A fast algorithm for powerful alliances in trees

Ararat Harutyunyan *

Department of Mathematics
Simon Fraser University
Burnaby, BC, V5A 1S6
Canada
aha43@sfu.ca

Abstract. Given a graph $G = (V, E)$ with a positive weight function w on the vertices of G, a global powerful alliance of G is a subset S of V such that for every vertex v at least half of the total weight in the closed neighborhood of v is contributed by the vertices of S. Finding the smallest such set in general graphs is NP-complete, even when the weights are all the same. In this paper, we give a linear time algorithm that finds the smallest global powerful alliance of any weighted tree $T = (V, E)$.

Keywords: Alliances, powerful alliances, weighted trees, algorithm

1 Introduction

The study of alliances in graphs was first introduced by Hedetniemi, Hedetniemi and Kristiansen [9]. They introduced the concepts of defensive and offensive alliances, global offensive and global defensive alliances and studied alliance numbers of a class of graphs such as cycles, wheels, grids and complete graphs. The concept of alliances is similar to that of *unfriendly partitions*, where the problem is to partition $V(G)$ into classes such that each vertex has at least as many neighbors outside its class than its own (see for example [1] and [10]). Haynes et al. [7] studied the global defensive alliance numbers of different classes of graphs. They gave lower bounds for general graphs, bipartite graphs and trees, and upper bounds for general graphs and trees. Rodriquez-Velazquez and Sigarreta [15] studied the defensive alliance number and the global defensive alliance number of line graphs. A characterization of trees with equal domination and global strong defensive alliance numbers was given by Haynes, Hedetniemi and Henning [8]. Some bounds for the alliance numbers in trees are given in [6]. Rodriguez-Velazquez and Sigarreta [12] gave bounds for the defensive, offensive, global defensive, global offensive alliance numbers in terms of the algebraic connectivity, the spectral radius, and the Laplacian spectral radius of a graph. They also gave bounds on the global offensive alliance number of cubic graphs in [13]

* Research supported by FQRNT (Le Fonds québécois de la recherche sur la nature et les technologies) doctoral scholarship.

and the global offensive alliance number for general graphs in [14] and [11]. The concept of powerful alliances was introduced recently in [3].

Given a simple graph $G = (V, E)$ and a vertex $v \in V$, the *open neighborhood* of v, $N(v)$, is defined as $N(v) = \{u : (u, v) \in E\}$. The *closed neighborhood* of v, denoted by $N[v]$, is $N[v] = N(v) \cup \{v\}$. Given a set $X \subset V$, the *boundary* of X, denoted by $\delta(X)$, is the set of vertices in $V - X$ that are adjacent to at least one member of X.

Definition 1. *A set $S \subset V$ is a* defensive alliance *if for every $v \in S$, $|N[v] \cap S| \geq |N[v] \cap (V - S)|$. For a weighted graph G, where each vertex v has a non-negative weight $w(v)$, a set $S \subset V$ is called a* weighted defensive alliance *if for every $v \in S$, $\sum_{u \in N[v] \cap S} w(u) \geq \sum_{u \in N[v] \cap (V-S)} w(u)$. A (weighted) defensive alliance S is called a* global (weighted) defensive alliance *if S is also a dominating set.*

Definition 2. *A set $S \subset V$ is an* offensive alliance *if for every $v \in \delta(S)$, $|N[v] \cap S| \geq |N[v] \cap (V - S)|$. For a weighted graph G, where each vertex v has a non-negative weight $w(v)$, a set $S \subset V$ is called a* weighted offensive alliance *if for every $v \in \delta(S)$, $\sum_{u \in N[v] \cap S} w(u) \geq \sum_{u \in N[v] \cap (V-S)} w(u)$. A (weighted) offensive alliance S is called a* global (weighted) offensive alliance *if S is also a dominating set.*

Definition 3. *A* global (weighted) powerful alliance *is a set $S \subset V$ such that S is both a global (weighted) offensive alliance and a global (weighted) defensive alliance.*

Definition 4. *The* global powerful alliance number *of G is the cardinality of a minimum size global (weighted) powerful alliance in G, and is denoted by $\gamma_p(G)$. A minimum size global powerful alliance is called a $\gamma_p(G)$-set.*

There are many applications of alliances. One is military defence. In a network, alliances can be used to protect important nodes. An alliance is also a model of suppliers and clients, where each supplier needs to have as many reserves as clients to be able to support them. More examples can be found in [9].

Balakrishnan et al. [2] studied the complexity of global alliances. They showed that the decision problems for global defensive and global offensive alliances are both NP-complete for general graphs. It is clear that the decision problems to find global defensive and global offensive alliances in weighted graphs are also NP-complete for general graphs.

The problem of finding global defensive, global offensive and global powerful alliances is only solved for trees. [5] gives a $O(|V|^3)$ dynamic programming algorithm that finds global defensive, global offensive and global powerful alliances of any weighted tree $T = (V, E)$.

In this short paper we give a $O(|V|)$ time algorithm that finds the global powerful alliance number of any weighted tree $T = (V, E)$. Combining our result with the obvious lower bound, we actually show that the problem of finding global powerful alliance number of any weighted tree $T = (V, E)$ is $\Theta(|V|)$.

In the next section, we present a linear time algorithm for minimum cardinality powerful alliance in weighed trees.

2 Powerful Alliances

In this section, we give a linear time algorithm that finds the minimum cardinality weighted global powerful alliance number of any tree. We assume all the weights are positive - the algorithm can be easily modified for the case where the weights are non-negative.

For a set $S \subset V$ define $w(S) := \sum_{u \in S} w(u)$.

It is clear that when the weight function is positive, the global powerful alliance problem can be formulated as follows.

Observation 1 *Let $G = (V, E)$ be a graph, and $w : V \to R^+ \backslash \{0\}$ a weight function. Then a global powerful alliance in G is a set $S \subset V$ such that for all $v \in V$,*

$$\sum_{u \in N[v] \cap S} w(u) \geq \frac{w(N[v])}{2}.$$

Note that the condition that S is a dominating set is automatically guaranteed because the weights of all vertices are positive. We will use the above formulation in our algorithm.

Definition 5. *For $v \in V$, the* alliance condition for v *is the condition that* $\sum_{u \in N[v] \cap S} w(u) \geq \frac{w(N[v])}{2}$.

2.1 Satisfying the alliance condition for a vertex

Throughout the algorithm, for every vertex v we need to find the smallest number of vertices necessary in the closed neighborhood of v to satisfy the alliance condition for v. To solve this problem, we use the following algorithm.

Given positive numbers $a_1, a_2, ..., a_n$, $FindMinSubset(a_1, a_2..., a_n)$ is the problem of finding the minimum k such that there is a k-subset of $\{a_1, a_2, ..., a_n\}$ the elements of which sum up to at least $\frac{1}{2} \sum_{i=1}^{n} a_i$. We give an algorithm that solves this problem in time $O(n)$. The algorithm also finds an instance of such a k-subset. Furthermore, out of all the optimal solutions it outputs a set with a maximum sum.

Algorithm FindMinSubset[A,n,T].
Input: An array A of size n of positive integers; a target value T.
Output: The least integer k such that some k elements of A add up to at least T.

1. If $n = 1$, return 1.
2. Set $i = \lceil \frac{n}{2} \rceil$.
3. Find the set A' of the i largest elements of A and compute their sum M.
4. If $M > T$ return $FindMinSubset[A', i, T]$; if $M = T$ return i; else return $i + FindMinSubset[A - A', n - i, T - M]$.

Lemma 1. *The above algorithm solves the problem $FindMinSubset(a_1, a_2, ..., a_n)$ in time $O(n)$. Furthermore, if the solution is k, the algorithm finds the k largest elements.*

Proof. Let $S = \sum_{i=1}^{n} a_i$. It is clear that $FindMinSubset[A, n, T = \frac{S}{2}]$ where A is the array of the elements $(a_1, a_2, ..., a_n)$ will solve the desired problem. The analysis of the running time is as follows. Note that finding the largest k elements in an array of size n can be done in linear time for any k (see [4]). Therefore, in each iteration of $FindMinSubset$, Step 2 and Step 3 take linear time, say Cn. Since the input size is always going down by a factor of 2, we have that the running time $T(n)$ satisfies $T(n) \leq T(n/2) + Cn$. By induction, it is easily seen that $T(n) = O(n)$. It is clear that out of all the possible solutions, the algorithm picks the one with the largest weight.

We now describe the algorithm for weighted powerful alliances in trees.

2.2 An overview of the algorithm

We assume the tree is rooted. For each k, we order the vertices of depth k from left to right. By $C(v)$ we denote the set of children of v. We define $p(v)$ to be the parent of v.

In each iteration of the algorithm, we may label some vertices with "+", with "?p", or with "?c". The "+" vertices are going to be part of the powerful alliance. Under some conditions, we may also pick some of the vertices labelled with "?p" or "?c".

Now, we give a brief intuition behind the algorithm. As noted above, when all the weights are positive, the global powerful alliance problem is simply finding the smallest set $S \subset V$ such that for every vertex $v \in V$, $w(N[v] \cap S) \geq \frac{w(N[v])}{2}$. Our algorithm is essentially a greedy algorithm. We root the tree at a vertex, and start exploring the neighborhoods of vertices starting from the bottom level of the tree. The vertices which have already been chosen to be included in the alliance set S are labelled with "+". For each vertex v, we find the smallest number of vertices in its closed neighborhood that need to be added to the vertices labelled "+" in v's closed neighborhood to satisfy the alliance condition for v. We do this using the algorithm FindMinSubset. In some cases we may get more than one optimal solution and it will matter which solution we pick (for example, if there is an optimal solution containing both v and $p(v)$ then this solution is preferable when we consider the neighborhood of $p(v)$). The complication arises when there is an optimal solution containing v, but not $p(v)$, and there is an optimal solution containing $p(v)$. If $w(v) > w(p(v))$, then it is not clear which solution is to be preferred because v is at least as good as $p(v)$ for satisfying the alliance condition for $p(v)$, but choosing $p(v)$ is preferable for satisfying the alliance condition for parent of $p(v)$, $p(p(v))$. In general, when we have to choose between a solution that contains v and one that contains $p(v)$ we give preference to the solution containing $p(v)$ unless: (i) we can immediately gain by choosing the solution with v due to choices made in previous iterations

(ii)when satisfying the alliance condition for $p(v)$, it might theoretically be better to have chosen v. In case (ii), we label both v and $p(v)$ with "?", v with "?c", and $p(v)$ with "?p", and delay satisfying v's alliance condition for later iterations.

2.3 Labels and Sets

In the algorithm, we use four labels for vertices: "+", "?c", "?p" and "?c+". We assign a vertex v a label "+" when we can claim that v is contained in some minimum cardinality powerful alliance. Generally, we assign a vertex v a label "?c" when we have a choice of taking v or $p(v)$ to satisfy the alliance condition for v (but we can't choose both v and $p(v)$) and $w(v) > w(p(v))$. In this case, we also label $p(v)$ with "?p". For a vertex u, we define $D(u)$ to be the set of all "?c" children of u. Note that for every vertex u, eventually we must take u or $D(u)$ in our alliance, regardless whether these vertices are the best in the sense that they help satisfy the alliance condition for u. We change the label of a vertex v from "?c" to "?c+" if for satisfying the alliance condition for $p(v)$ it is better to choose v than $p(v)$. The reason we label it "?c+" and not "+" is that it may turn out that $p(p(v))$, the parent of $p(v)$, will be labelled "+", and and this may allow labelling $p(v)$ with "+" and unlabelling of v.

There are two occasions when we label a vertex v with "?p". The first is when a child of v is labelled "?c", as described above. The second case is when we see no optimal solution containing $p(v)$ that would satisfy v's alliance condition, but if $p(v)$ were later labelled "+" for another reason, then there would be an optimal solution containing v that would satisfy v's alliance condition. This solution would be preferable since it can decrease the number of vertices required to satisfy $p(v)$'s alliance condition.

We denote by $N^+[v]$ to be the set of all vertices in the closed neighborhood of v which are labelled with "+" at the current stage in the algorithm. For a vertex $v \in V$, we define $Findmin(v)$ to be the function that finds the smallest set of vertices in $N[v] \backslash N^+[v]$(i.e. the set of vertices not labelled with "+") that need to be added to the set of vertices in already labelled "+" in v's neighborhood, $N^+[v]$, to satisfy the alliance condition for v. If there is more than one such set, $Findmin(v)$ returns the set with maximum total weight. In the algorithm, by a *solution* for v we mean either the set $Findmin(v)$ or a set S such that $|S| = |Findmin(v)|$ and $S \cup N^+[v]$ satisfies the alliance condition for v. Note that it could be that $Findmin(v) = \emptyset$ since the alliance condition could already have been satisfied for v. Also, note that finding $Findmin(v)$ is done using the algorithm $FindMinSubset$ defined previously.

For a set of vertices X none of which are labelled "+", define $Findmin(v) : X$ to be the set $Findmin(v)$ under the assumption that the vertices of X are now labelled "+". We also have a special set X_v for every vertex v. This is the set of all children u of v labelled with "?p" with the property that $|Findmin(u) : \{u, v\}| < |Findmin(u)|$. This means that when trying to satisfy the alliance condition for v, if we see a solution that contains v and u, then we can safely label u and v with "+", regardless whether this solution contained $p(v)$ since we will gain a vertex when solve the alliance condition for u. Therefore, the

X_v children of v can be used under some circumstances to satisfy the alliance condition for v. We have also two recursive functions, $Xcollect(v)$ and $Clear(v)$, that are used in the algorithm. $Xcollect(v)$ chooses all the X vertices in the subtree rooted at v, and finally settles the alliance condition for these vertices. It is used when we know that we can label v with a "+". $Clear(v)$ erases the labels of all the non X vertices that have label "?p" in the subtree rooted at v, and settles the alliance condition for them. Once we reach the root $r(T)$ of the tree, we no longer have the problem of deciding whether to give the parent of $r(T)$ a priority over $r(T)$ or its X_r children, since the parent does not exist. We can then settle the alliance conditions of all the "?" labelled vertices.

The functions $Clear(v)$ and $Xcollect(v)$ are as follows:

$Clear(v)$
For every child u of v labelled "?p" AND $u \notin X_v$,
Remove the label "?p" (and "?c" if it exists) of u
Replace all the labels of "?c+" from its children by "+".
$Clear(u)$.
$Findmin(u)$ and label the chosen vertices by "+".

$Xcollect(v)$
For every X_v child u of v
Label u with "+".
Remove "?p" label from u
Remove all "?c+" labels from children of u.
$Xcollect(u)$.
$Findmin(u)$, and label chosen vertices with "+".

If vertex r is the root, we assume that $p(r) = \emptyset$ and $w(p(r)) = 0$. Also, we will say that $p(r)$ has label "+" so that we are in Case 1 or in Case 2.1.

2.4 The Algorithm

The algorithm has two cases: the vertex v under consideration is labelled "+", or "?p" or unlabelled (it can only receive the label "?c" during the iteration). Each case has two subcases depending on the label of $p(v)$. We often switch between the cases. For example, if the vertex v under consideration was unlabelled (Case 2) and gets a label "+", we jump to Case 1 and continue from there.
All X_v's are initially set to $X_v := \emptyset$.

Algorithm Weighted Powerful Alliances in Trees

for($i = 0$ to d) AND **for all** (vertices v at depth $d - i$) **do**

Algorithm 1 Algorithm for finding Minimum Cardinality Weighted Powerful Alliance in a tree T. **Case 1: v is labelled with "+".**

Case 1.1: v has no "?p" children.
if $p(v)$ is labelled "+" **then**
  $Findmin(v)$ and take any solution. Label all the picked vertices with "+".
else if [$p(v)$ is unlabelled or labelled "?p"] **then**
  $Findmin(v)$
  if $\exists$ solution containing $p(v)$ **then**
    choose this solution. Label all picked vertices with "+".
    if v had label "?p" **then**
      remove this label, and remove the labels "?c" from all its children.
  else
    $Findmin(v)$
    Label picked vertices with "+"
{**Case1.2: v has at $\geq$ 1 "?p" child.**}
Xcollect(v)
Clear(v)
Go back to Case 1.1

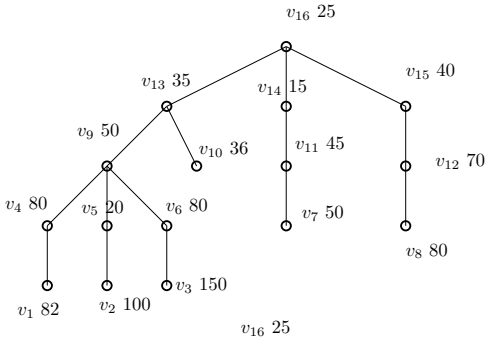
Algorithm 2 Case 2: v is labelled "?p" or unlabelled.

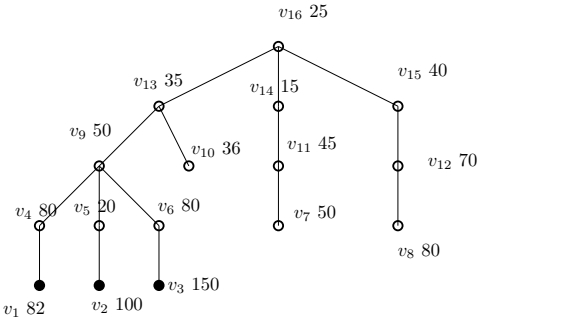
Case 2.1: $p(v)$ is labelled "+".
Let $D(v)$ be the set of children of v that have a label "?c".
$X_v := X_v \cup \{u \in C(v) : label(u) =$ "?p", $|Findmin(u) : \{u, v\}| < |Findmin(u)|\}$
Define $X'_v = X_v \cup v \cup p(v)$
$Findmin(v)$.
if $\exists$ solution $\ni v$ OR $\nexists$ solution $\supset D(v)$ OR $|Findmin(v) : \{X_v \cup v\}| < |Findmin(v)|$ **then**
  Label v with "+".
  Remove "?c" label's from v's children.
  Go to Case 1.
else
  Choose a solution $S \supset D(v)$. Label vertices of S with "+"
  Remove "?c" labels from vertices in $D(v)$, remove "?p" label from v
  Set $X_u := \emptyset$ for every vertex u in the subtree rooted at v
  Clear(v).

Algorithm 3 Case 2: v is labelled "?p" OR is unlabelled.

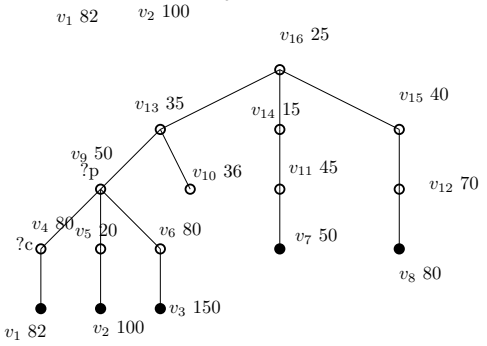
Case 2.2: $p(v)$ is labelled "?p" OR is unlabelled.
Let $D(v)$ be the set of children of v that have a label "?c".
$Findmin(v)$
$X_v := X_v \cup \{u \in C(v) : label(u) =$ "?p", $|Findmin(u) : \{u, v\}| < |Findmin(u)|\}$
$X'_v = X_v \cup \{v\} \cup \{p(v)\}$
if $|Findmin(v) : \{X_v \cup v\}| \leq |Findmin(v)| - 2$ OR $\exists$ solution S such that $|S \cap X'_v| \geq 2$ OR $\exists$ solution S such that $S \supset \{v, p(v)\}$. **then**
  Label v with "+", remove its "?p" label. Remove "?c" labels from v's children. Go to Case 1.
else if $\exists$ a solution S such that $S \supset \{D(v) \cup p(v)\}$ **then**
  Let R be a solution $\ni p(v)$ with maximum total weight.
  if $X_v \cup \{v\} \cup R \backslash u \geq w(N[v])/2$ for some $u \in R$, $u \neq p(v)$ **then**
    Label v with "+", remove its "?p" label.
    Remove "?c" labels from v's children.
    Go to Case 1.
  else if $X_v \cup \{v\} \cup R \backslash p(v) < w(N[v])/2$ OR $w(v) \leq w(p(v))$ **then**
    Label $p(v)$ with "+".
    if $p(v)$ had label "?p" **then**
      remove it, and remove "?c" labels from $p(v)$'s children.
    Go to Case 2.1
  else
    Label $p(v)$ "?p" if it is not already. Add a label "?c" to v. Relabel "?c" vertices in $D(v)$ by "?c+".
    if $|Findmin(v) : X'_v| < |Findmin(v) : \{p(v)\}|$ **then**
      $X_{p(v)} := X_{p(v)} \cup \{v\}$. Label v with "?p", if v is unlabelled.
    Clear(v)
else if $\exists$ a solution S such that $S \supset D(v)$ **then**
  if $|Findmin(v) : \{X_v \cup v\}| < |Findmin(v)|$ **then**
    Label v with "+", remove its "?p" label.
    Remove "?c" labels from v's children.
    Go to Case 1.
  else
    Relabel "?c" labels of vertices in $D(v)$ by "?c+".
    if $|Findmin(v) : X'_v| < |Findmin(v) : \{p(v)\}|$ **then**
      $X_{p(v)} := X_{p(v)} \cup \{v\}$. Label v with "?p", if v is unlabelled
    else
      Label vertices of S with "+".
      Remove "?c+" labels from vertices in $D(v)$
    Clear(v).
else
  Label v with "+". Remove "?p" label from v. Remove "?c" labels from vertices in $D(v)$. Go to Case 1.

An illustration of the algorithm by an example (Fig. 1).

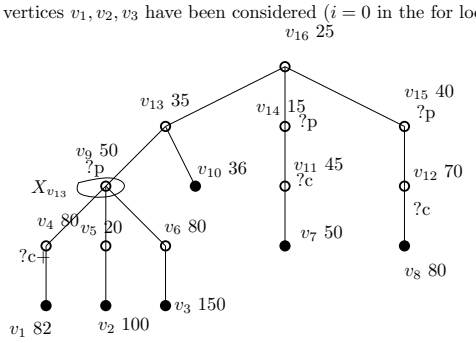




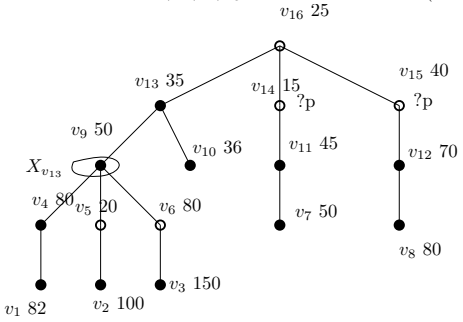
After vertices v_1, v_2, v_3 have been considered ($i = 0$ in the for loop)



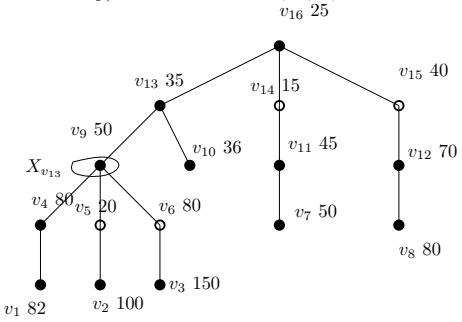
After vertices $v_1, v_2, ..., v_8$ have been considered ($i = 1$ in the for loop)



After vertices $v_1, v_2, ..., v_{12}$ have been considered ($i = 2$)



After vertices $v_1, v_2, ..., v_{15}$ have been considered ($i = 3$)



After vertices $v_1, v_2, ..., v_{16}$ have been considered ($i = 4$)

Theorem 1. *Algorithm Powerful Alliances for Trees correctly computes the minimum cardinality weighted powerful alliance of a weighted tree $T = (V, E)$ in time $O(|V|)$.*

Due to space restrictions, we omit the proof.

Acknowledgements I wish to thank Dr. Jacques Verstraete for helpful discussions, and Leonid Chindelevitch for indicating Lemma 1.

References

1. R. Aharoni, E. C. Milner, K. Prikry, Unfriendly partitions of a graph. J. Combin. Theory Ser. B 50 (1990), no. 1, 1-10.
2. H. Balakrishnan, A. Cami, N. Deo, and R. D. Dutton, On the complexity of finding optimal global alliances, J. Combinatorial Mathematics and Combinatorial Computing, Volume 58 (2006), 23-31.
3. R.C. Brigham, R. D. Dutton, T. W. Haynes, S. T. Hedetniemi, Powerful alliances in graphs, Discrete Mathematics Volume 309 (2009) Issue 8, 2140-2147.
4. T. H. Cormen, C. E. Leiserson, R. L. Rivest, C. Stein Introduction to Algorithms, McGraw-Hill, 2002.
5. B.C. Dean, L. Jamieson, Weighted Alliances in Graphs, Congressus Numerantium 187: 76-82, 2007
6. A. Harutyunyan, Some bounds in alliances in trees, Cologne Twente Workshop on Graphs and Combinatorial Optimization 2010, accepted.
7. T. W. Haynes, S. T. Hedetniemi, and M. A. Henning, Global defensive allliances in graphs, Electronic Journal of Combinatorics 10 (2003), no. 1, R47.
8. T. W. Haynes, S. T. Hedetniemi, and M. A. Henning, A characterization of trees with equal domination and global strong alliance numbers, Utilitas Mathematica, Volume 66(2004), 105-119.
9. S. M. Hedetniemi, S. T. Hedetniemi, and P. Kristiansen, Alliances in graphs, Journal of Combinatorial Mathematics and Combinatorial Computing, Volume 48 (2004), 157-177.
10. E. C. Milner and S. Shelah, Graphs with no unfriendly partitions. A tribute to Paul Erdos, 373-384, Cambridge Univ. Press, Cambridge, 1990
11. J. A. Rodriguez-Velazquez, J.M. Sigarreta, On the global offensive alliance number of a graph, Discrete Applied Mathematics, Volume 157 (2009), Issue 2, 219-226.
12. J. A. Rodriguez-Velazquez, J.M. Sigarreta, Spectal study of alliances in graphs, Discussiones Mathematicae Graph Theory 27 (1) (2007) 143-157.
13. J. A. Rodriquez-Velazquez and J. M. Sigarreta, Offensive alliances in cubic graphs, International Mathematical Forum Volume 1 (2006), no. 36, 1773-1782.
14. J. A. Rodriguez-Velazquez, J.M. Sigarreta, Global Offensive Alliances in Graphs, Electronic Notes in Discrete Mathematics, Volume 25 (2006), 157-164.
15. J. A. Rodriguez-Velazquez, J. M. Sigarreta, On defensive alliances and line graphs. Applied Mathematics Letters, Volume 19 (12) (2006) 1345-1350.