Accurate Modeling of Continuous-time SAT Solvers in SPICE

Yuriy V. PERSHIN, Dyk Chung NGUYEN

Dept. of Physics and Astronomy, University of South Carolina, Columbia, SC 29208 USA

pershin@physics.sc.edu, dykchung@email.sc.edu

Submitted May 20, 2025 / Accepted June 20, 2025 / Online first August 4, 2025

Abstract. *Recently, there has been an increasing interest in employing dynamical systems as solvers of NP-complete problems. In this paper, we present accurate implementations of two continuous-time dynamical solvers, known in the literature as analog SAT and digital memcomputing, using advanced numerical integration algorithms of SPICE circuit simulators. For this purpose, we have developed Python scripts that convert Boolean satisfiability (SAT) problems into electronic circuits representing the analog SAT and digital memcomputing dynamical systems. Our Python scripts process conjunctive normal form (CNF) files and create netlists that can be directly imported into LTspice. We explore the SPICE implementations of analog SAT and digital memcomputing solvers by applying these to a selected set of problems and present some interesting and potentially useful findings related to digital memcomputing and analog SAT. In this work, we also introduce networks of continuous-time solvers with potential applications extending beyond the solution of Boolean satisfiability problems.*

Keywords

SPICE, nonlinear dynamical systems, computing technology, boolean satisfiability problem, 3-SAT

1. Introduction

In recent years, there has been growing attention towards unconventional computing [1], which includes the use of physical systems for computation. In this area, the evolution of a suitable classical [2] or quantum [3] system is seen as a computation process. Examples of computing with physical systems include quantum computing [4], [5], memcomputing [6], [7], analog SAT [8], Toshiba bifurcation machine [9], and circuits involving stochastic tunnel junctions [10]. In certain tasks, physical computing systems have the potential to surpass traditional von Neumann computing systems, as suggested by Arute et al. [11].

The present paper focuses on two continuous-time dynamical solvers known in the literature as analog SAT [8] and digital memcomputing [12]. Chronologically, the first solver was the analog SAT developed by Ercsey-Ravasz and Toroczkai [8], who first proposed a mapping of k-SAT to a deterministic continuous-time dynamical system. In this approach, Boolean variables are extended to continuous ones within a particular dynamical system, such that finding a solution to the k-SAT problem is equivalent to identifying a stable fixed point (or points) of the system. To avoid local minima, the authors of [8] used a modified energy function depending on auxiliary variables. The idea is that the growth of auxiliary variables increases the weight of unresolved clauses and promotes their resolution[1]. For hard instances, the analog SAT dynamics is characterized by transiently chaotic trajectories that, however, all converge to the solution [8]. It is argued that the analog SAT has a polynomial analog-time complexity [8], [18].

Digital memcomputing [12] is a continuous-time dynamical solver based on different equations. We emphasize that while several designs of digital memcomputing are available in the literature [6][2], [12], here we consider the equations introduced by Bearden, Pei, and Di Ventra in [12] that tackle the 3-SAT problem. According to the authors, the new equations improve the analog SAT by avoiding exponential fluctuations in the energy function [12]. Several properties of digital memcomputing solvers have been identified. It is argued that they rely on self-organizing logic gates to find the problem solution [12]. Internally, the solver operates through instantonic jumps that couple less stable critical points with more stable ones so that the number of unstable directions reduces after each jump [20], [21]. Furthermore, the dynamics of digital memcomputing machines with solution(s) does not exhibit chaos and/or periodic orbits [22], [23].

Analog SAT and digital memcomputing dynamical systems are typically realized using numerical techniques to integrate their equations. Either conventional or dedicated hardware (such as GPUs or FPGAs) is used for this purpose.

[1] It is worth mentioning that the *clause weighting* is also employed in certain stochastic local search algorithms [13–17].

[2] See [19] for some crucial information to reproduce the first design [6].

DOI: 10.13164/re.2025.0526

	Analog SAT [24]	Digital memcomputing [12]
Equations	$\dot{s}_i = \sum_{m=1}^{M} 2a_m c_{mi} K_{mi}(\mathbf{s}) K_m(\mathbf{s})$ † (1) $\dot{a}_m = a_m K_m^2(\mathbf{s})$ (2)	$\dot{v}_n = \sum_m x_{l,m} x_{s,m} G_{n,m}(v_n, v_j, v_k) + (1 + \zeta x_{l,m}) \cdot (1 - x_{s,m}) R_{n,m}(v_n, v_m, v_k)$ (3) $\dot{x}_{s,m} = \beta (x_{s,m} + \epsilon)(C_m(v_i, v_j, v_k) - \gamma)$ (4) $\dot{x}_{l,m} = \alpha (C_m(v_i, v_j, v_k) - \delta)$ (5)
Functions	$K_m = \frac{1}{2^3} \prod_{j=1}^{N} (1 - c_{mj} s_j)$ (6) $K_{mi} = \frac{K_m}{1 - c_{mi} s_i}$ (7)	$G_{n,m} = \frac{1}{2} q_{n,m} \min\left[(1 - q_{j,m} v_j), (1 - q_{k,m} v_k)\right]$ (8) $R_{n,m} = \begin{cases} \frac{1}{2}(q_{n,m} - v_n), & \text{if } C_m(v_n, v_j, v_k) = \frac{1}{2}(1 - q_{n,m} v_n), \\ 0, & \text{otherwise.} \end{cases}$ (9) $C_m(v_i, v_j, v_k) = \frac{1}{2} \min\left[(1 - q_{i,m} v_i), (1 - q_{j,m} v_j), (1 - q_{k,m} v_k)\right]$ (10)
Variables	$s_i \in [-1, 1]$, $i = 1, \ldots, N$ $a_m \in (0, \infty)$, $m = 1, \ldots, M$	$v_i \in [-1, 1]$, $i = 1, \ldots, N$ $x_{s,m} \in [0, 1]$, $x_{l,m} \in [1, 10^4 M]$, $m = 1, \ldots, M$
Properties	Polynomial analog-time complexity [8], [18], transient chaos [8]	Polynomial time to solution [6], [12], absence of chaos and and periodic orbits in DMMs with solutions [12], relies on instantonic dynamics [25] of SO logic gates
Hardware	GPU [24], ASIC [26]	FPGA [27], [28]

† In this table, the dots above variables are used to indicate the time derivative.

Tab. 1. Summary of the analog SAT and digital memcomputing approaches.

For nearly five decades, SPICE (Simulation Program with Integrated Circuit Emphasis) circuit simulators have been in use since the publication of the original SPICE report [29]. Their operation relies on sophisticated numerical integration algorithms that have been refined and tested thoroughly throughout these years. In this work, we develop Python scripts to convert the equations of analog SAT and digital memcomputing dynamical systems into equivalent electronic circuits that can be directly imported into a SPICE environment. While our work is based on LTspice XVII (Analog Device), other simulators can also be used (with suitable minor modifications).

Although classical ODE solvers and dedicated SAT solvers remain the best in time-critical and large-scale applications, our approach uses the SPICE for a very different purpose, which is to provide a hardware-oriented platform to explore the dynamics of continuous-time SAT solvers. The use of SPICE to implement SAT solvers bridges the gap between theoretical dynamical system models and possible hardware implementations. Therefore, our work is not a competitor to state-of-the-art SAT solvers, but rather a conceptual and experimental framework to investigate the behavior of solvers and their potential circuit-level implementations.

The structure of this paper is as follows. Section 2 provides an overview of the problem, solvers, and information on the SPICE modeling approach. Examples of LTspice simulations are given in Sec. 3. These include the application of solvers to a variety of SAT problems, modification of solvers, and simulation of digital memcomputing networks. Section 3 is followed by a discussion in Sec. 4. The paper ends with an acknowledgment.

2. Methods

2.1 3-SAT

In this paper, we limit ourselves to 3-SAT problems, wherein each clause is formed by the disjunction of 3 literals. A literal is a Boolean variable, x_i, or its negation, $\bar{x}_i$. Basically, the problem is to find the assignment of N Boolean variables to satisfy M clauses forming a formula. An example of a 3-SAT formula is

$$(x_1 \vee \bar{x}_2 \vee x_3) \wedge (\bar{x}_7 \vee \bar{x}_5 \vee x_2) \wedge (x_3 \vee \bar{x}_1 \vee x_6) \wedge \ldots \quad (11)$$

where $\vee$ stands for the disjunction (OR) and $\wedge$ stands for the conjunction (AND). Here we just show explicitly the first three clauses.

In the above expression, the first clause can be satisfied by x_1 = TRUE, etc. However, since each variable (directly or negated) enters on average into $3M/N$ clauses, finding the solution, in general, is difficult, especially at M/N close to 4.26 [30]. In fact, it is well known that 3-SAT is NP-complete [31].

A simple example of a 3-SAT problem can be found in the Listing 1 of Appendix.

2.2 Solvers

The continuous-time solvers considered in this work are summarized in Tab. 1.

2.2.1 Analog SAT

The dynamical system in the analog SAT approach [24] is characterized by (1) and (2) [3]. In these equations, s_i-s are continuous extensions of Boolean variables x_i-s ($s_i = 1$ if x_i is TRUE, $s_i = -1$ if x_i is FALSE), and a_m are auxiliary variables. Moreover, $c_{mi} = 1$ (-1) if i-th variable enters into m-the clause in the direct (negated) form, respectively, and $c_{mi} = 0$ otherwise.

We note that Equation (1) implements the gradient descent [32] for a modified energy function $V(\mathbf{s}, \mathbf{a}) = \sum_{m=1}^{M} a_m K_m(\mathbf{s})^2$ with a_m defining the weight of m-th clause. According to (2), the weights of unsatisfied clauses increase with time. This promotes their resolution through (1).

Overall, the analog SAT is a deterministic non-local search algorithm wherein the auxiliary variables provide extra dimensions along which the trajectories escape from local minima. Several points related to (1)–(2) should be mentioned (for more details, see [8]). First, the dynamics of $\mathbf{s}$ is confined to the continuous domain $[-1, 1]^N$ (for an arbitrary initial condition for $\mathbf{s}$ within $[-1, 1]^N$). Second, 3-SAT solutions are stable fixed points of (1) and all trajectories converge to a solution [18]. Third, the system has no limit cycles. Fourth, for problems with solution, the only stable fixed points of (1)–(2) are the ones corresponding to the global minimum of $V(\mathbf{s}, \mathbf{a})$ with $V = 0$ [8].

2.2.2 Digital Memcomputing

The dynamical system in the digital memcomputing approach [12] is characterized by (3)–(5) (the original notation is used). In (3)–(5), v_n-s are continuous variables (similar to s_i-s in the analog SAT), $x_{s,m}$-s and $x_{l,m}$-s are the short and long memory variables, and $q_{j,m}$-s are the constants defining the clauses (same as c_{mj}-s in the analog SAT). The performance of digital memcomputing depends on the parameters α, β, γ, δ, ϵ, and ζ [12]. The intervals for v_n, $x_{s,m}$, and $x_{l,m}$ are provided in Tab. 1. Moreover, Equation (10) defines the clause function, $C_m(v_i, v_j, v_k)$, used in (4), (5), and (9). This function characterizes the state of the variable that most closely satisfies clause m. For more information, see [7], [12].

The first term in (3) can be interpreted as a "gradient-like" term, while the second - as a "rigidity" term [12]. The purpose of the "rigidity" term is to suppress the evolution of v_n when its value is the best to satisfy clause m. Compared to a single variable a_m associated with each clause in the analog SAT, in digital memcomputing, two memory variables, the short (s) and long (l), are associated with each clause.

2.3 SPICE Implementations

Two Python scripts were developed to generate electronic circuits that implement the analog SAT and digital memcomputing dynamical systems in SPICE. These scripts can be downloaded from [33]. Each script takes a CNF file representing a 3-SAT problem as input and generates the corresponding solver circuit as output. Python scripts are limited to clauses that contain three literals. In the case of analog SAT, the factor $1/2^3$ in (6) was omitted for compactness.

In SPICE, ordinary differential equations are numerically integrated employing 1 F capacitors driven by voltage-controlled current sources, which is a widely-used technique (see, e.g., [34], [35]). While capacitor voltages represent variables, voltage-controlled current sources implement the right-hand sides of dynamical equations (such as (1)–(2)). As described in [35], to ensure a DC path to the ground, the capacitors are shunted by high-resistance resistors, which do not affect the outcome of the integration. This approach is illustrated in Fig. 1 for the variable v_1 in the digital memcomputing solver.

The initial conditions are set as follows. In the analog SAT solver, s_i-s are chosen at random from a uniform distribution in the interval $[-1, 1]$ and $a_m(0) = 1$. In the digital memcomputing solver, we select v_i-s at random from a uniform distribution within the interval $[-1, 1]$, and set $x_{s,m}(0) = 0.5$ and $x_{l,m}(0) = 1$. Our analog SAT and memcomputing SPICE models utilize $N + M$ and $N + 2M$ capacitors, respectively, equal to the number of dynamical variables in these models.

Additionally, two control circuits are used to achieve a compact representation of the dynamics. The analog control voltage (node "contra") is obtained according to

$$V_{\text{contra}} = \sum_{m=1}^{N} C_m(v_i, v_j, v_k) \tag{12}$$

where $C_m(v_i, v_j, v_k)$ is given by (10). The digital control voltage (node "contrd") is derived from an analogous expression adjusted by incorporating the unit step functions, $u(\dots)$. In principle, $V_{\text{contrd}}(t)$ represents the number of unsatisfied clauses at time t.

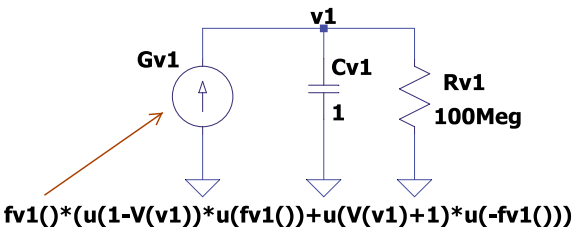

Fig. 1. SPICE model for the integration of (3) with $n = 1$. Here, fv1() is the function that represents the right-hand side of (3). The step functions, u(...), are used to confine the variable v_1 to the interval $[-1, 1]$.

[3] Note that in earlier publications [8], [18], Equation (2) had the form $\dot{a}_m = a_m K_m(\mathbf{s})$.

Examples of SPICE netlists for the problem in the Listing 1 are given in the Listings 2 and 3 in Appendix.

To enable random initial values for variables, the user must make sure the "Use the clock to reseed the MC generator" option is turned on in LTspice XVII. Moreover, the “uic” option should be selected in the transient analysis.

In addition to LTspice XVII, the functionality of our netlists has been verified in LTspice's newest release (version 24.1.9). A minor necessary change was the replacement of the “.probe” instruction with “.save”. Although numerical convergence issues have not been encountered in LTspice XVII, it might be necessary to employ smoothing techniques, rather than relying on the statements “if” and “unity step”, to execute our netlists in alternative SPICE environments.

2.4 Generation of Instances

To illustrate the approach, we used LTspice to solve easy, difficult, and very difficult 3-SAT instances with planted solutions. Easy and difficult 3-SAT instances were generated using the method of Barthel et al. [36] at $p_0 = 0.08$ selecting $M/N = 7$ and $M/N = 4.3$, respectively. For more information, see Barthel et al. [36].

The 3 regular 3-XORSAT instances are considered challenging for various 3-SAT solvers [5]. To create such an instance, a set of modulo 2 addition equations was generated via a random assignment of variables. Each equation included three variables, with their negations determined randomly. Furthermore, each variable appeared in precisely three equations. Finally, each equation was transformed into four 3-SAT clauses.

3. Simulations

In this section, we present examples of SPICE simulations for both the analog SAT and digital memcomputing solvers. Unless otherwise specified, the digital memcomputing results were obtained using the following parameter values: $\alpha = 5$, $\beta = 20$, $\gamma = 0.25$, $\delta = 0.05$, $\epsilon = 0.001$, and $\zeta = 0.01$. These parameter values are close to the optimal ones mentioned in [12].

3.1 Main Simulations

Ten instances were created for each type of the problem (easy, difficult, and very difficult) and size of the problem (we considered problems of $N = 10$, 20, 30, 40, and 50). In each run, the circuit was simulated for up to $t_{ev} = 300\,\mathrm{s}$ of its evolution. Each solver was used on each problem just once, resulting in either a solved or unsolved outcome. For unsolved outcomes, there were two distinct scenarios: one in which the time interval t_{ev} was insufficient to reach the solution and the other in which a convergence to zero was observed, as described below. The problems used in this study and corresponding netlists can be downloaded from [33].

		# Unsolved (out of 10)		
		B4.3	B7	X
$N = 10$	analog SAT	0	0	10 (10)
	memcomputing	0	0	0
$N = 20$	analog SAT	0	0	7 (7)
	memcomputing	0	0	0
$N = 30$	analog SAT	0	0	8 (8)
	memcomputing	0	0	4
$N = 40$	analog SAT	0	0	9 (9)
	memcomputing	0	0	10
$N = 50$	analog SAT	0	0	10 (9)
	memcomputing	0	0	10

Tab. 2. Summary of LTspice simulations.

Table 2 provides a summary of the outcomes from our SPICE simulations. In the "Unsolved" column, the numbers in parentheses represent the number of cases when the convergence to zero was observed. The convergence to zero was only observed with the analog SAT solver.

According to Tab. 2, all Barthel instances were solved by the two approaches. The convergence to zero was the reason for all unsolved cases by analog SAT, except one (at $N = 50$). We explain the remaining unsolved cases (by both methods) by the finite evolution time used in this study, $t_{ev} = 300\,\mathrm{s}$. Presumably, this value was too short for the solvers to reach the solution. At the same time, we have not verified whether the solution can be found at large values of t_{ev} in such cases.

Figure 2 presents examples of the temporal evolution of variables in solved problems of various complexity levels. According to Fig. 2, there is a clear correlation between the difficulty of the problem (easy, difficult, and very difficult) and the time to the solution. Noticeably, at any given difficulty of the problem, it takes more time/transitions for digital memcomputing to reach the solution in comparison to the analog SAT. However, this observation is not universal, since the opposite behavior was also observed in some other cases.

3.2 Convergence to Zero

It was unexpected to observe the convergence to zero in the dynamics of analog SAT. By the “convergence to zero” we mean a long-term behavior when all main variables in the solver, s_i, approach zero as in Fig. 3(a). This transition was only observed when the analog SAT was applied to the 3 regular 3-XORSAT instances. According to the right column of Tab. 2, the convergence to zero was the most common scenario in the case of very difficult problems.

Figure 3 illustrates how the main variables, s_i, and the auxiliary variables, a_i, behave during the convergence to zero of the main variables. It can be observed that while the main variables rapidly decrease to zero (Fig. 3(a)), the auxiliary variables exhibit exponential growth (Fig. 3(b)). It is important to point out that the convergence to zero is not the only possible scenario. It can be avoided with an alternative random selection of initial conditions.

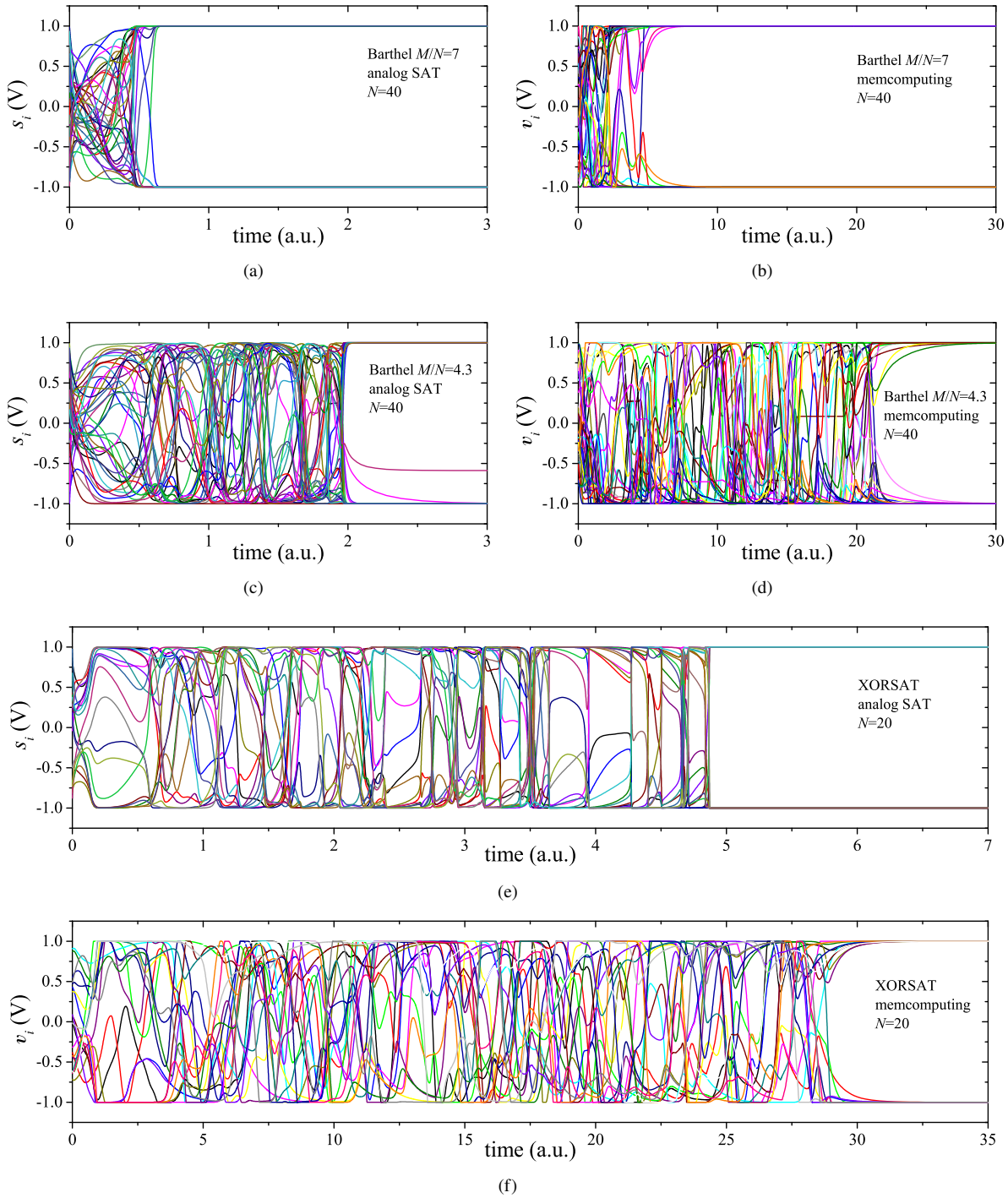

Fig. 2. Solving (a), (b) easy, (c), (d) difficult, and (e), (f) very difficult 3-SAT problems in SPICE. These plots show the dynamics of the variables s_i and v_i in the analog SAT and digital memcomputing algorithms, respectively. These plots were obtained using $N = 40$ 3-SAT problems (panels (a)–(d)) and one $N = 20$ problem (panels (e)–(f)). Each pair of simulation results was derived from solving the same problem.

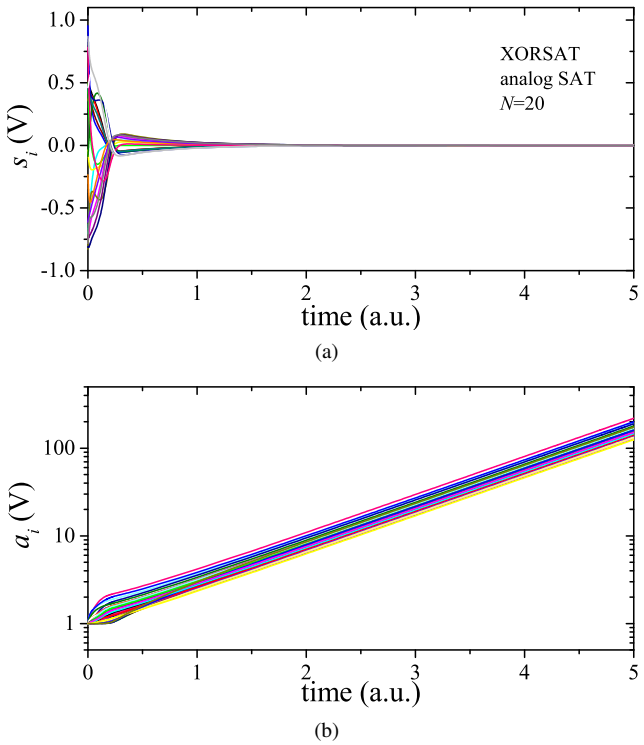

Fig. 3. An example of the converging to zero solution: time-dependencies of (a) main and (b) auxiliary variables. This figure was obtained using an $N = 20$, 3 regular 3-XORSAT problem.

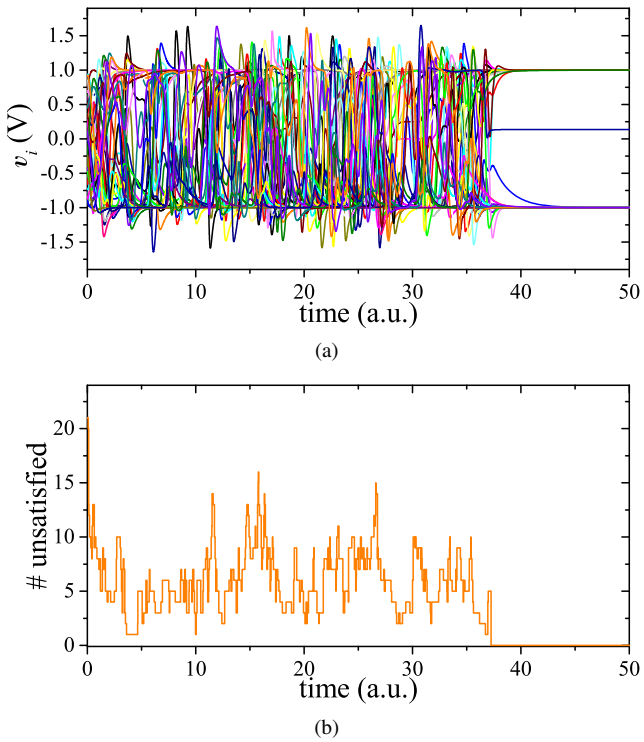

Fig. 4. Evolution of (a) voltage variables and (b) number of unsatisfied clauses in a modified memcomputing solver (without restrictions on v_i). The problem considered in this example is the same as in Fig. 2(c) and (d).

3.3 Modified Solvers

The techniques we have devised for modeling continuous-time solvers in SPICE offer a straightforward way to investigate and improve these solvers. In fact, different modifications to the solvers can be easily made, either through Python scripts or directly within the final electronic circuits. Presented below are two examples demonstrating this.

3.3.1 Digital Memcomputing

For illustration purposes, the memcomputing solver was modified by eliminating the constraints on v_i (refer to the line below (10) in Tab. 1). This adjustment was made directly in the solver's net list of a specific problem by removing the second multiplier from the equations that govern relevant current sources, similar to the one shown in Fig. 1. Specifically, in Fig. 1 case, the expression was simplified to fv1().

Figure 4 shows that the digital memcomputing solver without imposing restrictions on the values v_i may find the solution to the problem. Figure 4(b) depicts the number of unsatisfied clauses as a function of time (node "contrd" of the circuit).

3.3.2 Analog SAT

In an attempt to understand the generality of the convergence to zero (Sec. 3.2), the following versions of (2) were considered:

$$\dot{a}_m = a_m K_m(\mathbf{s}), \tag{13}$$

$$\dot{a}_m = K_m(\mathbf{s}), \tag{14}$$

$$\dot{a}_m = K_m(\mathbf{s})^2. \tag{15}$$

Using SPICE simulations, in all three cases, we have cases of the convergence to zero, as in Fig. 3(a). However, in the case of (14) and (15), the growth of the auxiliary variables was linear instead of exponential. Therefore, the convergence to zero seems to be unrelated to the exponential growth of the auxiliary variables shown in Fig. 3(b).

3.4 Networks of SAT Solvers

It might be worthwhile to explore networks composed of continuous-time SAT solvers, where these solvers serve as nodes that interact via some of their variables or combinations of them. On the one hand, SAT solver networks can be utilized to solve SAT problems whose size exceeds the capacity of an individual SAT solver, a particularly crucial approach for physical implementations of these solvers that have finite sizes [26]. On the other hand, SAT solvers can be viewed as distant analogs to neurons, possessing complex functionalities determined by the SAT problems they solve. Therefore, one can anticipate that networks of SAT solvers could find applications in artificial neural network architectures or reservoir computing.

An especially useful feature of SPICE for this application is the subcircuit declaration. Fundamentally, the subcircuit functions as a collection of elements in SPICE that can

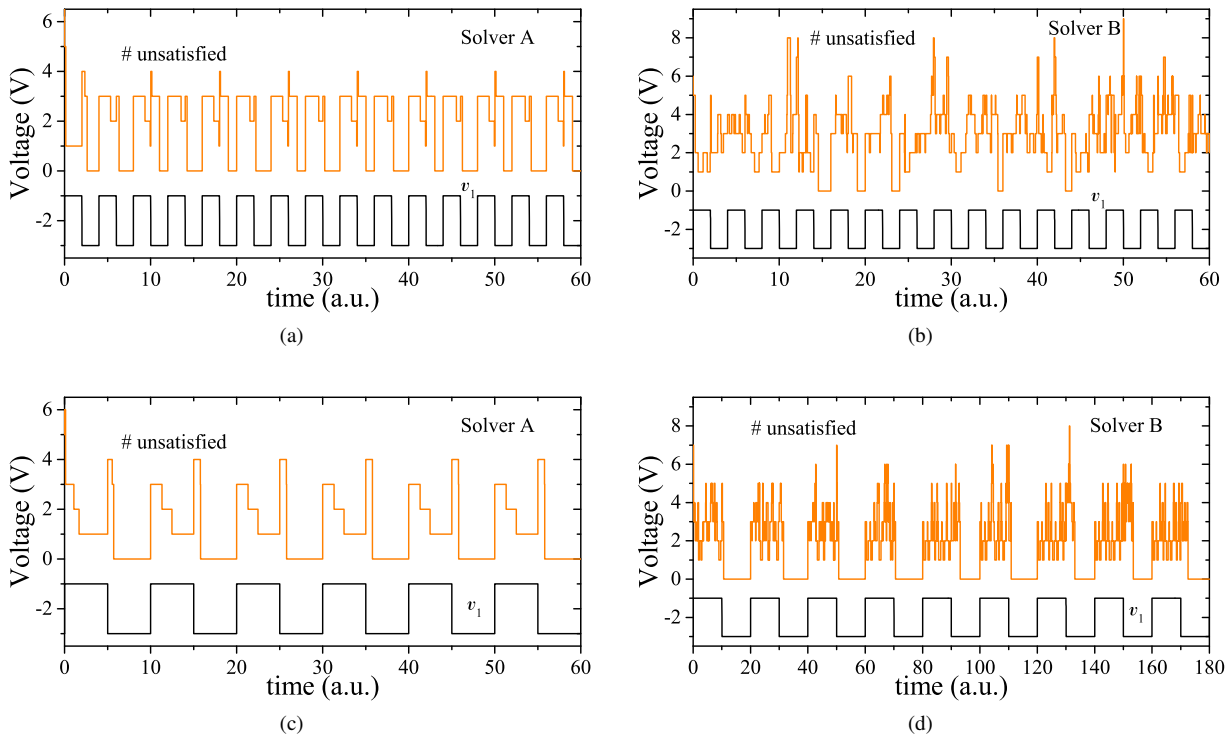

Fig. 5. Solvers A and B's response to a square wave voltage (represented by v_1). For the sake of clarity, the applied voltage curves were shifted downward by 2 V. See Sec. 3.4 for more details. These simulations were performed using $\alpha = 0$.

be referred to in a manner similar to device models. Externally, the subcircuit is accessed through its external nodes. For SAT solvers, a potential approach is to define P input nodes and Q output nodes ($P + Q \leq N$), such that, in the digital memcomputing case, the input nodes simply set the values of P voltage variables (their dynamics equations are disregarded), while the output nodes are used to output the values of some other Q voltage variables. To monitor the solver's state, the signal $V_{\text{contrd}}(t)$ may be included in the output nodes. Of course, there are other choices available to connect the solvers.

As an example, we considered two instances, say A and B, of the 3-regular 3-XORSAT problem with $N = 10$. The digital memcomputing circuits for A and B were created using our Python script [33]. First, we simulated uncoupled solvers and found the following solutions (which are not necessarily unique):

$$0101100111 \quad \text{for instance A,} \tag{16}$$
$$0100111101 \quad \text{for instance B.} \tag{17}$$

Here, each digit from left to right represents x_1 through x_{10}. By modifying several lines in the netlists of solvers A and B, we created the subcircuits for subsequent simulations. The sets $\{v_1\}$, $\{v_i, V_{\text{contrd}}(t)\}$ with $i = 2$ or 3 were selected as input and output nodes, respectively.

Next, we studied how individual solvers react to an applied square wave voltage. To achieve this, a square wave voltage source (−1 V to +1 V) was connected to the solver's input node. We have observed that when the applied voltage is at logic zero and there is sufficient time, both solvers can successfully identify the solution. When the applied voltage is at logic one, the solvers exhibit some interesting dynamics, which can be described as deterministic for solver A and chaotic for solver B. These observations are evident in Fig. 5. A notable characteristic associated with solver A is the period doubling behavior depicted in Fig. 5(a), which vanishes at a longer period, as demonstrated in Fig. 5(b).

Finally, the solvers were connected in a network as shown in Fig. 6(a). Two cases were considered: in the first case, v_2 was used as the output node in both solvers, while in the second case, v_3 was employed for the same purpose. The networks were simulated for 300 time units of the circuit dynamics. Figure 6(b) demonstrates that selecting v_2 as the output node results in no solution for the whole network, whereas using v_3 as the output node successfully yields a solution. These observations align with the solutions provided by (16) and (17) and can be explained by them.

4. Discussion

In this work, we have developed a platform for high-accuracy modeling of continuous-time SAT solvers within the SPICE framework. To achieve this, we utilized Python scripts to produce electronic circuits that implement continuous-time solvers of specific problems. Cutting-edge SPICE simulation tools, such as LTspice, used in this research

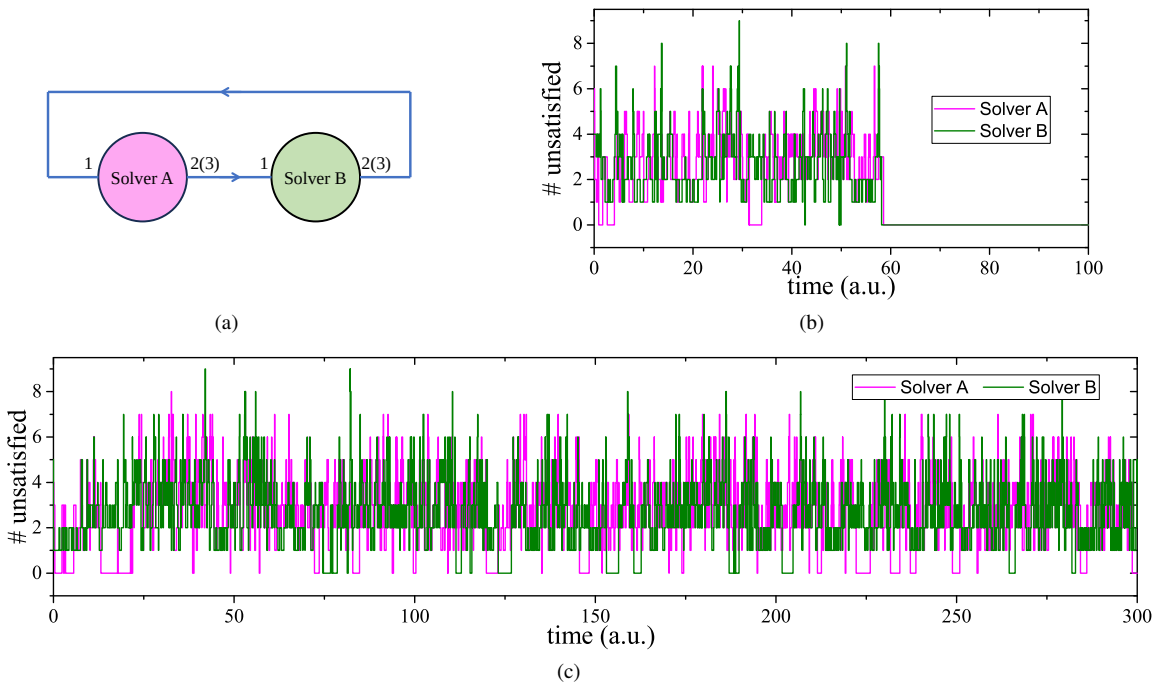

Fig. 6. (a) Schematics of a simple network of two digital memcomputing solvers; (b) Number of unsatisfied clauses as a function of time when v_2 is used as the output node in both solvers; (c) Number of unsatisfied clauses as a function of time when v_3 is used as the output node in both solvers.

are based on advanced numerical algorithms to simulate circuit dynamics. These algorithms being a part of our platform are used to solve NP-complete problems via circuit simulations. On a standard PC, LTspice can effectively handle problems with $N \lesssim 50$. For larger 3-SAT problems, massively parallel SPICE circuit simulators such as Xyse^{TM} [37] could offer potential solutions.

Through the application of analog SAT and digital memcomputing techniques to 3-SAT problems of several levels of complexity, we have observed, for the first time, the transition to zero in the behavior of certain analog SAT dynamical systems. Such behavior is robust with respect to the equations that define the dynamics of auxiliary variables and have a large basin of attraction. This observation challenges the statement that "eventually all trajectories converge to a solution" [18].

Additionally, we have observed that the digital memcomputing solver continues to function effectively when the limitation on voltage variables is removed. This observation could prove beneficial in the design of specialized digital memcomputing hardware [38].

Lastly, we have demonstrated that it is quite straightforward to simulate networks of continuous-time solvers in SPICE. It is expected that such networks may have potential applications extending beyond the solution of Boolean satisfiability problems.

In a wider perspective, although digital memcomputing and analog SAT belong to the same group of algorithms, their development has occurred largely in isolation. This presents opportunities for mutual enrichment of these methodologies at both theoretical and applied levels. For example, digital memcomputing operations are believed to incorporate instantonic jumps that connect less stable critical points with more stable ones, thereby decreasing the number of unstable directions after each jump [21]. We propose that analogous instantonic jumps may be present in analog SAT as well. In fact, in Figs. 2(c) and (e) showing the dynamics of analogue SAT, it is easy to identify distinct groups of transitions resembling those previously attributed to instantons in digital memcomputing [21]. A similar argument applies to the role of self-organizing gates in these solvers.

Acknowledgments

The authors acknowledge the support of the National Science Foundation grant number ECCS-2229880. They thank M. Di Ventra and Y.-H. Zhang for comments on the manuscript.

References

[1] FINOCCHIO, G., INCORVIA, J. A. C., FRIEDMAN, J. S., et al. Roadmap for unconventional computing with nanotechnology. *Nano Futures*, 2024, vol. 8, no. 1, p. 1–41. DOI: 10.1088/2399-1984/ad299a

[2] SIEGELMANN, H. T., FISHMAN, S. Analog computation with dynamical systems. *Physica D: Nonlinear Phenomena*, 1998, vol. 120, no. 1, p. 214–235. DOI: 10.1016/S0167-2789(98)00057-8

[3] FEYNMAN, R. P. Simulating physics with computers. *International Journal of Theoretical Physics*, 1982, vol. 21, no. 6, p. 467–488. DOI: 10.1007/BF02650179

[4] BIAN, Z., CHUDAK, F., MACREADY, W., et al. Solving SAT (and MaxSAT) with a quantum annealer: Foundations, encodings, and preliminary results. *Information and Computation*, 2020, vol. 275, p. 1–34. DOI: 10.1016/j.ic.2020.104609

[5] KOWALSKY, M., ALBASH, T., HEN, I., et al. 3-regular three-XORSAT planted solutions benchmark of classical and quantum heuristic optimizers. *Quantum Science and Technology*, 2022, vol. 7, no. 2, p. 1–18. DOI: 10.1088/2058-9565/ac4d1b

[6] TRAVERSA, F. L., DI VENTRA, M. Polynomial-time solution of prime factorization and NP-complete problems with digital memcomputing machines. *Chaos: An Interdisciplinary Journal of Nonlinear Science*, 2017, vol. 27, no. 2, p. 1–22. DOI: 10.1063/1.4975761

[7] DI VENTRA, M. *MemComputing: Fundamentals and Applications*. Oxford (UK): University Press, 2022. ISBN: 9780192845320

[8] ERCSEY-RAVASZ, M., TOROCZKAI, Z. Optimization hardness as transient chaos in an analog approach to constraint satisfaction. *Nature Physics*, 2011, vol. 7, no. 21, p. 966–970. DOI: 10.1038/nphys2105

[9] GOTO, H., TATSUMURA, K., DIXON, A. R. Combinatorial optimization by simulating adiabatic bifurcations in nonlinear Hamiltonian systems. *Science Advances*, 2019, vol. 5, no. 4, p. 1–9. DOI: 10.1126/sciadv.aav2372

[10] BORDERS, W. A., PERVAIZ, A. Z., FUKAMI, S., et al. Integer factorization using stochastic magnetic tunnel junctions. *Nature*, 2019, vol. 573, no. 7774, p. 390–393. DOI: 10.1038/s41586-019-1557-9

[11] ARUTE, F., ARYA, K., BABBUSH, R., et al. Quantum supremacy using a programmable superconducting processor. *Nature*, 2019, vol. 574, no. 7779, p. 505–510. DOI: 10.1038/s41586-019-1666-5

[12] BEARDEN, S. R. B., MANUKIAN, H., TRAVERSA, F. L., et al. Efficient solution of boolean satisfiability problems with digital MemComputing. *Scientific Reports*, 2020, vol. 10, p. 1–8. DOI: 10.1038/s41598-020-76666-2

[13] MORRIS, P. The breakout method for escaping from local minima. In *Proceedings of the AAAI'93: Eleventh National Conference on Artificial Intelligence*. Washington D.C. (USA), 1993, p. 40–45. DOI: 10.5555/1867270.1867277

[14] SELMAN, B., KAUTZ, H. A. An empirical study of greedy local search for satisfiability testing. In *Proceedings of the AAAI'93: Eleventh National Conference on Artificial Intelligence*. Washington D.C. (USA), 1993, p. 46–51. DOI: 10.5555/1867270.1867278

[15] CHA, B., IWAMA, K. Local search for SAT with preprocessing. In *Proceedings of the IJCAI'95: 14th International Joint Conference on Artificial Intelligence*. Montreal, Quebec (Canada), 1995, p. 304–310. DOI: 10.5555/1625855.1625895

[16] FRANK, J., CHEESEMAN, P., ALLEN, J. Weighting for Godot: Learning heuristics for GSAT. In *Proceedings of the AAAI'96: Thirteenth National Conference on Artificial Intelligence*. Portland, Oregon (USA), 1996, p. 338–343. DOI: 10.5555/1892875.1892926

[17] FRANK, J. Learning short-term weights for GSAT. In *Proceedings of the Fifteenth International Joint Conference on Artificial Intelligence*. Nagoya, Aichi (Japan), 1997, p. 384–391.

[18] MOLNAR, B., MOLNAR, F., VARGA, M., et al. A continuous-time MaxSAT solver with high analog performance. *Nature Communications*, 2018, vol. 9, no. 1, p. 1–15. DOI: 10.1038/s41467-018-07327-2

[19] PERSHIN, Y. V. SPICE modeling of memcomputing logic gates. *Radioengineering*, 2023, vol. 32, no. 4, p. 542–558. DOI: 10.13164/re.2023.0542

[20] BEARDEN, S. R. B., MANUKIAN, H., TRAVERSA, F. L., et al. Instantons in self-organizing logic gates. *Physical Review Applied*, 2018, vol. 9, no. 3, p. 1–10. DOI: 10.1103/PhysRevApplied.9.034029

[21] PRIMOSCH, D., ZHANG, Y.-H., DI VENTRA, M. Self-averaging of digital memcomputing machines. *Physical Review E*, 2023, vol. 108, no. 3, p. 1–8. DOI: 10.1103/PhysRevE.108.034306

[22] DI VENTRA, M., TRAVERSA, F. L. Absence of chaos in digital memcomputing machines with solutions. *Physics Letters A*, 2017, vol. 381, no. 38, p. 3255–3257. DOI: 10.1016/j.physleta.2017.08.040

[23] DI VENTRA, M., TRAVERSA, F. L. Absence of periodic orbits in digital memcomputing machines with solutions. *Chaos: An Interdisciplinary Journal of Nonlinear Science*, 2017, vol. 27, no. 10, p. 1–13. DOI: 10.1063/1.5004431

[24] MOLNAR, F., KHAREL, S. R., HU, X. S., et al. Accelerating a continuous-time analog SAT solver using GPUs. *Computer Physics Communications*, 2020, vol. 256, p. 1–14. DOI: 10.1016/j.cpc.2020.107469

[25] DI VENTRA, M., OVCHINNIKOV, I. V. Digital memcomputing: from logic to dynamics to topology. *Annals of Physics*, 2019, vol. 409, p. 1–12. DOI: 10.1016/j.aop.2019.167935

[26] CHANG, M., YIN, X., TOROCZKAI, Z., et al. An analog clock-free compute fabric based on continuous-time dynamical system for solving combinatorial optimization problems. In *Proceedings of the 2022 IEEE Custom Integrated Circuits Conference (CICC)*. Newport Beach (USA), 2022, p. 1–6. DOI: 10.1109/CICC53496.2022.9772850

[27] NGUYEN, D. C., ZHANG, Y.-H., DI VENTRA, M., et al. Hardware implementation of digital memcomputing on small-size FPGAs. In *Proceedings of the 2023 IEEE 66th International Midwest Symposium on Circuits and Systems (MWSCAS)*. Tempe (USA), 2023, p. 346–350. DOI: 10.1109/MWSCAS57524.2023.10405845

[28] NGUYEN, D. C., PERSHIN, Y. V. Fully parallel implementation of digital memcomputing on FPGA. In *Proceedings of the 2024 IEEE 67th International Midwest Symposium on Circuits and Systems (MWSCAS)*. Springfield (USA), 2024, p. 263–266. DOI: 10.1109/MWSCAS60917.2024.10658882

[29] NAGEL, L., PEDERSON, D. *SPICE (Simulation Program with Integrated Circuit Emphasis)*. Technical report, University of California, Berkeley, 1973, UCB/ERL M382. Available at: https://www2.eecs.berkeley.edu/Pubs/TechRpts/1973/ERL-382.pdf

[30] CRAWFORD, J. M., AUTON, L. D. Experimental results on the crossover point in random 3-SAT. *Artificial Intelligence*, 1996, vol. 81, no. 1, p. 31–57. DOI: 10.1016/0004-3702(95)00046-1

[31] GAREY, M. R., JOHNSON, D. S. Computers and intractability: A guide to the theory of NP-completeness. In *Series of Books in the Mathematical Sciences*. 1st ed. New York (USA): Freeman, 2002. ISBN: 0716710455

[32] GU, J., GU, Q., DU, D. On optimizing the satisfiability (SAT) problem. *Journal of Computer Science and Technology*, 1999, vol. 14, no. 1, p. 1–17. DOI: 10.1007/BF02952482

[33] *LTspice_SAT GitHub repository*. Available at: https://github.com/chunchic/LTspice_SAT

[34] BIOLEK, Z., BIOLEK, D., BIOLKOVA, V. SPICE model of memristor with nonlinear dopant drift. *Radioengineering*, 2009, vol. 18, no. 2, p. 210–214.

[35] BIOLEK, D., DI VENTRA, M., PERSHIN, Y. V. Reliable SPICE simulations of memristors, memcapacitors and meminductors. *Radioengineering*, 2013, vol. 22, no. 4, p. 945–968.

[36] BARTHEL, W., HARTMANN, A. K., LEONE, M., et al. Hiding solutions in random satisfiability problems: A statistical mechanics approach. *Physical Review Letters*, 2002, vol. 88, no. 18, p. 1–4. DOI: 10.1103/PhysRevLett.88.188701

[37] *Xyce(^{TM}) Parallel Electronic Simulator*. Sandia National Laboratories, Albuquerque, Livermore (USA), 2023. Available at: https://github.com/Xycedev/Xyce. DOI: 10.11578/dc.20171025.1421

[38] ZHANG, Y.-H., DI VENTRA, M. Implementation of digital memcomputing using standard electronic components. *International Journal of Circuit Theory and Applications*, 2025, vol. 53, no. 4, p. 2447–2465. DOI: 10.1002/cta.4220

About the Authors . . .

Yuriy V. PERSHIN received the M. Sc. degree in Theoretical Physics from the V. N. Karazin Kharkiv National University, Ukraine in 1996 and the Ph. D. degree from the University of Konstanz, Germany in 2002. Currently, he is a Professor of Physics at the University of South Carolina, Columbia, USA. He has published approximately 150 papers in peer-reviewed journals, several review articles, and coauthored a book. At present, his research focuses on the physics and applications of memory circuit elements, specifically memristors and memcapacitors.

Dyk Chung NGUYEM is currently pursuing Ph.D. in Physics at the University of South Carolina. He previously received his bachelor's degree in Applied Physics from the V. N. Karazin Kharkiv National University, Ukraine. His research interests include molecular dynamics simulations, FPGA-based memcomputing, and chaos in dynamical systems.

Appendix A: File Listings

Listing 1. A simple 3-SAT problem in CNF notation.

```
c p0=0.08
p cnf 5 15
5 -4 3 0
5 -2 -3 0
3 5 1 0
-2 5 3 0
-2 1 -5 0
-4 -1 3 0
5 -1 -4 0
5 3 4 0
-5 2 -4 0
-3 -5 -2 0
-3 5 4 0
3 -4 2 0
-2 4 5 0
1 3 5 0
1 3 4 0
```

Listing 2. LTspice netlist for the analog SAT algorithm for the problem in Listing 1.

```
* Control circuit
ESAT1 contra 0 value={fsat1()}
RSAT1 contra 0 100meg
ESAT2 contrd 0 value={fsat2()}
RSAT2 contrd 0 100meg

* Main variables
Cs1 s1 0 1 IC={-1+mc(1,1)}
Gs1 0 s1 value={fs1()}
Rs1 s1 0 100meg
Cs2 s2 0 1 IC={-1+mc(1,1)}
Gs2 0 s2 value={fs2()}
Rs2 s2 0 100meg
Cs3 s3 0 1 IC={-1+mc(1,1)}
Gs3 0 s3 value={fs3()}
Rs3 s3 0 100meg
Cs4 s4 0 1 IC={-1+mc(1,1)}
```

```
Gs4 0 s4 value={fs4()}
Rs4 s4 0 100meg
Cs5 s5 0 1 IC={-1+mc(1,1)}
Gs5 0 s5 value={fs5()}
Rs5 s5 0 100meg

* Memory variables
Ca1 a1 0 1 IC={1}
Ga1 0 a1 value={fa1()}
Ra1 a1 0 100meg
Ca2 a2 0 1 IC={1}
Ga2 0 a2 value={fa2()}
Ra2 a2 0 100meg
Ca3 a3 0 1 IC={1}
Ga3 0 a3 value={fa3()}
Ra3 a3 0 100meg
Ca4 a4 0 1 IC={1}
Ga4 0 a4 value={fa4()}
Ra4 a4 0 100meg
Ca5 a5 0 1 IC={1}
Ga5 0 a5 value={fa5()}
Ra5 a5 0 100meg
Ca6 a6 0 1 IC={1}
Ga6 0 a6 value={fa6()}
Ra6 a6 0 100meg
Ca7 a7 0 1 IC={1}
Ga7 0 a7 value={fa7()}
Ra7 a7 0 100meg
Ca8 a8 0 1 IC={1}
Ga8 0 a8 value={fa8()}
Ra8 a8 0 100meg
Ca9 a9 0 1 IC={1}
Ga9 0 a9 value={fa9()}
Ra9 a9 0 100meg
Ca10 a10 0 1 IC={1}
Ga10 0 a10 value={fa10()}
Ra10 a10 0 100meg
Ca11 a11 0 1 IC={1}
Ga11 0 a11 value={fa11()}
Ra11 a11 0 100meg
Ca12 a12 0 1 IC={1}
Ga12 0 a12 value={fa12()}
Ra12 a12 0 100meg
Ca13 a13 0 1 IC={1}
Ga13 0 a13 value={fa13()}
Ra13 a13 0 100meg
Ca14 a14 0 1 IC={1}
Ga14 0 a14 value={fa14()}
Ra14 a14 0 100meg
Ca15 a15 0 1 IC={1}
Ga15 0 a15 value={fa15()}
Ra15 a15 0 100meg

* functions
.func Cm(x,y,z)={0.5*min(1-x,min(1-y,1-z))}
.func Cm1(x,y,z)={min(1-u(x),min(1-u(y),1-u(z)))}

.func fsat1()=Cm(V(s5),-V(s4),V(s3))+Cm(V(s5),-V(s2),-V(s3))+Cm(V(s3),V(s5),V(s1))+Cm(-V(s2),V(s5),V(
     s3))+Cm(-V(s2),V(s1),-V(s5))+Cm(-V(s4),-V(s1),V(s3))+Cm(V(s5),-V(s1),-V(s4))+Cm(V(s5),V(s3),V(s4
     ))+Cm(-V(s5),V(s2),-V(s4))+Cm(-V(s3),-V(s5),-V(s2))+Cm(-V(s3),V(s5),V(s4))+Cm(V(s3),-V(s4),V(s2)
     )+Cm(-V(s2),V(s4),V(s5))+Cm(V(s1),V(s3),V(s5))+Cm(V(s1),V(s3),V(s4))

.func fsat2()=Cm1(V(s5),-V(s4),V(s3))+Cm1(V(s5),-V(s2),-V(s3))+Cm1(V(s3),V(s5),V(s1))+Cm1(-V(s2),V(s5
     ),V(s3))+Cm1(-V(s2),V(s1),-V(s5))+Cm1(-V(s4),-V(s1),V(s3))+Cm1(V(s5),-V(s1),-V(s4))+Cm1(V(s5),V(
     s3),V(s4))+Cm1(-V(s5),V(s2),-V(s4))+Cm1(-V(s3),-V(s5),-V(s2))+Cm1(-V(s3),V(s5),V(s4))+Cm1(V(s3)
     ,-V(s4),V(s2))+Cm1(-V(s2),V(s4),V(s5))+Cm1(V(s1),V(s3),V(s5))+Cm1(V(s1),V(s3),V(s4))

.func fs1() = {2*V(a3)*(1)*pow(1-V(s3),2)*pow(1-V(s5),2)*pow(1-V(s1),1)+2*V(a5)*(1)*pow(1+V(s2),2)*
     pow(1-V(s1),1)*pow(1+V(s5),2)+2*V(a6)*(-1)*pow(1+V(s4),2)*pow(1+V(s1),1)*pow(1-V(s3),2)+2*V(a7)*
     (-1)*pow(1-V(s5),2)*pow(1+V(s1),1)*pow(1+V(s4),2)+2*V(a14)*(1)*pow(1-V(s1),1)*pow(1-V(s3),2)*pow
     (1-V(s5),2)+2*V(a15)*(1)*pow(1-V(s1),1)*pow(1-V(s3),2)*pow(1-V(s4),2)}
.func fs2() = {2*V(a2)*(-1)*pow(1-V(s5),2)*pow(1+V(s2),1)*pow(1+V(s3),2)+2*V(a4)*(-1)*pow(1+V(s2),1)*
```

```
     pow(1-V(s5),2)*pow(1-V(s3),2)+2*V(a5)*(-1)*pow(1+V(s2),1)*pow(1-V(s1),2)*pow(1+V(s5),2)+2*V(a9)*
     (1)*pow(1+V(s5),2)*pow(1-V(s2),1)*pow(1+V(s4),2)+2*V(a10)*(-1)*pow(1+V(s3),2)*pow(1+V(s5),2)*pow
     (1+V(s2),1)+2*V(a12)*(1)*pow(1-V(s3),2)*pow(1+V(s4),2)*pow(1-V(s2),1)+2*V(a13)*(-1)*pow(1+V(s2)
     ,1)*pow(1-V(s4),2)*pow(1-V(s5),2)}
.func fs3() = {2*V(a1)*(1)*pow(1-V(s5),2)*pow(1+V(s4),2)*pow(1-V(s3),1)+2*V(a2)*(-1)*pow(1-V(s5),2)*
     pow(1+V(s2),2)*pow(1+V(s3),1)+2*V(a3)*(1)*pow(1-V(s3),1)*pow(1-V(s5),2)*pow(1-V(s1),2)+2*V(a4)*
     (1)*pow(1+V(s2),2)*pow(1-V(s5),2)*pow(1-V(s3),1)+2*V(a6)*(1)*pow(1+V(s4),2)*pow(1+V(s1),2)*pow
     (1-V(s3),1)+2*V(a8)*(1)*pow(1-V(s5),2)*pow(1-V(s3),1)*pow(1-V(s4),2)+2*V(a10)*(-1)*pow(1+V(s3)
     ,1)*pow(1+V(s5),2)*pow(1+V(s2),2)+2*V(a11)*(-1)*pow(1+V(s3),1)*pow(1-V(s5),2)*pow(1-V(s4),2)+2*V
     (a12)*(1)*pow(1-V(s3),1)*pow(1+V(s4),2)*pow(1-V(s2),2)+2*V(a14)*(1)*pow(1-V(s1),2)*pow(1-V(s3)
     ,1)*pow(1-V(s5),2)+2*V(a15)*(1)*pow(1-V(s1),2)*pow(1-V(s3),1)*pow(1-V(s4),2)}
.func fs4() = {2*V(a1)*(-1)*pow(1-V(s5),2)*pow(1+V(s4),1)*pow(1-V(s3),2)+2*V(a6)*(-1)*pow(1+V(s4),1)*
     pow(1+V(s1),2)*pow(1-V(s3),2)+2*V(a7)*(-1)*pow(1-V(s5),2)*pow(1+V(s1),2)*pow(1+V(s4),1)+2*V(a8)*
     (1)*pow(1-V(s5),2)*pow(1-V(s3),2)*pow(1-V(s4),1)+2*V(a9)*(-1)*pow(1+V(s5),2)*pow(1-V(s2),2)*pow
     (1+V(s4),1)+2*V(a11)*(1)*pow(1+V(s3),2)*pow(1-V(s5),2)*pow(1-V(s4),1)+2*V(a12)*(-1)*pow(1-V(s3)
     ,2)*pow(1+V(s4),1)*pow(1-V(s2),2)+2*V(a13)*(1)*pow(1+V(s2),2)*pow(1-V(s4),1)*pow(1-V(s5),2)+2*V(
     a15)*(1)*pow(1-V(s1),2)*pow(1-V(s3),2)*pow(1-V(s4),1)}
.func fs5() = {2*V(a1)*(1)*pow(1-V(s5),1)*pow(1+V(s4),2)*pow(1-V(s3),2)+2*V(a2)*(1)*pow(1-V(s5),1)*
     pow(1+V(s2),2)*pow(1+V(s3),2)+2*V(a3)*(1)*pow(1-V(s3),2)*pow(1-V(s5),1)*pow(1-V(s1),2)+2*V(a4)*
     (1)*pow(1+V(s2),2)*pow(1-V(s5),1)*pow(1-V(s3),2)+2*V(a5)*(-1)*pow(1+V(s2),2)*pow(1-V(s1),2)*pow
     (1+V(s5),1)+2*V(a7)*(1)*pow(1-V(s5),1)*pow(1+V(s1),2)*pow(1+V(s4),2)+2*V(a8)*(1)*pow(1-V(s5),1)*
     pow(1-V(s3),2)*pow(1-V(s4),2)+2*V(a9)*(-1)*pow(1+V(s5),1)*pow(1-V(s2),2)*pow(1+V(s4),2)+2*V(a10)
     *(-1)*pow(1+V(s3),2)*pow(1+V(s5),1)*pow(1+V(s2),2)+2*V(a11)*(1)*pow(1+V(s3),2)*pow(1-V(s5),1)*
     pow(1-V(s4),2)+2*V(a13)*(1)*pow(1+V(s2),2)*pow(1-V(s4),2)*pow(1-V(s5),1)+2*V(a14)*(1)*pow(1-V(s1
     ),2)*pow(1-V(s3),2)*pow(1-V(s5),1)}

.func fa1() = {V(a1)*pow((1-V(s5))*(1+V(s4))*(1-V(s3)),2)}
.func fa2() = {V(a2)*pow((1-V(s5))*(1+V(s2))*(1+V(s3)),2)}
.func fa3() = {V(a3)*pow((1-V(s3))*(1-V(s5))*(1-V(s1)),2)}
.func fa4() = {V(a4)*pow((1+V(s2))*(1-V(s5))*(1-V(s3)),2)}
.func fa5() = {V(a5)*pow((1+V(s2))*(1-V(s1))*(1+V(s5)),2)}
.func fa6() = {V(a6)*pow((1+V(s4))*(1+V(s1))*(1-V(s3)),2)}
.func fa7() = {V(a7)*pow((1-V(s5))*(1+V(s1))*(1+V(s4)),2)}
.func fa8() = {V(a8)*pow((1-V(s5))*(1-V(s3))*(1-V(s4)),2)}
.func fa9() = {V(a9)*pow((1+V(s5))*(1-V(s2))*(1+V(s4)),2)}
.func fa10() = {V(a10)*pow((1+V(s3))*(1+V(s5))*(1+V(s2)),2)}
.func fa11() = {V(a11)*pow((1+V(s3))*(1-V(s5))*(1-V(s4)),2)}
.func fa12() = {V(a12)*pow((1-V(s3))*(1+V(s4))*(1-V(s2)),2)}
.func fa13() = {V(a13)*pow((1+V(s2))*(1-V(s4))*(1-V(s5)),2)}
.func fa14() = {V(a14)*pow((1-V(s1))*(1-V(s3))*(1-V(s5)),2)}
.func fa15() = {V(a15)*pow((1-V(s1))*(1-V(s3))*(1-V(s4)),2)}


.tran 0 300.000000 1u uic
.probe V(contra) V(contrd) V(s1) V(s2) V(s3) V(s4) V(s5)
```

Listing 3. LTspice netlist for the digital memcomputing algorithm for the problem in Listing 1.

```
* parameters
.param alpha=5.000000 beta=20.000000 gamma=0.250000 delta=0.050000 epsilon=0.001000 xi=0.010000 xlmax
     =910000

* Control circuit
ESAT1 contra 0 value={fsat1()}
RSAT1 contra 0 100meg
ESAT2 contrd 0 value={fsat2()}
RSAT2 contrd 0 100meg

* Main variables
Cv1 v1 0 1 IC={-1+mc(1,1)}
Gv1 0 v1 value={fv1()*(u(1-V(v1))*u(fv1())+u(V(v1)+1)*u(-fv1()))}
Rv1 v1 0 100meg
Cv2 v2 0 1 IC={-1+mc(1,1)}
Gv2 0 v2 value={fv2()*(u(1-V(v2))*u(fv2())+u(V(v2)+1)*u(-fv2()))}
Rv2 v2 0 100meg
Cv3 v3 0 1 IC={-1+mc(1,1)}
Gv3 0 v3 value={fv3()*(u(1-V(v3))*u(fv3())+u(V(v3)+1)*u(-fv3()))}
Rv3 v3 0 100meg
Cv4 v4 0 1 IC={-1+mc(1,1)}
Gv4 0 v4 value={fv4()*(u(1-V(v4))*u(fv4())+u(V(v4)+1)*u(-fv4()))}
```

```
Rv4 v4 0 100meg
Cv5 v5 0 1 IC={-1+mc(1,1)}
Gv5 0 v5 value={fv5()*(u(1-V(v5))*u(fv5())+u(V(v5)+1)*u(-fv5()))}
Rv5 v5 0 100meg

* Short memory variables
Cs1 xs1 0 1 IC={0.5}
Gs1 0 xs1 value={fs1()*(u(1-V(xs1))*u(fs1())+u(V(xs1))*u(-fs1()))}
Rs1 xs1 0 100meg
Cs2 xs2 0 1 IC={0.5}
Gs2 0 xs2 value={fs2()*(u(1-V(xs2))*u(fs2())+u(V(xs2))*u(-fs2()))}
Rs2 xs2 0 100meg
Cs3 xs3 0 1 IC={0.5}
Gs3 0 xs3 value={fs3()*(u(1-V(xs3))*u(fs3())+u(V(xs3))*u(-fs3()))}
Rs3 xs3 0 100meg
Cs4 xs4 0 1 IC={0.5}
Gs4 0 xs4 value={fs4()*(u(1-V(xs4))*u(fs4())+u(V(xs4))*u(-fs4()))}
Rs4 xs4 0 100meg
Cs5 xs5 0 1 IC={0.5}
Gs5 0 xs5 value={fs5()*(u(1-V(xs5))*u(fs5())+u(V(xs5))*u(-fs5()))}
Rs5 xs5 0 100meg
Cs6 xs6 0 1 IC={0.5}
Gs6 0 xs6 value={fs6()*(u(1-V(xs6))*u(fs6())+u(V(xs6))*u(-fs6()))}
Rs6 xs6 0 100meg
Cs7 xs7 0 1 IC={0.5}
Gs7 0 xs7 value={fs7()*(u(1-V(xs7))*u(fs7())+u(V(xs7))*u(-fs7()))}
Rs7 xs7 0 100meg
Cs8 xs8 0 1 IC={0.5}
Gs8 0 xs8 value={fs8()*(u(1-V(xs8))*u(fs8())+u(V(xs8))*u(-fs8()))}
Rs8 xs8 0 100meg
Cs9 xs9 0 1 IC={0.5}
Gs9 0 xs9 value={fs9()*(u(1-V(xs9))*u(fs9())+u(V(xs9))*u(-fs9()))}
Rs9 xs9 0 100meg
Cs10 xs10 0 1 IC={0.5}
Gs10 0 xs10 value={fs10()*(u(1-V(xs10))*u(fs10())+u(V(xs10))*u(-fs10()))}
Rs10 xs10 0 100meg
Cs11 xs11 0 1 IC={0.5}
Gs11 0 xs11 value={fs11()*(u(1-V(xs11))*u(fs11())+u(V(xs11))*u(-fs11()))}
Rs11 xs11 0 100meg
Cs12 xs12 0 1 IC={0.5}
Gs12 0 xs12 value={fs12()*(u(1-V(xs12))*u(fs12())+u(V(xs12))*u(-fs12()))}
Rs12 xs12 0 100meg
Cs13 xs13 0 1 IC={0.5}
Gs13 0 xs13 value={fs13()*(u(1-V(xs13))*u(fs13())+u(V(xs13))*u(-fs13()))}
Rs13 xs13 0 100meg
Cs14 xs14 0 1 IC={0.5}
Gs14 0 xs14 value={fs14()*(u(1-V(xs14))*u(fs14())+u(V(xs14))*u(-fs14()))}
Rs14 xs14 0 100meg
Cs15 xs15 0 1 IC={0.5}
Gs15 0 xs15 value={fs15()*(u(1-V(xs15))*u(fs15())+u(V(xs15))*u(-fs15()))}
Rs15 xs15 0 100meg

* Long memory variables
Cl1 xl1 0 1 IC={1}
Gl1 0 xl1 value={fl1()*(u(xlmax-V(xl1))*u(fl1())+u(V(xl1)-1)*u(-fl1()))}
Rl1 xl1 0 100meg
Cl2 xl2 0 1 IC={1}
Gl2 0 xl2 value={fl2()*(u(xlmax-V(xl2))*u(fl2())+u(V(xl2)-1)*u(-fl2()))}
Rl2 xl2 0 100meg
Cl3 xl3 0 1 IC={1}
Gl3 0 xl3 value={fl3()*(u(xlmax-V(xl3))*u(fl3())+u(V(xl3)-1)*u(-fl3()))}
Rl3 xl3 0 100meg
Cl4 xl4 0 1 IC={1}
Gl4 0 xl4 value={fl4()*(u(xlmax-V(xl4))*u(fl4())+u(V(xl4)-1)*u(-fl4()))}
Rl4 xl4 0 100meg
Cl5 xl5 0 1 IC={1}
Gl5 0 xl5 value={fl5()*(u(xlmax-V(xl5))*u(fl5())+u(V(xl5)-1)*u(-fl5()))}
Rl5 xl5 0 100meg
Cl6 xl6 0 1 IC={1}
Gl6 0 xl6 value={fl6()*(u(xlmax-V(xl6))*u(fl6())+u(V(xl6)-1)*u(-fl6()))}
Rl6 xl6 0 100meg
Cl7 xl7 0 1 IC={1}
```

```
Gl7 0 xl7 value={fl7()*(u(xlmax-V(xl7))*u(fl7())+u(V(xl7)-1)*u(-fl7()))}
Rl7 xl7 0 100meg
Cl8 xl8 0 1 IC={1}
Gl8 0 xl8 value={fl8()*(u(xlmax-V(xl8))*u(fl8())+u(V(xl8)-1)*u(-fl8()))}
Rl8 xl8 0 100meg
Cl9 xl9 0 1 IC={1}
Gl9 0 xl9 value={fl9()*(u(xlmax-V(xl9))*u(fl9())+u(V(xl9)-1)*u(-fl9()))}
Rl9 xl9 0 100meg
Cl10 xl10 0 1 IC={1}
Gl10 0 xl10 value={fl10()*(u(xlmax-V(xl10))*u(fl10())+u(V(xl10)-1)*u(-fl10()))}
Rl10 xl10 0 100meg
Cl11 xl11 0 1 IC={1}
Gl11 0 xl11 value={fl11()*(u(xlmax-V(xl11))*u(fl11())+u(V(xl11)-1)*u(-fl11()))}
Rl11 xl11 0 100meg
Cl12 xl12 0 1 IC={1}
Gl12 0 xl12 value={fl12()*(u(xlmax-V(xl12))*u(fl12())+u(V(xl12)-1)*u(-fl12()))}
Rl12 xl12 0 100meg
Cl13 xl13 0 1 IC={1}
Gl13 0 xl13 value={fl13()*(u(xlmax-V(xl13))*u(fl13())+u(V(xl13)-1)*u(-fl13()))}
Rl13 xl13 0 100meg
Cl14 xl14 0 1 IC={1}
Gl14 0 xl14 value={fl14()*(u(xlmax-V(xl14))*u(fl14())+u(V(xl14)-1)*u(-fl14()))}
Rl14 xl14 0 100meg
Cl15 xl15 0 1 IC={1}
Gl15 0 xl15 value={fl15()*(u(xlmax-V(xl15))*u(fl15())+u(V(xl15)-1)*u(-fl15()))}
Rl15 xl15 0 100meg

* functions
.func Cm(x,y,z)={0.5*min(1-x,min(1-y,1-z))}
.func Cm1(x,y,z)={min(1-u(x),min(1-u(y),1-u(z)))}

.func fsat1()=Cm(V(v5),-V(v4),V(v3))+Cm(V(v5),-V(v2),-V(v3))+Cm(V(v3),V(v5),V(v1))+Cm(-V(v2),V(v5),V(
     v3))+Cm(-V(v2),V(v1),-V(v5))+Cm(-V(v4),-V(v1),V(v3))+Cm(V(v5),-V(v1),-V(v4))+Cm(V(v5),V(v3),V(v4
     ))+Cm(-V(v5),V(v2),-V(v4))+Cm(-V(v3),-V(v5),-V(v2))+Cm(-V(v3),V(v5),V(v4))+Cm(V(v3),-V(v4),V(v2)
     )+Cm(-V(v2),V(v4),V(v5))+Cm(V(v1),V(v3),V(v5))+Cm(V(v1),V(v3),V(v4))

.func fsat2()=Cm1(V(v5),-V(v4),V(v3))+Cm1(V(v5),-V(v2),-V(v3))+Cm1(V(v3),V(v5),V(v1))+Cm1(-V(v2),V(v5
     ),V(v3))+Cm1(-V(v2),V(v1),-V(v5))+Cm1(-V(v4),-V(v1),V(v3))+Cm1(V(v5),-V(v1),-V(v4))+Cm1(V(v5),V(
     v3),V(v4))+Cm1(-V(v5),V(v2),-V(v4))+Cm1(-V(v3),-V(v5),-V(v2))+Cm1(-V(v3),V(v5),V(v4))+Cm1(V(v3)
     ,-V(v4),V(v2))+Cm1(-V(v2),V(v4),V(v5))+Cm1(V(v1),V(v3),V(v5))+Cm1(V(v1),V(v3),V(v4))

.func fv1() = {V(xl3)*V(xs3)*0.5*min(1-V(v3),1-V(v5))+(1+xi*V(xl3))*(1-V(xs3))*0.5*(1-V(v1))*if(V(v1)
      > V(v3), if(V(v1) > V(v5),1,0),0)+V(xl5)*V(xs5)*0.5*min(1+V(v2),1+V(v5))+(1+xi*V(xl5))*(1-V(xs5
     ))*0.5*(1-V(v1))*if(V(v1) > -V(v2), if(V(v1) > -V(v5),1,0),0)+V(xl6)*V(xs6)*(-0.5)*min(1+V(v4)
     ,1-V(v3))+(1+xi*V(xl6))*(1-V(xs6))*0.5*(-1-V(v1))*if(-V(v1) > -V(v4), if(-V(v1) > V(v3),1,0),0)+
     V(xl7)*V(xs7)*(-0.5)*min(1-V(v5),1+V(v4))+(1+xi*V(xl7))*(1-V(xs7))*0.5*(-1-V(v1))*if(-V(v1) > V(
     v5), if(-V(v1) > -V(v4),1,0),0)+V(xl14)*V(xs14)*0.5*min(1-V(v3),1-V(v5))+(1+xi*V(xl14))*(1-V(
     xs14))*0.5*(1-V(v1))*if(V(v1) > V(v3), if(V(v1) > V(v5),1,0),0)+V(xl15)*V(xs15)*0.5*min(1-V(v3)
     ,1-V(v4))+(1+xi*V(xl15))*(1-V(xs15))*0.5*(1-V(v1))*if(V(v1) > V(v3), if(V(v1) > V(v4),1,0),0)}
.func fv2() = {V(xl2)*V(xs2)*(-0.5)*min(1-V(v5),1+V(v3))+(1+xi*V(xl2))*(1-V(xs2))*0.5*(-1-V(v2))*if(-
     V(v2) > V(v5), if(-V(v2) > -V(v3),1,0),0)+V(xl4)*V(xs4)*(-0.5)*min(1-V(v5),1-V(v3))+(1+xi*V(xl4)
     )*(1-V(xs4))*0.5*(-1-V(v2))*if(-V(v2) > V(v5), if(-V(v2) > V(v3),1,0),0)+V(xl5)*V(xs5)*(-0.5)*
     min(1-V(v1),1+V(v5))+(1+xi*V(xl5))*(1-V(xs5))*0.5*(-1-V(v2))*if(-V(v2) > V(v1), if(-V(v2) > -V(
     v5),1,0),0)+V(xl9)*V(xs9)*0.5*min(1+V(v5),1+V(v4))+(1+xi*V(xl9))*(1-V(xs9))*0.5*(1-V(v2))*if(V(
     v2) > -V(v5), if(V(v2) > -V(v4),1,0),0)+V(xl10)*V(xs10)*(-0.5)*min(1+V(v3),1+V(v5))+(1+xi*V(xl10
     ))*(1-V(xs10))*0.5*(-1-V(v2))*if(-V(v2) > -V(v3), if(-V(v2) > -V(v5),1,0),0)+V(xl12)*V(xs12)*0.5
     *min(1-V(v3),1+V(v4))+(1+xi*V(xl12))*(1-V(xs12))*0.5*(1-V(v2))*if(V(v2) > V(v3), if(V(v2) > -V(
     v4),1,0),0)+V(xl13)*V(xs13)*(-0.5)*min(1-V(v4),1-V(v5))+(1+xi*V(xl13))*(1-V(xs13))*0.5*(-1-V(v2)
     )*if(-V(v2) > V(v4), if(-V(v2) > V(v5),1,0),0)}
.func fv3() = {V(xl1)*V(xs1)*0.5*min(1-V(v5),1+V(v4))+(1+xi*V(xl1))*(1-V(xs1))*0.5*(1-V(v3))*if(V(v3)
      > V(v5), if(V(v3) > -V(v4),1,0),0)+V(xl2)*V(xs2)*(-0.5)*min(1-V(v5),1+V(v2))+(1+xi*V(xl2))*(1-V
     (xs2))*0.5*(-1-V(v3))*if(-V(v3) > V(v5), if(-V(v3) > -V(v2),1,0),0)+V(xl3)*V(xs3)*0.5*min(1-V(v5
     ),1-V(v1))+(1+xi*V(xl3))*(1-V(xs3))*0.5*(1-V(v3))*if(V(v3) > V(v5), if(V(v3) > V(v1),1,0),0)+V(
     xl4)*V(xs4)*0.5*min(1+V(v2),1-V(v5))+(1+xi*V(xl4))*(1-V(xs4))*0.5*(1-V(v3))*if(V(v3) > -V(v2),
     if(V(v3) > V(v5),1,0),0)+V(xl6)*V(xs6)*0.5*min(1+V(v4),1+V(v1))+(1+xi*V(xl6))*(1-V(xs6))*0.5*(1-
     V(v3))*if(V(v3) > -V(v4), if(V(v3) > -V(v1),1,0),0)+V(xl8)*V(xs8)*0.5*min(1-V(v5),1-V(v4))+(1+xi
     *V(xl8))*(1-V(xs8))*0.5*(1-V(v3))*if(V(v3) > V(v5), if(V(v3) > V(v4),1,0),0)+V(xl10)*V(xs10)*
     (-0.5)*min(1+V(v5),1+V(v2))+(1+xi*V(xl10))*(1-V(xs10))*0.5*(-1-V(v3))*if(-V(v3) > -V(v5), if(-V(
     v3) > -V(v2),1,0),0)+V(xl11)*V(xs11)*(-0.5)*min(1-V(v5),1-V(v4))+(1+xi*V(xl11))*(1-V(xs11))*0.5*
     (-1-V(v3))*if(-V(v3) > V(v5), if(-V(v3) > V(v4),1,0),0)+V(xl12)*V(xs12)*0.5*min(1+V(v4),1-V(v2))
     +(1+xi*V(xl12))*(1-V(xs12))*0.5*(1-V(v3))*if(V(v3) > -V(v4), if(V(v3) > V(v2),1,0),0)+V(xl14)*V(
     xs14)*0.5*min(1-V(v1),1-V(v5))+(1+xi*V(xl14))*(1-V(xs14))*0.5*(1-V(v3))*if(V(v3) > V(v1), if(V(
```

```
      v3) > V(v5),1,0),0)+V(xl15)*V(xs15)*0.5*min(1-V(v1),1-V(v4))+(1+xi*V(xl15))*(1-V(xs15))*0.5*(1-V
      (v3))*if(V(v3) > V(v1), if(V(v3) > V(v4),1,0),0)}
.func fv4() = {V(xl1)*V(xs1)*(-0.5)*min(1-V(v5),1-V(v3))+(1+xi*V(xl1))*(1-V(xs1))*0.5*(-1-V(v4))*if(-
      V(v4) > V(v5), if(-V(v4) > V(v3),1,0),0)+V(xl6)*V(xs6)*(-0.5)*min(1+V(v1),1-V(v3))+(1+xi*V(xl6))
      *(1-V(xs6))*0.5*(-1-V(v4))*if(-V(v4) > -V(v1), if(-V(v4) > V(v3),1,0),0)+V(xl7)*V(xs7)*(-0.5)*
      min(1-V(v5),1+V(v1))+(1+xi*V(xl7))*(1-V(xs7))*0.5*(-1-V(v4))*if(-V(v4) > V(v5), if(-V(v4) > -V(
      v1),1,0),0)+V(xl8)*V(xs8)*0.5*min(1-V(v5),1-V(v3))+(1+xi*V(xl8))*(1-V(xs8))*0.5*(1-V(v4))*if(V(
      v4) > V(v5), if(V(v4) > V(v3),1,0),0)+V(xl9)*V(xs9)*(-0.5)*min(1+V(v5),1-V(v2))+(1+xi*V(xl9))*
      (1-V(xs9))*0.5*(-1-V(v4))*if(-V(v4) > -V(v5), if(-V(v4) > V(v2),1,0),0)+V(xl11)*V(xs11)*0.5*min
      (1+V(v3),1-V(v5))+(1+xi*V(xl11))*(1-V(xs11))*0.5*(1-V(v4))*if(V(v4) > -V(v3), if(V(v4) > V(v5)
      ,1,0),0)+V(xl12)*V(xs12)*(-0.5)*min(1-V(v3),1-V(v2))+(1+xi*V(xl12))*(1-V(xs12))*0.5*(-1-V(v4))*
      if(-V(v4) > V(v3), if(-V(v4) > V(v2),1,0),0)+V(xl13)*V(xs13)*0.5*min(1+V(v2),1-V(v5))+(1+xi*V(
      xl13))*(1-V(xs13))*0.5*(1-V(v4))*if(V(v4) > -V(v2), if(V(v4) > V(v5),1,0),0)+V(xl15)*V(xs15)*0.5
      *min(1-V(v1),1-V(v3))+(1+xi*V(xl15))*(1-V(xs15))*0.5*(1-V(v4))*if(V(v4) > V(v1), if(V(v4) > V(v3
      ),1,0),0)}
.func fv5() = {V(xl1)*V(xs1)*0.5*min(1+V(v4),1-V(v3))+(1+xi*V(xl1))*(1-V(xs1))*0.5*(1-V(v5))*if(V(v5)
       > -V(v4), if(V(v5) > V(v3),1,0),0)+V(xl2)*V(xs2)*0.5*min(1+V(v2),1+V(v3))+(1+xi*V(xl2))*(1-V(
      xs2))*0.5*(1-V(v5))*if(V(v5) > -V(v2), if(V(v5) > -V(v3),1,0),0)+V(xl3)*V(xs3)*0.5*min(1-V(v3)
      ,1-V(v1))+(1+xi*V(xl3))*(1-V(xs3))*0.5*(1-V(v5))*if(V(v5) > V(v3), if(V(v5) > V(v1),1,0),0)+V(
      xl4)*V(xs4)*0.5*min(1+V(v2),1-V(v3))+(1+xi*V(xl4))*(1-V(xs4))*0.5*(1-V(v5))*if(V(v5) > -V(v2),
      if(V(v5) > V(v3),1,0),0)+V(xl5)*V(xs5)*(-0.5)*min(1+V(v2),1-V(v1))+(1+xi*V(xl5))*(1-V(xs5))*0.5*
      (-1-V(v5))*if(-V(v5) > -V(v2), if(-V(v5) > V(v1),1,0),0)+V(xl7)*V(xs7)*0.5*min(1+V(v1),1+V(v4))
      +(1+xi*V(xl7))*(1-V(xs7))*0.5*(1-V(v5))*if(V(v5) > -V(v1), if(V(v5) > -V(v4),1,0),0)+V(xl8)*V(
      xs8)*0.5*min(1-V(v3),1-V(v4))+(1+xi*V(xl8))*(1-V(xs8))*0.5*(1-V(v5))*if(V(v5) > V(v3), if(V(v5)
      > V(v4),1,0),0)+V(xl9)*V(xs9)*(-0.5)*min(1-V(v2),1+V(v4))+(1+xi*V(xl9))*(1-V(xs9))*0.5*(-1-V(v5)
      )*if(-V(v5) > V(v2), if(-V(v5) > -V(v4),1,0),0)+V(xl10)*V(xs10)*(-0.5)*min(1+V(v3),1+V(v2))+(1+
      xi*V(xl10))*(1-V(xs10))*0.5*(-1-V(v5))*if(-V(v5) > -V(v3), if(-V(v5) > -V(v2),1,0),0)+V(xl11)*V(
      xs11)*0.5*min(1+V(v3),1-V(v4))+(1+xi*V(xl11))*(1-V(xs11))*0.5*(1-V(v5))*if(V(v5) > -V(v3), if(V(
      v5) > V(v4),1,0),0)+V(xl13)*V(xs13)*0.5*min(1+V(v2),1-V(v4))+(1+xi*V(xl13))*(1-V(xs13))*0.5*(1-V
      (v5))*if(V(v5) > -V(v2), if(V(v5) > V(v4),1,0),0)+V(xl14)*V(xs14)*0.5*min(1-V(v1),1-V(v3))+(1+xi
      *V(xl14))*(1-V(xs14))*0.5*(1-V(v5))*if(V(v5) > V(v1), if(V(v5) > V(v3),1,0),0)}

.func fs1() = {beta*(V(xs1)+epsilon)*(Cm(V(v5),-V(v4),V(v3))-gamma)}
.func fs2() = {beta*(V(xs2)+epsilon)*(Cm(V(v5),-V(v2),-V(v3))-gamma)}
.func fs3() = {beta*(V(xs3)+epsilon)*(Cm(V(v3),V(v5),V(v1))-gamma)}
.func fs4() = {beta*(V(xs4)+epsilon)*(Cm(-V(v2),V(v5),V(v3))-gamma)}
.func fs5() = {beta*(V(xs5)+epsilon)*(Cm(-V(v2),V(v1),-V(v5))-gamma)}
.func fs6() = {beta*(V(xs6)+epsilon)*(Cm(-V(v4),-V(v1),V(v3))-gamma)}
.func fs7() = {beta*(V(xs7)+epsilon)*(Cm(V(v5),-V(v1),-V(v4))-gamma)}
.func fs8() = {beta*(V(xs8)+epsilon)*(Cm(V(v5),V(v3),V(v4))-gamma)}
.func fs9() = {beta*(V(xs9)+epsilon)*(Cm(-V(v5),V(v2),-V(v4))-gamma)}
.func fs10() = {beta*(V(xs10)+epsilon)*(Cm(-V(v3),-V(v5),-V(v2))-gamma)}
.func fs11() = {beta*(V(xs11)+epsilon)*(Cm(-V(v3),V(v5),V(v4))-gamma)}
.func fs12() = {beta*(V(xs12)+epsilon)*(Cm(V(v3),-V(v4),V(v2))-gamma)}
.func fs13() = {beta*(V(xs13)+epsilon)*(Cm(-V(v2),V(v4),V(v5))-gamma)}
.func fs14() = {beta*(V(xs14)+epsilon)*(Cm(V(v1),V(v3),V(v5))-gamma)}
.func fs15() = {beta*(V(xs15)+epsilon)*(Cm(V(v1),V(v3),V(v4))-gamma)}

.func fl1() = {alpha*(Cm(V(v5),-V(v4),V(v3))-delta)}
.func fl2() = {alpha*(Cm(V(v5),-V(v2),-V(v3))-delta)}
.func fl3() = {alpha*(Cm(V(v3),V(v5),V(v1))-delta)}
.func fl4() = {alpha*(Cm(-V(v2),V(v5),V(v3))-delta)}
.func fl5() = {alpha*(Cm(-V(v2),V(v1),-V(v5))-delta)}
.func fl6() = {alpha*(Cm(-V(v4),-V(v1),V(v3))-delta)}
.func fl7() = {alpha*(Cm(V(v5),-V(v1),-V(v4))-delta)}
.func fl8() = {alpha*(Cm(V(v5),V(v3),V(v4))-delta)}
.func fl9() = {alpha*(Cm(-V(v5),V(v2),-V(v4))-delta)}
.func fl10() = {alpha*(Cm(-V(v3),-V(v5),-V(v2))-delta)}
.func fl11() = {alpha*(Cm(-V(v3),V(v5),V(v4))-delta)}
.func fl12() = {alpha*(Cm(V(v3),-V(v4),V(v2))-delta)}
.func fl13() = {alpha*(Cm(-V(v2),V(v4),V(v5))-delta)}
.func fl14() = {alpha*(Cm(V(v1),V(v3),V(v5))-delta)}
.func fl15() = {alpha*(Cm(V(v1),V(v3),V(v4))-delta)}

.tran 0 300.000000 1u uic
.probe V(contra) V(contrd) V(v1) V(v2) V(v3) V(v4) V(v5)
```