Applied Mathematics & Information Sciences
An International Journal

http://dx.doi.org/10.12785/amis/070402

Creating Unorganised Machines from Memristors

Gerard Howard[∗], *Larry Bull, Ben de lacy Costello and Andrew Adamatzky*

Unconventional Computing Group, University of the West of England, Coldharbour Lane Bristol BS16 1QY UK

Received: 5 Dec. 2012, Revised: 20 Feb. 2013, Accepted: 10 Mar. 2013
Published online: 1 Jul. 2013

Abstract: There is growing interest in memristive devices following their recent nanoscale fabrication. This paper describes initial consideration of the implementation of artificial intelligence within predominantly memristive hardware. In particular, versions of Alan Turing's discrete dynamical network formalism — the unorganised machine — are used as the knowledge representation scheme and a population-based search technique is used to design appropriate networks. Issues including memristor count and global network synchrony are compared for two memristive logic implementations (NAND and IMP) on a well-known simulated robotics benchmark task. It is shown that IMP networks are harder to design than NAND, but are simpler to implement and require fewer processor cycles.

Keywords: Memristors, Unorganised Machines, Discrete Dynamical Systems, Evolutionary Computing, Logic Gates

1. Introduction

The memory-resistor or "memristor", identified experimentally by Widrow [19] and then theoretically by Chua [6] has become the focus of significant attention after the fabrication of nano-scale devices by Williams et al. through sandwiching Titanium Dioxide between two platinum electrodes (e.g., see [20] for details). Two of the anticipated applications of this fourth fundamental circuit element are non-volatile memory and neuromorphic architectures, the latter almost exclusively as synapse analogues in conjunction with standard Complementary Metal Oxide Semiconductor (CMOS) neurons. We are interested in the hardware implementation of artificial intelligence within predominantly memristive technology, e.g., as low-energy devices, and this article presents initial results from that endeavour.

Borghetti et al. [2] have recently described how their aforementioned memristors can be used for Boolean logic operations (see also [12][13] for related work). In particular, they demonstrate how two-input material implication (IMP) can be implemented using two memristors and a load resistor, further showing how this enables the implementation of two-input NAND. In 1948 Alan Turing produced an internal paper in which he presented a formalism he termed "unorganised machines" (UM) by which to represent intelligence within computers (eventually published as [16]). These consisted of various types, the simplest being "A-type" unorganised machines (AUM), which were composed of two-input NAND gates connected into disorganised networks. Here each NAND gate node updates in parallel on a discrete time step, with the output from each node arriving at the input of the node(s) on each connection for the next time step. The structure of unorganised machines is therefore very much like a simple artificial neural network with recurrent connections and this has led to their also being known as "Turings connectionism" (e.g., [7]). Moreover, as Teuscher [15] has noted, AUM are (discrete) nonlinear dynamical systems and therefore have the potential to exhibit complex behaviour despite their construction from simple elements. The current work aims to explore the use of AUM as a general representation scheme for implementation within memristive hardware. Turing suggested a number of mechanisms by which to program UM but a culture-inspired search technique is used here, presented following the highlighting of the analogy he makes between culture and intellectual search in the 1948 paper [3].

The main contribution of this paper is the comparison of memristive IMP-based circuitry to traditional NAND circuitry, and extensions using single-memristor nodes and asynchronous updating. IMP and NAND circuits are tested on a suite of logic problems and two robotics navigation tasks. Overall, it is shown that IMP circuits are harder to automatically design, but provide benefits in that fewer components are required for a given implementation. Single-memristor elements are shown to

[∗] Corresponding author e-mail: david4.howard@uwe.ac.uk provide a more complex design challenge, yet require fewer memristors overall to solve a given problem. Asynchronous updating is shown to require larger networks but bring an order-ambivalence which may be highly beneficial to physical circuit implementations. The simulated circuits generated herein are intended to serve as a precursor to deployment of purely hardware circuits, where the beneficial characteristics of non-volatility and charge-dependent resistance are implicit in their chemical makeup.

2. Materials and Methods

2.1. Representation: Unorganised Machines

A-type unorganised machines have a finite number of possible states and they are deterministic, hence such networks eventually fall into a basin of attraction. It has previously been shown that the typical percentage of nodes/gates which change state per update cycle for randomly created networks, for various numbers of nodes N, rises to near 100% after around 15 update cycles, which is when an attractor is typically reached [3]. As such, networks are run for 15 update cycles here before the state of the output node(s) is ascertained. The I inputs of a given task are applied as constant signals to the first connection of the first I nodes in an AUM. This scheme follows that indicated by Turing, although others are possible [3]. Figure 1 shows a simple example of an AUM for a task with one input and one output.

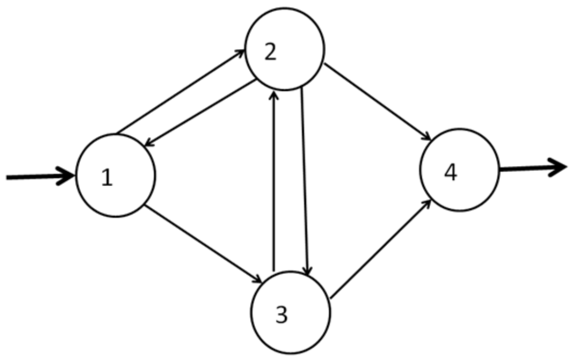
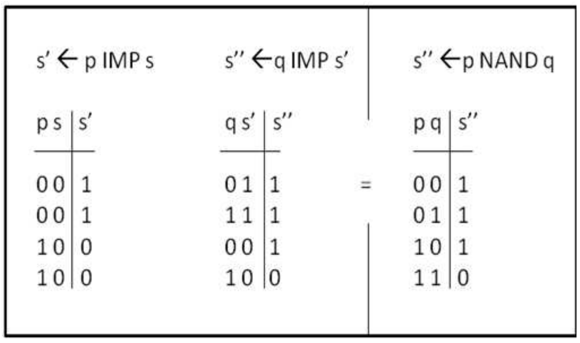


Figure 1: Example A-type unorganised machine consisting of four two-input NAND gate nodes (N=4), with one input (node 1) and one output (node 4) as indicated by the bold arrows.

2.2. Target Hardware: Memristors

A memristor can be formally defined as a passive two-terminal electronic device that is described by the non-linear relation between the device terminal voltage, v, terminal current, i (which is related to the charge q transferred onto the device), and magnetic flux, $i = W(\varphi)v$. Memristance (M) is a nonlinear function: $M(q) = d\varphi(q)/dq$.

As noted above, Borghetti et al. [2] have presented a scheme by which memristors can be used as switches to implement Boolean logic. They use two memristors to realise material implication (IMP), a much-forgotten function originally highlighted by Whitehead and Russel [18]. The authors then construct two-input NAND, using two IMP gates in serial from three memristors and a constant False signal as shown in Figure 2. The reader is referred to their paper for full circuit and voltage details.



$s' \leftarrow p \text{ IMP } s$

p s	s'
0 0	1
0 0	1
1 0	0
1 0	0

$s'' \leftarrow q \text{ IMP } s'$

q s'	s''
0 1	1
1 1	1
0 0	1
1 0	0

=

$s'' \leftarrow p \text{ NAND } q$

p q	s''
0 0	1
0 1	1
1 0	1
1 1	0

Figure 2: Showing how NAND can be created from two IMP gates.

As NAND is computationally complete, anything computable can therefore be implemented using memristors in principle. However, Turings AUM can be seen as a low-level representation scheme which can be mapped directly onto memristive hardware due to its use of two-input NAND gates. As will be shown, AUM are also amenable to design using automated search techniques.

2.3. Search: Imitation Programming

Imitation Programming (IP) is a population-based stochastic search process which has been found to be competitive with related evolutionary search techniques for network design [3]. Pseudocode is shown in Figure 3.

For AUM design, IP utilizes a variable-length representation of pairs of integers defining node inputs, each with an accompanying single bit defining the nodes start state. There are three imitation operators - copy a node connection, copy a node start state, and change size through copying. In this paper, each operator can occur with or without error, with equal probability, such that an individual performs one of the six during the imitation process as follows:

```
BEGIN
INITIALISE population with random candidate solutions
EVALUATE each candidate
REPEAT UNTIL (TERMINATION CONDITION) DO
        FOR each candidate solution DO
                SELECT candidate(s) to imitate
                CHOOSE component(s) to imitate
                COPY the chosen component(s) with ERROR
                EVALUATE new solution
                REPLACE IF (UPDATE CONDITION) candidate with new solution
        OD
OD
END
```

Figure 3: Showing pseudocode of the Imitation Programming algorithm.

To copy a node connection, a randomly chosen node has one of its randomly chosen connections set to the same value as the corresponding node and its same connection in the individual it is imitating. When an error occurs, the connection is set to the next or previous node (equal probability, bounded by solution size). Imitation can also copy the start state for a randomly chosen node from the corresponding node, or do it with error (bit flip here). Size is altered by adding or deleting nodes and depends upon whether the two individuals are the same size. If the individual being imitated is larger than the copier, the connections and node start state of the first extra node are copied to the imitator, a randomly chosen node being connected to it. If the individual being imitated is smaller than the copied, the last added node is cut from the imitator and all connections to it re-assigned. If the two individuals are the same size, either event can occur (with equal probability). Node addition adds a randomly chosen node from the individual being imitated onto the end of the copier and it is randomly connected into the network. The operation can also occur with errors such that copied connections are either incremented or decremented. For a problem with a given number of binary inputs I and a given number of binary outputs O, the node deletion operator has no effect if the parent consists of only $O + I + 2$ nodes. The extra two inputs are constant True and False lines. Similarly, there is a maximum size (100) defined beyond which the growth operator has no effect.

In this article, each individual in the population P creates one variant of itself and it is adopted if better per iteration. In the case of ties, the solution with the fewest number of nodes is kept to reduce size, otherwise the decision is random. The individual to imitate is chosen using a roulette-wheel scheme based on proportional solution utility, i.e., the traditional reproduction selection scheme used in Genetic Algorithms [10]. Other forms of updating, imitation processes, and imitation selection are, of course, possible [3].

3. Results I: Logic

In the following, three well-known logic problems are used to begin to explore the characteristics and capabilities of the general approach. The multiplexer task is used since they can be used to build many other logic circuits, including larger multiplexers. These Boolean functions are defined for binary strings of length $l = k + 2k$ under which the k bits index into the remaining $2k$ bits, returning the value of the indexed bit. Hence the multiplexer has multiple inputs and a single output. The demultiplexer and adders have multiple inputs and multiple outputs. As such, simple examples of each are also used here. In all cases, the correct response to a given input results in a quality increment of 1, with all possible binary inputs being presented per solution evaluation. Upon each presentation of an input, each node in an AUM has its state set to its specified start state. The input is applied to the first connection of each corresponding I input node. The AUM is then executed for 15 cycles. The value on the output node(s) is then taken as the response. All results presented are the average of 20 runs, with P=20. Experience found giving initial random solutions $N = O + I + 2 + 30$ nodes was useful across all the problems explored here, i.e., with the other parameter/algorithmic settings described.

Figure 4(a), (c), and (e) show the performance of IP to design NAND AUM on k=2 versions of the three tasks: the 6-bit multiplexer (opt. 64), 2-bit adder (opt. 16) and 6-bit demultiplexer (opt. 8) respectively. As can be seen, optimal performance is reached in all cases, well within the allowed time, and that the AUM solution sizes are adjusted to the given task. Hence any of the resultant (dynamical) circuits may be implemented almost directly into appropriately configured memristive hardware, three per NAND gate.

As noted above, Borghetti et al.[2] have implemented material implication as the basic logic function within memristive hardware, using two per IMP gate. The same experiments were repeated using IMP at each node, as opposed to NAND as Turing specified. Figure 4(b), (d), and (f) show the comparative performance. As can be

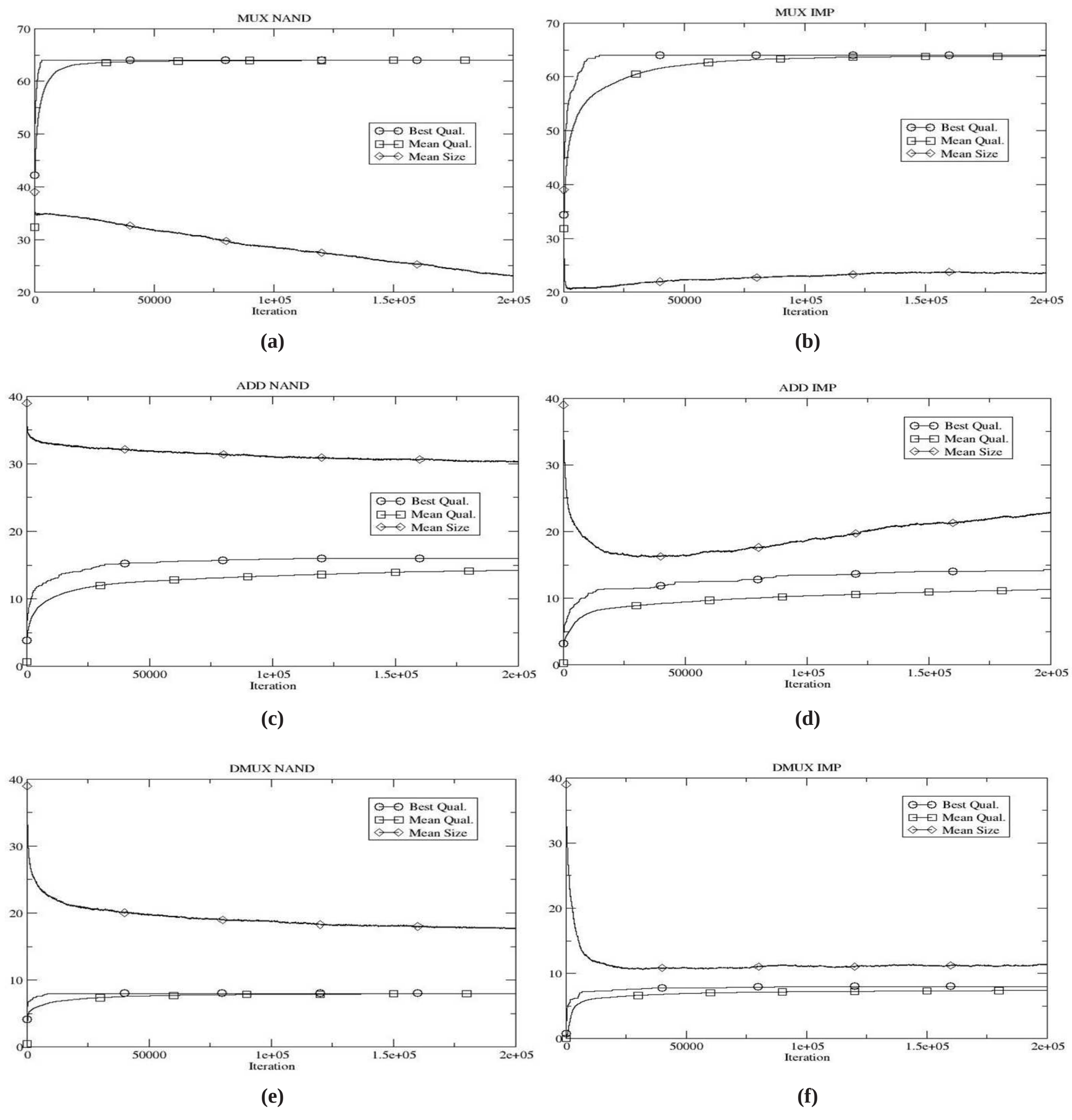


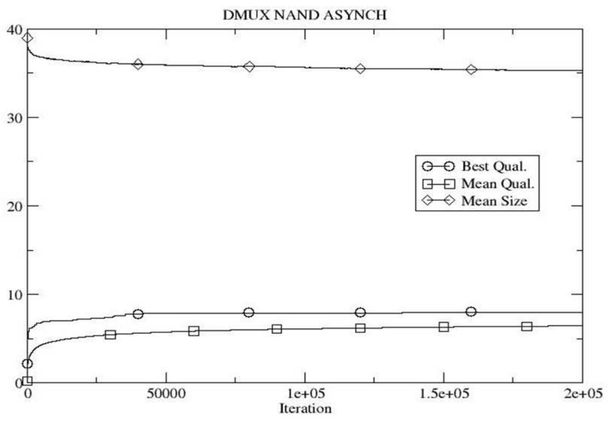
Figure 4: Showing the performance of AUM on the three logic tasks, and comparative performance when IMP is used at each node as opposed to NAND.

seen, use of IMP means it takes longer to discover an optimal solution in all cases (T-test, p<0.05). Indeed, optimality for the 2-bit adder is not reached in the allotted time (longer runs needed, not shown). However, when optimality is reached, the size of the AUM is smaller for all tasks in terms of the nodes used with IMP (T-test, p<0.05). This implies IP does not construct NAND gates from two IMP gates. Moreover, given only two memristors are needed per gate, the equivalent circuits are more efficient when hardware implementation is considered (T-test, p<0.05). The same has been found for *k*=3 versions of the three problems (not shown).

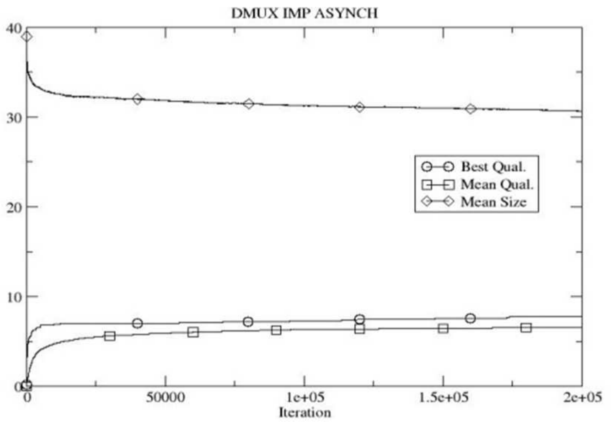
4. Results II: Asynchronous Logic

As stated above, Turings unorganized machines were originally defined as updating synchronously in discrete time steps. However, there is no reason why this should be the case and it has recently been shown [3] that their behaviour changes significantly when nodes are updated asynchronously. That is, in a random order, with replacement, and N updates allowed per equivalent cycle. Around 20% of nodes are found to change state per update cycle, as opposed to around 100% in the synchronous case. There may be significant benefits from relaxing the synchrony constraint: asynchronous CMOS devices are known to have the potential to consume less power and dissipate less heat (e.g. [17]); they are also known to have the potential for improved fault tolerance, particularly through delay insensitive schemes (e.g. [8]); removing the need for a global clock signal may ease the use of novel physical memristive devices; and, given the non-volatile nature of memristors, efficient power “pulsing” schemes may be envisaged for updating.

Figure 5 shows an example of the performance of IP to design asynchronous AUM using the aforementioned random order updating, together with those using IMP gates per node on tasks from above. That is, an assumption of local synchrony per node in the AUM is assumed, but not global synchrony. As can be seen, the same general result as in the synchronous case is again true IMP solutions are smaller but take longer to find (T-test, $p<0.05$). It can also be noted that the use of asynchronous node updating means the design task is significantly slower in each case (T-test, $p<0.05$) and it has altered the topology of the networks, with more nodes (T-test, $p<0.05$) being exploited. This is perhaps to be expected since redundancy, e.g., through sub-network duplication, presumably provides robustness to exact updating order during computation.



(a)



(b)

Figure 5: (a) shows the performance of asynchronous AUM on one of the logic tasks. (b) Shows the comparative performance when IMP is used at each node as opposed to NAND.

5. Results III: Synapse

As noted above, one of the largest areas of current interest in memristors is their use as hardware implementations of synapse within neuromorphic hardware (e.g., [1]). The first known example of such work was undertaken [19] with a device termed a “memistor” within a hardware implementation of his seminal Adaline neural network. A memistor was used to store the current weight setting of each neuron input and created by the electro-plating of a pencil lead with copper; the conductance of the memistor was varied by varying the amount of copper plating on the lead at any time.

Given their temporally dynamic nature, a very simple approximation of a single memristive element has been included within AUM along with the logic gate nodes. These may be seen as synapse-like but, in keeping with AUM, less prescriptive in placement. This is done using the Widrow-Hoff delta rule in the form of single-input nodes. Of course, the actual non-linear behaviour of a given memristive device primarily depends upon the substrate in which it is fabricated (e.g., see[11]). The resistive state (M) of a node is maintained using the running average of inputs to the node: $M \leftarrow M + \beta(currentinput - M)$, with learning rate β=0.2. If $M \leq 0.5$, the state of the node is equal to the current input and is logical ‘0’ otherwise. Hence the resistive behaviour of the node varies based upon the temporal sequence of inputs it receives. The imitation process is altered to include the potential copying of node type, with and without error. Nodes have a 50% chance of being either logic gates or single memristors at initialization.

Figure 6 shows example results on the same logic tasks as before, using synchronous updating. As can be seen, compared to the results shown in Figure 6, the additional single memristive nodes appear to make the design problem harder as it takes longer to find optimality

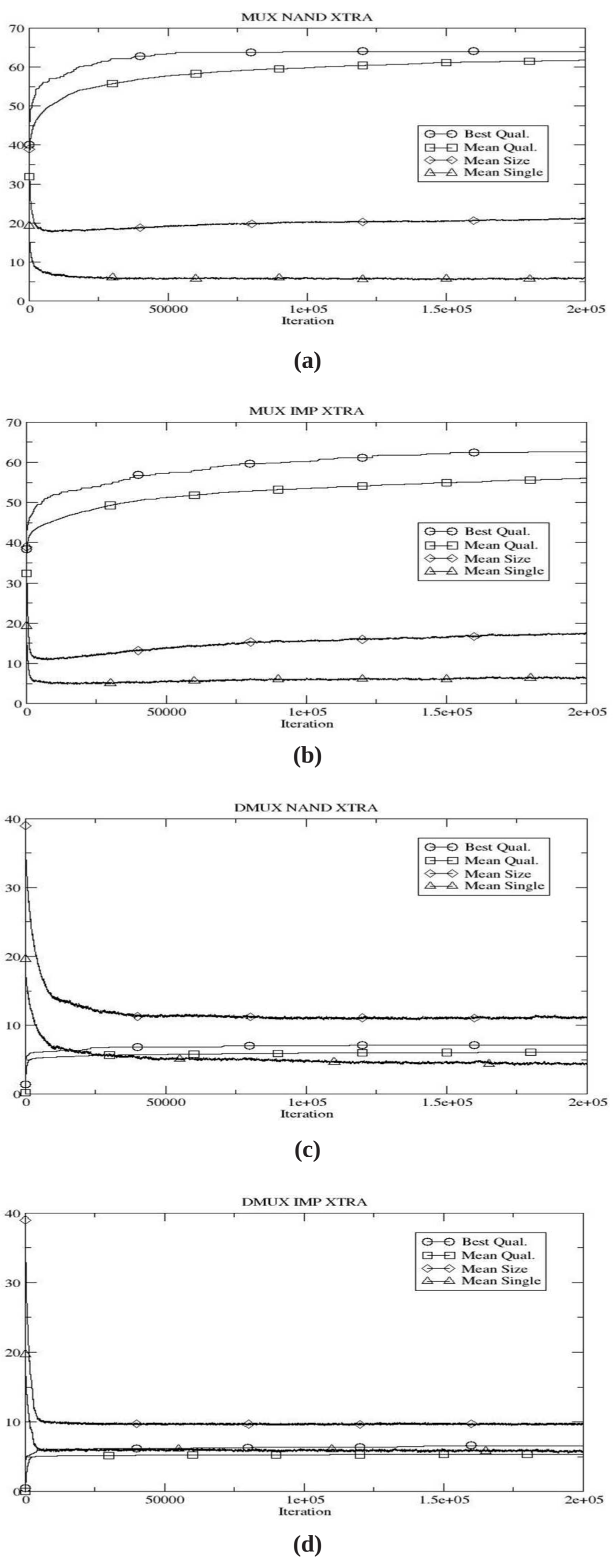


Figure 6: (a) shows the performance of AUM augmented with single memristor nodes on two of the logic tasks. (b) shows the comparative performance when IMP is used at each node as opposed to NAND.

in all cases (T-test, p<0.05). For example, on the 6-bit multiplexer, the NAND node AUM takes almost 100,000 iterations here whereas it previously took 3000 iterations. Similarly, the IMP node AUM takes over 200,000 iterations here whereas it previously took around 15,000. However, the resulting AUM contain fewer nodes at optimality in all cases (T-test, p<0.05). Again, given only one memristor is needed in the new type of nodes, the equivalent circuits are more efficient when hardware implementation is considered (T-test, p<0.05). Similar results were found using asynchronous updating (not shown).

6. Results IV: Robotics

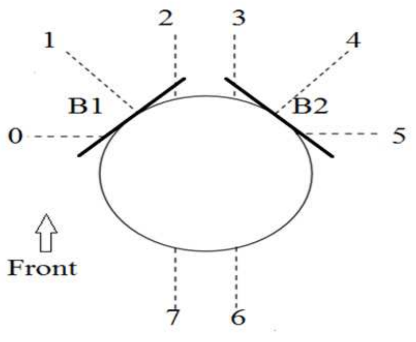
For this initial study, AUM have also been applied to a simple multi-step control problem in which a robot must navigate an obstacle to reach a light source. The chosen robotics simulator was Webots [14].

6.1. The Agent

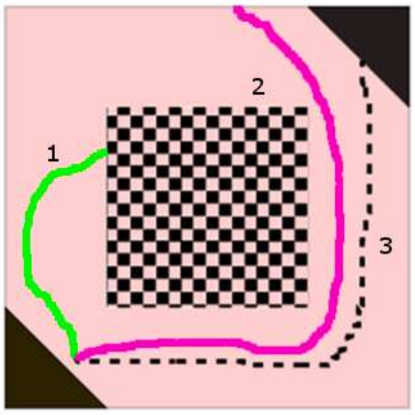
The agent was a simulated Khepera II robot with eight light sensors and eight IR distance sensors; three of each sensor type were used as input (sensors at positions 0, 2 and 5 as shown in Figure 7(a)). At each step (64ms in simulation time), the agent sampled its light and IR sensors, whose scaled response values ranged from 0 (no light / no object detected) to 1 (fully illuminated / object very close). To make this continuous-valued input amenable to processing by an AUM, each of the 6 sensors was encoded as two-bit binary (sensor value<0.25 = 00, between 0.25 and 0.5 = 01, 0.5 to 0.75 = 10 and >0.75 = 11). The input state was then applied as the first connection of the required 12 input nodes, such that the first input to the first two nodes encodes the first sensor reading, etc. Two further input nodes carry constant “1” and “0” as before. Three actions were possible: forward, (11 or 00 at the two output nodes) and continuous turns to both the left (01) and right (10). Additionally, two bump sensors were added to the front-left and front-right of the agent to prevent it from becoming stuck against an object. If either bump sensor was activated, an interrupt was sent causing the agent to reverse 0.1 units and the agent to be penalised by 10 steps.

6.2. The Environment

The agent was placed within a walled arena which it could not leave, with coordinates ranging from [-1,1] in both *x* and *y* directions. A box was placed centrally in the arena and a light source was placed at *x*=1, *y*=1. The agent’s random start position was constrained (initial $x + y < -1.5$), forcing the agent to learn obstacle



(a)



(b)

Figure 7: (a) Showing the sensor configuration of the Khepera II robot; 3 light sensors and 3 IR sensors at positions 0, 2 and 5 are used to create the environmental input. Each real-valued sensor response is thresholded into two-bit binary for network processing. Bumpers B1 and B2, angled at 45 degrees to the front of the agent, prevent the agent from becoming stuck in the environment. (b) Showing the test environment used in the experiment. The start zone is in the lower left, the goal state is in the upper right. Checkered boxes indicate obstacles. Paths indicate the best-evolved behaviour for asynchronous NAND synapse networks from generation 0 (path 1), generation 250 (path 2), and the final generation (path 3).

avoidance behaviour. The environment is shown in Figure 7(b). The fitness function f ($f > 0$) is shown in (1), the denominator accounts for the position of the goal state (1.6) and the agent's position (pos_x and pos_y); t_s is the current number of steps.

$$f = 1/1.6 - (|pos_x - pos_y|) * 1000 - t_s. \qquad (1)$$

For this task, reaching the goal state (where $x + y > 1.6$) provided the AUM with constant fitness bonus of 2500; optimal performance gives $f \approx$11800, or $\approx$700 steps.

6.3. Experimental Setup

All experiments had a population size of 20. As this was a multi-step problem, start states were used only once (at the start of the trial). An experiment began with the generation of 20 networks of a given node type (IMP or NAND). Every network in the population was then trialled on the test problem, with t_s= 4000 (long enough to allow for initial exploration). A trial began with the placement of the agent in the arena and ended with either (a) location of the reward or (b) a time out. Each step of processing consisted of the receipt of the current state at the input nodes, lasted 15 update cycles, and ended with determination of an action based upon the states of the output nodes.

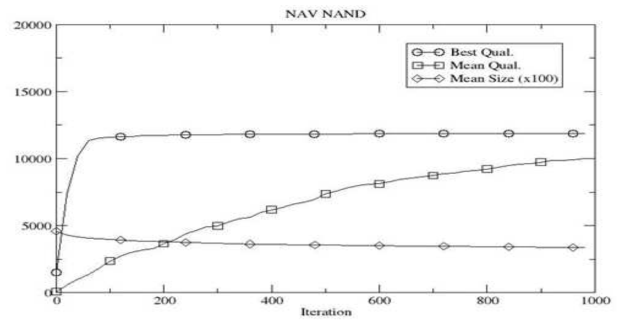
6.4. Results

Figure 8 shows the performance of AUM and the IMP node AUM on the simple navigation task. As can be seen, and as predicted by the above results for logic gate design, the IMP node AUM learns (significantly) more slowly than the original NAND version, with optimality not found in the allowed time; the NAND node AUM are typically optimal around 950 iterations. Figure 8 also shows the performance of the asynchronous AUM with and without additional single memristive nodes. Again, relative performance and behaviour matches that seen above on the logic tasks, as do the IMP node versions (not shown).

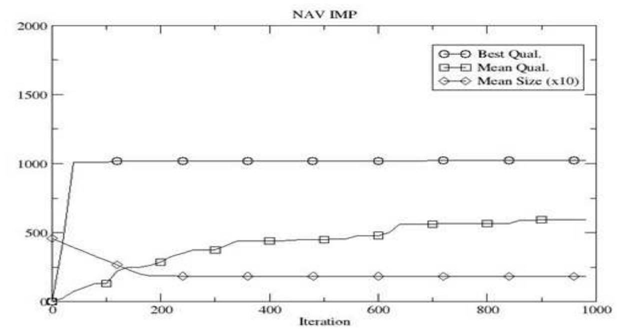
Following [11], to further test the capabilities of the system a dynamic version of the task was also explored. Here, once the agent finds the location of the goal state in the top right corner (Figure 8(b)) it is able to increase its fitness by then moving to the top left corner. It should be noted that the light source does not move and if the agent did not locate the goal in the first part, it cannot receive reward when the goal is moved. The results (not shown) were again as those above with synchronous NAND node AUM learning the task optimally more quickly than any other combination but smaller networks were found with IMP nodes. A 100% change in network states was still possible with 15 updates per AUM cycle, however note that we do not reset start states in these multi-step scenarios, so the networks in these experiments undergo many more state changes in their lifetimes when compared to the logic experiments..

7. Conclusions

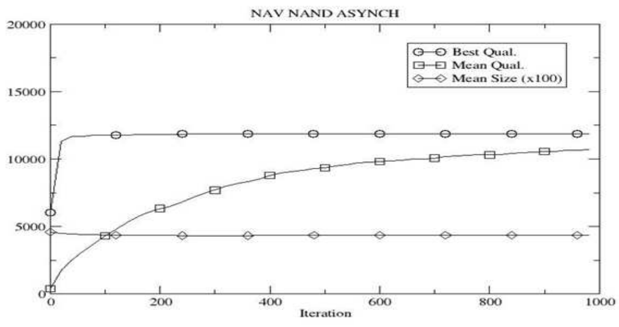
We are interested in the hardware implementation of artificial intelligence within predominantly memristive technology and this research presents initial results from that endeavour using versions of Turings unorganised machine representation.



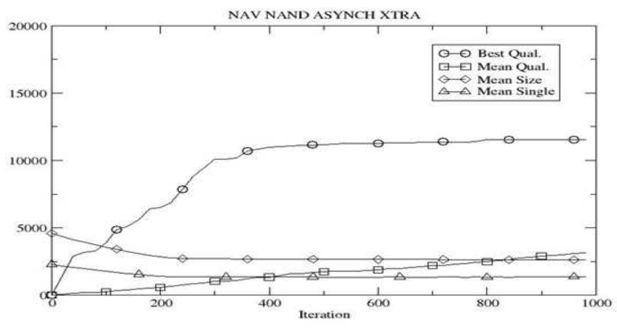
(a)



(b)



(c)



(d)

Figure 8: (a) Showing the performance of the NAND AUM on the robot task. (b) When IMP AUMs are used as opposed to NAND. The other two figures show the utility of the asynchronous NAND AUM (c) and synapse NAND AUM (d).

Results were found to significantly vary based on the type of logic gate used. Pertinent findings include, on all tasks, (i) IMP-based solutions being more compact (in terms the required number of components) than the equivalent NAND circuitry. (ii) Asynchronous solutions were highlighted as an attractive implementation paradigm, as they are required to update less frequently and therefore consume less power and generate less heat then synchronous circuits. Inclusion of the single-memristor synapse (iii) was shown to further reduce the required number of memristors for a given circuit whilst more fully embodying the dynamic nonvolatile state of the device.

We are currently exploring the manufacture of memristive hardware using sol-gel chemistry and drop-coating (e.g., see [9]) with the aim of implementing some of the designs discovered. Not least for this reason, the subjects of asynchrony and memristor count covered here are particularly significant. It can also be noted that the former issue of asynchrony remains an important topic in neuroscience (e.g.,[4]) and has been explored in some neural network models (e.g.,[5]). We suggest our work will highlight this for consideration within neuromorphic hardware. Future work will consider more complex applications.

Acknowledgement

This work was supported by EPSRC Grant no. EP/H014381/1.

References

[1] A. Afifi, A. Ayatollahi, and F. Raissi. Stdp implementation using memristive nanodevice in cmos-nano neuromorphic networks. *IEICE Electronics Express*, 6(3):148–153, 2009.

[2] Julien Borghetti, Gregory S. Snider, Philip J. Kuekes, Joshua J. Yang, Duncan R. Stewart, and R. Stanley Williams. /'Memristive/' switches enable /'stateful/' logic operations via material implication. *Nature*, 464(7290):873–876, April 2010.

[3] Larry Bull. Using genetical and cultural search to design unorganised machines. *Evolutionary Intelligence*, 5:23–33, 2012.

[4] G. Buzsaki. *Rhythms of the Brain*. Oxford University Press, 2006.

[5] Shannon R. Campbell, DeLiang L. Wang, and Ciriyam Jayaprakash. Synchrony and desynchrony in integrate-and-fire oscillators. *Neural Comput.*, 11(7):1595–1619, October 1999.

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

[7] B. J. Copeland and D. Proudfoot. On Alan Turing's anticipation of connectionism. *Synthese*, 108:361–377, 1996.
[8] Jia Di and Parag K. Lala. Cellular array-based delay-insensitive asynchronous circuits design and test for nanocomputing systems. *J. Electron. Test.*, 23(2-3):175–192, June 2007.
[9] Ella Gale, Dave Pearson, Stephen Kitson, Andrew Adamatzky, and Ben de Lacy Costello. Different behaviour seen in flexible titanium dioxide sol-gel memristors dependent on the choice of electrode material. In Schlom D. Tokura Y. Waser R. Heber, J. and M. Wutting, editors, *Technical Digest of Frontiers in Electronic Materials*, pages 577–578. Nature Conferences & Wiley VCH, 2012.
[10] J.H. Holland. *Adaptation in Natural and Artificial Systems*. University of Michigan Press, 1975.
[11] G. Howard, E. Gale, L. Bull, B. Costello, and A. Adamatzky. Evolution of plastic learning in spiking networks via memristive connections. *Evolutionary Computation, IEEE Transactions on*, 2012.
[12] Makoto Itoh and Leon O Chua. Memristor cellular automata and memristor discrete-time cellular neural networks. *International Journal Of Bifurcation And Chaos*, 19(11):3605, 2009.
[13] E Lehtonen, J H Poikonen, and M Laiho. Two memristors suffice to compute all boolean functions. *Electronics Letters*, 46(3):230, 2010.
[14] Olivier Michel. WebotsTM: Professional mobile robot simulation. *International Journal of Advanced Robotic Systems*, 1(1):39–42, 2004.
[15] C Teuscher. *Turing's Connectionism*. Springer, 2002.
[16] A. Turing. Intelligent machinery. pages 91–102. Butterworths, 1968.
[17] Tony Werner and Venkatesh Akella. Asynchronous processor survey. *Computer*, 30(11):67–76, November 1997.
[18] A. N. Whitehead and B Russell. *Principia Mathematica Vol I, 7*. Cambridge University Press, 1910.
[19] B. Widrow. An adaptive 'adaline' neuron using chemical 'memistors'. Technical Report 1533-2, Stanford Electronics Laboratories, Stanford, CA, oct 1960.
[20] J Joshua Yang, Matthew D Pickett, Xuema Li, Douglas A A Ohlberg, Duncan R Stewart, and R Stanley Williams. Memristive switching mechanism for metal/oxide/metal nanodevices. *Nature Nanotechnology*, 3(7):429–433, 2008.

Gerard Howard is a research associate based in the Department of Computer Science and Creative Technologies at UWE. His research to this point has been focused on natural computing and the application of evolutionary approaches to control systems.

Larry Bull is a Professor of Artificial Intelligence and based in the Department of Computer Science and Creative Technologies at UWE. His research interests are in intelligent and unconventional systems, with an emphasis on evolution. He is the founding Editor-in-Chief of the Springer journal Evolutionary Intelligence.

Ben de lacy Costello received his B.Sc. in Applied Chemical Sciences and Ph.D. in Materials Science from the University of the West of England, Bristol. He is now a Senior Research Fellow in the Faculty of Health and Life Sciences and his research is split between producing sensors and systems for early disease diagnosis and nature inspired unconventional computing.

Andrew Adamatzky is a Professor in Unconventional Computing in the Department of Computer Science, ranger of the Unconventional Computing Centre, and a member of Bristol Robotics Lab. He does research in reaction-diffusion computing, cellular automata, physarum computing, massive parallel computation, applied mathematics, collective intelligence and robotics.