Graphs and Combinatorics 1, 305–310 (1985)

Graphs and Combinatorics

Asynchronous Threshold Networks

Noga Alon

Department of Mathematics, Tel Aviv University, Tel Aviv, Israel and
Bell Communications Research, Morristown, NJ 07960, USA

Abstract. Let $G = (V, E)$ be a graph with an initial sign $s(v) \in \{\pm 1\}$ for every vertex $v \in V$. When a vertex v becomes *active*, it resets its sign to $s'(v)$ which is the sign of the majority of its neighbors ($s'(v) = 1$ if there is a tie). G is in a *stable state* if $s(v) = s'(v)$ for all $v \in V$. We show that for every graph $G = (V, E)$ and every initial signs, there is a sequence $v_1, v_2, \ldots, v_r$ of vertices of G, in which no vertex appears more than once, such that if v_i becomes active at time i, $(1 \leq i \leq r)$, then after these r steps G reaches a stable state. This proves a conjecture of Miller. We also consider some generalizations to directed graphs with weighted edges.

1. Introduction

A *threshold network* N is a directed graph $G = (V, E)$ with a fixed real weight $w((u, v))$ assigned to each (directed) edge $(u, v) \in E$ and a variable sign $s(v) \in \{\pm 1\}$ assigned to each vertex $v \in V$. For a vertex $v \in V$ put $N(v) = \{u \in V: (u, v) \in E\}$. Initially, the vertices of N are not *active* and each sign has an initial value. When a node $v \in V$ becomes active it changes its sign from $s(v)$ to $s'(v)$ according to the following rule:

$$s'(v) = \begin{cases} -1 & \text{if } \sum \{s(u)w(u, v): u \in N(v)\} < 0 \\ +1 & \text{if } \sum \{s(u)w(u, v): u \in N(v)\} \geq 0 \\ s(v) & \text{if } N(v) = \varnothing. \end{cases} \tag{1.1}$$

Threshold networks can simulate every boolean circuit and their computational power is studied in [2]. Neurons also seem to act according to a similar threshold rule and various types of neural networks have been used to simulate associative memory [1]. The main difference between neural networks and boolean circuits is the time at which the vertices become active. The timing is *synchronous* if all vertices become active simultaneously. The timing is *asynchronous* when only one vertex becomes active at a time. The network is *symmetric* if $(u, v) \in E$ implies $(v, u) \in E$ and $w((u, v)) = w((v, u))$. Poljak and Sura [4] showed that symmetric threshold networks, when run *synchronously*, reach a cycle of size at most two. Using an "energy" function one can show that symmetric networks, when run asynchronously, must always reach a *stable state*, i.e., a state where no vertex will change its sign once it becomes active. In the general case, however, there are simple examples of threshold networks with no stable states at all. (A directed odd cycle with all edge weights negative is such an example.)

The situation changes radically if all edge weights are positive. Our main result, stated below, is that in this case our network can always reach, under a suitable asynchronous run, a stable state with at most one sign change per vertex. This result was conjectured by G. Miller [3].

Theorem 1.1. *Let N be a threshold network with positive edge weights and arbitrary initial sign values. Then there is an asynchronous run with at most one sign change per vertex which leads N to a stable state.*

We prove this result in Section 2. Section 3 contains some concluding remarks.

2. The Proof

Let $N = (V, E)$ be our threshold network where $w((u, v)) > 0$ for all $(u, v) \in E$. Consider the following algorithm for constructing an asynchronous run of N.

Algorithm 1: Stabilize (N)

1. While there is some $v \in V$ that needs to change its sign from -1 to $+1$ do:
   activate the first such v
   end.
2. While there is some $v \in V$ that needs to change its sign from $+1$ to -1 do:
   activate the first such v
   end.

We need the following simple lemma.

Lemma 2.1. *N reaches a stable state via Algorithm 1. During the algorithm each node changes its sign at most twice.*

Proof. For $v \in V$ let $s(v)$ denote its sign in the end of the algorithm and define $s'(v)$ by (1.1). We must show that $s(v) = s'(v)$ for all $v \in V$. If $v \in V$ and $s(v) = +1$ this is obvious, since otherwise $s'(v) = -1$ and step 2 of the algorithm has not yet been completed. Assume that $s(v) = -1$ and $s'(v) = +1$ for some $v \in V$. Notice that during step 2 of our algorithm the only sign changes that occur are from plus to minus. Since all edge weights are positive we conclude that v certainly needed to change its sign to $+1$ also before step 2 or any part of it. Hence at the end of step 1 the sign of v should have been $+1$, and this sign should have remained unchanged during step 2, contradicting the hypothesis $s(v) = -1$. Hence our assumption is false and N reaches a stable state via Algorithm 1. The second part of the lemma is obvious. □

Algorithm 1 supplies an asynchronous run of N, in which first the vertices of some subset $A \subseteq V$ change their signs from -1 to $+1$ and then the vertices of $B \subseteq V$ change their signs from $+1$ to -1. Put $C = A \cap B$ and suppose $C \neq \emptyset$. The vertices of C are the only ones that change their signs twice during the algorithm. Consider now the following new algorithm, that depends on A and C.

Algorithm 2: Stabilize (N, A, C)

1. While there is some $v \in A - C$ that needs to change its sign from -1 to $+1$ do:
   activate the first such v
   end.
2. While there is some $v \in V$ that needs to change its sign from $+1$ to -1 do:
   activate the first such v
   end.

Lemma 2.2. *N reaches a stable state via Algorithm 2. The vertices that change their signs during step 1 of the algorithm form a subset of $A - C$ and those that change their signs during step 2 form a superset of $B - C$.*

Remark 2.3. The stable state reached by Algorithm 2 can be different from the one reached by Algorithm 1. An example is given in Section 3. The same example shows that it is possible that some vertices change their sign twice in Algorithm 2.

Proof of Lemma 2.2. The only difference between step 1 of Algorithm 1 and that of Algorithm 2 is that in the second algorithm we do not allow to change the signs of vertices in C from -1 to $+1$. Therefore, every plus at the end of step 1 of the second algorithm is certainly a plus at the end of step 1 of the first algorithm. However, this implies, (since all edge weights are positive), that every vertex in $B - C$ changes sign from plus to minus during step 2 of Algorithm 2. This proves the second part of the lemma, and shows that any vertex that has a minus sign at the end of Algorithm 1 has a minus sign at the end of Algorithm 2 as well. Thus, certainly no $v \in C$ needs to change its sign from minus to plus at the end of Algorithm 2, since this v was satisfied with its minus even at the end of Algorithm 1, where at most as many minuses were around. It remains to show that no $v \in V - C$ needs to change its sign at the end of Algorithm 2. This certainly holds, as in the proof of Lemma 2.1, for $v - s$ whose sign is plus. Assume $v \in V - C$ has a minus sign and needs to change it to a plus. Then, as in the proof of Lemma 2.1, its sign was minus also at the end of step 1 of our Algorithm 2. If $v \in A$ we would change the sign to plus then, hence $v \notin A$. But then v had a minus sign during all Algorithm 2, and also during all Algorithm 1. Since it was satisfied with its minus at the end of Algorithm 1, it is certainly satisfied at the end of Algorithm 2, where there are at least as many minuses. This completes the proof of the lemma. □

Note that the proof of Lemma 2.2 does not depend on the special choice of the sets A, $B \subseteq V$ done in Algorithm 1. For any subsets A, $B \subseteq V$, and $C = A \cap B$, suppose that we have an asynchronous run of N in which first all vertices of A change their signs from minus to plus and then all vertices of B change their signs from plus to minus. Suppose, further, that N reaches a stable state in this run. Then a close look at our last proof shows that it implies that the algorithm Stabilize (N, A, C) will satisfy all the assertions of Lemma 2.2.

We can now prove Theorem 1.1. The algorithm Stabilize (N) supplies an initial asynchronous run of N to a stable state, where at first the vertices of $A \subseteq V$ change signs from minus to plus and then the vertices of $B \subseteq V$ change signs from plus to minus. Put $A \cap B = C$. If $C = \varnothing$ we have our desired run. Else consider the algo-

rithm Stabilize (N, A, C) and let A', B' be its corresponding A, B subsets. By Lemma 2.2, in this algorithm N reaches a stable state, and since $A' \subseteq A - C, |A'| \leq |A| - 1$. By repeatedly applying the process, (which must terminate since $|A|$ decreases by at least 1 in each iteration) we obtain an asynchronous run with at most one sign change per vertex, in which N reaches a stable state. □

3. Concluding Remarks

1) Note that the proof of Theorem 1.1 supplies an efficient algorithm to find an asynchronous run of the desired type. As an illustration of the proof (and as promised in Remark 2.3) we apply our method to find an asynchronous run of the desired type in the following network N on the vertices $\{1, 2, 3, 4, 5, 6\}$, given in Fig. 1. (All edge weights are $+1$.)

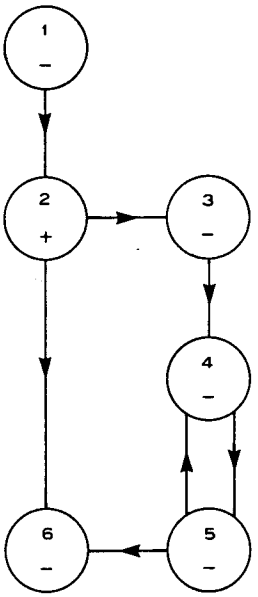

Fig. 1

By the algorithm Stabilize (N) we get Fig. 2(a) in the end of step 1 and Fig. 2(b) in the end of step 2. Hence $A = \{3, 4, 5, 6\}$, $B = \{2, 3\}$, $C = A \cap B = \{3\}$.

By the algorithm Stabilize (N, A, C) we get Fig. 3(a) in the end of step 1 and Fig. 3(b) in the end of step 2. Here $A' = \{6\}$, $B' = \{2, 6\}$ and $C' = A' \cap B' = \{6\}$.

By the algorithm Stabilize (N, A', C') we get Fig. 1 in the end of step 1 and Fig. 3(b) in the end of step 2. This is an asynchronous run with at most one sign change per vertex.

2) As we have already mentioned, the situation changes radically when we allow

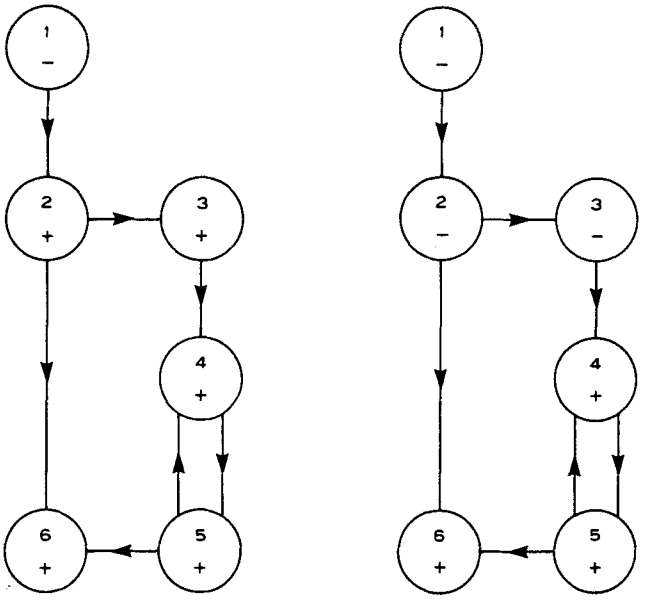

Fig. 2

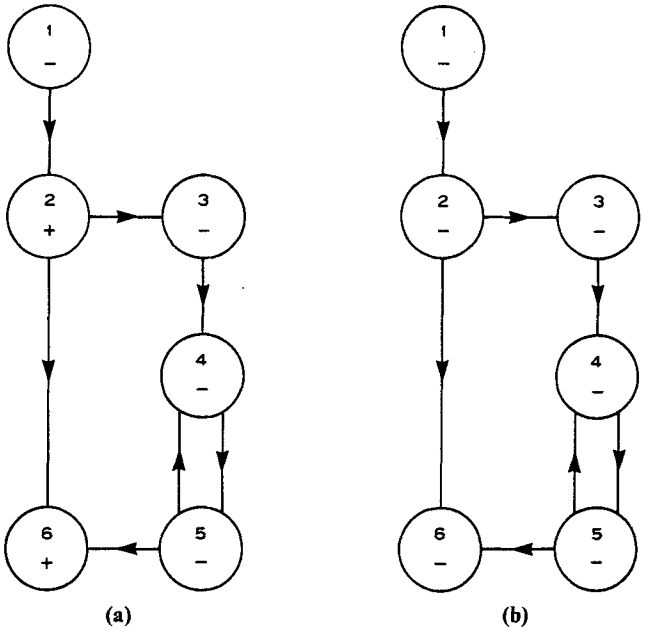

Fig. 3

also negative edge weights and there are such threshold networks with no stable states at all. We can show that the decision problem: "given a threshold network N, decide whether it has a stable state" is NP-complete. Similarly, the problem: "given a threshold network N with initial sign values, decide whether it can reach a stable state by some asynchronous run" is NP-complete. We omit the details.

References

1. Hopfield, J.J.: Neural networks and physical systems with emergent collective computational abilities. Proc. Natl. Acad. Sci. USA, **79**, 2554–2558 (1982)
2. Lepley, M., Miller, G.: Computational power for networks of threshold devices in an asynchronous environment (to appear)
3. Miller, G.: Private communication
4. Poljak, S., Sura, M.: On periodical behavior in societies with symmetric influences. Combinatorica **3**, 119–121 (1983)