Solving AI problems with memristors: A case study for optimal “bin packing”

Dimitrios Stathis, Ioannis Vourkas, and Georgios Ch. Sirakoulis
Department of Electrical and Computer Engineering (ECE), Democritus University of Thrace (DUTh)
Panepistimioupoli, Kimmeria, ECE Building B, Room 122, GR-67100, Xanthi, GREECE
Tel: +30-25410-79547; Fax: +30-25410-79540
{dstathis, ivourkas, gsirak}@ee.duth.gr

ABSTRACT

This paper presents a novel circuit-level Cellular Automata (CA)-inspired computational scheme capable of executing computations within memory. The proposed computing structures exploit the threshold-based resistance switching behavior of memristors and of their multi-state composite components. Array-like circuit structures with memristors are designed and their ability to efficiently solve the classic “bin packing” problem is verified via a simulation-based validation using a published memristor device model. A fundamental memristive cell which implements a one-dimensional CA rule is described in detail and then employed in a sophisticated two-dimensional array able to execute the “First-Fit” (decreasing) bin packing algorithm.

Categories and Subject Descriptors

B.6.1 [**Logic Design**]: Design Styles – Cellular arrays and automata. B.6.3 [**Logic Design**]: Design Aids – Simulation. C.0 [**Computer Systems Organization**]: General – System architectures. F.1.1 [**Computation by Abstract Devices**]: Models of Computation – Unbounded-action devices (e.g., cellular automata, circuits, networks of machines). I.2.m [**Artificial Intelligence**]: Miscellaneous.

General Terms

Theory, Algorithms, Experimentation, Performance, Design, Verification

Keywords

memristor, memristive circuits, cellular automata, bin packing, artificial intelligence

1. INTRODUCTION

“Bin Packing” is a classic problem where a collection of objects of different volumes must be packed into a finite number of fixed-size containers (bins) in a way that minimizes the number of total used bins [1]. In computational complexity theory, it is a combinatorial NP-hard problem with many variations (e.g. linear, 2D, or 3D packing) which find application in several every-day practical problems related to minimization of space or time. Despite its NP-hard computational complexity, optimal solutions can be produced with heuristic algorithms [2]. A straightforward greedy approximation algorithm is the “First-Fit” which provides a fast solution by processing the items in arbitrary order and attempting to place each one into the first bin in which it will fit. If no bin is found, it puts the item in a new empty bin. The required time complexity is $O(n \log n)$, where n is the number of elements to be packed. It has been shown that this algorithm achieves an approximation factor of 2, i.e. the number of bins used by is no more than 2× the optimal number of bins. However, in real applications the sizes to be packed may all be known in advance, hence better results are achieved by packing the largest objects first. Therefore, “First-Fit” can be made much more effective by first sorting the list of elements into decreasing order, thus giving the “First-Fit decreasing” algorithm [3].

Cellular automata (CA) constitute a well-studied inherently parallel computing paradigm of high efficiency and robustness [4]. Owing to their potential to capture globally emerging behavior from collective interaction of simple and local components, they have found successful application in several computational problems [5-9]. When CA-based models are implemented in hardware (HW), circuit design reduces to the design of a single cell and the overall layout results regular with exclusively local interconnections [10-11]; the models are executed fast by exploiting the parallelism of the CA structure, thus meeting the necessary information processing demands. However, memristors, a recently discovered class of two-terminal passive nonvolatile resistance-switching devices, have so far shown abilities that could revolutionize HW computing architectures [12-16]. Their unique and adaptive properties have motivated the exploration of novel computing paradigms [17].

This paper addresses the classic “bin packing” problem by proposing a CA-inspired circuit-level approach, capable of executing computation within memory. We consider array-like circuits where the sparse nature of computations resembles certain operational features of CA. We describe a fundamental memristive cell which implements the desired one-dimensional CA rule and then employ it to create a sophisticated two-dimensional computational structure able to execute the “First-Fit” (decreasing) algorithm. Within the cells, information is encoded in the resistive states of threshold-type multi-state memristor-based composite components. A simulation-based verification of the proposed design is given. The main contribution of this work consists in the combination of a powerful computational tool with the unique circuit properties of

PCI '14, October 02 - 04 2014, Athens, Greece
Copyright 2014 ACM 978-1-4503-2897-5/14/10…$15.00.
http://dx.doi.org/10.1145/2645791.2645814

memristors within a CA-inspired implementation for NP-hard artificial intelligence (AI) problems.

2. BASIC BACKGROUND

2.1 Cellular Automata (CA)

CA are computational models of physical systems, where space and time are discrete and interactions are local. We present here a formal definition of CA [4]. In general, CA require:

1. A regular lattice of cells covering a portion of a *d*–dimensional space;
2. A set $\mathbf{C}(\vec{r},t)=\{C_1(\vec{r},t), C_2(\vec{r},t), \ldots, C_{\mathrm{m}}(\vec{r},t)\}$ of variables attached to each site $\vec{r}$ of the lattice, giving the local state of each cell at a specific time *t*;
3. A rule $\boldsymbol{R} = \{R_1, R_2, \ldots, R_m\}$ which specifies the time evolution of the states $\boldsymbol{C}(\vec{r},t)$ in the following way: $C_j(\vec{r},t+1)=R_j(\boldsymbol{C}(\vec{r},t),\ldots,\boldsymbol{C}(\vec{r}+\vec{\delta}_q,t))$, where $\vec{r}+\vec{\delta}_k$ designates the cells which belong to the given neighborhood of cell $\vec{r}$.

In the above definition, the rule ***R*** is identical for all sites and it is applied simultaneously to all of them, leading to synchronous dynamics. However, spatial/temporal inhomogeneities can be introduced. In the given definition, the new state of a particular cell at time *t*+1 is a function of the previous state of the specific cell and of the cells within its defined neighborhood. Here we focus on one–dimensional (1-d) elementary CA, where the typical neighborhood consists of the central cell (which is to be updated) and the two adjacent cells on both sides. The neighbourhood size *n* is usually taken to be *n*=2*r*+1, where *r* is a positive integer parameter known as the radius. CA-based systems allow for decentralized spatially extended complex computations with a high degree of efficiency and robustness.

2.2 The memristor (memory resistor)

Memristor, the 4th fundamental circuit element (joining the resistor, the capacitor, and the inductor) represents one of today's latest technological achievements. Memristor (here used to refer both to an ideal memristor as well as to a generalized memristive system [18]) is a passive two-terminal electronic device whose behavior is described by a nonlinear constitutive relation:

$$v = R(x,i)i \tag{1}$$

between the voltage drop at its terminals *v* and the current flowing through the device *i*, where:

$$\dot{x} = f(x,i). \tag{2}$$

$x = (x_1, x_2, \ldots, x_n)$ denotes *n* state-variables $x_1, x_2, \ldots x_n$, which do not depend on any external voltages or currents, whereas the nonlinear function *R*(*x*,*i*) defines the memory resistance (memristance) which varies between two limiting values, i.e. the less resistive (R_{ON}) and the high resistive (R_{OFF}) state. The reason why memristors are substantially different from the other fundamental circuit elements is that, when the applied voltage is turned off, they still remember how much voltage was applied before and for how long; thus presenting memory of their past. The nonvolatile memory property of memristors is a direct consequence of the state-dependent Ohm's law given in (1).

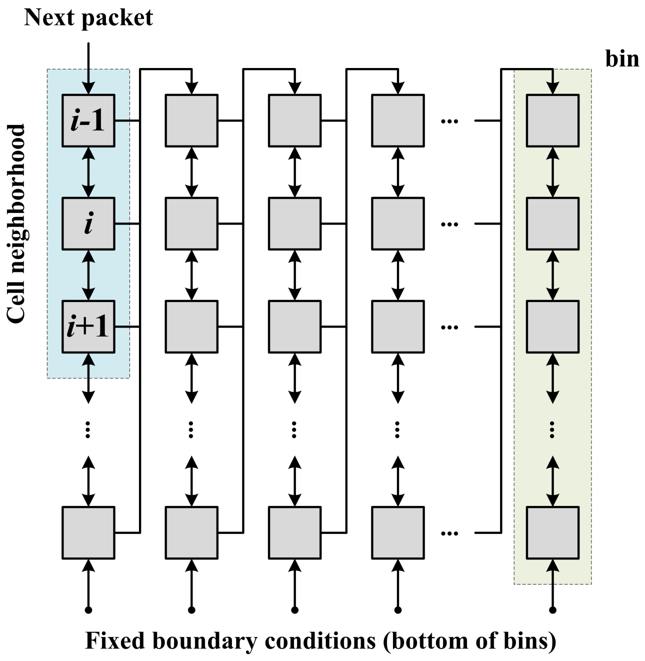


Figure 1. Block diagram describing the setup of the CA-inspired circuit approach to the bin-packing problem.

Threshold-type switching is closer to the actual behavior of most experimentally realizable memristive devices. Throughout this work, all conducted simulations employ a threshold-based device model of a voltage-controlled memristor which attributes its behavior to a tunneling distance modulation [19]. Such model is based on the assumption that the resistance switching rate of a memristor is small below (fast above) a voltage threshold (namely V_{SET} or V_{RESET}) which is viewed as the minimum voltage required to induce a change to the memristance of the device.

3. MEMRISTOR-BASED "BIN PACKING"

The general setup of the circuit which implements the "First Fit" algorithm for a set of packets (inputs) which are processed sequentially, either in arbitrary or decreasing order ("First Fit Decreasing"), is shown in Figure 1 (the sorting process is out of the scope of this work). Overall, this 2-d array consists of a "chain" of vertically-placed 1-d CA arrays representing separate equally-sized bins.

The CA neighborhood of the *i*th cell of each bin includes the top and bottom adjacent cells, as shown in Figure 1. Fixed boundary conditions are applied to the last cells of every 1-d CA to denote the bottom of the bins. All employed 1-d CA are identical and the total number of their cells (i.e. the rows of the 2-d array) equals *bin_size*+1, where *bin_size* is a positive integer. During the packing process, each new packet is introduced to the array from the top cell of the first bin. The top cells of the rest of the bins do not receive any input directly from outside but instead they are fed by the cells of the previous (lower-indexed) adjacent bin. In the following subsections the entire function of the 2-d array and the particular circuits which implement the update rule of the CA cells, are given in detail.

3.1 Algorithm Description

The overall function of the proposed system which implements the packing process is described in the flow chart of Figure 2. After initialization, for every recently introduced packet, all bins function as 1-d CA under a common evolution rule which consists of three phases: *set*, *reset*, and *read*. A new packet enters the array only when the previous packet is packed, i.e. it has reached a constant final position in one of the available bins. The entire system is globally controlled and stops functioning only when all packets are finally packed according to the "First Fit" algorithm.

The set of parameters which characterize the state of the i^{th} CA cell at time moment t is the following:

- *Packet_Size_on_Grid* ($PSoG_i^t$)
- *Space_Used* (SU_i^t)
- *Switch* (SW_i^t).

$PSoG_i^t$ is a positive integer whose values correspond to the size of the assigned packet, SU_i^t is also a positive integer which corresponds to the total utilized space inside the bin where the cell belongs, and SW_i^t is a particular flag which can be either '1' or '0'. After initialization all cells have $\{PSoG_i^t, SU_i^t, SW_i^t\}$ = '0'.

Every new packet sets the $PSoG_i^t$ parameter of the top cell of the leftmost bin to its particular size before the packing procedure is initiated. Then the packet moves sequentially downwards along each bin until it finally settles in an empty cell. Whenever the moving packet encounters another packet in the cell below it, it checks if there is enough space left in this bin by consulting the *SU* parameter of the next cell (SU_{i+1}^t). If it fits in the bin then it remains in the current position and updates its *SU* parameter to $SU_i^{t+1}=SU_{i+1}^t+PSoG_i^t$. However, if it does not fit in the bin, i.e. if it is $(SU_{i+1}^t+PSoG_i^t)>bin\_size$, then the packet continues the search in the next available bin. If the packet reaches the last cell of the bin without having encountered any other packets on his way, i.e. if it is the first packet inside the particular bin, is settles there and its *SU* parameter is updated as $SU_i^{t+1}=PSoG_i^t$. All CA cells located along the bottom boundary receive constant inputs $PSoG_{i+1}^t$!='0' and SU_{i+1}^t='0' from below; such combination corresponds to a "virtual" existing packet which however does not occupy any of the available space of the bin, i.e. the bin is still considered empty. The update rule for all 1-d CA cells is described in detail with the following pseudo-code:

Algorithm 1. 1-d CA update rule.

```
repeat
if ($PSoG_i^t$ =='0')                        //the cell is empty
    if ($PSoG_{i-1}^t$=='0')                  //previous cell is empty
        do nothing;
    else
        set $PSoG_i^{t+1}=PSoG_{i-1}^t$;
    end
else                                         //the cell has a packet
    if ($SW_i^t$=='1')                        //flag is up
        reset the cell {$PSoG_i^{t+1}$, $SU_i^{t+1}$, $SW_i^{t+1}$}='0';
        trigger the next bin;
    elsif ($PSoG_{i+1}^t$!='0')               //next cell has a packet
        if ($SU_i^t$=='0')                    //not in final position
            if ($SU_{i+1}^t+PSoG_i^t>bin\_size$)
                set $SW_i^{t+1}$='1';
            else
                $PSoG_i^{t+1}=PSoG_i^t$;
                $SU_i^{t+1}=SU_{i+1}^t+PSoG_i^t$;
            end
        else                                  //if in final position
            do nothing;
        end
    else                                      //next cell is empty
        reset the cell {$PSoG_i^{t+1}$, $SU_i^{t+1}$, $SW_i^{t+1}$}='0';
    end
end
until (packets_left_for_packing=='0');
```

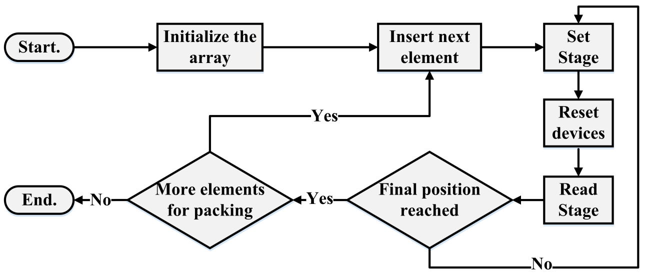


Figure 2. Flow chart describing the algorithm (work flow) during the bin-packing process.

3.2 Circuit-Level Implementation

As shown in Figure 2, the update rule for all 1-d CA cells consists of three discrete consecutive computational stages. The first stage is the *set* stage where the new state of the cell is computed. The second stage is the *reset* stage where the memristors inside the cell are reset (i.e. set to R_{OFF}). The final stage is the *read* stage where the recently calculated state of the cell is stored and the outputs of the cell are defined. For readability reasons we have separated the whole circuit schematic in three simpler schematics corresponding to each one of the aforementioned evolution stages.

3.2.1 Set Stage

The dedicated circuit that implements the *set* stage is shown in Figure 3. The circuit includes two composite memristive devices which are used in order to store the *Packet_Size_on_Grid* ($PSoG_i^t$) and the *Space_Used* (SU_i^t) parameters of the cell. Such memristive circuit components consist of three memristors connected in parallel which are considered to have different V_{SET} thresholds, namely $V_{SET,1} < V_{SET,2} < V_{SET,3}$, but equal memristance ranges within [R_{ON}, R_{OFF}]. So when a positive voltage is applied to the composite device, it operates as a multi-threshold memristive device [20]; assuming a high enough memristance ratio (R_{OFF}/R_{ON}), the equivalent memristance will approximately take either of the following values: $\{R_{OFF}/3, R_{ON}, R_{ON}/2, R_{ON}/3\}$ depending on the number of memristors that are set to R_{ON}.

More specifically, a positive voltage whose amplitude falls between $V_{SET,1}$ and $V_{SET,2}$ sets one of the memristors in R_{ON}. Similarly, if the voltage amplitude is between $V_{SET,2}$ and $V_{SET,3}$ it forces two of the memristors to switch their state, whereas a voltage higher than $V_{SET,3}$ causes all of them to switch to R_{ON}. $PSoG_i^t$ and SU_i^t parameters are encoded in the state of these composite devices. Therefore, for the particular circuit snapshot of Figure 3, the maximum $PSoG_i^t$ and SU_i^t value is three units since three memristors are used. However, this can be adjusted by modifying the number of parallel memristors and their voltage thresholds. Finally, a single memristor is used for the SW_i^t flag.

Moreover, three current-controlled DC voltage sources are used to store the next state of the cell, both for internal use as well as to define cell's output. A voltage adder and three switches are used to define the voltage that will be applied to each composite device, according to the current cell state and the states of the adjacent cells, as described in detail in Algorithm 1. A comparator is used to check if there is enough space inside the particular bin for the current packet. If there is enough space, then the SU_i^t parameter of the cell is updated; otherwise, the SW_i^t is set up and in the next time step the packet moves to the next bin.

3.2.2 *Reset Stage*

Similarly, the part of the circuit which is dedicated to re*set* the memristors is shown in Figure 4. Since gradual resetting is not important for our application, we assume a common negative threshold for all memristors, i.e. V_{RESET}. This stage consists in the "conditional" application of a single negative voltage pulse of appropriate amplitude above $|V_{\mathrm{RESET}}|$ to the multi-threshold devices in order to reset the state of all memristors. The application of such voltage is controlled by three switches which operate according to the CA update rule.

3.2.3 *Read Stage*

The last part of the proposed circuit implementation is about the last stage of the CA update rule, i.e. the *read* stage, and it is particularly presented in Figure 5. In this stage the state of the composite devices is read by applying a positive voltage V_{READ} of low enough amplitude so that it does not exceed any of the threshold values, thus it does not affect the state of the memristors. For each composite device we include a current-to-voltage converter (inverting amplifier) whose external gain is modified according to the composite state of the memristors; the output of the converter will be each time approximately given by $n \times V_{\mathrm{READ}}$, where n is the number of memristors that are in R_{ON}. Each cell state parameter is related to a corresponding DC voltage source and all of them are adjusted accordingly to hold the next cell state which is also its output. Flag SW_i^{t+1} is read using a voltage divider using a series resistor; its value defines whether or not the $PSoG_i^{t+1}$ value will be passed to the top of the next bin.

4. SIMULATION RESULTS

The effectiveness of all presented circuit designs was validated via simulations conducted using the Easy Java Simulations (EJS) environment [21]. All simulations employ a threshold-type model of a voltage-controlled memristor proposed in [19]; all differential equations of the model were numerically solved using a 4th order Runge-Kutta integration method, as implemented in [21]. All assumptions regarding both threshold and programming voltage values, as well as for the considered memristance of the employed devices, have been made only in the context of this study; thus, they do not relate to any real, manufactured or measured devices.

The parameters of the model were set to the following common values for all memristors: $\{a, b, c, m, f_0, L_{MAX}\} = \{5\times10^4, 10, 0.1, 310, 5\}$ and $\{r_{\mathrm{MIN}}, r_{\mathrm{MAX}}\} = \{100, 1000\}$, corresponding to $R_{\mathrm{OFF}} \approx$ 650KOhm and $R_{\mathrm{ON}} \approx$ 2KOhm, respectively. Such a high $R_{\mathrm{OFF}}/R_{\mathrm{ON}}$ ratio makes it easier to distinguish the different composite states of the multi-state components. The number of parallel memristors in the composite devices was selected in order to have a maximum packet size of three units and a maximum bin size of either three or four units. The voltage thresholds for the employed memristors forming the composite devices were chosen as $\{V_{\mathrm{SET,1}}$, $V_{\mathrm{SET,2}}$,

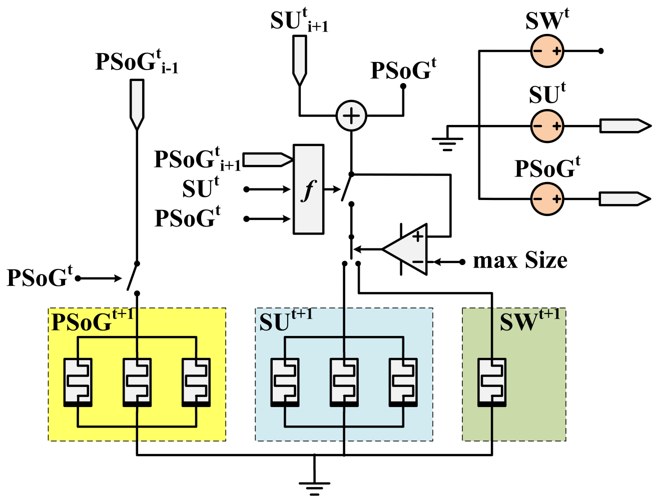


Figure 3. Schematic of the circuit corresponding to the *set* stage of the 1-d CA update rule of the bin-packing process.

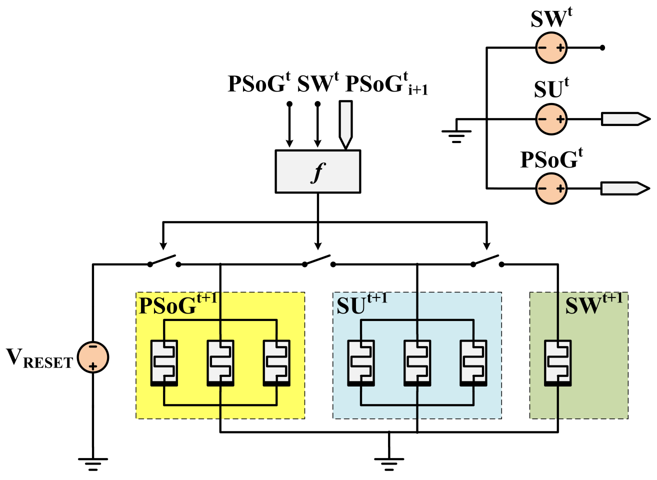


Figure 4. Schematic of the circuit corresponding to the re*set* stage of the 1-d CA update rule of the bin-packing process.

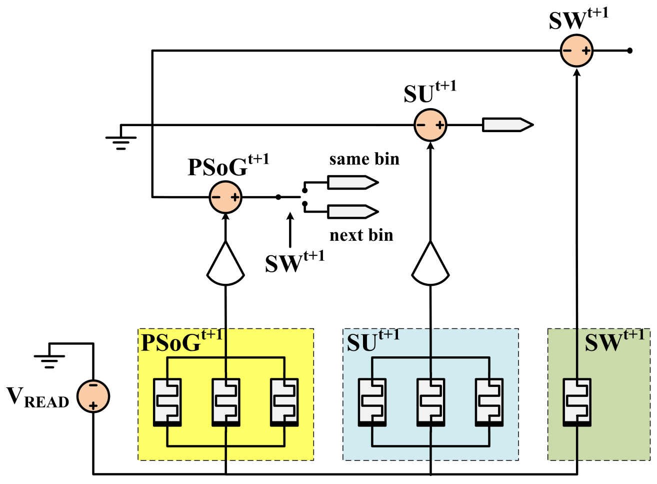
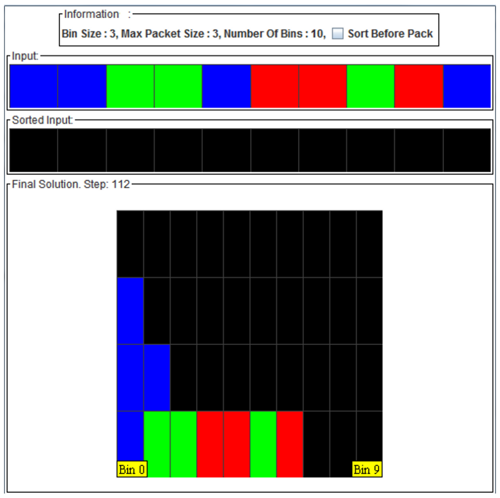
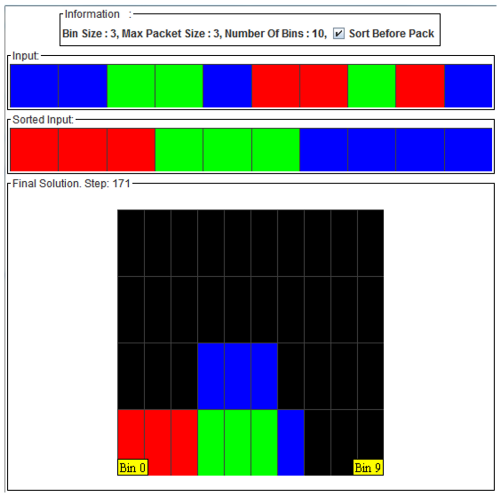


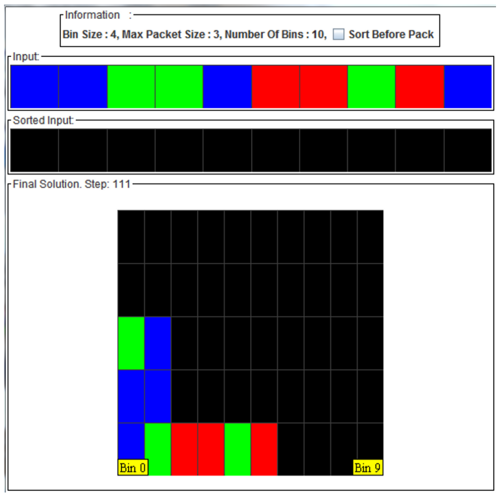
Figure 5. Schematic of the circuit corresponding to the *read* stage of the 1-d CA update rule of the bin-packing process.


Information :
Bin Size : 3, Max Packet Size : 3, Number Of Bins : 10, ☐ Sort Before Pack
Input
Sorted Input
Final Solution. Step: 112

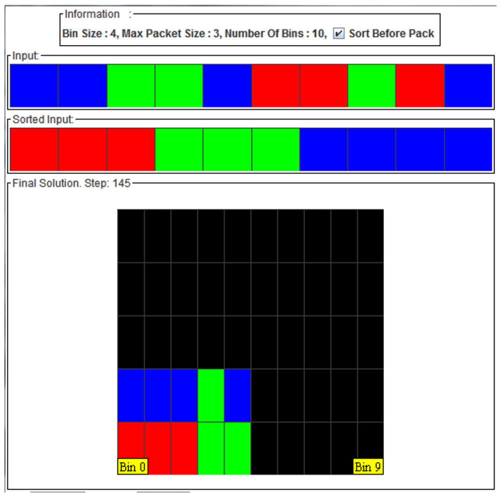
(a)


Information :
Bin Size : 3, Max Packet Size : 3, Number Of Bins : 10, [✓ Sort Before Pack]
Input
Sorted Input
Final Solution. Step: 171

(b)


Information :
Bin Size : 4, Max Packet Size : 3, Number Of Bins : 10, ☐ Sort Before Pack
Input
Sorted Input
Final Solution. Step: 111

(c)


Information :
Bin Size : 4, Max Packet Size : 3, Number Of Bins : 10, [✓ Sort Before Pack]
Input
Sorted Input
Final Solution. Step: 145

(d)

Figure 6. Simulation results after the bin-packing process. Two shelves of objects appear in the program display. The top shelf shows the candidate objects to be packed where the sorted entry presents the objects from largest to smallest. Colors {R, G, B} correspond to {3, 2, 1} packet sizes, respectively. The bottom shelf shows the results of packing in the 2-d grid (see Figure 1).

$V_{SET,3}$} = {0.5, 1.5, 2.5}V with a common reset threshold V_{RESET} = -3V. Depending on the state of the memristors, the applied read voltage produces four different voltage levels via the inverting amplifiers, which are approximately equal to{0, 1, 2, 3}V. The duration of the *set* and the *reset* stage was selected after experimentation with the memristor model and is enough for the memristors to completely switch their states.

Simulation results are shown in Figure 6. In the final state of the 2-d array (corresponding to that of Figure 1), each colored cell corresponds to a packet whose color indicates its size. Overall four simulations took place for a particular set of ten packets of various sizes and ten bins. The first two in Figure 6 a,b concern a set of ten bins of capacity equal to three units when the packets are processed either (a) in arbitrary or (b) descending order. In this example both results are nearly-optimal resulting in the same number of used bins. However, starting with the largest objects first leads to better utilization of the available space; six bins are completely filled compared to five when the packets are arbitrarily introduced to the packing system. We repeated the same simulation after having increased the bin capacity to four units. The corresponding results are given in Figure 6 c,d. Here the impact of the prior sorting process is evident; one less bin is finally occupied while the total space utilization is much better with 4 out of 5 used bins being completely filled compared to only 2 out of 6 when no sorting takes place before packing. For each simulation scenario we include the duration of computations in time steps (Figure 6 b,d include the steps of the prior sorting process [9]); increasing the capacity of the bins significantly lowers the necessary computational time when the candidate packets are sorted, whereas it has no significant effect for mixed entries.

5. CONCLUSIONS

This work presented a Cellular Automata (CA)-inspired circuit-level early approach to the solution of the classic “bin packing” problem (BPP) using memristors and memristive compositions. A fundamental memristive cell which implements a 1-d CA rule was designed and then employed in a two-dimensional computing structure. Simulation results based on a published device model proved the correctness of the presented CA-based circuit implementing the “First-Fit” packing algorithm.

Architectural improvements and modifications to the algorithm which will facilitate parallel processing of the candidate packets, leading to faster computations compared to published 1-d BPP benchmark data sets and classic CA designs, as well as real circuit simulations using environments with integrated-circuit-emphasis (SPICE), will be part of our future work. Also, cost-related issues concerning the implementation of multiple memristor devices with different thresholds in the same dice will be investigated.

6. ACKNOWLEDGMENTS

This work was supported in part by a scholarship from the BODOSSAKI Foundation in Greece.

7. REFERENCES

[1] Lewis, R. 2009. A General-Purpose Hill-Climbing Method for Order Independent Minimum Grouping Problems: A Case Study in Graph Colouring and Bin Packing. *Comput. Oper. Res.* 36, 7 (July 2009), 2295-2310. DOI= http://dx.doi.org/10.1016/j.cor.2008.09.004

[2] Coffman Jr., E. G., Csirik, J., Galambos, G., Martello, and S., Vigo, D. 2013. Bin Packing Approximation Algorithms: Survey and Classification. In *Handbook of Combinatorial Optimization*, P M. Pardalos, D.-Z. Du, and R. L. Graham, Ed. Springer, New York, NY, 455-531. DOI= http://dx.doi.org/10.1007/978-1-4419-7997-1_35

[3] Johnson, D. S. 1974. Fast Algorithms for Bin Packing. *J. Comput. Syst. Sci*. 8, 3 (June 1974), 272-314. DOI= http://dx.doi.org/10.1016/S0022-0000(74)80026-7

[4] Chopard, B. 2012. Cellular Automata Modeling of Physical Systems. In Computational Complexity, R. A. Meyers, Ed. Springer, New York, NY, 407-433. DOI= http://dx.doi.org/10.1007/978-1-4614-1800-9_27

[5] Adamatzky, A.I. 1996. Computation of shortest path in cellular automata. *Math. Comput. Modell*. 23, 4, (February 1996), 105–113. DOI= http://dx.doi.org/10.1016/0895-7177(96)00006-4

[6] Ioannidis, K., Sirakoulis, G. Ch., and Andreadis, I. 2011. A path planning method based on Cellular Automata for Cooperative Robots. *Appl. Artif. Intell*. 25, 8, (September 2011), 721–745. DOI= http://dx.doi.org/10.1080/08839514.2011.606767

[7] Golzari, S., and Meybodi, M.R. 2006. A Maze Routing Algorithm Based on Two Dimensional Cellular Automata. In *Proceedings of the 7th international conference on Cellular Automata for Research and Industry* (Perpignan, France, September 20-23, 2006). In *Lect. Notes Comput. Sc*. 4173, S. El Yacoubi, B. Chopard, S. Bandini, Ed. Springer Berlin Heidelberg, 564-570. DOI= http://dx.doi.org/10.1007/11861201_65

[8] Charalampous, K., Amanatiadis, A., and Gasteratos, A. 2012. Efficient Robot Path Planning in the Presence of Dynamically Expanding Obstacles. In *Proceedings of the 10th international conference on Cellular Automata for Research and Industry* (Santorini Island, Greece, September 24-27, 2012). In *Lect. Notes Comput. Sc*. 7495, G. Ch. Sirakoulis, S. Bandini, Ed. Springer Berlin Heidelberg, 330-339. DOI= http://dx.doi.org/10.1007/978-3-642-33350-7_34

[9] Gordillo, J. L., and Luna, J. V. 1994. Parallel sort on a linear array of cellular automata. In *Proccedings of 1994 IEEE International Conference on Systems, Man, and Cybernetics, Humans, Information and Technology* (San Antonio, TX, October 2-5, 1994). 1903 - 1907 vol.2. DOI= http://dx.doi.org/10.1109/ICSMC.1994.400129

[10] Vourkas, I., and Sirakoulis, G. Ch. 2012. FPGA based cellular automata for environmental modeling. In *Proceedings of 19th IEEE Int. Conf. Electronics, Circuits, and Systems* (Seville, Spain, December 9-12, 2012). *ICECS* '12. 93–96. DOI= http://dx.doi.org/10.1109/ICECS.2012.6463791

[11] Halbach, M., and Hoffmann, R. 2004. Implementing cellular automata in FPGA logic. In *Proceedings of 18th International Parallel and Distributed Processing Symposium*. (Santa Fe, New Mexico, April 26-30, 2004). IPDPS '04. DOI= http://dx.doi.org/10.1109/IPDPS.2004.1303324

[12] Strukov, D. B., Snider, G. S., Stewart, D. R., and Williams, R.S. 2008. The missing memristor found. *Nature* 453, (May 2008), 80–83. DOI= http://dx.doi.org/10.1038/nature06932

[13] Vourkas, I., and Sirakoulis, G. Ch. 2013. Recent progress and patents on computational structures and methods with memristive devices. *Recent Patents on Electrical & Electronic Engineering* 6, 2, (August 2013), 101–116. DOI= http://dx.doi.org/10.2174/22131116113069990004

[14] Yang, J. J., Strukov, D. B., and Stewart, D. R. 2013. Memristive devices for computing. *Nat. Nanotechnol*. 8, (January 2013), 13–24. DOI= http://dx.doi.org/10.1038/nnano.2012.240

[15] Zhao, W., Querlioz, D. Klein, J.-O. Chabi, D. and Chappert, C. 2012. Nanodevice-based Novel Computing Paradigms and the Neuromorphic Approach. In *Proceedings of 2012 IEEE International Symposium on Circuits and Systems* (Seoul, South Korea, May 20-23 2012). ISCAS '12. 2509-2512. DOI= http://dx.doi.org/10.1109/ISCAS.2012.6271812

[16] Ye, Z., Wu, S. H. M., and Prodromakis, T. 2013. Computing shortest paths in 2D and 3D memristive networks http://arxiv.org/abs/1303.3927

[17] Di Ventra, M., and Pershin, Y. V. 2013. The parallel approach. *Nat. Phys*. 9, (April 2013), 200-202. DOI= http://dx.doi.org/10.1038/nphys2566

[18] Chua, L. O., and Kang, S. M. 1976. Memristive devices and systems. *Proc. IEEE* 64, 2, (February 1976), 209-223. DOI= http://dx.doi.org/10.1109/PROC.1976.10092

[19] Vourkas, I., Batsos, A., and Sirakoulis, G. Ch. 2013. SPICE modeling of nonlinear memristive behavior. *Int. J. Circ. Theor. Appl*. DOI= http://dx.doi.org/10.1002/cta.1957

[20] Vourkas, I., and Sirakoulis, G. Ch. 2013. On the Analog Computational Characteristics of Memristive Networks. In *Proceedings of 20th IEEE International Conference on Electronics, Circuits, and Systems* (Abu Dhabi, UAE, December 8-11, 2013). ICECS '13. 309-312. DOI= http://dx.doi.org/10.1109/ICECS.2013.6815416

[21] Easy Java Simulations (EJS). (2014). [Online]. Available: http://fem.um.es/Ejs/