SANTA FE INSTITUTE &
UNIVERSITY OF NEW MEXICO

EXTENDED ABSTRACT

Analyzing Average-Case Properties of Boolean Functions via Random Circuits

Author:
Ronnie GARDUÑO
Department of Computer Science, UNM
& Santa Fe Institute
ronnie.garduno@gmail.com

Primary Investigator:
Dr. Simon DEDEO
Department of Informatics, Indiana University
& Santa Fe Institute
sdedeo@indiana.edu

March 28, 2014

1 Introduction

In this work, we present a new method for studying the structural complexity of Boolean functions. This method depends upon the ability to generate random circuits of a given size and number of inputs. Using this ability and tools from statistical mechanics, we shed some light on a particular class of Boolean function, the 'shuffled PARITY' functions. The presence of phase transitions in this class' computational power (measured against certain functions) suggests a deep connection between the statistical properties of these circuits and the inherent difficulty of the various functions under analysis. These results could have far-reaching consequences; our method may help us understand the amount of resources a given problem really needs to be solved, as well as the amount required to solve the problem efficiently. This could be important to any field which uses computers, especially those that use embedded systems, which must be small and efficient to achive their intended goals. In the following sections, we present the basic method we are using, as well as the theoretical justification for it.

2 Methods

In this section, we describe the basic method we are using to study the Boolean functions we are interested in. There are a lot of different steps in the process, but it all begins with a setup phase. In this phase, we must choose which Boolean function we are studying, as well as the number of inputs and gates in the circuits we wish to analyze the function with. The function we are studying is called the **target function**. We must also choose a **basis function** for the nodes to compute with.

2.1 Boolean circuits

A **Boolean circuit** is an abstraction of the type of circuits used in electronics design. These are represented by DAGs (directed acyclic graphs) with the given number of input nodes and intermediate 'gate' nodes, linked together by randomly-chosen connections which determine the workings of the circuit. The input nodes are represented by **sources**, nodes which do not have any incoming edges, and the singular output is represented by a **sink**, which is a node without any outgoing edges. There may be multiple sinks in any given DAG, but there is always one which is designated as the output for the entire circuit. Every gate has exactly two incoming edges in our method. Each gate's outgoing edges carry the output of the basis function computed on the two incoming edges to the gate. To interpret a given DAG as a Boolean circuit, then, we assign an input value (0 or 1) to each input node, which is passed on to the outgoing edges from each input. Next, for each gate with incoming edges from the inputs, we compute the output of the gate as the result of computing the basis function with the inputs chosen. This process is repeated with every gate until the output gate is reached, and the output of that gate is treated as the output of the circuit acting on the given inputs. This is a particular way of representing a computation, and any computation may be represented in this way, given enough gates and a sufficiently-powerful basis function. In this work, we mostly use the basis function NAND, which is functionally complete ([2]), and can therefore compute any computable function with enough space, but we also use XOR for comparison purposes.

2.2 Circuit ensembles

Once the setup phase is complete, our method proceeds by exploring the **circuit ensemble** defined by the choices we made in the setup phase. A statistical ensemble is a mathematical object, invented for use in the field of statistical mechanics, composed of all of the possible arrangements of the parts in a given system. For us, the ensemble is made up of all of the different circuits that can be formed by using the given number of inputs and gates, with each gate computing the given basis function on the two incoming edges assigned to it. The ensemble of circuits with I inputs and N gates using the basis function BOOL is written as $E_{I,N}(\mathrm{BOOL})$. Therefore, the setup phase defines the ensemble we will be exploring in any given run of our method, as well as the target function we are analyzing. The next step is to actually explore this ensemble.

2.3 MCMC sampling

Our strategy for exploring a given circuit ensemble is to utilize the Markov chain we've developed in an **MCMC** (Monte Carlo Markov Chain) sampler. We have proved that the Markov chain in question is 'ergodic', which essentially means that it samples from the complete space of circuits in the ensemble, and that it does so uniformly. The Markov chain works in the following manner. First, we need to start by generating some circuit in the ensemble. This is easily achieved by labeling each of the gates in the circuit with a number from 1 to *I* + *N*, starting with the inputs and ending with the output node. We then assign to each gate (the nodes with labels greater than *I*) two incoming edges, chosen randomly from the set of all gates with labels less than that gate's label. In this way, we can generate a DAG that fits into the ensemble, since this method ensures that there are no cycles in the resulting graph.

Once we have this initial graph, we proceed by performing a move we have called the *Root-Flip/Tip-Complement*(RFTC). The *RFTC* move works by randomly choosing a random gate (again, a node with a label greater than *I*) from the circuit, then randomly picking one of its two incoming edges. We then randomly choose another gate in the circuit and make sure that it does not lie along any directed path from our first gate to the output node. Once we have this gate, we simply change the edge we picked earlier by moving the root, which is the end that determines where the edge comes from, to come from the new gate instead. This process gives us another DAG in the same ensemble. We keep this new

DAG with a probability determined by its weighting in the partition function of the system, as we will describe in the next subsection. If we do not keep the new DAG, then, we simply keep the old DAG and treat it as a new sample.

The MCMC algorithm we are using, then, works by performing the *RFTC* move repeatedly, storing the fitness of each graph we generate in this fashion in a big list. This gives us a random sample from the ensemble, which is important because the ensembles can grow to contain many graphs. They can easily be so large that simply enumerating every member of the ensemble would be computationally infeasible, i.e., it would take too long. Once we have this random sample of graphs, we can compute an estimate of the **partition function** for the ensemble.

2.4 Partition functions

The partition function of a statistical ensemble is a function computed from the energy values of each member in that ensemble. This function is of the form $\sum_i e^{-\beta E_i}$, where *i* ranges over each member of the ensemble and E_i is the energy of the i^{th} member. In this application, we choose to make the 'energy' of a circuit equal to (the negative of) its 'fitness' with respect to our target function. The **fitness** of a given circuit is calculated by evaluating the circuit's outputs for every possible set of input values, then counting the number of these outputs which match the output of the target function over the same inputs.

We use the Boltzmann function to construct our partition function, which means the following. For each circuit *C* with fitness *F*, we add to the overall partition function a term of the form $e^{-\beta F}$, where β is an inverse 'temperature' variable.

This term, then, is the probability of transitioning to circuit *C* from an adjacent circuit using our *RFTC* MCMC sampler. In this context, the 'temperature' determines how much weight the sampler places on circuits with higher fitness. A higher value for β gives us a higher probability of transitioning to circuits that have higher fitness than others in the same ensemble.

For the purposes of this work, we wish to estimate the true partition function, which has one term for every distinct circuit in the ensemble. Therefore, we set $\beta = 0$ in our sampler, giving every circuit the same weight and therefore inducing the sampler to take a uniform sample over the entire ensemble. The purpose of computing these partition functions, in this case, is to enable us to search for phase transitions in the sampler's output, which are of some interest.

2.5 Phase transitions

The notion of a **phase transition** is another tool from statistical mechanics. The idea is that a phase transition denotes a point where the microscopic details of the system in question change is such a way that it results in a qualitative macroscopic change in the properties of the system. The most familiar phase transition in everyday life is the freezing of water, but there are countless other examples. In this work, then, phase transitions are changes in the circuits being sampled by our algorithm that denote important changes in the overall sample.

A true phase transition is said to take place at temperatures where a derivative of the (logarithm of the) partition function has a discontinuity. This can only take place when the system being analyzed has an infinite number of components. Unfortunately, we cannot sample an infinitely large ensemble with any precision. Therefore, we sample from finite ensembles instead, and point out the location of local maxima as being points where there *appears* to be a phase transition. The locations of these phase transitions are the overall outputs of the general method, and they are called critical β values. This method should work for any target function and basis function, so how do we choose which problems to study? The theoretical basis of our choices are briefly outlined in the next subsection.

2.6 PARITY and SPARITY

In the field of circuit complexity, researchers study the computational complexity of various Boolean problems by looking at the circuits which can and cannot implement those problems ([3]). One of the major results in the field says that PARITY, the question of whether the number of inputs with value 1 is even or odd, requires a circuit whose depth grows at least logarithmically in the number of its inputs ([1]). This result gives a lower bound on the difficulty of computing the PARITY function, which is a rare thing. In this work, the problems studied are all from a set of problems we call SPARITY, or 'shuffled PARITY', which are formed by taking the bits in PARITY's truth table and shuffling them around to different input values. The main thing that can be said of these problems currently is that they are all *balanced*, which is to say that they have the same number of 0s and 1s in their truth tables. The work is still in progress, but the next section describes some of the results this method has yielded for us thus far.

3 Results

As stated previously, what we get from the application of our method right now are critical β values for each ensemble we analyze. We have computed these results for several ensemble groups. The first two groups are the set of ensembles with 2 inputs and 2-15 gates, one using NAND and the other using XOR. There are six SPARITY functions at this size, but only two are shown here. The phase transition locations are plotted along with the function values associated with them.

There are many other plots like these, but we have not analyzed them all yet.

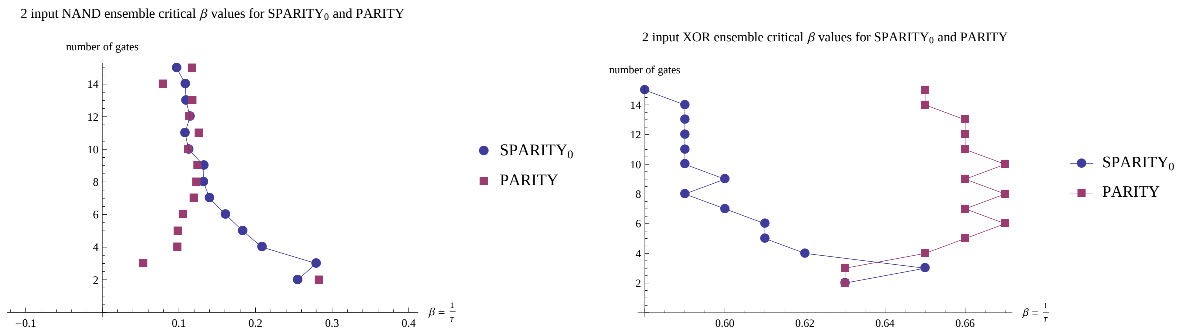

Figure 1: Phase diagrams

4 Discussion and Conclusion

The preliminary results presented in the previous section contained an interesting surprise. Looking closely at the plot for the ensembles of 2 inputs, 2-15 gates NAND circuits, one will discover that there are some sizes for which there are, not just one, but *three* critical β values. This implies that there are even more than the expected two phases, and, therefore, that the behavior of our sampler when sampling NAND circuits against the PARITY function is quite complex. We are working to characterize these phases with respect to some value like average fitness. What we are looking for in the mass of data is a type of SPARITY function whose critical β values are consistently either higher or more interesting than PARITY's. We have not yet found this, but if we do, it would imply the existence of Boolean functions which are harder than PARITY. This could help researchers clarify the bounds between different circuit complexity classes, and could even provide a way to try to separate P and NP, a famously difficult problem ([1]). Another potential application for this work would be to compare natural systems to each other in terms of this work's β values. We could find some system which has been analyzed down to the level of a circuit, then see what β value the sampler would need to find this circuit. These comparisons may help theoretical biologists, for one, to better understand the complexity of the systems they study.

5 Acknowledgements

The author would like to acknowledge the huge amount of help provided by his mentor, Dr. Simon DeDeo at the Santa Fe Institute. This work owes its very existence to his suggestions and guidance. The author is also grateful for stimulating discussions on this work with Kirstin Harriger, Christopher Aicher and Robert Hawkins. The author would also like to thank the NSF (National Science Foundation) for supporting this work, first in the REU (Research Experience for Undergraduates) program, then later through Dr. DeDeo's grant #EF-1137929, "The Small-Number

Limit of Biological Information Processing."

References

[1] Sanjeev Arora and Boaz Barak. *Computational Complexity: A modern approach*. Cambridge Univ Press.

[2] Herbert B Enderton. *A Mathematical Introduction to Logic*. Academic Press, University of California, Los Angeles.

[3] Heribert Vollmer. *Introduction to Circuit Complexity*. Springer-Verlag.