Analog Boolean Satisfiability

Jared Smith

University of Tennessee, Knoxville

jms@vols.utk.edu

Contents

Abstract

Analog computation is rooted in the early period of electronics and computer technology. Once at the pinnacle of computation in the early 20th century, analog computation is now largely obsolete for conventional and particularly personal computing, yet it remains a promising area of research for computer scientists involved in advanced computational methods. Analog computation operates on the set of all real numbers and draws its power from this, rather than the discrete quantities used in digital computation. Additionally, analog computers perform very well in massively parallel systems, use comparatively small amounts of power, and disperse less heat. All of these properties combined make analog computation promising for a diverse array of applications, including simulating the human brain and building highly efficient artificial neural networks and constructing complex robotics systems. In this report, I present another application of analog computation, solving boolean satisfiability (SAT) problems, which is the basis for all NP-Complete problems. We present an overview of analog computation and its history, a study of the classic K-SAT problem, an algorithm for solving these problems involving the use of a continuous time artificial neural network, an overview of the analog circuit and the software simulation that implements this algorithm, and results from the demonstration of this software.

I. Introduction

Analog computers are machines that enable a user to reason about a complex physical system through interacting with another physical system, which is analgous to the first physical system. Distinct from digital, analog computing was an alternative form of computer technology that had varying degrees of success. In general, analog computers store data as a continuous physical quantity and perform computations by manipulating measures that represent numbers. On the other hand, a digital computer operates on symbolic numerals that represent numbers and operate on discrete quantities. This distinction makes analog and digital computing vastly different with differing benefits and drawbacks to each [6].

Analog computing flourished in the early 20th century including advancements and uses such as Nordsieck's differential analyzer and NASA and Lockheed Martin using Analog computers for its Missle and Space program, but it would be gradually replaced in most systems by digital computers following advancements in digital computing models like the Turing Machine. However, in the 1990's, analog computing regained popularity for several areas of research in computer science and advanced computational models. This rise in popularity led to the work that will be described in this paper as analog computation is used to more effectively represent a dynamical model of a neural network for solving constraint satisfaction problems than traditional digital hardware [5].

Analog computation can more effectively model this type of problem because the problem we will present can be solved using a type of neural network represented as a continuous dynamical system of equations. One of the advancements in analog computation research that started in the 1990's was using analog computation for modeling neural networks, since neural networks are very well represented by analog models of computation [4]. We will first discuss the history of analog computation, then we will explore work by Molnár and Ercsey-Ravas [5] that describes an asymmetric continuous time neural network to solve constrain satisfaction problems, then we will explore an analog algorithm for this model by Molnár implemented in both software and hardware and results from the simulation of this software, and finally we will discuss future work in this area of research.

II. History of Analog Computing

I. The Antikythera mechanism

The Antikythera mechanism was found housed in a 340 mm x 180 mm x 90 mm wooden box in 1901 in a shipwreck off the Greek island of Antikythera, the device is a complex clockwork mechanism composed of at least 30 interacting bronze gears.

This device represents the earliest known analog computer. It is suspected to have been used to predict astronomical positions for calendrical purposes. Other uses were for the Olympiads, the cycles of the ancient Olympic Games [2].

II. Nordsieck's Differential Analyzer

Much later in the 20th century, Arnold Nordsieck started building a differential analyzer when many other computing researchers were experimenting with vacuum-tube digital electronic computers, one of the earliest forms of digital computers. Nordsieck needed a practical machine

Figure 1: *The Antikythera mechanism from the Ancient Greek Civilization*

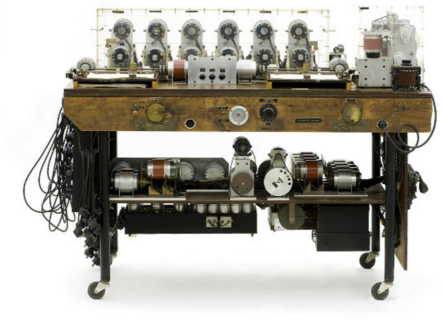

Figure 2: *Nordsieck's Differential Analyzer*

that was simple to use and cheap to build. Thus, he came up with his differential analyzer. He used $700 worth of surplus World War II supplies, Arnold Nordsieck assembled an analog computer in 1950. His device, shown in Fig. 1, was modeled on differential analyzers built since the 1930s. However, it had some key differences.

For example, Nordsieck's computer used electrical connections instead of mechanical shafts. His device was just as he imagined, simple, cheap, practical, and could be used as a general-purpose computing device. The differential analyzer was a key milestone on the path to analog computing, though it began much earlier with the integrator, an important component of analog systems.

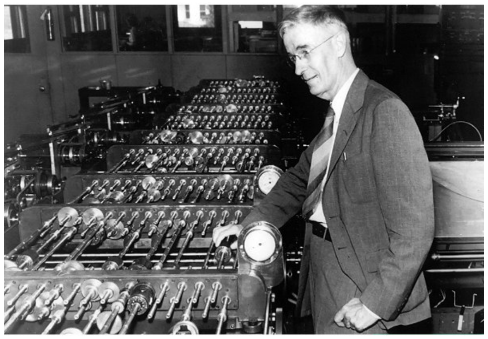

Figure 3: *Vannevar Bush's differential analyzer*

III. Planimeters, Integrators, and Bush's Differential Analyzer

Johann Martin Hermann built an early planimeter, a type of integrator, in 1814. These didn't gain a large amount of use until Casper Wetli built a version based on a wheel-and-disk integrator in 1849. In 1876, Scottish engineer James Thomson introduced a simplified integrator. At tha time, he also coined the term "integrator." Thomson's brother, Lord Kelvin, recognized his brother's instrument's value as a general-purpose mathematical device.

Vannevar Bush joined MIT at age 29 as an electrical engineering professor and led the design of the first differential analyzer. Bush "was trying to solve some of the problems of electric circuitry... I was thoroughly stuck because I could not solve the tough equations...". He didn't abandon his task for lack of a tool, rather he invented a new tool. The MIT professor created a differential analyzer to model power networks in 1931, and he quickly saw its value as a general-purpose analog computer, which would be one of the many sparks that would start the fire for analog computing's rise in popularity.

Bush's Differential Analyzer filled a room with a complicated array of gears and shafts driven by electric motors. Wheel-and-disc "integrators" at its heart could be connected to 18 long, rotating shafts. The project was started in 1928 by Bush's student Harold Hazen, the machine could solve, approximately, an arbitrary sixth-order differential equation. However, it had to be meticulously set up for each new problem. In addition to analyzing power transmission networks, Bush's analyzer solved problems in many other fields including physics, seismology, and ballistics. It inspired similar devices that were used in the US, Britain, Europe, the Soviet Union, and Australia

Figure 4: *George Philbrick's K2-W op-amp*

for many years to come.

IV. Operational Amplifiers

Later, the operational amplifier, or "op-amp", a key component of an electronic analog computer, was invented in the early 1940s and allowed for mechanical contraptions to be replaced by silent and faster electronic hardware.

An op-amp is a high-gain voltage amplifier with differential inputs. Using the correct negative feedback, a single op-amp can add or subtract two voltage signals, multiply by a constant, or integrate voltage over time. By chaining together many op-amps the machine can compute complicated formulas using continuous values.

Many analog computers relied on vacuum-tube op-amps, available commercially from George A. Philbrick's company in 1952. In 1963, Bob Widlar at Fairchild Semiconductor made an op-amp on a single integrated circuit, which was a major advancement in engineering at the time. Widlar's 1965 μ A709 became a huge commercial success. He later was known as the "analog wizard-inresidence" at National Semiconductor, a large electronics company, widely known for his colorful personality.

V. To the Present

Since the time aroun the breakthroughs in analog computing in the mid-1900's, analog computing was largely replaced by digital computers for most general purpose computing. However, later in the 1990's and to the present, analog computing made a resurgence in many areas of research in advanced computational methods, ranging from modeling neural networks to building complex robotics systems. In this paper, we will explore an application of analog computers for solving constraint satisfaction problems.

III. Boolean Satisfiability and K-SAT

First, we will introduce the Boolean Satisfiability problem (SAT), since that is directly related to constraint satisfaction. The goal is to find a set of 0/1 assignments to the Boolean variables $X_1, \ldots, X_n$ so that a specified Boolean expression is true (= 1). The Boolean expressions are in *conjunctive normal form*, e.g.:

$$(X_1 + X_3 + X_4) \cdot (\overline{X_2} + X_3 + X_4) \cdot (X_2 + X_4 + \overline{X_5}).$$

In this instance, there are $N = 5$ variables and $M = 3$ *clauses*. Positive or complemented variables (e.g., $X_2, \overline{X_2}$) are called *literals*. If every clause contains exactly k literals (as in the above example, $k = 3$), then it is an instance of the k-SAT problem. k-SAT is an NP-complete problem.

The difficulty of a particular instance is related to the *constraint density*, $\alpha = M/N$ [3].

IV. ACTNN k-SAT Algorithm

I. Overview

The algorithm we present for solving k-SAT problems described above is modeled after work done by a research group led by Ercsey-Ravasz and her colleagues [5, 7]. In their work, they present a continuous-time dynamical system for solving k-SAT. This algorithm intends to solve SAT problems as detailed above.

II. The Algorithm

In the model, there are N bounded variables s_i which evolve to a solution, positive for Boolean 1, and negative for Boolean 0. In addition, there are M bounded variables a_m that measure the "urgency" of satisfying a clause. A particular problem instance is represented by an $M \times N$ constraint matrix $\mathbf{c}$, where $c_{mi} \in \{-1, 0, 1\}$. $c_{mi} = +1$ if X_i is positive in clause m, $c_{mi} = -1$ if $\overline{X_i}$ is in clause m, and $c_{mi} = 0$ if X_i does not occur in clause m.

The dynamical system is defined by the differential equations:

$$\dot{s}_i(t) = -s_i(t) + Af[s_i(t)] + \sum_{m=1}^{M} c_{mi} g[a_m(t)],$$
$$\dot{a}_m(t) = -a_m(t) + Bg[a_m(t)] - \sum_{i=1}^{N} c_{mi} f[s_i(t)] + 1 - k.$$

The *self-coupling parameters* A and B are two real constants that depend weakly on k. The f and g activation functions are piece-wise linear squashing functions that map the s and a values into $[-1, 1]$ and $[0, 1]$, respectively:

$$\begin{aligned} f(s) = &\quad (|s+1| - |s-1|)/2 &(1)\\ = &\left\{ \begin{array}{cc} \text{-1} & \text{if } s < -1, \\ s & \text{if } -1 \le s \le 1, \\ \text{+1} & \text{if } s > 1. \end{array} \right\} &(2)\\ g(a) = &\quad (1 + |a| - |1-a|)/2 &(3)\\ = &\left\{ \begin{array}{cc} 0 & \text{if } a < 0, \\ a & \text{if } 0 \le a \le 1, \\ \text{+1} & \text{if } a > 1. \end{array} \right\} &(4) \end{aligned}$$

Molnár and Ercsey-Ravasz [5] prove that the only stable fixed points of the system are solutions to the problem, and they give numerical evidence that there are no limit cycles, provided that the A and B parameters are in the appropriate range. Typical good ranges for the constants are

$1 < A < 2$ and $2 < B < 2\lfloor k/2 \rfloor + 2$. If in fact there are no limit cycles in these ranges, then unsolvable instances would have chaotic attractors. This is consistent with the observed transient chaotic behavior for hard instances [5].

Molnár and Ercsey-Ravasz [5] prove the following bounds on the variables:

$$|s_i(t)| \leq 1 + A + \sum_m |c_{mi}|,$$
$$-2k \leq a_m(t) \leq 2 + B,$$

provided that they are initially in the ranges $|s_i(0)| \leq 1$ and $0 \leq a_m(0) \leq 1$. This guaranteed bounding of variables is critical for analog implementations, because variables in analog computational systems are bounded by their physical representation. The initial values are otherwise arbitrary.

The progess toward solution can be tracked by the following "energy function" which depends on the number of unsatisfied clauses [3, 5]:

$$E[f(\mathbf{s})] = \mathbf{K}^{\mathrm{T}}\mathbf{K} \text{ where } K_m = 2^{-k} \prod_{i=1}^{N} [1 - c_{mi} f(s_i)].$$

Note that E is not a Lyapunov function, and energy does not decrease monotonically.

V. Implementation

The implementation of this algorithm was completed in both software and hardware. The software implementation was done on a traditional digital computer in two different programming languages, Python and C, and visualizations were produced for analyzing the simulation results. The hardware implementation was built on a small, 4×4 analog circuit in the popular simulation tool, Cadence. Neither implementation is near optimal, and additional work is being done to correct the imperfections on each platform.

I. Overview of Analog Implementation

Figure 5 displays an analog algorithm for implementing this dynamical system proposed by Ercsey-Ravasz. The overall structure is a cross-bar between the M integrators for the a_m and the N integrators for the s_i; thus $M + N$ integrators are required. A particular instance is programmed by

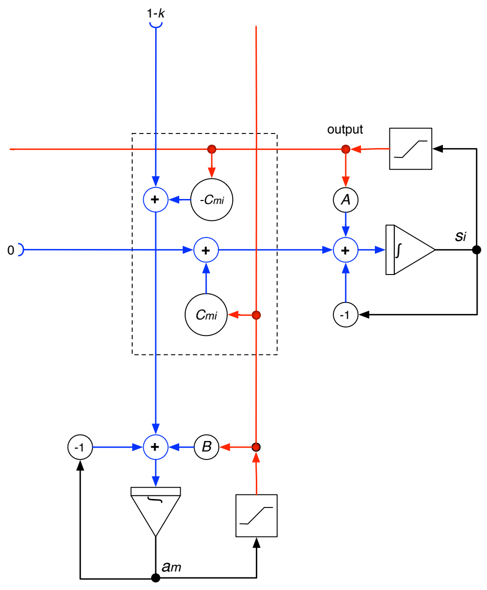

Figure 5: *Analog Implementation of ACTNN-k-SAT.*

setting the c_{mi} and $-c_{mi}$ connections to $-1, 0$, or $+1$, as required for the problem. The integrators are initialized to small values to start the computation; non-zero offset or noise in the hardware integrators might have the same effect.

This structure is easily implemented in analog form, as it comprises a small number of simple components repeated in a regular pattern. Integrators are a ubiquitous element of analog signal processing systems, with numerous switched-capacitor[10][11], continuous-time voltage-mode, and current-mode implementations [12]. Clamping, scaling, and summation are likewise easily implemented, particularly using current-mode signaling. The weights $c_{m,i}$ can easily be implemented using cross-bar switches.

Analog computation has been demonstrated to provide outstanding energy efficiency at moderate resolutions, often yielding energy efficiency improvements of multiple orders of magnitude over digital implementations [13][14]. However, with that efficiency comes imperfect computation, as every element introduces error into a calculation. These errors can be reduced, but at the cost of increased silicon area and power consumption. In order to ensure that circuit performance does not degrade algorithmic performance, while avoiding excessive power and area due to over-engineering, designers must carefully model the effects of non-ideal computation. In this paper, we address that modeling task for the ACTNN-k-SAT solver, focusing primarily on noise and offsets.

II. Software

The software implementation of the analog algorithm implements the dynamical system described above. The mathematics are defined by the model in Molnár's paper [5].

$\dot{s}_i(t)$ and $\dot{a}_m(t)$ in the algorithm are the primary equations being computed by the code. M and N are constants representing the number of clauses and variables, respectively, in the SAT problem and define the shape of C, which is represented as a two-dimensional array. S and A are also represented as one-dimensional arrays and are initialized randomly. A is initialized in the range of 0.01 to 0.1 and S is in the same range but multiplied by a random choice of 1 or -1. F and G are the activation functions and are essentially implementing as clipping functions where G constrains the matrix between 0 and 1 and F between -1 and 1.

It was implemented in both Python and C using an extensive open-source linear algebra library for Python called NumPy and a custom library in C, rather than opting for the cumbersome Boost library for C and C++. The lines of code in total for both were under 1000 lines, including the overhead of generating visualizations and parsing configuration for the simulation. Regarding performance, both perform well, though the C implementation performs much better given that it is a compiled and takes advantage of multi-threading using Linux's pthreads standard library. For visualization, the popular Python scientific library, Matplotlib, was used for graphing the results.

III. Hardware

The design was implemented for simulation as a four-by-four array in an 130nm process. The circuit is capable of solving problems with up to a maximum of four each of variables, clauses, and variables per clause, the main primitives in SAT problems. Larger problems may be solved by increasing the dimensions of the array. The circuit is fully differential, and the majority of the junctions were implemented as current summing nodes to minimize voltage swings. Current mirrors were used extensively to copy the current outputs of each cell to the inputs of the junctions across the array. The circuit operates at a VDD of $1V$ and consumes an average of 140μW. The unit of differential current was chosen to be $100nA$. The integrator was formed using a fully differential OTA with common mode feedback. A pair of $100fF$ feedback capacitors was used to achieve a slope of $7.69mV/\mu s$ for a $1nA$ differential input. A pair of series diodes from each input to ground provided the common mode bias voltage for the OTAs inputs.

The $f(s)$ function was implemented using an NFET differ- ential pair. The bias current was provided by an NFET current sink which limited the output current swing when the inputs were driven beyond the linear range.

The $g(a)$ functions implementation was similar to that of $f(s)$ except that differential output current was only allowed to change for positive input. This was accomplished by splitting the differential and common mode output currents from the differential pair. The differential current was then recombined with the common mode current through a pair of MOSFET diodes. At full positive input an NFET diode allowed up to $50nA$ to be added to the positive current output, and a PFET diode would subtract up to $50nA$ from the negative output. For negative input the diodes were cut off, so only the common mode current remained at the output.

The outputs of $f(s)$ and $g(a)$ are scaled by the constants A and B respectively to be used as feedback around their respective integrators. This was performed using scaled current mirrors. The values of A and B are 1.3 and 2.3 respectively as these were determined to be optimal for solution accuracy and time [5].

The negative unity feedback around the integrators required conversion from voltage to current, so it was implemented using a linearized transconductor. The circuit is similar to that of the f cell except that the current sinks have been split and a resistor attached between the sources of the differential pair. The common mode current was increased to allow a larger output current swing, and the transistor sizes were adjusted so that the slope was close to the linear region of the f cell. The output still saturates for a large enough input, but the limit is much larger compared to the f cell.

The $c_m i$ function was implemented using crossbar switches to connect a pair of NFET current mirrors alternately to the outputs. The inputs to the current mirrors are from the appropriate f or g signals. Two logic bits were used to control the switching action. The enable bit $c_m i_e n$ determines if the variable exists in the clause, and the polarity bit $c_m i_p ol$ determines whether or not the variable is complemented. If $c_m i_e n$ and $c_m i_p ol$ bits are both high, then the differential current is allowed through unaltered. If the $c_m i_e n$ is high and $c_m i_p ol$ is low, then the current outputs are swapped. If $c_m i_e n$ is low, then $c_m i_p ol$ is ignored and the outputs of both current mirrors are split between both outputs. This effectively cancels the differential current output and leaves the common mode unchanged to avoid upsetting bias conditions of the following stage.

The $1-k$ input was formed by using a linearized transcon- ductor like before except that a fixed $100nA$ differential current is subtracted from the output. The outputs were then swapped to achieve the $1-k$ function. The slope for a differential input is $100nA/70mV$. The common mode current of $435nA$ allows values of k up to 5.

The 0 input shown on the left side of Fig. 1 can actually be used as a means of energy input to the system. Another f cell was used to provide input to the row blocks. A pulse at this input will cause the row integrators to leave their initial stable states.

VI. Simulation Results

We will now present results from the simulation of the algorithm using the software described above. The visualizations were all generated using a scientific computing package in Python. Particularly being studied is the effect of noise and offset on the algorithm's performance for various values of α and k. Previous simulations have demonstrated that the algorithm is resilient in the face of significant noise in the interconnection weight c_{mi} and integrations [7].

For the small instances of k-SAT ($N \approx 10, M < 50$) we use a simple forward-Euler method. We established that a time step $\Delta t = 0.02$ gives reliable results, and it was used in the simulations described here. We focused on 4-SAT problems and used self-coupling parameters $A = 1.54$ and $B = 2.18$, which are in the optimal region established by Molnár and Ercsey-Ravasz [5]. Simulations were run for a maximum of $t_{\max} = 100$ time units, and were deemed to have failed to find a solution if unable to do so within this time.

Fig. 6 displays the evolution of the ten s variables over time for an example 4-SAT problem with a constraint density $\alpha = 4$. The graph show how some variables tentatively assume values early in the simulation, but change them later as the system is attracted to a solution state. Fig. 7 displays the evolution of the forty a variables for the same simulation. Careful comparison with

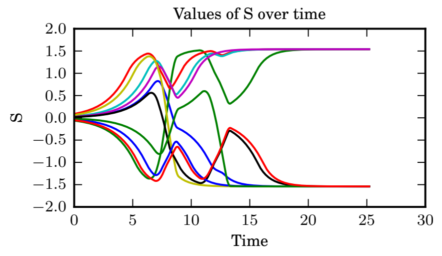

Figure 6: *Evolution of s variables over time for a typical $\alpha = 4$ 4-SAT problem.*

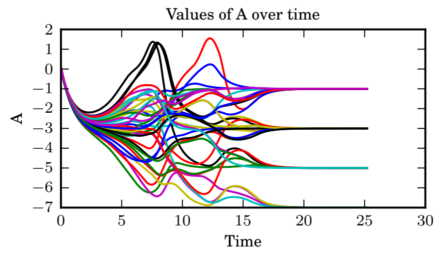

Figure 7: *Evolution of a variables over time for a typical $\alpha = 4$ 4-SAT problem.*

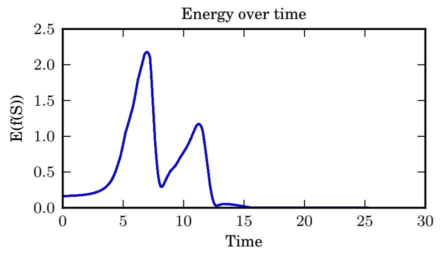

Figure 8: *Evolution of E "energy" function over time for a typical $\alpha = 4$ 4-SAT problem.*

Fig. 6 shows that increase in an a variable (corresponding to an unsatisfied constraint) leads to a sign change in an s variable. Finally, Fig. 8 shows the time evolution of the energy function, which reflects the urgency of satisfying unsatisfied constraints. A partial solution is found at $t \approx 8$ before a complete solution is discovered at $t \approx 18$. Fig. 9 displays the distribution of convergence times versus constraint density α for 100 randomly chosen constraint matrices for 4-SAT instances with no noise or offset. For $\alpha < 3.5$, all instances are solved in the allotted time, but for $\alpha \geq 3.5$ a few remained unsolved when the simulation was terminated. Therefore, in the following simulations

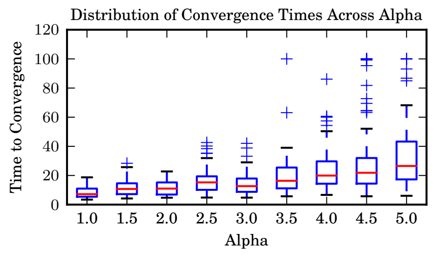

Figure 9: *Distribution of convergence time versus difficulty measured by α for 100 random 4-SAT instances. ($t = 100$ represents non-convergence in the allotted time.)*

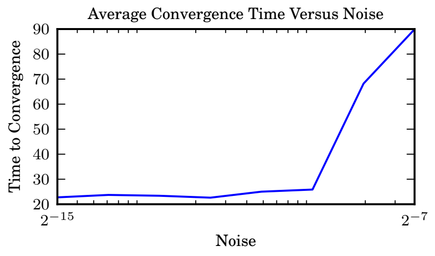

Figure 10: *Convergence time versus noise for 100 random 4-SAT instances.*

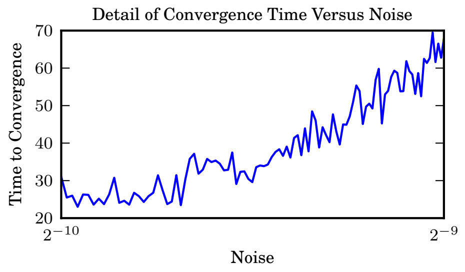

Figure 11: *Detail of convergence time versus noise for 100 random 4-SAT instances, noise $\sigma = 2^{-10}$ to 2^{-9}.*

we used $\alpha = 4.0$ to provide a baseline against which to compare the effects of noise and offset.

Fig. 10 displays the time to convergence for the same 100 random 4-SAT problems, but with Gaussian noise σ varying from 2^{-15} to 2^{-7} of full range of the s and a variables. Noise had little effect on the system up through values of 2^{-10} (nearly 100% solved). At 2^{-9}, however, only about 40% of the problems were solved in the time set for the simulation. Additional noise levels were simulated in range of $2^{-10} - 2^{-9}$ where the critical transition seems to occur, as illustrated in

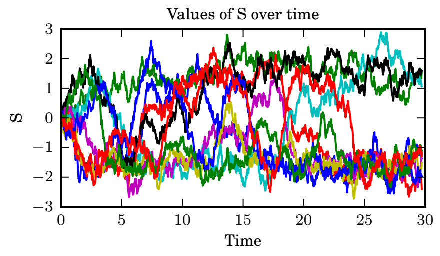

Figure 12: *Effect of noise on evolution of s variables.*

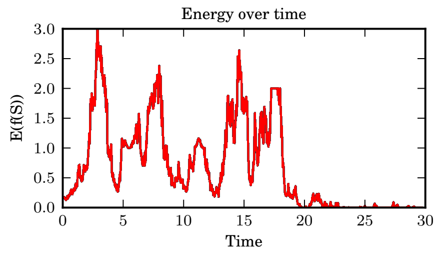

Figure 13: *Effect of noise on evolution of energy.*

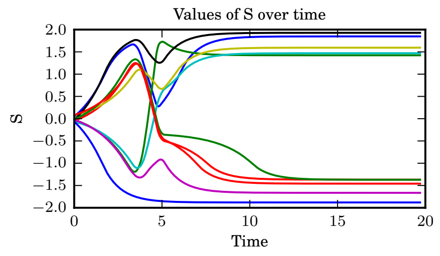

Figure 14: *Effect of offset on evolution of s variables.*

Fig. 11

Figs. 12 and 13 show the effect of noise ($\sigma = 2^{-9}$ of full range of s and a variables) on the problem instance depicted in Figs. 6–8. Figs. 14 and 15 show the effect of offset ($\sigma = 2^{-13}$) on this same instance. The combined effects of noise and offset are shown in Figs. 16 and 17 ($\sigma = 2^{-10}$ noise and $\sigma = 2^{-12}$ offset). In all of these cases, despite the considerable noise in the s and a variables, the algorithm found a solution in about the same time as the error-free version.

The effects of noise and offset on convergence are summarized in Fig. 18, which displays the

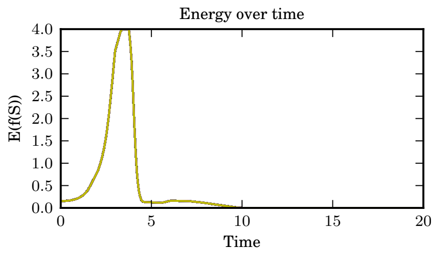

Figure 15: *Effect of noise on evolution of energy.*

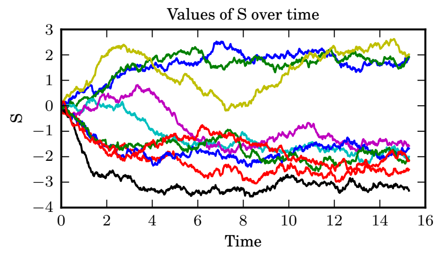

Figure 16: *Combined effects of noise and offset on evolution of s variables.*

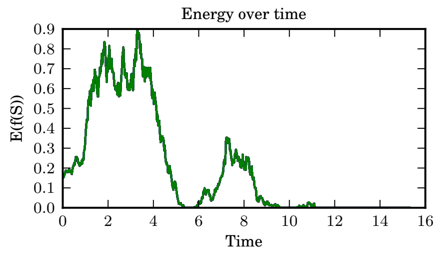

Figure 17: *Combined effects of noise and offset on evolution of energy.*

percentage of this random selection of $\alpha = 4$ problems that were solved within the allotted time for a variety of combinations of noise and offset.

One of the more interesting parts of these results is the effect of noise and offset on the effectiveness of finding solutions to the k-SAT problems, as described above. Fig. 18 seems to show that as more noise and offset is added, the number of problems solved goes down in both directions. However, we will explore in the next section some expectations we have from the original algorithm and its response to noise and offset and why we aren't seeing those met in this

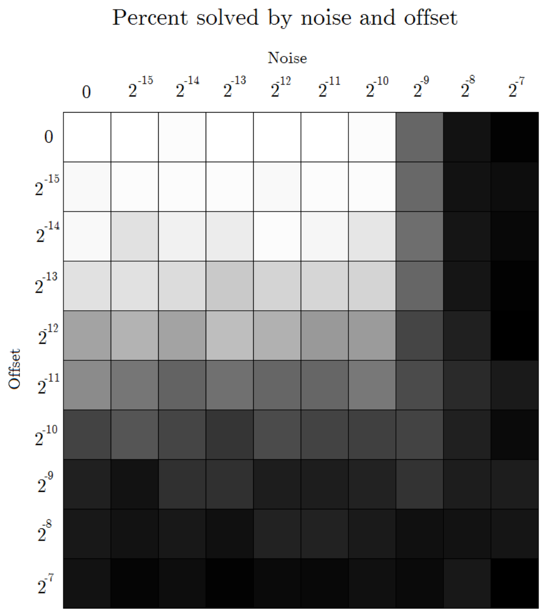

Figure 18: *Percent of $\alpha = 4$ problems solved for various combinations of noise and offset. Key: white = 100%, black = 0%.*

simulation. Perhaps, there are some improvements that could be made to take advantage of noise and offset to solve more problems.

VII. Improving the Implementation

One of the key features of this experiment is the study of the effects of noise and offset on the algorithm. Currently, the timestep that determines the noise and offsets is an arbitrary unit without any parallel to a real world quantity such as seconds. A major improvement for the implementation of the model would be representing the timestep as a realistic quantity in both the hardware and software. Work is being done to solve this by attempting to pull out discrete quantities for time out of the model by finding a time-difference equation for the analog implementation.

We also believe that noise and offset *should* help the algorithm find solutions to the problems by settling in the global solution states. Fig. 13 shows the effects of noise and offset on the algorithm's ability to find a solution state, and as it currently stands the noise and offset seem to push the algorithm in the wrong direction. This may be due to the values we are using for the noise and offset currently, specifically the noise values. The noise is currently incremented on every timestep, which may not be the best solution. A better solution may be randomly alternating the solution on every timestep between a positive and negative value at a very small magnitude. We believe that the current magnitude of the noise may be too large, and a smaller value and realistic timestep may allow the noise to help the algorithm.

VIII. Future Work

Analog computation is very promising, and applications such as the one shown in this paper demonstrate the use of analog computers beyond what early computer scientists used analog hardware for. Future work in this area would include several improvements to the current analog hardware and software representations, including adding a realistic timestep to the software simulation and implementing the analog hardware on larger quantities of variables, clauses, and variables per clauses. Additionally, finding specific applications of this algorithm that we can use constraint satisfaction to solve will be useful once the implementation is perfected by finding a realistic timestep and helpful values of noise and offset. Beyond the scope of the work presented in this paper, applications for analog computation are far reaching. Research efforts also focus on robotics and analog hardware, which could yield promising results in the near future [8], as well as optical computing on analog hardware [9].

IX. Conclusion

Despite analog hardware not being nearly as ubiquitous as the digital computers we see on a regular basis, analog computers and computational methods based on analog principles still have a broad and important impact on many aspects of modern computer science. We have explored how analog computation plays a role in optimally modeling a dynamical system for solving the root of all NP-Complete problems, Boolean Satisfiability, though analog computation is useful for far more applications. In our presentation of results from the simulation and modeling of the algorithm, it is clear that the software and hardware are not nearly complete regarding optimal performance and scale, but they are a small start on what could be a path to finally settling the legendary P=NP problem in computational complexity theory.

References

[1] Connor, R. Joseph, Jeremy Holleman, Bruce J. MacLennan, and Jared M. Smith. "Simulation of Analog Solution of Boolean Satisfiability." *UTK EECS Tech Reports* (2015): 1-4. UTK Electrical Engineering and Computer Science Department, 29 Sept. 2015. Web. 29 Nov. 2015. http://www.eecs.utk.edu/resources/library/593.

[2] Freeth, Tony, Alexander Jones, John M. Steele, and Yanis Bitsakis. "Calendars with Olympiad Display and Eclipse Prediction on the Antikythera Mechanism." Nature 454.7204 (2008): 614-17. Web.

[3] M. Ercsey-Ravasz and Z. Toroczkai, "Optimization hardness as transient chaos in an analog approach to constraint satisfaction," *Nature Physics*, vol. 7, pp. 966–970, 2011.

[4] Siegelmann, Hava T., and Eduardo D. Sontag. "Analog computation via neural networks." Theory and Computing Systems, 1993., Proceedings of the 2nd Israel Symposium on the. IEEE, 1993.

[5] B. Molnár and M. Ercsey-Ravasz, "Asymmetric continuous-time neural networks without local traps for solving constraint satisfaction problems," *PLoS ONE*, vol. 8, no. 9, p. e73400, 2013.

[6] MacKay, Donald MacCrimmon, and Michael E. Fisher. Analogue Computing at Ultra-high Speed; an Experimental and Theoretical Study. New York: Wiley, 1962. Print.

[7] R. Sumi, B. Molnár, and M. Ercsey-Ravasz, "Robust optimization with transiently chaotic dynamical systems," *EPL (Europhysics Letters)*, vol. 106, p. 40002, 2014.

[8] Glasius, Roy, Andrzej Komoda, and Stan C.a.m. Gielen. "Neural Network Dynamics for Path Planning and Obstacle Avoidance." Neural Networks 8.1 (1995): 125-33. Web.

[9] Solli, Daniel R., and Bahram Jalali. "Analog optical computing." Nature Photonics 9.11 (2015): 704-706.

[10] Y. Zhang, C.-H. Chen, and G. Temes, "Accuracy-enhanced switched-capacitor stages using low-gain opamps," *Electronics Letters*, vol. 49, no. 1, pp. 22–23, 2013.

[11] D. J. Allstot, R. W. Brodersen, and P. R. Gray, “Mos switched capacitor ladder filters,” *Solid-State Circuits, IEEE Journal of*, vol. 13, no. 6, pp. 806–814, 1978.

[12] W. A. Serdijn, M. Broest, J. Mulder, A. C. van der Woerd, and A. H. van Roermund, “A low-voltage ultra-low-power translinear integrator for audio filter applications,” *Solid-State Circuits, IEEE Journal of*, vol. 32, no. 4, pp. 577–581, 1997.

[13] J. Lu and J. Holleman, “A 1 TOPS/W analog deep machine-learning engine with floating-gate storage in 0.13 $upmu$ m CMOS,” in *IEEE International Solid-State Circuits Conference Digest of Technical Papers*, February 2014.

[14] R. Chawla, A. Bandyopadhyay, V. Srinivasan, and P. Hasler, “A 531 nw/mhz, 128$\times$ 32 current-mode programmable analog vector-matrix multiplier with over two decades of linearity,” in *Custom Integrated Circuits Conference, 2004. Proceedings of the IEEE 2004*. IEEE, 2004, pp. 651–654.