Computational Power for Networks of Threshold Devices in an Asynchronous Environment

Margaret Lepley & Gary Miller

Department of Mathematics
Massachusetts Institute of Technology
Cambridge, Massachusetts 02139

Abstract: A boolean circuit is a network of threshold devices which is run either synchronously or asynchronously in an analog fashion. If any gates were to sit idle for long periods of time, the computation might be incorrect. We will examine the behavior of such networks when they are run in various types of asynchronous environments. Different types of asynchronous behavior change the computational power of a circuit. We exhibit polynomial size circuits which when run in a random asynchronous environment simulate PSPACE machines with only exponentially small probability of error. We conjecture that worst case asynchronous behavior will work only on problems in DTIME(N).

Keywords: asynchronous, boolean circuit, double-rail logic, latch, network, threshold device, PSPACE-complete, DTIME(N).

1. Introduction

All boolean functions can be composed of *and, or,* and *not,* which are threshold functions. Neurons also seem to act as threshold devices, and various types of neural networks have been used to simulate associate memory [2]. The difference between neural networks and boolean circuits is the time at which the processors respond to their inputs. When a network is highly asynchronous, the result is dependent on the timing of the processors relative to each other.

General networks of threshold devices can be depicted by directed graphs with weighted edges. These graphs have varing behaviors depending upon whether the weights of the edges are positive or negative, whether the edges are directed, and whether the devices update synchronously or asynchronously. We will show how powerful these networks are under several combinations of these conditions. Inparticular we will show that *any* PSPACE computation can be performed with exponentially small error on a network of threshold devices with random asynchronous update.

The remainder of the paper is divided into three sections. In Section 2, we describe the mathematical model and some previous results in greater detail. In Section 3, we restrict our study to the asynchronous environment. We conclude in Section 4, with a table of results and related problems.

This research was supported in part by the National Science Foundation Grant NSF CS 80-07756 and the Air Force Office of Scientific Research AFOSR-82-0326.

2. Preliminaries

2a) Mathematical model

A *threshold device* acts like a simple step function. For each device, d, there is an associated threshold, T_d. For any input vector, $\boldsymbol{x}$, the output is

$$s_d(x) = \begin{cases} a & \text{if } \Sigma x_i < T_d \\ b & \text{if } \Sigma x_i \geq T_d. \end{cases}$$

For example, in *and* and · or gates: $a=0$, $b=1$, $T_{or} = 1$ and $T_{and} = \text{fan-in}$.

A *network* of N of these processors can be thought of as a directed graph where each node is a threshold device and an incoming edge denotes an input to the device. Every edge is assigned a fixed weight, $w_{ij} \in R$, and each node is labelled with a binary value, s_i, the output of that device. Since the edge weights are real, we can assume without loss of generality that $s_i = \pm 1$. Furthermore all the thresholds can be changed to zero by introducing "forcing" nodes (see Fig.1). The inputs to processor j are the products, $s_i w_{ij}$, from the incoming edges.

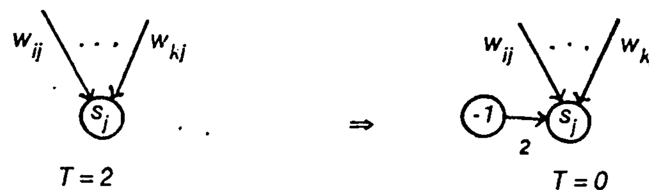

Figure 1: *Changing the threshold to zero by adding a "forcing" node.*

The values of all the nodes, therefore change according to the following rule

$$s_j = \begin{cases} -1 & \text{if } \Sigma s_i w_{ij} < 0 \\ +1 & \text{if } \Sigma s_i w_{ij} \geq 0. \end{cases}$$

But the nodes do not update their values continuously. Instead each node is "dormant" for a period of time before becoming active for an instant. The relative times at which different nodes are active will control the behavior of the network. The timing is *synchronous* if all the nodes become active simultaneously. The timing is *asynchronous* when only one node becomes active at a time.

2b) Previous results

If $w_{ij} = w_{ji}$ for all i,j then the graph is said to be *undirected*. Polak & Sura[3] showed that undirected graphs, when run synchronously, reach a cycle of size at most two. Using an "energy" function, it is also easy to see that undirected graphs when run asynchronously, must always reach a *stable configuration* where no node needs to change. In fact, if all the edge weights are positive then a stable configuration can be reached quickly. The following result by Alon[1] shows how this can be done even in the directed case.

Proposition: *A directed network with only positive edge weights can reach a stable configuration via this algorithm:*

1. *Activate all nodes that need to change to +1.*

   When there are no more nodes needing to become +1, then

2. *Activate all nodes that need to change to −1.*

This algorithm requires at most 2N value changes.

When the graph is directed the situation changes drastically. One can easily construct from a PSPACE Turing machine and its input a poly-sized circuit with feedback, composed entirely of *and* and *or* gates and *negations* which when run synchronously, simulates the Turing machine. Obviously these networks can have large cycles. Since the timing is synchronous, all negations can be removed by using *double-rail* logic [4]. In double-rail logic each line is doubled and $01 \equiv 1$ and $10 \equiv 0$. We have just shown that synchronous networks with only positive edges are PSPACE-hard. Since any network of threshold devices running synchronously can easily be simulated in poly-space, we see that directed graphs with non-negative edge weights are PSPACE-complete.

Asynchronous directed networks though behave very differently when all the edges are non-negative. By the above Proposition a stable configuration can still be reached with at most one change per node. When negative edges are present cycles can appear and the network never reaches a stable configuration. We will now discuss the computational ability of these types of networks.

3. Directed Networks in an Asynchronous Environment

There are two types of asynchronous behavior that we will examine here. First is *random asynchronous* behavior, where each node that needs to change has equal probability of becoming active at any time. This means that there is a possibility of some node remaining unchanged long enough to invalidate a computation with feedback. The second type of asynchronous behavior is a type of worst case analysis. We would like these machines to be able to run for a long time without cycling. An adversary will try to choose the order in which the nodes become active so that the computation stabilizes or cycles quickly. The question then is: How powerful is the network when an adversary is running it?

3a) Random asynchronous update

When all the edges have positive weights then it is always possible to reach a stable configuration. The number of value changes when the active nodes are chosen at random can be larger than linear. For instance the network in Fig.2 needs $O(N^2)$ changes on average with random asynchronous updating.

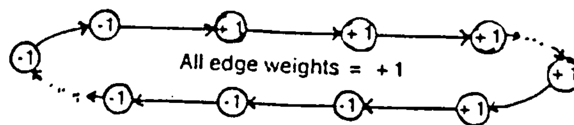

Figure 2: *Random asynchronous network averaging $O(N^2)$ changes to reach stability (random walk).*

Negative edges allow a network to enter a cycle. Simple clocks can be made from these cycles, so it is possible to perform PSPACE computations with high accuracy.

Theorem 1: *For any $L \epsilon PSPACE$ and for any allowable error $\varepsilon > 0$ there is a network of threshold devices of poly-size which decides L with error at most ε.*

Proof sketch: As was remarked before, there is a poly-sized logical circuit, C, which simulates a Turing machine to decide L. Asynchronous update will produce errors in the feedback loop. One possible solution to this problem is shown in Fig.3.

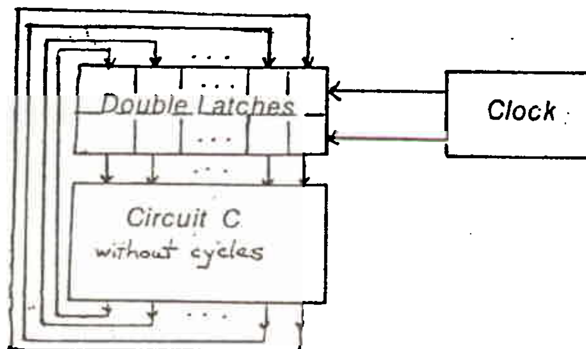

Figure 3: *Random asynchronous network with high probability of simulating logical circuit C.*

This circuit would work perfectly if we could be sure that the latches and the clock always performed their functions correctly. Unfortunately this is not the case. The latch in particular creates problems. Fig.4a shows a circuit which acts as a latch under normal circumstances.

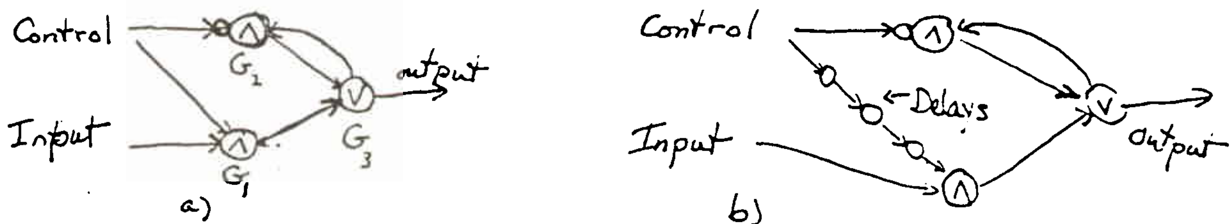

Figure 4: *a) A latch for a synchronous circuit*
b) A latch for a random asynchronous network.

But if G_1 and G_3 get updated before G_2 when the control variable changes from *1* to *0*, all information is lost and only *0* can be output.

To make this occurance extremely unlikely we add some delay nodes on the edge leading from the control node to G_1. Now we need to know how long a delay is needed and whether we have enough time to add it to the circuit.

Lemma 1: *A latch will perform correctly for an exponential number of repetitions with a delay line of polynomial length.*

Proof: This proof makes use of the fact that each node is equally likely to become active at any time. So the dormant time between activities is exponentially distributed. If the delay line has length D then the probability that G_1 is active before G_2 is

$$\int_0^{\infty}\int_x^{\infty} x^D e^{-x}/\Gamma(D+1) * e^{-y} \, dy \, dx = 2^{-D-1}$$

To perform the computation correctly the latch must work at each iteration, so after an exponential number of iterations the error must still be exponentially small. The above statement produces a D a polynomial. □

In Fig.4 there are double latches so that there won't be feedback while the latch is trying to sample the new value. Because of this we need a clock with four different outputs. Fig.5 shows one possibility for such a clock.

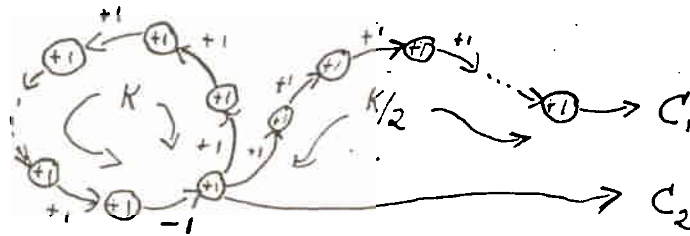

Figure 5: *A clock with four different nearly equal length periods.*

The length of the clock cycle will be a function of the depth and the width of main section of the circuit, as well as the number of latches and the error allowed. By an analysis similar to the one in Lemma 1 we can show that

Lemma 2: *A clock that will ensure that random asynchronous update runs this circuit with only exponentially small error, has at most polynomial size.*

This finishes the sketch of the proof of Theorem 1. That is, any problem in PSPACE can be done with only exponentially small error on a network of threshold devices if random asynchronous updating is used.

3b) Worst-case asynchronous update

A network with only non-negative edges is not very powerful under worst-case conditions.

Theorem 2: *Worst-case asynchronous networks with no negative edges are a subset of DTIME(N).*

Proof: The algorithm in the Proposition requires only $O(N)$ changes. This can be simulated by a Turing machine in linear time. □

Unfortunately double-rail logic does not lend itself well to asynchronous systems, so the negative edges cannot be eliminated in this manner. When negative edges do occur we believe the worst case behavior is still linear. By that we mean that only a linear number of changes are required to reach a cycle and the cycle length is $\leq N$.

4. Conclusions

The following table summarizes the results we have so far for directed networks of threshold devices.

Edges \ Timing	Synch.	Asynch.	
		random	worst-case
Non-negative	PSPACE-complete	$\Omega(N^2)$	DTIME(N)
Real	PSPACE-complete	PSPACE with ε error	?

Better bounds are needed in both the positive edge, random asynchronous environment and the worst-case asynchronous environment with both positive and negative edges.

In undirected networks, we still do not know how many changes are necessary in either worst -case or random asynchronous environments. Besides determining the number of changes in the values s_i, it would be useful to know what the parallel computational time is for a network of threshold devices.

Acknowledgements

We would like to thank Noga Alon, and Bruce Bayly for many helpful discussions.

References

[1] Alon,N., Personal communication.

[2] Hopfield,J.J., "Neural networks and Physical systems with emergent collective computational abilities," *Proc Natl Acad Sci USA*, Vol.79, April 1982, pp. 2554-2558.

[3] Poljak,S., Sura,M., "On Periodical Behaviour in Societies with Symmetric Influences," *Combinatorica*, Vol.3, No.1 (1983), pp. 119-121.

[4] Leiserson,C., Personal communication.