is rooted at v. When every child subtree of v is attached, T_1 is transformed into T_v, and $\alpha(T_1)$ also contains the desired values of $\alpha(T_v)$.

We now discuss initial (base case) values for $\alpha(T_1)$, as well as how to compute the new values of $\alpha(T_1)$, given its old values and $\alpha(T_2)$. Initially, T_1 is a leaf node that contains v. Since v has no children, the set $S_v = \{v\}$ is a valid configuration only if $j \geq 0$, in which case $|S_v| = 1$. The initial values of $\alpha_j(T_1)$ are then

$$\alpha_j(T_1) = \begin{cases} 1 & \text{if } 0 \leq j \leq \lceil \frac{c_v}{2} \rceil + 1 \\ \infty & \text{if } -\lfloor \frac{c_v}{2} \rfloor - 1 \leq j \leq -1 \end{cases}$$

Let $T_2 = T_{u_i}$ be an arbitrary child subtree of v. Let $\alpha(T_2)$ be given and let $r_2 = u_i$ be the root of T_2. Consider $T_{12} = T_1 \circ T_2$. When combining the trees T_1 and T_2, we must decide on the *role* of r_2 in this new tree T_{12}. There are 3 cases to be considered.

1. $r_2 \in S_v$ and r_2 defends v. In this case, $S_2 \subset S_v$ and $|S_2| = \alpha_0(T_2)$. In other words, among the children of r_2, the number of attackers of r_2 is at most the number of defenders, since r_2 defends v and not itself.

2. $r_2 \in S_v$ and r_2 defends itself. In this case r_2 is included in S_v so that it does not attack v. That is, r_2 is a neutral vertex with respect to v. Then, $S_2 \subset S_v$ and

$|S_2| = \alpha_1(T_2)$. In other words, among the children of r_2, there may be one more attackers than defenders, accounting for the fact that r_2 is an additional defender of itself.

3. $r_2 \notin S_v$. Here, r_2 is an attacker of v and $S_2 = \emptyset$.

The associated values $\alpha_j(T_{12})$ can be computed as follows.

$$\alpha_j(T_{12}) = \min \begin{cases} \alpha_{j+1}(T_1) + \alpha_0(T_2) & \text{(Type 1)} \\ \alpha_j(T_1) + \alpha_1(T_2) & \text{(Type 2)} \\ \alpha_{j-1}(T_1) & \text{(Type 3)} \end{cases}$$

The cardinality of a minimum secure set that contains r is $\alpha_1(T_r)$, since r has no parent and thus must defend itself. Note that during the construction of $\alpha(T_v)$, for each $u_i \in U$ we only utilized $\alpha_0(u_i)$ and $\alpha_1(u_i)$. The cardinality of a minimum rooted secure set is $\alpha_1(r)$. Thus, an algorithm does not need to compute all entries of $\alpha(T_v)$, but only $\{\alpha_0(T_v), \alpha_1(T_v)\}$. Next, we present the pseudocode of the algorithm, follows with a justification of the range $(-\lfloor \frac{c_v}{2} \rfloor - 1$ to $\lceil \frac{c_v}{2} \rceil + 1)$ of $\alpha_j(T_v)$ used in the algorithm.

Input: A rooted tree T_v.

Output: $\alpha_0(T_v)$ and $\alpha_1(T_v)$ as defined above.

Algorithm 10.

RootedSecure(T_v)

1. *Initialize* $\alpha_j(T_1) = \begin{cases} 1 & \text{if } 0 \le j \le \lceil \frac{c_v}{2} \rceil + 1 \\ \infty & \text{if } -\lfloor \frac{c_v}{2} \rfloor - 1 \le j \le -1 \end{cases}$
2. *For* $i = 1$ *to* c_v
    1. $\{\alpha_0(T_2), \alpha_1(T_2)\} \leftarrow$ *RootedSecure(T_{u_i})*
    2. *For* $j = \max\{i - c_v, -i\}$ *to* $\min\{c_v + 1 - i, i\}$
       $\alpha_j(T_{12}) \leftarrow \min\{\alpha_{j+1}(T_1) + \alpha_0(T_2), \alpha_j(T_1) + \alpha_1(T_2), \alpha_{j-1}(T_1)\}$
    3. *For* $j = \max\{i - c_v, -i\}$ *to* $\min\{c_v + 1 - i, i\}$
       $\alpha_j(T_1) \leftarrow \alpha_j(T_{12})$
3. *Return* $\{\alpha_0(T_1), \alpha_1(T_1)\}$

As aforementioned, the algorithm only needs to return the values of $\alpha_0(T_v)$ and $\alpha_1(T_v)$. However, the entry of $\alpha_j(T_1)$ needs to be computed for other values of j as an intermediate step of the algorithm, as each child of v is attached. Consider Figure 1, for $\alpha_0(T_v)$,

$S_v = \{v, u_3, u_4\}$. As u_1 and u_2 are attached, both subtrees belong to Type 3 of the recurrence, and the instance of T_1 after T_{u_1} and T_{u_2} are attached to v looks like that of Figure 2. The set marked in Figure 2 is not a valid α_0 or α_1 configuration, but it must be computed since it will lead to a valid α_0 or α_1 configuration of T_v. In this case, we store the partial tree in Figure 2 as $\alpha_{-2}(T_1)$ and it will lead to a solution of $\alpha_0(T_v)$ eventually, as shown in Figure 1. This is the reason that other values of j, besides $j \in \{0, 1\}$, are maintained by the algorithm.

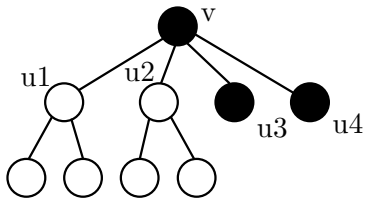

Figure 1: A rooted secure set

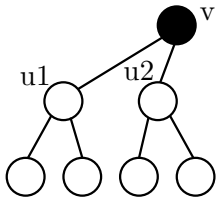

Figure 2: A partial solution

It remains to ensure that the ranges (range for initialization of $\alpha_j(T_1)$, as well as the ranges of for-loops in step 2.2 and 2.3) are exactly what is needed in order to correctly compute $\alpha_0(T_v)$ and $\alpha_1(T_v)$. As child i is attached to v, there can be at most i attackers and at most i defenders of v, thus we may bound the for loop in step 2.2 and 2.3 by the range $[-i, i]$. In addition, note that the value of $\alpha_j(T_{12})$ depends on $\alpha_{j-1}(T_1)$, $\alpha_j(T_1)$ and $\alpha_{j+1}(T_1)$. As shown in Figure 3, $\alpha_0(T_v)$ and $\alpha_1(T_v)$ (on row c_v, column 0 and 1) depends on the cells marked with left diagonal shade, and the cells marked with right diagonal shades are the non-trivial values for α_j. Thus, the algorithm proceeds and computes exactly those cells with cross shade, which are the cells with non-trivial values and affect results $\alpha_0(T_v)$ or $\alpha_1(T_v)$. The range for initialization is then derived from the ranges of loops and the j-indices being referenced: $-\left\lfloor \frac{c_v}{2} \right\rfloor - 1 \leq j - 1$ and $j + 1 \leq \left\lceil \frac{c_v}{2} \right\rceil + 1$.

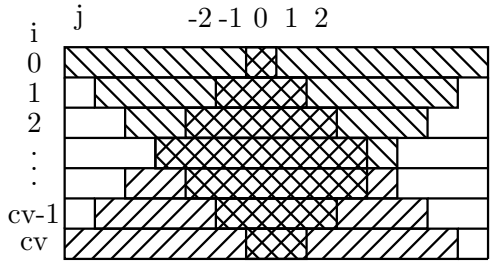

Figure 3: Solution Table

The above discussion verifies the correctness of Algorithm 10.

Lemma 11. *Algorithm 10 has time complexity* $O(\Delta n)$.

Proof. For each vertex $v \in V(T)$, the algorithm is called with T_v. Step 2 is executed c_v times, once for each child of v. Step 2.2 and 2.3 are executed at most $c_v + 3$ times each for every iteration of step 2. There are, in total, $O((c_v)^2)$ operations for each vertex $v \in V(T)$. The total number of operations required for solving T_r is then proportional to $\sum_{v \in T}(c_v)^2 \leq \sum_{v \in T}(c_v \times \Delta) = (n-1) \times \Delta \in O(\Delta n)$, where Δ is the maximum degree of T_r. □

In the next section, we describe an $O(n \lg(\Delta))$ algorithm, using a strategy that deviates from the Wimer Method.

3. An $O(n \lg(\Delta))$ Algorithm

Algorithm 10 uses the Wimer Method, which calculates the result of the subtree T_v by appending the subtrees of v, $\{T_{u_1}, T_{u_2}, \ldots, T_{u_k}\}$, to v one at a time. An alternate strategy, which we describe below, constructs the result for T_v by considering the results of $\{T_{u_1}, T_{u_2}, \ldots, T_{u_k}\}$ all at once. This requires more extensive analysis and sophisticated data structures, but results in an $O(n \lg(\Delta))$ algorithm.

Let v be the root of tree T_v and $U = \{u_1, u_2, \ldots, u_k\}$ be the children of v. As shown in Section 2, there are three cases to be considered for each child u_i of v. Among the three choices, at least one will result in an optimal solution for S_v. Let D, N, A be a partition of U and defined as follows.

Definition 12.

1. $D = \{u_i : u_i \in S_v$ *and* u_i *defends* $v\}$
2. $N = \{u_i : u_i \in S_v$ *and* u_i *defends itself* $\}$

3. $A = \{u_i : u_i \notin S_v\}$

The partition is defined such that D contains exactly the vertices that are defending v, N contains exactly the vertices that are neutral with respect to v (these vertices are included in S_v so that they will not attack v), and A contains exactly the vertices that are attacking v (these vertices are not in S_v).

An optimal solution for $|S_v|$ is a partition (D, N, A) of $\{u_1, u_2, \ldots, u_k\}$, where

$$|S_v| = 1 + \sum_{u \in D} \alpha_0(T_u) + \sum_{u \in N} \alpha_1(T_u)$$

is minimum. For $\alpha_0(T_v)$, a partition with $|A| = |D|$ is required; for $\alpha_1(T_v)$, a partition with $|A| = |D| + 1$ is required. In the following, we will explore how to compute $\alpha_0(T_v)$. That is, how to efficiently obtain an optimal partition such that $|A| = |D|$. The computation of $\alpha_1(T_v)$ follows in a similar manner.

Definition 13. *A partition* (D, N, A) *of* $\{u_1, \ldots, u_k\}$ *is* feasible *if* $|A| = |D|$. *The* value *of the partition is*

$$1 + \sum_{u \in D} \alpha_0(T_u) + \sum_{u \in N} \alpha_1(T_u)$$

A partition is optimal *if it is feasible and its value is minimum.*

The following six *elementary rules* (E-rules) can be used to transform one partition (D, N, A) into another.

Definition 14. Elementary Rule Set

1. $E_1 : A \to D$. *Move an element from* A *to* D. *That is, let* $x \in A$ *and modify* $A \leftarrow A - \{x\}$ *and* $D \leftarrow D \cup \{x\}$. *Rules* $E_2, \ldots, E_6$ *are defined similarly.*
2. $E_2 : D \to A$.
3. $E_3 : A \to N$.
4. $E_4 : N \to A$.
5. $E_5 : D \to N$.
6. $E_6 : N \to D$.

The E-rules allow any element from D, N or A to be moved to any other set. Using the E-rules, we can transform any partition into any other one. Hence, given any feasible partition and an optimal solution, we may apply the E-rules to transform the feasible

partition into the optimal partition. This can be done by moving any element that is misplaced, and put it in the correct set, using the optimal partition as a reference. Note that each element in $\{u_1, \ldots, u_k\}$ is moved at most once, and as a result the order of applications is irrelevant.

Since the order of applications is irrelevant, the E-rules can be grouped together into several groups such that each group of E-rules transform one feasible partition into another, improved, feasible partition. In Definition 15, we provide a set of these groups and follow with a justification that these are sufficient for transforming any feasible partition into a known optimal solution. Then, when the optimal solution is *not* given, the difficulty is determining which of the rules should be used, to which elements, and in which order. We will later describe how these rules may be applied to an arbitrary feasible partition and produce an optimal partition, without pre-knowledge of an optimal solution.

Definition 15. Feasibility Preserving Rule Set

1. $R_1 : D \to A, A \to D$. *Exchange elements between* A *and* D. *Let* $x \in A, y \in D$ *and modify* $A \leftarrow A - \{x\} \cup \{y\}$, $D \leftarrow D - \{y\} \cup \{x\}$. *The other* R*-rules are defined similarly.*
2. $R_2 : D \to N, N \to D$.
3. $R_3 : N \to A, A \to N$.
4. $R_4 : A \to N, D \to N$.
5. $R_5 : N \to A, N \to D$.
6. $R_6 : A \to D, D \to N, N \to A$.
7. $R_7 : A \to N, N \to D, D \to A$.
8. $R_8 : A \to D, N \to A, N \to A$.
9. $R_9 : D \to A, N \to D, N \to D$.
10. $R_{10} : A \to D, D \to N, D \to N$.
11. $R_{11} : D \to A, A \to N, A \to N$.

Note that, for example, in R_8, *two* distinct *elements are moved from* $N \to A$. *The case is similar for* R_9, R_{10} *and* R_{11}. *In* R_5, *the element that moves from* $N \to A$ *must be different from the one moving from* $N \to D$.

Each R-rule is composed of 2 or 3 E-rules. For example, R_1 is E_2 followed by E_1. In addition, if (D, N, A) is a feasible partition, then (D, N, A) remains feasible after any number of applications of R_1 through R_{11}.

In what follows, let (D, N, A) be an arbitrary feasible partition and (D_o, N_o, A_o) be an optimal one. Our next goal is to show that starting from any feasible partition (D, N, A), there exists a sequence of R-rule applications that transforms (D, N, A) into an optimal partition, given pre-knowledge of such a partition, namely (D_o, N_o, A_o). This will be done in Theorem 21. First, we introduce some definitions and three lemmas (18, 19 and 20) needed in the proof of Theorem 21. These lemmas will help determine when a sequence of rule applications has resulted in an optimal partition.

Definition 16. *Let (D, N, A) and (D_o, N_o, A_o) be given as described above. Let X_Y denote the set of elements that must be moved from set X to set Y, for $X, Y \in \{D, N, A\}$, in order to transform (D, N, A) into (D_o, N_o, A_o). Elements that do not need to be moved are denoted X_X. With this notation, for example, $D_A = \{u : u \in D \text{ and } u \in A_o\} = D \cap A_o$, which are the elements currently in D that must be moved to A. $D = D_D \cup D_N \cup D_A$ and $D_o = D_D \cup N_D \cup A_D$.*

Definition 17. *Let $(D, N, A) \Rightarrow^* (D', N', A')$ denote that there exists a (possibly empty) sequence of R-rule applications that can be applied to (D, N, A) and result in (D', N', A').*

Lemma 18. *Let (D, N, A) be an arbitrary feasible partition and assume $(D, N, A) \Rightarrow^* (D_t, N_t, A_t)$. Then, $(D_t, N_t, A_t) = (D_o, N_o, A_o)$ if and only if $D_t \subseteq D_o$ and $A_o \subseteq A_t$ (or $D_o \subseteq D_t$ and $A_t \subseteq A_o$).*

Proof. If $(D_t, N_t, A_t) = (D_o, N_o, A_o)$, then $D_t = D_o$, $A_t = A_o$ and the claim holds.

Suppose $D_t \subseteq D_o$ and $A_o \subseteq A_t$. $|D_o| = |A_o|$ because (D_o, N_o, A_o) is a feasible partition. $|D_t| \leq |D_o|$ and $|A_o| \leq |A_t|$ due to set inclusion. So, $|D_t| \leq |D_o| = |A_o| \leq |A_t|$. But, $|D_t| = |A_t|$ because (D_t, N_t, A_t) is also a feasible partition. Thus we have $|D_t| = |D_o| = |A_o| = |A_t|$. $D_t = D_o$ and $A_t = A_o$. Similarly, if $D_o \subseteq D_t$ and $A_t \subseteq A_o$, then $D_t = D_o$ and $A_t = A_o$. □

Lemma 19. *The following are each sufficient conditions indicating that (D, N, A) has been transformed into (D_o, N_o, A_o).*

1. *$N_D = A_D = A_N = \emptyset$, or*
2. *$D_A = D_N = N_A = \emptyset$*

Proof.

1. $N_D = A_D = \emptyset \Rightarrow N \cap D_o = A \cap D_o = \emptyset \Rightarrow D_o \subseteq D$. $A_D = A_N = \emptyset \Rightarrow A \cap D_o = A \cap N_o = \emptyset \Rightarrow A \subseteq A_o$. The conclusion follows from Lemma 18.
2. $D_A = D_N = \emptyset \Rightarrow D \cap A_o = D \cap N_o = \emptyset \Rightarrow D \subseteq D_o$. $D_A = N_A = \emptyset \Rightarrow D \cap A_o = N \cap A_o = \emptyset \Rightarrow A_o \subseteq A$. The conclusion follows from Lemma 18.

□

Lemma 20. *If $N_D = D_N = A_N = \emptyset$, then $|N_A| \neq 1$.*

Proof. Assume $N_D = D_N = A_N = \emptyset$ and $|N_A| = 1$.

Since $D_N = A_N = \emptyset$, $N_o \subseteq N$ and $N_o = N_N$. Recall that $N = N_D \cup N_N \cup N_A$, therefore

$$\begin{aligned} |N| &= |N_D \cup N_N \cup N_A| \\ &= |N_D| + |N_N| + |N_A| && ((D_o, N_o, A_o) \text{ is a partition}) \\ &= 0 + |N_o| + 1 && (N_D = \emptyset, N_o = N_N, |N_A| = 1) \\ &= |N_o| + 1 \end{aligned}$$

We have established that $|N| = |N_o| + 1$, or $|N| \not\equiv |N_o| (\text{mod } 2)$. Since (D, N, A) and (D_o, N_o, A_o) are feasible partitions of $\{u_1, \ldots, u_k\}$, $|D| + |A| + |N| = 2|D| + |N| = k$, and $k \equiv |N| (\text{mod } 2)$. On the other hand, $|D_o| + |A_o| + |N_o| = 2|D_o| + |N_o| = k$, and $k \equiv |N_o| (\text{mod } 2)$. This is a contradiction. □

Next, we show that, with perfect knowledge (pre-knowledge of an optimal partition), the R-rules are sufficient for transforming an arbitrary feasible partition into an optimal partition.

Theorem 21. *Let (D, N, A) be a feasible partition and (D_o, N_o, A_o) be an optimal one. With perfect knowledge, there exists a sequence of R-rule applications that transform (D, N, A) to (D_o, N_o, A_o). In addition, each element in $\{u_1, \ldots, u_k\}$ is moved at most once.*

Sketch of proof. We give a description of the essential idea that results in 8 exhaustive cases. We verify here only the first, since the remaining cases follow in a similar manner.

Given (D, N, A) and knowing the optimal partition (D_o, N_o, A_o), check if any R-rules may be applied on any 2 or 3 elements of $\{u_1, \ldots, u_k\}$ in a way that the rule brings the corresponding elements to the correct set with respect to (D_o, N_o, A_o). When none of the R-rules apply, we show that (D, N, A) has been transformed into (D_o, N_o, A_o) (denoted by $(D, N, A) \Rightarrow^* (D_o, N_o, A_o)$).

First, apply R_1, R_2 and R_3 whenever possible. Recall that the contents of (D, N, A) are changed as each R-rule is applied. As a result the sets X_Y for $X, Y \in \{D, N, A\}$ also change.

1. While D_A and A_D are both non-empty, apply R_1.
2. While D_N and N_D are both non-empty, apply R_2.
3. While A_N and N_A are both non-empty, apply R_3.

This results in at least one of the following 8 possible situations, depending on which of the two sets becomes empty in each of the three cases.

(1) $D_A = D_N = A_N = \emptyset$.

(2) $D_A = D_N = N_A = \emptyset$.

(3) $D_A = N_D = A_N = \emptyset$.

(4) $D_A = N_D = N_A = \emptyset$.

(5) $A_D = D_N = A_N = \emptyset$.

(6) $A_D = D_N = N_A = \emptyset$.

(7) $A_D = N_D = A_N = \emptyset$.

(8) $A_D = N_D = N_A = \emptyset$.

For Case (1), $D_A = D_N = A_N = \emptyset$. At this point, the only applicable rules are R_5 and R_8. Apply R_5 repeatedly until either $N_A = \emptyset$ or $N_D = \emptyset$.

(1) $N_A = \emptyset$. By Lemma 19 condition 2, $(D, N, A) \Rightarrow^* (D_o, N_o, A_o)$.

(2) $N_D = \emptyset$. The only applicable rule is R_8. Apply R_8 repeatedly until either $A_D = \emptyset$ or $|N_A| \leq 1$.

   (1) $A_D = \emptyset$. By Lemma 19 condition 1, $(D, N, A) \Rightarrow^* (D_o, N_o, A_o)$.

   (2) $N_A = \emptyset$. By Lemma 19 condition 2, $(D, N, A) \Rightarrow^* (D_o, N_o, A_o)$.

   (3) $|N_A| = 1$. By Lemma 20, we cannot arrive at this situation.

Cases (2) through (8) are handled in a similar way to that of Case (1). These case analyses are mechanical and will not be presented. This completes the justification of Theorem 21. □

In the justification of Theorem 21, each element is moved at most once to the correct set with respect to (D_o, N_o, A_o). As a result, the order in which the R-rules are applied is not important. Note that Theorem 21 requires knowing an optimal partition in advance and thus cannot be directly used in an algorithm that computes the optimal partition. But, we have obtained the following corollary.

Corollary 22. *Let (D, N, A) be an arbitrary partition. There exists a sequence of R-rule applications that transforms (D, N, A) to an optimal partition, such that each element of $\{u_1, \ldots, u_k\}$ is moved at most once.*

So far, we have the following results with regard to obtaining an optimal partition of $\{u_1, \ldots, u_k\}$.

1. The E-rules are given in Definition 14. If we know the content of an arbitrary feasible partition (D, N, A) and the content of an optimal partition (D_o, N_o, A_o) (we have perfect knowledge), then we can apply the E-rules, one at a time, to (D, N, A) and transform it into (D_o, N_o, A_o).
2. Every element that is misplaced in the initial partition (D, N, A) will be moved, exactly once, to the correct set in (D_o, N_o, A_o). The order of E-rules application is not important.
3. A set of R-rules are developed and given in Definition 15. The R-rules are designed in a way that applying any R-rule to a feasible partition will result in another feasible partition.
4. In Theorem 21 and Corollary 22, we established that the R-rules are sufficient to transform an arbitrary feasible partition into an optimal partition, given preknowledge of such an optimal partition. The order of applications of R-rules is not important.

In order to obtain the $O(n \lg(\Delta))$ algorithm, we will

1. Develop a set of evaluation functions for the R-rules, reflecting the change in objective function of the partition as a R-rule is applied to the partition, and
2. Show that given an arbitrary feasible partition, there is a sequence of R-rule applications such that successive applications decrease the objective function of the partition, and eventually will transform it into an optimal partition.

In the next step, we develop evaluation functions for R_1 through R_{11}.

Definition 23. *When an element is moved from X to Y, for $X, Y \in \{D, N, A\}$, the value of the partition as given in Definition 13 changes. Define $e : (X \to Y) \to \mathbb{Z}$ to be the evaluation function of a single transformation $X \to Y$ applied to (D, N, A). When applying the rule $X \to Y$, any element in X may be moved to Y, $e(X \to Y)$ is the smallest change that may result, among all choices in X. More specifically,*

1. $e(A \to D) = \min_{u \in A}\{\alpha_0(T_u)\}$
2. $e(D \to A) = \min_{u \in D}\{-\alpha_0(T_u)\}$
3. $e(A \to N) = \min_{u \in A}\{\alpha_1(T_u)\}$
4. $e(N \to A) = \min_{u \in N}\{-\alpha_1(T_u)\}$
5. $e(D \to N) = \min_{u \in D}\{-\alpha_0(T_u) + \alpha_1(T_u)\}$

6. $e(N \to D) = \min_{u \in N}\{-\alpha_1(T_u) + \alpha_0(T_u)\}$

Let $r : \{R_1, \ldots, R_{11}\} \to \mathbb{Z}$ *be the evaluation function of the set of transformations as indicated by* $R_1, \ldots, R_{11}$. *More specifically,*

1. $r(R_1) = e(D \to A) + e(A \to D)$
2. $r(R_2) = e(D \to N) + e(N \to D)$
3. $r(R_3) = e(N \to A) + e(A \to N)$
4. $r(R_4) = e(A \to N) + e(D \to N)$
5. $r(R_5) = e(N \to A) + e(N \to D)$
6. $r(R_6) = e(A \to D) + e(D \to N) + e(N \to A)$
7. $r(R_7) = e(A \to N) + e(N \to D) + e(D \to A)$
8. $r(R_8) = e(A \to D) + e(N \to A) + e(N \to A)$
9. $r(R_9) = e(D \to A) + e(N \to D) + e(N \to D)$
10. $r(R_{10}) = e(A \to D) + e(D \to N) + e(D \to N)$
11. $r(R_{11}) = e(D \to A) + e(A \to N) + e(A \to N)$

Note that the e functions, and hence r functions, are dependent only upon the current partition.

Similar to Definition 15, R_5, R_8, R_9, R_{10} and R_{11} require special treatment. For example, in R_8, two distinct elements are moved from $N \to A$. Thus, in the evaluation of $r(R_8)$, we should select an element from A and move it to D, and two *distinct* elements from N and move them to A, such that the objective function of the new partition is minimum. More precisely, $r(R_8) = \min\{\alpha_0(T_u) - \alpha_1(T_w) - \alpha_1(T_{w'}) : u \in A, w, w' \in N \text{ and } w \neq w'\}$; the sum of the smallest element in $\{\alpha_0(T_u) : u \in A\}$ and the smallest two elements in $\{-\alpha_1(T_w) : w \in N\}$. The case is similar for R_9, R_{10} and R_{11}.

In R_5, the element moved from N to A must be different from the one moved from N to D. To correctly evaluate $r(R_5)$, one must examine the result of $e(N \to A)$ and $e(N \to D)$. If the minimum values of both functions are achieved by the same element in N, then one of the movements needs to use the second smallest element in that set. The value of $r(R_5)$ is $\min\{-\alpha_1(T_w) - \alpha_1(T_{w'}) + \alpha_0(T_{w'}) : w, w' \in N \text{ and } w \neq w'\}$.

These evaluations are non-trivial, nonetheless they can be computed in $O(\lg(k))$ time with a priority queue implemented on a heap data structure (cf.[5], Chapter 6).

Lemma 24. *Let (D, N, A) be an arbitrary feasible partition. There exists an optimal partition (D_o, N_o, A_o) such that with perfect knowledge, there exists a sequence, $R_{x_1}, R_{x_2}, \ldots, R_{x_\ell}$ of R-rule applications that transforms (D, N, A) to (D_o, N_o, A_o), where $r(R_{x_t}) < 0$ for $1 \leq t \leq \ell$ and each element in $\{u_1, \ldots, u_k\}$ is moved at most once.*

Proof. By Theorem 21 and the discussion following it, we know the existence of a sequence that transforms (D, N, A) to (D_o, N_o, A_o) where each element of $\{u_1, \ldots, u_k\}$ is moved at most once. Let $R_{x_1}, R_{x_2}, \ldots, R_{x_\ell}$ be such a sequence with the minimum number of R-rule applications (minimum ℓ). Suppose $R_{x'}$ is a rule in the sequence where $r(R_{x'}) \geq 0$. Removing $R_{x'}$ from the sequence will yield a new, shorter sequence that transforms (D, N, A) to another feasible solution that is no worse than the optimal, i.e., it transforms (D, N, A) to a different optimal solution. This is not possible since $R_{x_1}, R_{x_2}, \ldots, R_{x_\ell}$ is shortest, so $r(R_{x'}) < 0$ for $x' \in \{x_1, \ldots, x_\ell\}$. □

Next, we propose an algorithm that finds an optimal partition, and then bound its time complexity.

Input: $\{\alpha_0(T_{u_1}), \ldots, \alpha_0(T_{u_k})\}$ and $\{\alpha_1(T_{u_1}), \ldots, \alpha_1(T_{u_k})\}$.

Output: An optimal partition for $\{u_1, \ldots, u_k\}$.

Algorithm 25.

1. *Let $(D, N, A) \leftarrow (\{\}, \{\}, \{\})$. (D, N, A) is an optimal partition for $\{\}$.*
2. *For $i = 1 \ldots k$*
   1. *$N \leftarrow N \cup \{u_i\}$*
   2. *While $\min_{1 \leq x \leq 11} r(R_x) < 0$*
      1. *Let x' be such that $r(R_{x'}) = \min_{1 \leq x \leq 11} r(R_x)$*
      2. *$(D, N, A) \leftarrow R_{x'}(D, N, A)$*
3. *Return (D, N, A).*

Lemma 26. *Algorithm 25 provides an optimal partition.*

Proof. At the start of each iteration of the for-loop in step 2, (D, N, A) is an optimal partition of $\{u_1, \ldots, u_{i-1}\}$. By adding u_i to N, (D, N, A) becomes a feasible solution for $\{u_1, \ldots, u_i\}$. Note that at each iteration of the while-loop in step 2.2, the objective function of the current partition (D, N, A) strictly decreases. Since the value of the objective function is at least 1, it cannot decrease indefinitely. Thus, the while-loop must terminate.

Next, assume that at the end of the i-th iteration of the for-loop, (D, N, A) is not an optimal partition of $\{u_1, \ldots, u_i\}$. By Lemma 24, there exists a sequence of R-rule applications that transforms (D, N, A) to an optimal solution, where for any rule $R_{x'}$ in that sequence, $r(R_{x'}) < 0$. This is a contradiction to the termination of the while-loop. □

Lemma 27. *The while-loop in Algorithm 25 is executed at most once for each iteration of the for-loop.*

Proof. Recall from Definition 13 that the value of a partition (D, N, A) is $1 + \sum_{u \in D} \alpha_0(T_u) + \sum_{u \in N} \alpha_1(T_u)$. Consider the i-th iteration of the for-loop in step 2. Let OPT_i be the value of an optimal partition for $\{u_1, \ldots, u_i\}$, let $(D_{i-1}, N_{i-1}, A_{i-1})$ be an arbitrary optimal partition of $\{u_1, \ldots, u_{i-1}\}$, and let (D, N, A) be $(D_{i-1}, N_{i-1} \cup \{u_i\}, A_{i-1})$. Note that (D, N, A) is a feasible partition of $\{u_1, \ldots, u_i\}$, which we constructed in step 2.1, before the execution of the while-loop.

By Lemma 24, there exists an optimal partition of $\{u_1, \ldots, u_i\}$ such that with perfect knowledge, there exists a sequence of R-rule applications $R_{x_1}, \ldots, R_{x_\ell}$ that transforms (D, N, A) to it, such that $r(R_{x_t}) < 0$ for $1 \le t \le \ell$. Let (D_i, N_i, A_i) be such an optimal partition. Let $f(D, N, A)$ be the value of partition (D, N, A). Note that $f(D, N, A) = f(D_{i-1}, N_{i-1}, A_{i-1}) + \alpha_1(T_{u_i})$. Two cases follow.

1. $u_i \in N_i$.

   In this case, we claim that $f(D, N, A) = f(D_i, N_i, A_i) = \mathrm{OPT}_i$, and the while-loop will execute for 0 iterations. Assume the opposite that $f(D, N, A) > f(D_i, N_i, A_i)$. Let (D_t, N_t, A_t) be $(D_i, N_i - \{u_i\}, A_i)$. Then, (D_t, N_t, A_t) is a feasible partition of $\{u_1, \ldots, u_{i-1}\}$ with value $f(D_t, N_t, A_t) = f(D_i, N_i, A_i) - \alpha_1(T_{u_i})$. Now,

   $$\begin{aligned} f(D, N, A) &> f(D_i, N_i, A_i) \\ f(D_{i-1}, N_{i-1}, A_{i-1}) + \alpha_1(T_{u_i}) &> f(D_i, N_i, A_i) \\ f(D_{i-1}, N_{i-1}, A_{i-1}) &> f(D_i, N_i, A_i) - \alpha_1(T_{u_i}) \\ f(D_{i-1}, N_{i-1}, A_{i-1}) &> f(D_t, N_t, A_t) \end{aligned}$$

   We have obtained a feasible partition of $\{u_1, \ldots, u_{i-1}\}$, (D_t, N_t, A_t), with value $f(D_t, N_t, A_t) < f(D_{i-1}, N_{i-1}, A_{i-1})$. This is a contradiction since $(D_{i-1}, N_{i-1}, A_{i-1})$ is an optimal partition of $\{u_1, \ldots, u_{i-1}\}$.

2. $u_i \notin N_i$.

   By the definition of (D_i, N_i, A_i), with perfect knowledge, there exists a sequence of R-rule applications $R_{x_1}, \ldots, R_{x_\ell}$ that takes (D, N, A) to (D_i, N_i, A_i), such that $r(R_{x_t}) < 0$ for $1 \le t \le \ell$. We claim that $\ell = 1$. Assume $\ell > 1$. Since each element is moved at most once, element u_i is moved at most once during the applications of $R_{x_1}, \ldots, R_{x_\ell}$. As the order of applications does not matter, let u_i be moved in the last rule. Then, R_{x_1} does not move u_i. Let (D_t, N_t, A_t) be constructed as follows. Apply R_{x_1} on (D, N, A), then remove u_i from N. This is valid since R_{x_1} does not move u_i, and removing $u_i \in N$ does not affect the feasibility of this new partition. Thus, (D_t, N_t, A_t) is a feasible partition of $\{u_1, \ldots, u_{i-1}\}$. Now,

$$\begin{aligned} f(D_t, N_t, A_t) &= f(D, N, A) + r(R_{x_1}) - \alpha_1(T_{u_i}) \\ &= f(D_{i-1}, N_{i-1}, A_{i-1}) + \alpha_1(T_{u_i}) + r(R_{x_1}) - \alpha_1(T_{u_i}) \\ &= f(D_{i-1}, N_{i-1}, A_{i-1}) + r(R_{x_1}) \end{aligned}$$

Since $r(R_{x_1}) < 0$, $f(D_t, N_t, A_t) < f(D_{i-1}, N_{i-1}, A_{i-1})$. This is a contradiction, since $(D_{i-1}, N_{i-1}, A_{i-1})$ is an optimal partition for $\{u_1, \ldots, u_{i-1}\}$.

□

Lemma 27 gives us the following algorithm.

Algorithm 28.

Input: *$\{\alpha_0(T_{u_1}), \ldots, \alpha_0(T_{u_k})\}$ and $\{\alpha_1(T_{u_1}), \ldots, \alpha_1(T_{u_k})\}$.*

Output: *An optimal partition for $\{u_1, \ldots, u_k\}$.*

1. *Let $(D, N, A) \leftarrow (\{\}, \{\}, \{\})$. (D, N, A) is an optimal partition for $\{\}$.*
2. *For $i = 1 \ldots k$*
   1. *$N \leftarrow N \cup \{u_i\}$*
   2. ***If*** $\min_{1 \le x \le 11} r(R_x) < 0$
      1. *Let x' be such that $r(R_{x'}) = \min_{1 \le x \le 11} r(R_x)$*
      2. *$(D, N, A) \leftarrow R_{x'}(D, N, A)$*
3. *Return (D, N, A).*

As a result, we have an $O(k \times m(k))$ time algorithm for computing an optimal partition that gives us the value of $\alpha_0(T_v)$, where $m(k)$ is the time of accessing (for smallest and second smallest) and modifying a priority queue (implemented with a heap, cf.[5], Chapter 6) of at most k elements. The value of $\alpha_1(T_v)$ can be computed in a similar way.

In turn, this gives us an algorithm for solving the Rooted Secure Set problem of trees, with running time $\sum_{v \in T} O(c_v \times m(c_v)) \le \sum_{v \in T} O(c_v \times m(\Delta)) = O((n-1) \times m(\Delta)) = O(n \times m(\Delta))$. In the sum, v is any internal node and c_v is the number of children of that node. With appropriate data structures, $m(\Delta) \in O(\lg(\Delta))$. Hence, we have an $O(n \lg(\Delta))$ algorithm for solving the Rooted Secure Set problem of trees.

4. Extensions and Conclusions

In this paper we focused on the development of an $O(n \lg(\Delta))$ algorithm for Rooted Secure Set of trees, where the root of the tree must be included in the secure set. Two extensions are possible following what is presented here. First, one may ask the following question.

Problem 29.

Given: *A graph* $G = (V, E)$, *sets* $\mathscr{I}, \mathscr{E} \subseteq V$, *integer* k.

Question: *Is there a secure set* $S \subseteq V$ *of size* k *or less, such that* $\mathscr{I} \subseteq S$ *and* $\mathscr{E} \cap S = \emptyset$*?*

That is, in addition to enforce a single vertex to be in the set, we would like to make sure every vertex in $\mathscr{I}$ is in S, and every vertex in $\mathscr{E}$ is not in S. This is a more general model for an application, as in many situations there are some vertices which are desired to be included in a secure set, while some other vertices are not desired. The Rooted Secure Set problem presented here is a special case where $\mathscr{I} = \{r\}, \mathscr{E} = \{\}$.

The second extension is to solve for a set that is secure, but with additional properties enforced. For instance a global secure set of a tree can be computed as an (non-trivial) extension of the rooted case, where ideas along similar lines can be applied.

In addition to solving secure sets of trees, a more interesting question is whether solving for a secure set on more complex graphs are polynomial. In was shown, in [7] that a minimum secure set of any outerplanar graph has cardinality at most 3. One may want to investigate for an algorithm that computes a rooted secure set of an outerplanar graph.

Most NP-complete problems are polynomial when restricted to series-parallel graphs, partial k-trees and graphs with bounded treewidth [1, 2, 15, 17]. It is interesting to ask if the Secure Set problem, a problem not believed to be in the class NP, can be solved in polynomial time on these graphs.

References

[1] Marshall W. Bern, Eugene L. Lawler, and A. L. Wong, Linear-time computation of optimal subgraphs of decomposable graphs, *J. Algorithms*, **8**(2) (1987), 216–235.

[2] Hans L. Bodlaender, Dynamic programming on graphs with bounded treewidth, In *Internat. Colloq. Automata, Languages and Programming*, Springer-Verlag, London, UK, (1988), 105–118.

[3] Robert C. Brigham, Ronald D. Dutton, Teresa W. Haynes, and Stephen T. Hedetniemi, Powerful alliances in graphs, *Discrete Math.*, **309** (2009), 2140–2147.

[4] Robert C. Brigham, Ronald D. Dutton, and Stephen T. Hedetniemi, Security in graphs, *Discrete Appl. Math.*, **155**(13) (2007), 1708–1714.

[5] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein, *Introduction to Algorithms*, MIT Press, 2001.

[6] Ronald D. Dutton, Secure set algorithms and complexity, *Congr. Numer.*, **180** (2006), 115–121.

[7] Ronald D. Dutton, On a graph's security number, *Discrete Math.*, **309** (2009), 4443–4447.

[8] Ronald D. Dutton, Robert Lee, and Robert C. Brigham, Bounds on a graph's security number, *Discrete Appl. Math.*, **156** (2008), 695–704.

[9] Michael R. Garey and David S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-completeness*, W. H. Freeman, 1979.

[10] Jonathan L. Gross and Jay Yellen, editors, *Handbook of Graph Theory (Discrete Mathematics and Its Applications)*, CRC, 2003.

[11] Tohru Kikuno, Noriyoshi Yoshida, and Yoshiaki Kakuda, A linear algorithm for the domination number of a series-parallel graph, *Discrete Appl. Math.*, **5** (1983), 299–311.

[12] Petter Kristiansen, Sandra M. Hedetniemi, and Stephen T. Hedetniemi, Alliances in graphs, *J. Combin. Math. Combin. Comput.*, **48** (2004), 157–177.

[13] Juan A. Rodrguez-Velzquez and Jos M. Sigarreta, Global defensive k-alliances in graphs, *Discrete Appl. Math.*, **157** (2009), 211–218.

[14] Juan A. Rodrguez-Velzquez, Ismael G. Yero, and Jos M. Sigarreta, Defensive k-alliances in graphs, *Appl. Math. Lett.*, **22** (2009), 96–100.

[15] K. Takamizawa, Takao Nishizeki, and Nobuji A. Saito, Linear-time computability of combinatorial problems on series-parallel graphs, *J. ACM*, **29**(3) (1982), 623–641.

[16] Douglas B. West, *Introduction to Graph Theory*, Prentice Hall, 2004.

[17] Thomas V. Wimer, *Linear algorithms on k-terminal graphs*, PhD thesis, Clemson University, Clemson, SC, USA, 1987.

[18] Thomas V. Wimer, Stephen T. Hedetniemi, and Renu C. Laskar, A methodology for constructing linear graph algorithms, *Congr. Numer.*, **50** (1985), 43–60.