An Algorithm for Testing a Signed Graph for Balance

Ioannis S. Xezonakis
Dept. of Electrical and Computer Engineering
Hellenic Mediterranean University
Heraklion, 71410, Greece

Danai Xezonaki
Faculty of Science
University of Amsterdam
1012 WX Amsterdam, The Netherlands

ABSTRACT
A signed graph consists of a graph together with a sign characterizing each vertex. A fundamental concept of signed graphs is that of balance. In this paper a programming algorithm is presented in order to detect balance in signed graphs. The algorithm traverses each vertex at most once and uses two stacks for the implementation, each having a size of at most the number of vertices of the graph. Moreover, the graph need not be stored in computer's memory.

General Terms
Graphs Algorithms, Signed graphs, Balanced graphs

Keywords
Balanced graphs, Signed graphs

1. INTRODUCTION
A *graph G=(V,E)* is a representation of objects, connected by links. The "objects" in graphs are called *vertices*, noted by the finite set *V* in the above representation, while links are called *edges*, noted by *E* in the same representation. A *signed graph* is one whose edges are characterized by a sign σ, either positive (+) or negative(-), so it in the following it will be denoted as $S=(G, \sigma) = (V, E, \sigma)$. The *sign of a cycle* in a signed graph is the product of the signs of all the edges participating in the cycle. A signed graph is called *balanced* if the sign of every cycle in the graph is positive [1].

The first mathematical notion for signed graphs is found in [2]. However, according to this paper, balanced signed graphs "obey" to the following theorem:
Suppose that S is a signed graph. The next conditions are equivalent:

i) S is balanced.

ii) Every cycle in S is positive

iii) All paths between every pair of dinstict vertices in V have the same sign.

iv) The set V of vertices can be separated into two disjoint sets of vertices, one of them may be empty. Every positive edge joins vertices of the same set and every negative edge joins vertices of different sets.

Thereafter, various references, as [3], [4] and [5], appear, whereas signed graphs find application in many fields such as psychology [6], turnover analysis [7] etc.

In [1] an algorithm is proposed, which uses a spanning tree of the graph. In this algorithm, all the vertices of the graph are examined, in order to obtain a sign. Subsequently, the edges are tested for sign, which is correct if it is equal to the product of the signs of the two adjacent vertices, else, a fault exists, the graph is non balanced and the algorithm is terminated.

In [8], another algorithm is proposed, which is based on the above condition (iii), examines each line of the graph at most once through a breadth-first search (BFS) traversal and does not require the graph to be stored in a computer's memory, since it examines the unused edges of a signed graph one at a time as they are read and tests the graph for balance.

Obviouslsy, extended research has been done for signed graphs. In recent papers, various related subjects are also examined, such as signed distance [9] and powers of signed graphs [10].

In this paper an algorithm is proposed, based also on the above condition (iii), which is a variation of the algorithm proposed in [8]. The algorithm uses two stacks. It is not necessary to examine for sign all the vertices of the graph and it is not based on the existence of closed subgraphs, as [8] does; also it is not required for the graph to be stored in a computer's memory.

2. THE ALGORITHM
In the proposed algorithm a sign is attributed to each vertex examined, in accordance to the sign of an edge adjacent to it. Then, reaching the vertex following different paths, the sign of the vertex is checked for consistency to the already attributed one. If such consistency does not exist, the graph is not balanced. The different paths, through which the examined vertex are reached, are formed by a depth first searching (DFS) traversal of the graph. Two stacks are used, named (for the needs of this paper) as ***POS*** and ***NEG***, in which the numbers of the vertices examined are put.

As already mentioned, the examined vertices are characterized by their sign as either *positive* (+) or *negative* (-). Suppose that a vertex *x* obtains the sign $\sigma(x)$. Departing from *x*, we reach vertex *z; z* obtains its sign $\sigma(z)$ by setting $\sigma(z) = \sigma(x) * \sigma(x,z)$, where $\sigma(x,z)$ holds for the sign of the edge adjacent to both *x* and *z* vertices. A vertex is also characterized as ***R*** (reached) or ***NR*** (non-reached) if it has already been reached or not, and as ***C*** (checked) or ***NC*** (non-checked) if checking of correct sign has already been performed to it or not. So, the status ***St*** of each vertex is descibed by a triad of symbols; for example, the status of a vertex *k*, denoted as $St(k)$, would be expressed as $St(k) = (+, R, NC)$ or $St(k) = (-, NR, NC)$ etc. When one (or more) of the three symbols has not yet been attributed to some vertex *k*, or its status does not matter, it is replaced by **?**. So, the status of some node may be $St(k) = (?,?, NC)$, or $St(k) = (?,R, ?)$ etc. Obviously, an ***NC*** vertex is possible to be ***R***; this occurs if we the vertex has already been reached, but checking of correct sign has not yet been completed. Sign is attributed only to ***R*** vertices, since a vertex gains sign at the moment of visiting it, that is, when its status changes from ***NR*** to ***R***.

The algorithm is described as follows and is implemented in a "quasi-C" pseudo-code form in Figure 1, where a variable named bal is used; this variable has the value true when the graph is balanced, else its value is false.

```
bal = true;
St(x) = (+, R, NC);
push (x, POS);
x = pop (POS);
while (St(x) == (?, ?, NC)) {
    St(x) = (?, R, C));
    while (St(z) == (?, ?, NC)) {
        if (σ(x) * σ(x, z) > 0) {
            if (St(z) == (?, R, ?) and (z in NEG)) {
                bal = false;
                break; }
            if (St(z) == (?, NR, ?)) {
                St(z) = (+, R, ?);
                push (z, POS); } }
        if (σ(x) * σ(x, z) < 0) {
            if (St(z) == (?, R, ?) and (z in POS)) {
                bal = false;
                break; }
            if (St(z) == (?, NR, ?)) {
                St(z) = (-, R, ?);
                push (z, NEG); } }
        next (z); }
        if (bal == false)
            break;
        if (POS not empty)
            x = pop (POS);
        if ((POS empty) and (NEG not empty))
            x = pop (NEG); }
if (bal == true)
   Graph is balanced;
else
   Graph is not balanced;
```

Fig 1: Algorithm for balance checking

While ***NC*** vertices exist in the graph, a vertex (say *x*) is arbitrarily chosen, which is characterized as positive (+). Its status is changed from ***NR*** to ***R*** and is pushed to the ***POS*** stack. The algorithm proceeds as below:

1. If ***POS*** is not empty, the vertex *x*, stored at the head of ***POS***, is recalled. This vertix is marked as ***C*** and continue with step 3.
2. If ***POS*** is empty, the vertex *x*, stored at the head of ***NEG***, is recalled. This vertix is marked as ***C*** and continue with step 3.
3. Departing from vertex *x*, there are visited one by one the vertices, to whom we arrive through edges having the same sign as *x*. For each such vertex *z*:
   a) If *z* is marked as ***R*** and belongs in the ***NEG*** stack, the graph is not balanced and the algorithm terminates.
   b) If *z* is marked as ***C***, it is ignored.
   c) If *z* is marked as ***R*** and belongs in the ***POS*** stack, it is also ignored.
   d) If *z* is marked as ***NR***, then the sign + is attributed to it, it is marked as ***R*** and is pushed in ***POS***.
4. Departing from vertex *x*, there are visited one by one the vertices, to whom we arrive through edges having the opposite sign than *x*. For each such vertex *z*:
   a) If *z* is marked as ***R*** and belongs in the ***POS*** stack, the graph is not balanced and the algorithm terminates.
   b) If *z* is marked as ***C***, it is ignored.
   c) If *z* is marked as ***R*** and belongs in the ***NEG*** stack, it is also ignored.
   d) If *z* is marked as ***NR***, the sign - is attributed to it, it is marked as ***R*** and is pushed in ***NEG***.

After following all the above steps, the non existence of ***NC*** vertices means that the graph is balanced.

In Figure 2 a signed graph is presented:

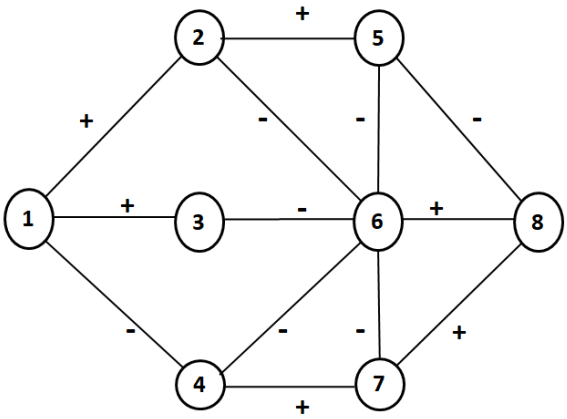


Fig 2: Signed graph

In the six sub-figures of Figure 3, the proposed algorithm is step by step applied for the graph of Figure 2. In each step, the vertices already checked for sign are shadowed. Each subgraph is accompanied by a matrix, having three columns; the first stands for the already checked vertices, while the second and the third ones stand for the contents of ***POS*** and ***NEG*** stacks (vertices in them are all reached (R) and non completed (NC)).

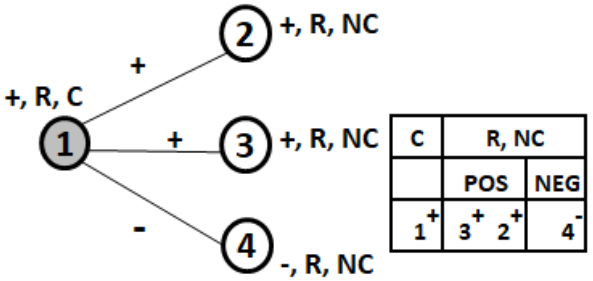


Fig. 3a: Step a

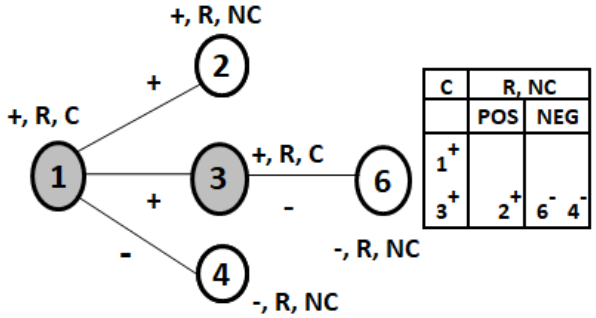
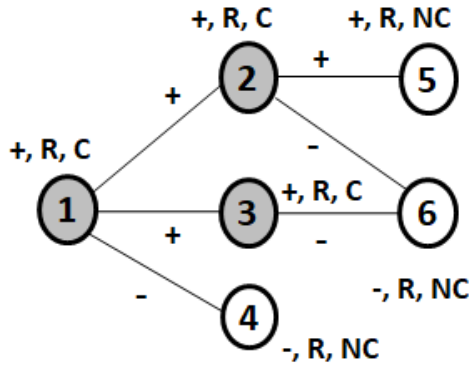
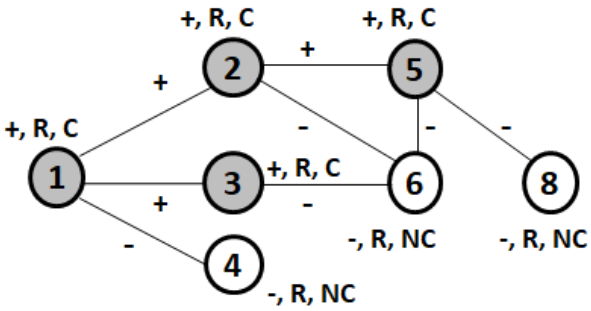


Fig. 3b: Step b



C	R, NC	
	POS	NEG
1+		
3+		
2+	5+	6- 4-

Fig. 3c: Step c

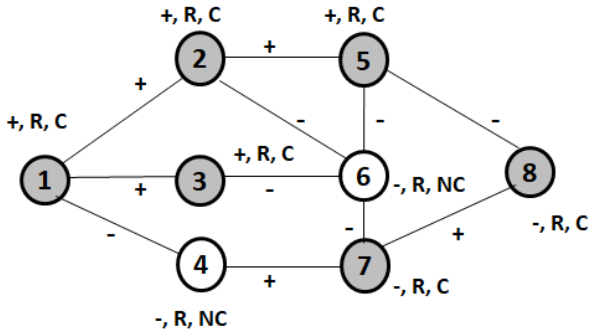


C	R, NC	
	POS	NEG
1+		
3+		
2+		
5+		
8-		7- 6- 4-

Fig. 3e: Step e

C	R, NC	
	POS	NEG
1+		
3+		
2+		
5+		8- 6- 4-

Fig. 3d: Step d



C	R, NC	
	POS	NEG
1+		
3+		
2+		
5+		
8-		
7-		6- 4-

Fig. 3f: Step f

In Figure 3f the algorithm ends; accessed from 7, vertex 6 must belong to ***POS***, since $\sigma(7) < 0$ and $\sigma(7, 6) < 0$. In contradiction, vertex 6 belongs to ***NEG***, so the graph is non-balanced.

3. RESULTS

The algorithm proposed, has obviously space complexity Θ(V). It is uses two stacks and has not to search for closed loops in order to decide whether the graph is balanced or not. It is not needed also to store the graph in memory. For all these reasons, the algorithm is considered as very effective. Moreover, the algoritmh has been checked using C programming language

Usung the proposed algorithm, the balanced power signed graphs can be a subject for further research.

4. REFERENCES

[1] F. Harary, and J. A. Kabell, 1980. A simple algorithm to detect balance in signed graphs. Mathematical Social Scienses 1, 131-136.

[2] F. Harary, 1953-1954. On the Notion of Balance of a Signed Graph. Michigan Mathematical Journal 2, 143-146.

[3] F. Harary, R.Z. Norman, and D. Cartwright, 1965. Structural Models: An Introduction to the Theory of Directed Graphs. Wiley.

[4] T. Zaslavsky, 1981. Characterization of Signed Graphs. Journal of Graph Theory, 5, 401-406.

[5] C. Hoede, 1992. A Characterization of Consistent Marked Graphs. Journal of Graph Theory, 16(1), 17-23.

[6] F. Heider, 1946. Attitudes and Cognitive Organization. The Journal of Psychology, 21, 107-112.

[7] B. Vasanthi et al., 2015. Applications of Signed Graphs to Portfolio Turnover Analysis. Procedia – Social and Behavioral Sciences, 211, 1203-1209.

[8] E. Loukakis, 2003. A Dynamic Programming Algorithm to Test a Signed Graph for Balance. Intern. J. Computer Math., 80(4), 499-507.

[9] S. Hameed et al., 2020. Signed Distance in Signed Graphs. Linear Algebra and its Applications, 608, 236-247.

[10] T. V. Shijin et al., 2022. On the powers of signed graphs. Communications in Combinatorics and Optimization. 7 (1), 45-51.