Memcomputing: Leveraging memory and physics

COT6410 : Spring 2019 : Literature Review

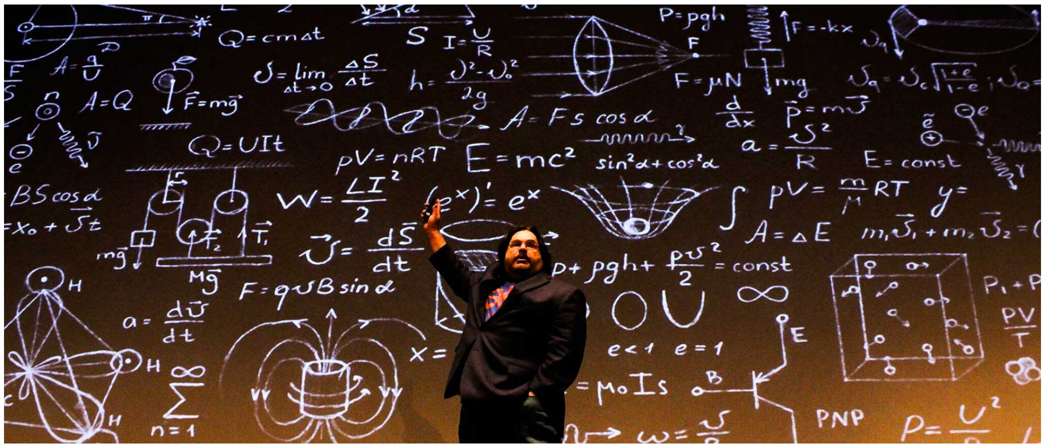


Figure 1. John Beane, CEO of MemComputing, Inc., presenting at EvoNexus Demo Day 2018.[1]

ABSTRACT

Just like Turing machines are not the only model of computation, Von Neumann architecture is not the only computing hardware out there, e.g. quantum computers, liquid-state machines, etc. Here we review the literature surrounding a novel hybrid analog-digital computing system (a memcomputer) built from memristors (memory-resistors), a basic electronics component with variable resistance, theorized about in 1971,[2] and used recently for various applications including fast non-volatile RAM.[3–5] We discuss the theory and application of digital memcomputing machines (DMM), which are non-general-purpose computing systems for solving some NP problems efficiently.[6,7] A universal memcomputing machine (UMM) is also discussed, which the authors claim is a Turing-complete, general purpose computing model.[8] One major advantage of these systems over quantum computers is the reuse of existing electronics versus specialized hardware. Simulation of their technology is conducted and results are presented showing increasing instance sizes of specific NP problems solved in polynomial ($O(n^4)$) time. The literature *does not* offer any formal proofs of P=NP, or make any such claims. The authors of the literature have also formed the company, MemComputing, Inc.,[1] and together with technology partners are attempting to leverage memcomputers for commercial applications.

1. INTRODUCTION

For at least 50 years now, commercial and personal computing has been dominated by digital, discrete-timestep computers, built around the Church-Turing model of computation and Von Neumann architectural principles.[9–11] We now call this computing system the *classical* computer, and to this day, computer scientists, architects, and engineers are still busy trying improve its performance, storage capacities, specialized hardware modules, etc. Even the mobile versions of such computers available today are more powerful than the fastest commercial mainframes of 15 years ago, a tribute to our commitment and faith in this classical computing paradigm.

And yet, despite our achievements, we still find ourselves limited (perhaps hopelessly so) by this computer. While it performs admirably well on polynomial (P) class problems, it fails miserably against non-deterministic-polynomial (NP) class problems, a set of problems with no known efficient (i.e. polynomial time) solutions.[*] Non-trivial instances of such problems scale at such dramatic rates, that it is not uncommon for the best known algorithms to take millennia to complete. We call such problems *intractable*. And unfortunately, these problems are ever present and growing.[12–15] 22 years ago, Christos Papadimitriou wrote, *"A keyword search [...] reveals that about 6,000 papers each year have the term 'NP-complete' on their title, abstract, or list of keywords. This is more than each of the terms 'compiler,' 'database,' 'expert,' 'neural network,' and 'operating system'."*[16] There is no reason to believe that the rate at which we have discovered NP (and harder = NP+) problems has decayed. The fact remains that unless we discover even a single polynomial running-time algorithm to an NP-complete problem, implying P=NP, then we simply cannot solve non-trivial instances of such problems with classical Church-Turing Von Neumann computers. Alternative computing architectures are needed.

When considering alternatives, it is important to remember that digital, discrete-time-step machines are not our only option. There is a long, overshadowed (still active) history of non-classical computers of analog, and even hybrid digital-analog, architectural design.[17–22] There are even biological and chemical architectures.[23–25] Unlike classical computing, a computer's state in such systems is often not expressed as an instantaneous description (ID) at a particular timestep, but rather as a function or set of functions (linear or non-linear) with time itself as a parameter.[26] Or in some cases as a chemical or biological composition.

Quantum computing is one such promising hybrid computing architecture that may be able to solve at least some problems that classical computers cannot seem to solve efficiently.[27, 28] Theoretical computer scientists have even carved out a new class of problems, called bounded quantum polynomial (BQP), which consists of problems solvable in polynomial time by quantum computers and yet *not efficiently solvable* by classical computers.[29]

[*]We purposely ignore heuristics and approximations when referring to pure solutions to NP problems.

Interest in quantum computers has been so great over the last few decades, that the research has led to a number of high profile attempts (most still ongoing) to build and compute with quantum computers.[30] And yet, for all their promise, these systems still seem too noisy to be of any practical use.[31] The current implementation of the most promising quantum computer requires cooling structures at near 0 kelvin, even for a small number of bits; or in other words, highly specialized, expensive hardware for limited payoff (so far).

One major advantage of the system discussed in this literature review, is that it is designed and built from the same electronics used by modern computers, e.g. CMOS logic gates, transistors, resistors, capacitors, etc. The novelty is the use of a non-traditional electronics components known as the *memristor*, a circuit element first theorized about in 1971 by Dr. Leon Chua,[2] and used recently in a myriad of applications.[3–5] This circuit element brings with it a variable resistance, which can be capitalized upon as a sort of memory. The authors combine it with other electronics components to construct what they call a ‘self-organizing logic gate,’ which when combined with others in circuit, allow electrical current to flow through them (not in a specific direction) until the entire circuit stabilizes into a solution.[6] In other words, the authors have shown that a problem, realized (wired) as a Boolean circuit, can be solved in polynomial time by simply allowing the electricity applied to it to stabilize. This novel computing architecture is simulated on various NP problems with polynomial running time results for many instances.[7,32] The authors also propose a Turing-complete universal memcomputing machine (UMM).[8]

2. MEMCOMPUTING

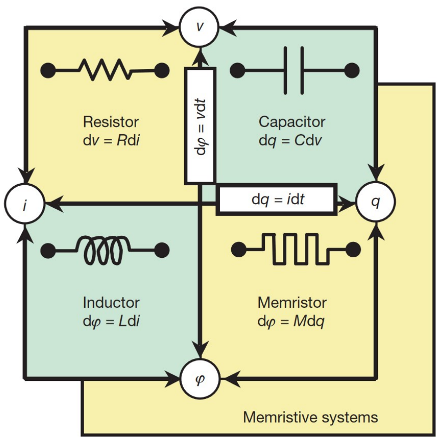
2.1 Memristors

Memristors are a two-terminal device which stores a history of the previous input to define its state.[33] The behavior of general memelements can be described by Eqs. 1 and 2.

$$y(t) = g(\tilde{x}, u, t)u(t) \quad , \tag{1}$$

$$\dot{\tilde{x}} = f(\tilde{x}, u, t) \quad , \tag{2}$$

$u(t)$ is the input signal, $y(t)$ is the output signal, g is the response function that depends on the current input and the internal state of the device $\tilde{x}$. The change in the internal state over time, $\dot{\tilde{x}}$, is governed by the function f which depends on the current internal state and the input. The internal state of a device can be captured by various physical phenomena including spin polarization, positional atomic defects, crystalline uniformity, etc. When these physical phenomena, when subjected to an external voltage, change their internal structure and cause a change in resistance between the two terminals, they display memristance capabilities.



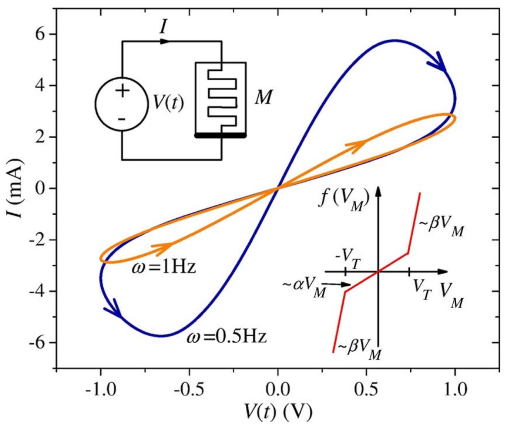


Figure 2. Left: Memristors can be seen as the fourth two-terminal, passive circuit element that relates the charge q to the magnetic flux linkage ψ. Right: The response curve of a memristor when subject to a varying input voltage. Notice the non-linear relationship between the current and voltage.

2.2 Architecture

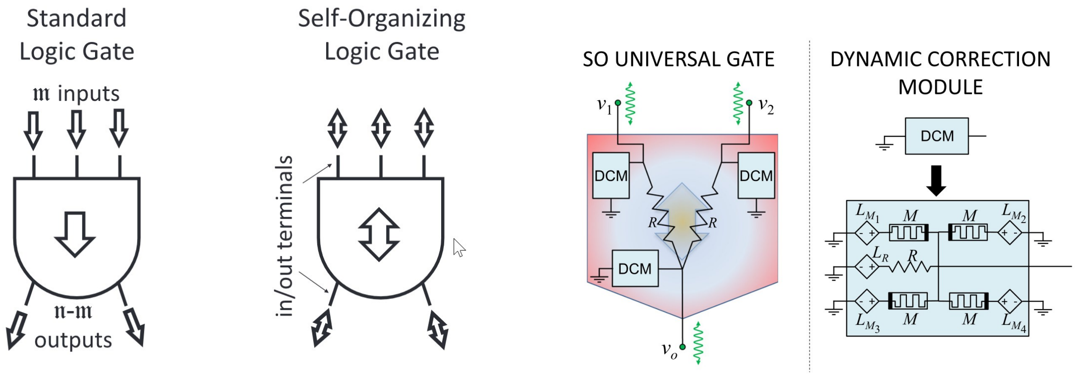


Figure 3. A terminal agnostic self organizing logic gate is like a traditional Boolean logic gate in terms of its input-output relationship, but it allows electrical current to flow in either direction. The dynamic correction modules ensure that Boolean relationship defined by the gate is always satisfied.

Leveraging the capabilities of memristors can be achieved by constructing Self-Organizing Universal Logic Gates (SOLG). SOLGs are terminal-agnostic devices where any of the terminals can be the input or the output. The SOLG self-organizes to satisfy the relationship between “inputs” and “outputs” of the conventional Boolean gate the SOLG mimics. This means that a signal can be applied to the “output” of the SOLG, and the “inputs” will be driven to a state that satisfies the Boolean relationship. For example, for a SOLG AND gate, if a one is applied to the “output”, the “inputs” will both converge to ones. If a zero is applied, the “inputs” will converge to one of three states: either (zero, zero), (zero, one), or (one, zero). The specific type of logic gate the SOLG takes on is defined by the voltage sources in the Dynamic Correction Module.[7]

By combining the SOLGs together, a Self-Organizing Logic Circuit (SOLC) can be constructed. Since the individual SOLGs can be "run in reverse", the composite SOLCs have the same capability. This property can be useful for solving problems that are easier in the reverse direction than the forward direction. For example, Prime Factorization is easy in reverse, simply a multiplication of all the prime factors to produce the product. However, the forward direction is more difficult and is known to be NP. Using SOLCs, the integer can be applied to the "output" terminals and the internal state of the machine will self-organize into a stable state to produce the solution on the "input" terminals.

2.3 Dynamical Systems Analysis

SOLCs can be viewed as a type of dynamical system that evolves in continuous time. The authors have shown that the dynamics governing UMMs will always find a solution if one exists, will converge within a bounded amount of time which depends on the problem size, and will not exhibit periodic or chaotic behavior unlike more general dynamical systems.[6]

2.4 Universality

Memcomputers are universal, in that they not only are Turing complete, but also that they can simulate both reservoir computers (liquid-state machine) and quantum computers, which means that Memcomputers are universal computers.[8]

2.4.1 Universal Memcomputing Machines

Univeral Memcomputing Machines (UMM) are an ideal machine formed by the interconnection of memprocessors (or memcells). The concept of a Universal Memcomputing Machine parallels that of a Universal Turing Machine (UTM).[34] These machines are intrinsically parallel. By intrinsically parallel, we mean that these machines do not operate on each input cell individually, rather, they read the input cells as a whole and write to the output cells as a whole, at once.[8] In addition, such machines are functionally polymorphic, meaning that the transition function which converts the input to the output may change dynamically after each time step.

A UMM, much like a UTM, is an ideal machine that instead of having an infinitely long memory tape, is instead a bank of interconnected memory cells (memprocessors), able to perform either digital or analog operations and controlled by a control unit.[34] The control unit is responsible for sending the signal which configures the internal state of the memprocessors. It is the memprocessors current state, coupled with this signal, which determines the next state of the processor, thus, the processor gives rise to the intrinsic parallelism and functional polymorphism as described above.[34]

2.4.2 Turing Completeness

A Universal Turing Machine has traditionally been realized by the von Neumann architecture, a computing paradigm whereby the memory and CPU are physically separate entities.[34] From there evolved the Harvard architecture, which is similar to the von Neumann architecture except that not only would the memory be separate from the CPU, the memory itself would be separated into data and instruction. Another architectural model, the most common one used today in our modern CPUs and GPUs, is known as the "pipelined architecture", where stages are connected to each other, and the output of one stage becomes the input of another.[34] Although it has not yet been proven that Turing machines and UMMs are equivalent, it has been shown that UMM's are Turing-Complete, that is, they are at least as powerful as Universal Turing machines and represent a model that can, at the very least, simulate the behavior of Turing machines.[34]

2.4.3 Liquid-State Machines

A liquid-state machine (LSM) is a type of neural network, although it is different than most modern neural networks because it does not require the hidden layers (the layers between input and output) to be trained.[8,35] Input and output therefore, is mapped non-linearly. An LSM can be thought of as a container of water, whereby sending an input signal into the LSM is the equivalent of dropping stones into the water, and the change in internal states is equivalent to the propagation of waves. Depending on which stone is dropped, different waveforms are created. A function is then trained to map the waveforms to output states. UMMs are liquid-state complete.[8] This means that they can simulate any liquid-state machine (although, this may require too many resources to be practical).

2.4.4 Quantum Computers

A quantum computer utilizes quantum mechanics to perform computational tasks that otherwise would be intractable. It performs computations in a probabilistic manner. UMMs are quantum-complete, that is, they can simulate any quantum computer.[8] Quantum computers take advantage of quantum entanglement between q-bits to achieve intrinsic parallelism. UMMs use instantions to achieve intrinsic parallelism using non-local interactions to communicate between SOLGs in real-time.

3. EXPERIMENTS & RESULTS

The authors perform simulations of self-organizing logic circuits to solve the integer factorization and the subset-sum problem.[7] All simulations performed were able to find a solution to the problem for the given parameters within a time polynomial with the input size. No periodic orbits or strange attractors were discovered when the problem instance had a solution and the simulations did not converge when a solution did not exist. For both problems, the circuit complexity scales polynomially with the problem instance and the time step does not scale since it only depends on the convergence time of the individual SOLG devices.

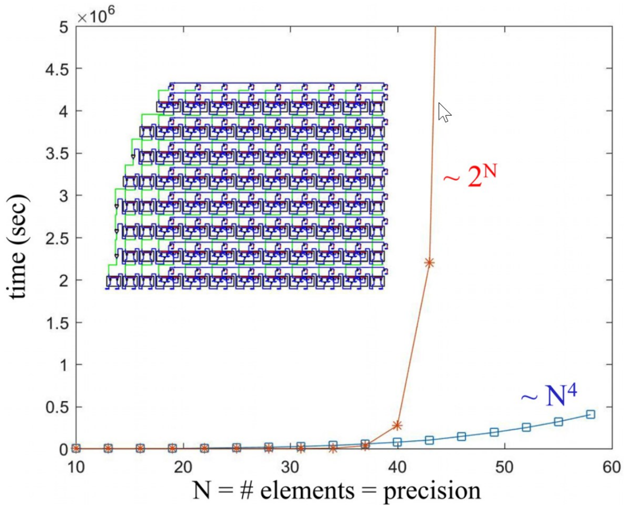


Figure 4. From Di Ventra and Traversa,[6] shows time to compute instances of the Subset-Sum problem where the number of elements is equal to the number of bits of precision. The classical computing time is shown in orange and the memcomputing approach in blue. The topology of the DMM network used is shown in the top-left corner.

One drawback of the simulations was on the size of the problem instances considered. Due to the need to simulate the dynamical systems that described the circuits and the large number of independent variables, the study only considered an integer factorization problem at most 18 bits in size. The subset-sum problem considered inputs and a target sum with at most 9 bits of precision. These size problems are not out of reach of classical computers and could be solved within a reasonable amount of time using a brute-force method.

A later study[36] studied how the performance of the approach scaled with larger problem sizes. The tests were performed on the Max-E3SAT problem, an optimization problem similar to 3SAT that attempts to, in addition to satisfying the expression, also find the most number of true variables. The authors focus on difficult instances of the problem for with no efficient approximation method is known. The tests showed a linear increase in time and memory usage up to problem instances with 64×10^6 variables and 320×10^6 clauses before the 128 Gb of system RAM was exhausted. These scalability tests provide more evidence to support the claims of the authors. Even more interesting is the fact that these results were achieved in *software* without the need for specialized hardware or restrictive constraints on the types of problems. If the capabilities of this approach are reproducible, this method for tackling difficult problems should garner more support in the coming future.

4. CONCLUSION

Computer science has long since reached a complexity wall with regards to NP (and harder = NP+) problems. Unless we discover that P=NP, we may never be able to (purely) solve interesting instances of NP+ problems with classical Church-Turing-Von Neumann computers. Because of this, it is in our best interest to seek out additional, supplementary, or even complementary models of computation to help us overcome such problems. And quantum computing, while promising in theory, has yet to reach implementation maturity. Throughout our rich computing history, analog and hybrid digital-analog computers have been designed, built and operated to overcome specific challenges. Perhaps these non-traditional, non-classical (i.e. non-digital-discrete-timestep-Turing-Von Neumann) machines can be harnessed to overcome the challenges of NP+ problems.

The authors behind the research in this literature review proposed and simulated a novel computing system (a memcomputer), built from existing electronics components and Boolean logic circuitry, with the apparent ability to solve interesting instances of NP problems efficiently. Their system does so by leveraging the intrinsic parallelism inherent to a network of memristors, which are influenced by each other as electrical current flows across the system to achieve a stable state. In other words, the authors have shown that an NP problem, realized as a Boolean circuit, can be solved in polynomial time by allowing the electric signal applied to it to stabilize.

While the authors of this research showed results of simulations to this effect, and have partnered with technology companies to build such systems, they have yet to show a physical version of such a system implemented in hardware. Although memcomputing is a newly proposed model of computation, it may be subject to implementation challenges, like quantum computing. However, the software simulations and the scalabilty of their approach for the problem instances tested do show promise and it will be interesting to see where this technique will lead in the future.

REFERENCES

[1] MemComputing, Inc., "MemComputing Website." https://memcpu.com (2019). Accessed: 2019-04-03.

[2] Chua, L., "Memristor-The missing circuit element," *IEEE Transactions on Circuit Theory* **18**(5), 507–519 (1971).

[3] Strukov, D. B., Snider, G. S., Stewart, D. R., and Williams, R. S., "The missing memristor found," *Nature* **453**, 80–83 (May 2008).

[4] Ventra, M. D., Pershin, Y. V., and Chua, L. O., "Circuit Elements With Memory: Memristors, Memcapacitors, and Meminductors," *Proceedings of the IEEE* **97**, 1717–1724 (Oct. 2009).

[5] Ho, Y., Huang, G. M., and Li, P., "Nonvolatile memristor memory: Device characteristics and design implications," in [*2009 IEEE/ACM International Conference on Computer-Aided Design - Digest of Technical Papers*], 485–490 (Nov. 2009).

[6] Di Ventra, M. and Traversa, F. L., "Perspective: Memcomputing: Leveraging memory and physics to compute efficiently," *Journal of Applied Physics* **123**, 180901 (May 2018).

[7] Traversa, F. L. and Di Ventra, M., "Polynomial-time solution of prime factorization and NP-complete problems with digital memcomputing machines," *Chaos: An Interdisciplinary Journal of Nonlinear Science* **27**, 023107 (Feb. 2017).

[8] Pei, Y. R., Traversa, F. L., and Di Ventra, M., "On the Universality of Memcomputing Machines," *arXiv:1712.08702 [cs]* (Dec. 2017).

[9] Turing, A. M., "On computable numbers, with an application to the Entscheidungsproblem," *Proceedings of the London mathematical society* **2**(1), 230–265 (1937).

[10] Von Neumann, J., "First Draft of a Report on the EDVAC," *IEEE Annals of the History of Computing* **15**(4), 27–75 (1993).

[11] Soare, R. I., "Turing oracle machines, online computing, and three displacements in computability theory," *Annals of Pure and Applied Logic* **160**(3), 368–399 (2009).

[12] Garey, M. R., Johnson, D. S., and Stockmeyer, L., "Some simplified NP-complete problems," in [*Proceedings of the sixth annual ACM symposium on Theory of computing*], 47–63, ACM (1974).

[13] Ullman, J. D., "Np-complete scheduling problems," *Journal of Computer and System sciences* **10**(3), 384–393 (1975).

[14] Černỳ, V., "Quantum computers and intractable (NP-complete) computing problems," *Physical Review A* **48**(1), 116 (1993).

[15] Bell, P. C., Hirvensalo, M., and Potapov, I., "The Identity Problem for Matrix Semigroups in SL2 () is NP-complete," in [*Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms*], 187–206, SIAM (2017).

[16] Papadimitriou, C. H., "Np-completeness: A retrospective," in [*International Colloquium on Automata, Languages, and Programming*], 2–6, Springer (1997).

[17] Bell, P., "HYCOMP 250 the first desktop hybrid analog/digital computing system,"

[18] Ocker, W. and Teger, S., "HYTRAN: a software system to aid the analog programmer," in [*Proceedings of the October 27-29, 1964, fall joint computer conference, part I*], 291–298, ACM (1964).

[19] Ross, L. W., "Analog Computers Literature Review," *SIMULATION* **4**(3), 201–205 (1965).

[20] Hubbard, A. E. and Geisler, C. D., "A hybrid-computer model of the Cochlear Partition," *The Journal of the Acoustical Society of America* **51**(6B), 1895–1903 (1972).

[21] Cyganek, B. and Gruszczyński, S., "Hybrid computer vision system for drivers' eye recognition and fatigue monitoring," *Neurocomputing* **126**, 78–94 (2014).

[22] Guo, N., Huang, Y., Mai, T., Patil, S., Cao, C., Seok, M., Sethumadhavan, S., and Tsividis, Y., "Continuous-time hybrid computation with programmable nonlinearities," in [*ESSCIRC Conference 2015-41st European Solid-State Circuits Conference (ESSCIRC)*], 279–282, IEEE (2015).

[23] Karl, L., "DNA computing: arrival of biological mathematics," *The mathematical intelligencer* **19**(2), 9–22 (1997).

[24] Păun, G., Rozenberg, G., and Salomaa, A., [*DNA computing: new computing paradigms*], Springer (1998).

[25] Matsumaru, N., Centler, F., di Fenizio, P. S., and Dittrich, P., "Chemical organization theory as a theoretical base for chemical computing," in [*Proceedings of the 2005 Workshop on Unconventional Computing: From Cellular Automata to Wetware*], 75–88, Luniver Press (2005).

[26] Platzer, A. et al., "Analog and hybrid computation: Dynamical systems and programming languages," *Bulletin of EATCS* **3**(114) (2014).

[27] Feynman, R. P., "Simulating physics with computers," *International journal of theoretical physics* **21**(6), 467–488 (1982).

[28] Shor, P. W., "Algorithms for quantum computation: Discrete logarithms and factoring," in [*Proceedings 35th annual symposium on foundations of computer science*], 124–134, Ieee (1994).

[29] Tal, A. and Raz, R., "Oracle separation of BQP and PH," (2018).

[30] Jha, R., Das, D., Dash, A., Jayaraman, S., Behera, B. K., and Panigrahi, P. K., "A novel quantum N-Queens solver algorithm and its simulation and application to satellite communication using IBM Quantum experience," *arXiv preprint arXiv:1806.10221* (2018).

[31] Preskill, J., "Quantum Computing in the NISQ era and beyond," *Quantum* **2**, 79 (2018).

[32] Traversa, F. L., Cicotti, P., Sheldon, F., and Di Ventra, M., "Evidence of an exponential speed-up in the solution of hard optimization problems," *arXiv:1710.09278 [nlin]* (Oct. 2017).

[33] Swenson, G., "Thanks for the Memory: NIST Takes a Deep Look at Memristors." `https://www.nist.gov/news-events/news/2018/01/thanks-memory-nist-takes-deep-look-memristors` (2018). Accessed: 2019-04-09.

[34] Traversa, F. L. and Ventra, M. D., "Universal memcomputing machines," *CoRR* **abs/1405.0931** (2014).

[35] Wilczek, F., "Hidden layers." `https://www.edge.org/response-detail/10351` (2011). Accessed: 2019-04-25.

[36] Sheldon, F., Cicotti, P., Traversa, F. L., and Di Ventra, M., "Stress-testing memcomputing on hard combinatorial optimization problems," *arXiv preprint arXiv:1807.00107* (2018).