



An automatic routing approach for NML circuits

Pedro Arthur R. L. Silva¹ · Ruan E. Formigoni² · Laysson O. Luz¹ · Ricardo Ferreira² · Omar P. Vilela Neto¹ · José Augusto M. Nacif²

Received: 7 April 2021 / Accepted: 4 September 2022 / Published online: 7 November 2022
© The Author(s), under exclusive licence to Springer Science+Business Media, LLC, part of Springer Nature 2022

Abstract

Nanomagnetic Logic (NML) is an emergent computation model based on the interactions between nanomagnets. However, there are several challenges and open problems in the NML design flow, where clocking schemes and routing play an essential role. We compare the tradeoffs between NML synchronized and unsynchronized routing strategies using the A* search in this work. Both algorithms outperform previous work execution times by orders of magnitude, scaling for circuits with more than 1500 logic gates. Furthermore, we compare the synchronized circuits generated by our algorithm with ToPoliNano. For different full adders sizes, we outperform their results in terms of number of nanomagnets, absolute area, and clock zones by up to 113 ×, 88 ×, and 1.7 ×, respectively.

Keywords Nanocomputing · NML · Routing · EDA

1 Introduction

CMOS has been the standard technology for the manufacturing of digital devices over the years. However, this technology is close to its physical limits, while at the same time, reliability and power issues are rising at an alarming pace [12]. Several studies propose new technologies in recent years to overcome these problems and continue increasing integration densities, such as Field Coupled Nanotechnologies (FCN) [1, 4]. One among these is Nanomagnetic Logic (NML), whose circuits are composed of arrays of

nanomagnets placed on a plane interacting through the magnetostatic dipolar coupling. Also, NML is nonvolatile and operates with ultra-low energy dissipation [5, 15].

In NML circuits, the information from one or more input particles propagates in the circuit as antiferromagnetic and ferromagnetic coupling. Particle geometries and position define their coupling interaction. Therefore, it is possible to tailor the design of the particles in such a way that the shape anisotropy energy (which is magnetostatic) influences the final magnetization direction allowing only two stable states, which are associated with ‘0’ and ‘1’ binary logic states, enabling the implementation of Boolean logic operations.

Over the years, as the complexity of the CMOS integrated circuits increases, Electronic Design Automation (EDA) teams develop tools capable of providing higher abstraction levels. These tools can perform the design and verification of complex integrated circuits and electronic systems. Since CMOS has been the leading technology to build digital electronic devices, the literature presents a well-established set of EDA tools and techniques that address the problem of automatically generating a physical layout for CMOS circuits. Still, NML is not fully mature yet, so there is considerable room for research that aims to use this technology to overcome circuit designing challenges at the nanometric scale. Therefore, the development of tools and techniques that can assist the designer while abstracting low-level details of the NML technology is central. Due to the

✉ Pedro Arthur R. L. Silva
pedro.rodrigues@dcc.ufmg.br

Ruan E. Formigoni
ruan.formigoni@ufv.br

Laysson O. Luz
layssonluz@dcc.ufmg.br

Ricardo Ferreira
ricardo@ufv.br

Omar P. Vilela Neto
omar@dcc.ufmg.br

José Augusto M. Nacif
jnacif@ufv.br

¹ Universidade Federal de Minas Gerais, Belo Horizonte, MG, Brazil

² Universidade Federal de Viçosa, Viçosa, MG, Brazil

differences between CMOS and NML technologies and the latter’s several constraints, current tools and techniques are not entirely suitable to generate an efficient physical layout for the new technology.

Recent works presented the first NML circuit simulation tools, such as ToPoliNano [17] and NMLSim 2.0 [8, 9, 20]. These tools enable the increase in the abstraction levels and the development of EDA tools for NML. ToPoliNano also explores a series of optimizations to achieve an efficient design in terms of area and delay. Although this represents an important step toward the automatic design of NML circuits, researchers must further investigate and apply other tools from the EDA field to fully automate the designing process.

In this work, we focus on routing, one of the stages of the circuit design workflow. This step consists of finding routes to connect the logic elements previously placed in a given space while minimizing area and satisfying other design constraints. Despite previous works that propose algorithms for placement and routing of a predecessor FCN, Quantum Dots Cellular Automata (QCA) circuits [7, 22, 23], these do not fit perfectly with NML, primarily due to features such as wire crossing and information propagation.

We base our routing strategy in the A* search [10], a well-known algorithm applied in graph search. This work’s main contribution is proposing a new routing algorithm for NML circuits that scales more than the previous work by Silva et.al [19] that is also based on the A*. While Silva et.al. can handle small circuits (less than 20 gates), the adaptation proposed in this work scales to more than 1,500 gates. Moreover, we improve the execution time by orders of magnitude, using the path-finding algorithm’s concurrent resource instances. We also compared our strategy with ToPoliNano, which provides a complete EDA tool for NML. Our results show that the new proposed algorithm is able to outperform ToPoliNano both in time and circuit area.

We organize this work as follows: Sect. 2 reviews the basics of the NML technology and routing process. Section 3 presents an algorithm for automatic routing of NML circuits. In Sect. 4, we compare our approach to the previous works [17, 19], and we present novel circuit routing results for ISCAS’85 benchmarks [2]. Also, we analyze the performance of our algorithm and the trade-off between circuit area and execution time. Finally, Sect. 5 concludes this work.

2 Background

2.1 Nanomagnetic logic

This section presents a brief overview of Nanomagnetic Logic (NML), explaining the basic concepts and devices of the technology.

The NML primary device is a bistable elongated nanomagnet whose magnetization is likely to lie alongside its long axis (also known as easy axis) to minimize the shape energy. Figure 1a depicts the possible configurations for the magnetization of a nanomagnet. We arbitrarily define the logic values ‘0’ and ‘1’ to magnetization points ‘down’ and

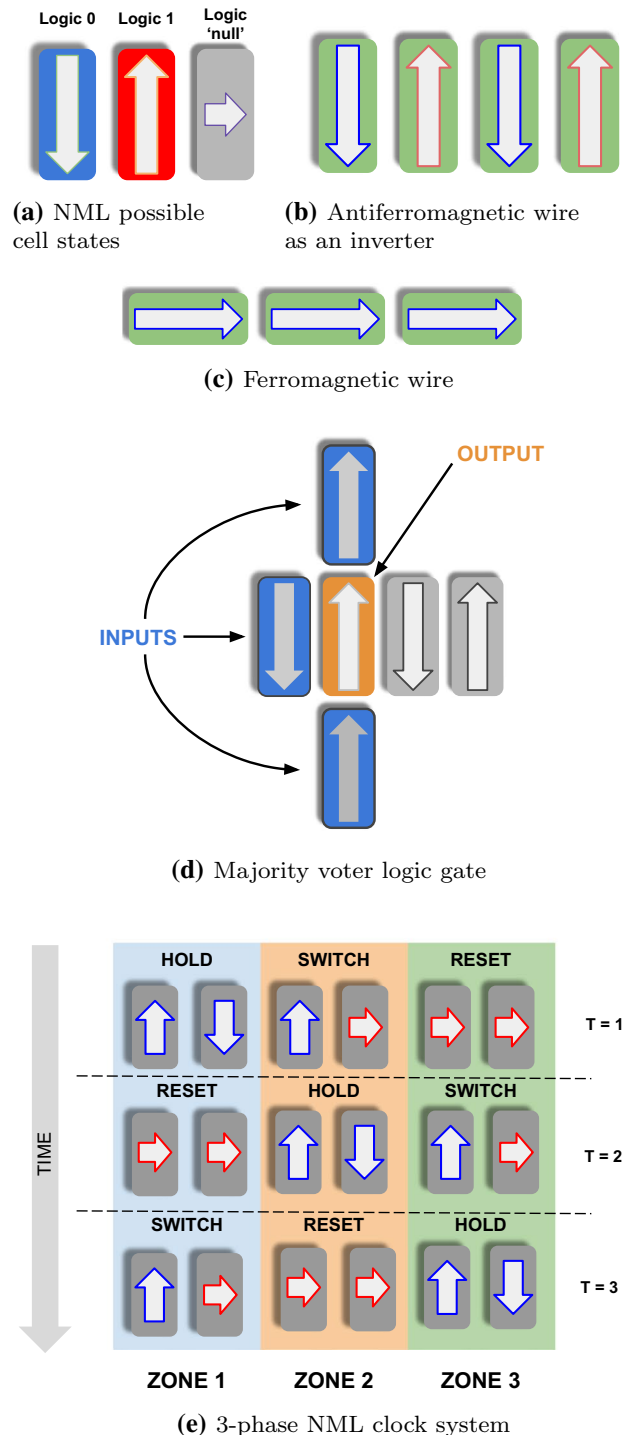


Fig. 1 NML basics

‘up’, respectively. In contrast, the nanomagnet changes to the ‘null’ logic state (metastable) while applying an external magnetic field to the short axis of the particle [8].

Wires are the essential element to propagate information through circuits. In NML, there are two configurations to arrange the nanomagnets to create wires: antiferromagnetic and ferromagnetic. The former structure presents the antiparallel direction of the magnetization vectors (Fig. 1b), while the latter exhibits a parallel orientation (Fig. 1c).

It is possible to exploit the wire structure in NML circuits to create an inverter. Antiferromagnetic wires with an odd number of cells propagate the logic values through the wire. On the other hand, a wire with an even number of nanomagnets inverts the original input logic level, as shown in the example of Fig. 1b.

Another essential element in NML circuits is the majority gate (MG), which replicates on the output the logic level that is most present in the inputs. Figure 1d depicts a 3-input MG where one of them presents an antiferromagnetic coupling with the output. Thus, when implementing a logic function, we should consider the complement of this input. We can reduce the MG to AND or OR gates by fixing one of the inputs to 0 or 1, respectively. Therefore, by relying on the wires, MG, and inverter, it is possible to implement any logic function [11].

To build more complex NML devices, one should select the placement of NML nanomagnets carefully and synchronize the information, avoiding a signal to reach a logic gate and propagate before the other inputs reach the gate. The utilization of an NML clock solves these issues, ensuring correct circuit operation. For simplicity, one can consider the clock as an external magnetic field that controls the particle’s magnetization. The clock is applied perpendicularly to the nanomagnet’s long axis to force it into a logic ‘null’ state, as shown in Fig. 1a. When these particular nanoparticles are horizontally oriented, they reach an unstable energy state. The nanomagnets get oriented vertically by removing the external magnetic field, assuming a minimum energy state corresponding to a ground state. Finally, the magnetization of a given nanomagnet points up or down, depending on its neighbors’ polarization.

The clock in NML solves three main issues: it allows the adiabatic change of the magnetization, generates direction and synchronization of information, and avoids signal errors in long arrays of nanomagnets due to non-nearest neighbor coupling and thermal noise. Also, with the increase in integration levels of NML circuits, the design becomes a complex process, being unfeasible to be performed manually.

As an example, Fig. 1e depicts a clocking system in NML composed of three clock zones. A periodic clock signal controls each zone with three phases: *Hold*, *Reset*, and *Switch*. In the *Hold* phase, the magnetization of the nanomagnets remains unchanged. In the *Reset* phase, the clocking system

applies an external magnetic field, inducing the nanomagnets into a *null* magnetization state. In the *Switch* phase, the clocking system gradually removes the external magnetic field, allowing the nanomagnets to polarize according to their neighbors. A four clock zones scheme is also allowed. In this case, as extra phase, called *Relax*, is included between phases *Reset* and *Switch*, keeping the nanomagnets in a null state.

When we split a circuit into clock zones, the magnetic fields act upon each zone independently, thus eliminating errors. A clock cycle in NML is the time a clock zone needs to pass through all the phases. A clock scheme is when one divides the nanomagnets in a circuit into very well organized, and distributed clock zones [7].

2.2 NML routing

Routing is the procedure of adding wires and creating the interconnections among circuit components. The synchronization is a substantial issue to build layouts that depicts the correct functionality. The synchronization of NML layouts can be achieved by using clocking schemes and balancing the circuit path lengths. A circuit is synchronized if we can ensure that all the paths leading to the gate inputs pass through the same number of clock zones, arriving at all gate inputs simultaneously. This phenomenon is known as the *layout-timing* problem, inherent to the FCN paradigm [13, 16].

At the graph level, in which the vertices are elements such as buffers, wires, and gates, while edges represent connections between these elements, we can apply some changes to simplify the *Routing* step. In the circuit graph, we name two paths as reconvergent if they diverge from and then reconverge to the same logic element or block [7, 17]. Therefore, we must guarantee that all the reconvergent paths leading to the same element have the same length, which increases the *Routing* complexity.

Figure 2 depicts an example of *Placement* and *Routing* at both graph and layout levels. In Fig. 2a, there are two unbalanced reconvergent paths starting in $buffer_1$ and $buffer_2$, and ending in And_1 and And_2 , respectively. We introduce wire nodes ($wire_1, wire_2$) to balance the graph as shown in Fig. 2b.

At the layout level, first, we need to place the graph inside a regular grid. Figure 2c, d presents two circuit layouts where we perform the *Routing* on the balanced graph from Fig. 2b. Consider a special clock zone for these circuits, while traversing the circuit, following the wires, from the inputs to the outputs, each tile is assigned a zone (or a group of equivalent clock zones) according to its distance from the primary inputs, x_1 and x_2 . So, tiles at the same distance are assigned to the same clock zone.

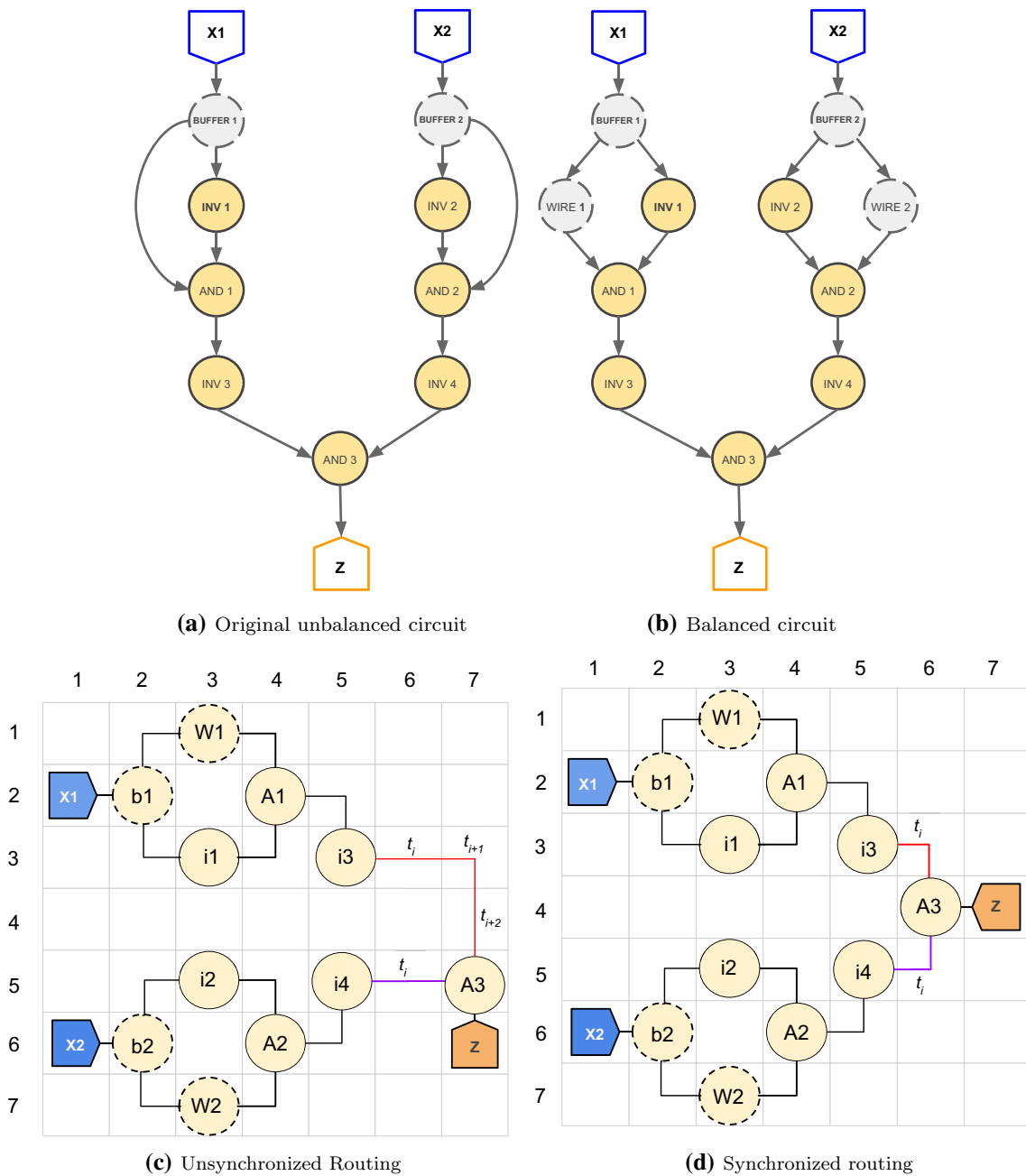


Fig. 2 Unsynchronized and synchronized routing example

However, a balanced graph does not guarantee a synchronized circuit. While the circuit in Fig. 2d is synchronized, Fig. 2c depicts that even if we deal with reconvergent paths at graph level, the *Routing* can generate an unsynchronized layout. In this case, the *Routing* path from I_3 to A_3 has a longer wire length in comparison to the path from I_4 to A_3 . A signal value from I_4 arrives at A_3 after the time t_i , but the signal from I_3 arrives after time t_{i+2} . On the other hand, this does not occur for the circuit in Fig. 2d, since both signals I_3 and I_4 arrive at A_3 at the same time.

The final area occupied by both layouts is the same, as one may notice the grids of Figs. 2c, d contains 7×7 tiles each. One tile represents the minimal layout unit, a logic element, or a wire with the same delay. All tiles have the same area, so we compute the total area of the layout by multiplying the number of tiles rows by the number of tiles columns, so the total area for the layouts depicted in Fig. 2c, d is 49 tiles each.

In summary, the definition of balanced or unbalanced relates to the graph level, and the term unsynchronized

or synchronized refers to the layout level. An unbalanced graph has not gone through *Pre-Routing* pre-processing steps, as we will discuss in Sect. 3. A balanced graph does not violate any of the restrictions imposed on the Pre-Routing. After the Placement and Routing, we get to the circuit's final layout, which can be either Unsynchronized or Synchronized. In a Synchronized layout, for each gate in the circuit, all of its inputs arrive at the same time, unlike the Unsynchronized one, which does not guarantee the simultaneous arrival of inputs at each gate.

It is essential to highlight that a balanced graph can have an unsynchronized layout. This situation occurs because an edge at graph level always has length 1. Its respective wire at layout level could cross several tiles, resulting in a length higher than one, as illustrated in Fig. 2c. The correct operation of unsynchronized circuits depends on modifying external clocking to guarantee that a signal arriving at a gate from a short path will wait for the signal arriving from a longer route, which results in lower throughput [18, 19].

3 Methodology

In this section, we present our *Routing* methodology for NML circuits. Sect. 3.1 details pre-processing, which includes placement, and Sect. 3.2 provides important information about our routing approach. Figure 3 shows a summary of the whole process.

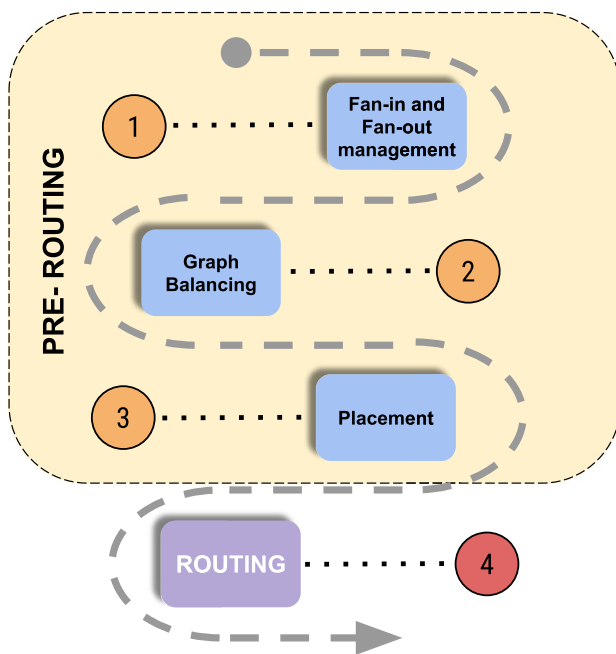


Fig. 3 NML placement and routing flow

3.1 Pre-routing

Our algorithm executes on a *Directed Acyclic Graph (DAG)*, manipulating the logic gates of the circuit, which we interchangeably refer to as a network. The DAG vertices can be gates, inputs, or outputs. The node can also represent buffers that the routing algorithm may create during the following steps to pre-process the circuit. The edge set E determines the connections between the logic elements. Thus, an edge from a vertex v to a vertex u means that v is an input of u .

The first step (1) is *Fan-in and Fan-out Management*, generating an equivalent network where all the gates have no more than two fan-ins and two fan-outs, this is similar to what has been done in Fontes et al. [7] and Riente et al. [17].

As detailed in Sect. 2.2, there is a relationship between circuit balancing and synchronization. Therefore, in the second step (2), we use a *balancing* algorithm to topologically order the network, providing a level-by-level view of the graph. Then, our algorithm travels the edges of the graph adding new edges if a given path is unbalanced, similarly to the works by Fontes et al. [7] and Riente et al. [17].

The final step (3) of pre-routing is the *Placement*, which assigns positions for each vertex of the graph in the circuit layout. The paths found at the *Routing* are intrinsically dependent on the *Placement* since our algorithm exploits the previous topological order to allocate children nodes as close as possible to their respective parent nodes, thus reducing the wire length. However, this step does not guarantee the synchronization of the circuit.

3.2 Routing

Our *Routing* algorithm applies the A* informed search algorithm [6] to build the connection between the vertices after the placement phase is complete. From here, we treat the grid returned by the placement as an undirected graph, where we define three types of adjacent positions related to position (i, j) : $(i, j + 1)$ and $(i, j - 1)$ are horizontally adjacent; $(i + 1, j)$ and $(i - 1, j)$ are vertically adjacent; $(i + 1, j + 1)$, $(i + 1, j - 1)$, $(i - 1, j + 1)$, and $(i - 1, j - 1)$ are diagonally adjacent. For the sake of understanding, we refer to the DAG representing the circuit as G_1 and the layout representing the placement grid as G_2 .

The *Placement* step creates the layout G_2 , where we execute our A* based approach to route the edges of G_1 . Consider an edge (V, U) from G_1 , both V and U were assigned positions (V_x, V_y) and (U_x, U_y) during the placement, respectively. Therefore, in G_2 we must find a route from (V_x, V_y) , the source, to (U_x, U_y) , the target. This is true for every edge of G_1 .

The A* algorithm builds the routes by relying on a heuristic function to estimate the total path cost from the source S , passing through a vertex N , to the destination D . We present the

evaluation function in Eq. 1, where $g(N)$ is the cost so far to move from $S \rightarrow N$ and $h(N)$ is the estimated cost from $N \rightarrow D$.

$$f(N) = g(N) + h(N), \tag{1}$$

Our algorithm splits the vertices into two sets: *open* and *closed*. The *open* set contains the vertices yet not explored, i.e., those available for expansion. Once we take the vertex out of the *open* and add it to the *closed* set, we also add its neighbors to the open set, and we cannot explore this node again. We first expand the vertices with lower $f(n)$ from the *open* set to guide the search. The algorithm stops when we fully expand the target, or the *open* set is empty. The latter case indicates that the target is not reachable from the source. Also, our A^* takes into account routing constraints such as the maximum number of wires passing through a position.

For the $H(n)$ estimated cost in the evaluation function, we assess four distance functions: *Manhattan*, *Euclidean*, *Chebyshev*, and *Octile distance*, described in Eqs. 2 to 5, respectively.

$$h(N) = \Delta x + \Delta y. \tag{2}$$

$$h(N) = \sqrt{\Delta x^2 + \Delta y^2}. \tag{3}$$

$$h(N) = \max(\Delta x, \Delta y). \tag{4}$$

$$h(N) = 1.414 * \min(\Delta x, \Delta y) + |\Delta x - \Delta y|. \tag{5}$$

Next, we present the unsynchronized and synchronized variations of our NML routing algorithm.

3.2.1 Unsynchronized routing

We present the unsynchronous routing in Algorithm 1. The *placement* creates a hash table \mathbf{P} to store the vertices positions, and Table \mathbf{E} stores the fan-ins and fan-outs of all vertices. The pre-processing phase creates a k -layered bipartite graph (*KLBG*) without intra-edge connections, then, instances of A^* execute between $lev(i)$ and $lev(i+1)$. The model receives a predefined number of threads T to use simultaneously. The number of layers of the *KLBG* splits into containers with sizes equal to T , and each one has indices to adjacent levels. Each thread inserts the result on a data-structure with a mutex-lock. Thus, each thread waits for an unlocked state, locks the mutex, stores the result, and unlocks it for the next thread.

Algorithm 1 Routing Algorithm

Input $\mathbf{P} \leftarrow$ Vertex \rightarrow Position Hash Table
Input $\mathbf{E} \leftarrow$ Edges Table
Input $\mathbf{H} \leftarrow$ An heuristic function for A^*
Output \leftarrow Vectors of Grid Paths

- 1: $\mathbf{V} \leftarrow$ Checks if a position is valid \propto Handles wire limits and regions with logic elements

- 2: $routes \leftarrow \{\}$
- 3: **for** $[v, u]$ **Of** \mathbf{E} **do**
- 4: $[src, dest] \leftarrow \mathbf{P}(v, u)$ \propto Get the positions of the placed vertices in the grid
- 5: $routes \leftarrow \mathbf{A_STAR}(src, dest, V, H)$ \propto Execute A^* and append a route to the routes array
- 6: **end for**

- 7: *return routes*

3.2.2 Synchronized routing

When we relax the synchronization constraint, it is not difficult to rely only on the A^* to find the connections. Nevertheless, an unsynchronized NML circuit has a lower throughput to guarantee valid outputs. We extended the A^* to generate a synchronized circuit, where all the route paths between adjacent levels have the same length. For synchronized layouts, the placement process already performs the first step generating a topological ordered graph. All vertices on the same topological level are on the same layout row. However, to generate a synchronized layout, we should organize the sequence of vertices (column ordering) in each row to mitigate the route paths, the wiring-cross between the levels. Also, we should guarantee path balancing. We propose a novel approximation algorithm for column assignment inside each layout row. Figure 4a shows an example. First, we partition the graph into a forest, as shown in Fig. 4b, where we use colors to highlight the sub-graph partitions. The algorithm traverses the graph in a depth-first manner. We also add a queue to keep track of columns during the traversal. We use the sub-graph partition and the topological order to create new columns. Figure 4c shows the final placement and routing.

In the worst case, the A^* search time complexity is exponential in the depth of the solution path, that is, $O(b^d)$, where b is the branching factor, and d is the depth of the solution [21]. The branching factor represents the number of children each node of the search may have, 8 in this case, one for each neighbor in the matrix. However, the algorithm’s performance strongly correlates with the heuristic choice. We are currently investigating the heuristics choice impact on the performance of the A^* in the context of circuit routing.

Regarding the algorithm to generate the physical layout for NMLSim 2.0, the time complexity is $O(V + E)$, where

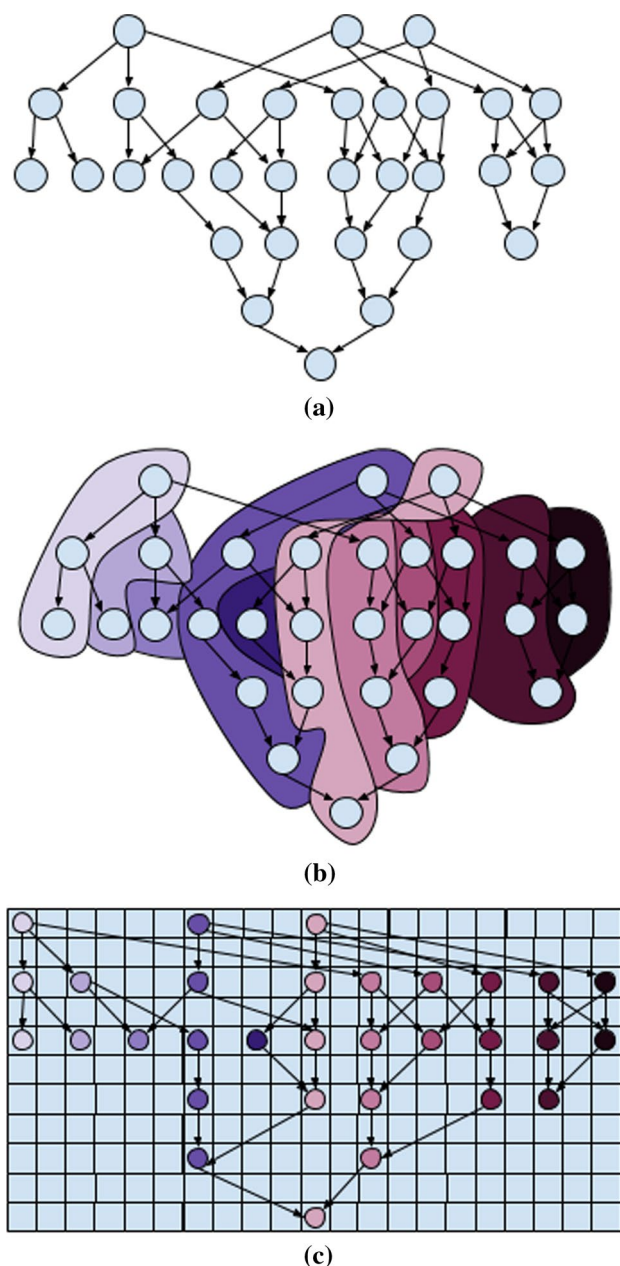


Fig. 4 **a** An example circuit graph. **b** The columns set by the algorithm. **c** The P&R approximation

V is the number of vertices and E is the number of edges of the graph returned by the routing algorithm.

4 Experiments and results

This section presents the analysis and comparison of our NML *Routing* algorithms. The focus of this work is to depict an algorithm capable of generate a valid placement and routing for a NML layout for any given circuit described in Verilog HDL. Although we do have an algorithm capable of

generate the circuit physical layout, it is out of the scope of this work to give more details about it. We choose to show only the physical layout depicted in Fig. 5a.

We have used the NMLSim 2.0 tool to simulate the circuits [8], and we set the number of clock zones to four. The nanomagnets geometries and the design rules are based on the work from Luz et al. [14]. To ensure that the generated layouts perform the desired logic functions, we have conducted extensive simulations in NMLSim 2.0. But unfortunately, we could only perform the simulation for all the possible combinations of inputs of small circuits since the simulation time is prohibitive for more complex circuits.

In Fig. 2c, d, the tiles that represent each clock zone (or a group of clock zones) all have the same size. However, the physical mapping uses tiles of different sizes to avoid errors and achieve a more robust layout. Therefore, for instance, two wires designated to the same clock zone (or group) may have a different number of nanomagnets.

When we remove the synchronization constraint, some paths on the circuit may take shorter routes. As depicted in Fig. 5a, we have two paths arriving at the same nanomagnet at different moments. The shorter one takes 1.5 clock cycles, while the longer one takes 2.5 clock cycles, yielding an undesired output. These unbalanced paths are not present in the circuit of Fig. 5b because the two paths have the same delay, 2.25 clock cycles.

4.1 Comparison with previous work

Here, we compare our results with previous works and discuss the advantages and drawbacks of our synchronized and unsynchronized algorithms in more complex benchmarks. We also used ToPolinano [17] to generate several circuits to compare. Although our focus rests mainly on the Routing phase and ToPolinano is a complete EDA tool, we see this as a relevant comparison since ToPolinano presents very solid results toward the automation of NML Routing.

We have implemented the algorithms in C++ and performed the experiments on an Intel Core i5-7200U with 2.50 GHz and 8 GB of RAM. Our implementation covers the four evaluation functions presented in Sect. 3. However, after preliminary experiments, we chose the *Octile* to generate the final results because this strategy outperforms the other evaluation functions in terms of execution time. We believe that the best performance of this algorithm is because it prioritizes horizontal and vertical paths over diagonal ones.

Table 1 compares our approach with the previous work by Silva et al. [19], and circuits from [2, 24]. In terms of area, for most benchmarks, our algorithm does not outperform the previous results. The difference is high for all circuits when using our unsynchronized algorithm. The main reason for this difference is that our new algorithm performs graph

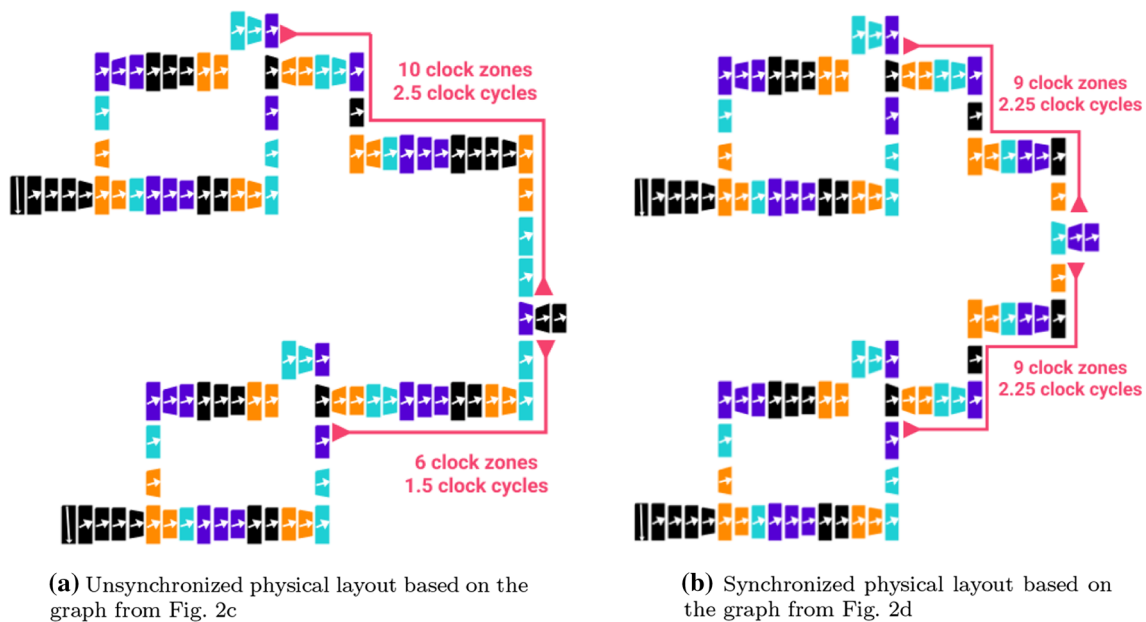


Fig. 5 Physical layout of **a** an unsynchronized, and **b** a synchronized layout

balancing, which increases the area, while the previous work takes another approach, ignoring the graph balancing.

The balancing considered now helps in the routing of larger circuits. When we consider the synchronized algorithm, the previous work outperforms the algorithm presented here for small circuits (*B1_r2*, *FA_AOIG*, *C17*, *t*, and *newtag*). Nevertheless, for the more complex circuits, especially with reconvergent paths, our algorithm presents very close or even better area results, as can be seen in the cases *xor5_r* and *xor5_r1*.

One of the main advantages of our algorithm is that it is much faster than the previous one. For the unsynchronized case we achieved speedups ranging from 65.21× (*newtag*) up to 9,600× (*b1*). For the synchronized case we achieved speedups ranging from 52,02× (*newtag*) up to 8,478× (*b1*). These significant speedups occur because our current approach simplifies the search and exploits parallelism during the A* path routing. Furthermore, our approach scales efficiently with the number of logic gates, while the previous work [19] was limited to a few logic gates (< 15, see Table 1). With this in mind, we can now apply our routing algorithm in larger circuits, as we describe later.

In Table 2, we present a comparison with ToPolinano using the same circuits of Table 1. In this case, we compare just synchronized circuits since ToPolinano only works on this kind of circuit. Besides the speedup, Table 1 also presents the bounding box area, which is the area of the minimum rectangle that bounds the layout. To measure it, we consider tiles with size 4x4, the default value for ToPolinano. The column **Runtime** for ToPolinano considers

more than the Routing, since the tool does not provide a way to measure the time spent only on the Routing.

One may notice that the speed up reached ranges from 1.52× to 50× in comparison with ToPolinano. Even when considering that the ToPolinano runtime encompasses the whole process of a CAD tool, this is a significant improvement and depicts that our algorithm is very suitable for the Routing of NML circuits. Regarding the area of the circuits, we consider the bounding area in terms of tiles. Thus, we can realize this comparison in a fairway. Our approach reaches a much smaller area than ToPolinano, achieving an improvement of up to 18.80 times.

We also performed a comparison with the adders presented in the ToPolinano original paper [17]. In this work, the authors depict results for multiples ripple-carry adders varying the number of bits. Table 3 shows the results of this comparison, including the absolute area, which is the area occupied by all the nanomagnets, i.e., the sum of the areas of each magnet. Our approach needs much fewer nanomagnets than ToPolinano for all the benchmarks, especially for larger circuits, reaching 113× fewer nanomagnets for 64-bit full adder. Consequently, our approach always presents a smaller circuit area. It's worth mentioning that to calculate the area for our approach, we consider that all magnets have dimensions 150 nm × 50 nm, following the details of the cell library designed by Luz et al. [14].

The increase in the number of nanomagnets and clock zones is higher for ToPolinano than our approach. Our bounding area has shown worse results for only two cases, the 2 and 4-bit adders. However, for larger circuits, our

Table 1 Execution time and grid area for our approach in comparison to previous work [19]

Circuits	Our approach										
	Silva et al. [19]					Our approach					
	Gates		Unsynchronized		Synchronized		Unsynchronized		Synchronized		
Name	Runtime (s)	Bounding area	Runtime (s)	Bounding area	Runtime (s)	Bounding area	Runtime (s)	Bounding area	Speedup	Bounding area	
b1 [24]	4	9.66	16	0.16	50	1.00×10^{-4}	100	1.14×10^{-3}	$96,600 \times$	$8,473 \times$	120
1-bit Full-Adder [19]	5	0.10	24	9.44	56	5.52×10^{-4}	96	7.54×10^{-4}	181.15X	132.62X	128
C17 [2]	6	0.73	32	0.13	40	9.21×10^{-4}	96	1.33×10^{-3}	792.61X	548.87X	120
t [19]	6	0.09	32	0.15	40	1.06×10^{-3}	80	1.32×10^{-3}	65.21X	68.18X	100
newtag [19]	8	0.09	70	0.14	70	1.38×10^{-3}	168	1.73×10^{-3}	65.21X	52.02X	168
xor5_r [19]	14	0.18	88	18.68	216	1.23×10^{-3}	224	3.41×10^{-3}	146.34X	52.78X	224
xor5_r1 [19]	12	0.2	88	0.27	216	1.93×10^{-3}	180	2.62×10^{-3}	103.62X	76.33X	180

bounding area gets similar or better than ToPoliNano. For example, the bounding area for the 64-bit full adder uses $3.77 \times$ less area in our approach compared to ToPoliNano. Regarding the absolute area, our approach has shown improvement for most cases. We have improvements of several orders of magnitude, reaching $88 \times$ smaller area for the 64-bit full adder. In terms of clock zones, our approach presents better results for the larger circuits (32 and 64-bit full adders). In ToPoliNano, when we double the full adder size, the number of nanomagnets is multiplied by approximately 4 times, and the number of clock zones is multiplied by about 3 times. In our approach, this multiplying factor is around 2 for both the number of nanomagnets and the number of zones.

4.2 Routing larger circuits

Table 4 presents results for a set of circuits from different benchmarks [2, 3, 24], where the number of logic gates ranges from 22 up to 1669. Therefore, we increase the circuit size up to two orders of magnitude. The first five columns present the circuit name, function, number of logic gates, number of inputs, and outputs, respectively. The sixth and seventh columns show the run time and grid size (area) for the unsynchronized routing. The next two columns present the same information for the synchronized routing. Finally, the last two columns show the speedup and area overhead between the two cases.

As we can observe, the unsynchronized algorithm is faster since balanced paths are not required. It presents a speedup average equal to $3.06 \times$ when compared with the synchronized one. Also, to synchronize the paths, the algorithm needs more cells to route the wires, occupying a larger area. It is important to highlight that we limited the maximum number of wires per cell to two in our approaches. It is unfeasible to handle more than this in NML due to magnetostatic interactions between the nanomagnets.

5 Conclusion

In this work, we presented synchronized and unsynchronized routing algorithms using the A^* search, with custom heuristics, to find paths to connect the gates in an NML circuit. Our algorithms scaled better than the previous work proposed by Silva et al. [19], being able to handle circuits with more than 1500 gates and reducing the execution times by orders of magnitude. Furthermore, our synchronized algorithm outperformed the state-of-the-art ToPoliNano tool [17]. For larger circuits, our strategy presented area reduction of up to $113 \times$.

Table 2 Execution time and bounding area for our approach in comparison to ToPoliNano [17]

Circuits		ToPoliNano		Our approach		Comparison	
Name	Gates	Runtime (s)	Bounding area	Runtime (s)	Bounding area	Speedup time	Area
b1 [24]	4	6×10^{-3}	1260	1.14×10^{-3}	120	5.26x	10.5x
1-bit Full-Adder [19]	5	4×10^{-3}	900	7.54×10^{-4}	128	5.30x	7.03x
C17 [2]	6	5×10^{-3}	572	0.1×10^{-3}	120	50x	4.77x
t [19]	6	5×10^{-3}	572	1.32×10^{-3}	100	3.78x	5.72x
newtag [19]	8	5×10^{-3}	1672	1.73×10^{-3}	168	2.89x	9.95x
xor5_r [19]	14	7×10^{-3}	4212	3.41×10^{-3}	224	2.05x	18.80x
xor5_r1 [19]	12	4×10^{-3}	2508	2.62×10^{-3}	180	1.52x	13.93x

Table 3 Adders generated by our approach in comparison to ToPoliNano [17]

Bits	Nanomagnets		Clock zones		Bound area (mm ²)		Absolute area (mm ²)	
	ToPoliNano	Our approach	ToPoliNano	Our approach	ToPoliNano	Our approach	ToPoliNano	Our approach
2	1572	1151	52	181	3.84×10^{-5}	8.1×10^{-5}	9.12×10^{-6}	8.63×10^{-6}
4	6,021	2,307	129	365	1.59×10^{-5}	3.16×10^{-4}	3.49×10^{-5}	1.73×10^{-5}
8	28,513	4619	409	733	1.12×10^{-3}	1.15×10^{-3}	1.65×10^{-4}	3.46×10^{-5}
16	116,827	9243	932	1469	4.56×10^{-3}	4.37×10^{-3}	6.78×10^{-4}	6.93×10^{-5}
32	676,969	18,491	3022	2941	2.97×10^{-2}	1.70×10^{-2}	3.93×10^{-3}	1.39×10^{-4}
64	4,207,667	36,987	10,009	5885	2.54×10^{-1}	6.73×10^{-2}	2.44×10^{-2}	2.77×10^{-4}

Table 4 Our synchronized and unsynchronized approaches for larger circuits: execution time and grid area

Circuits			Unsynchronized				Synchronized		Speedup	Overhead
Name	Function	Gates	In	Out	Time (s)	Bounding area	Time (s)	Bounding area	Time	Area
Decod [24]	Decoder	22	5	16	0.0243	920	0.0247	1080	1.01x	1.14x
Mux [24]	Multiplexer	91	21	1	0.08	5568	0.12	5568	1.5x	–
s208 [3]	Frac. Multiplier	96	11	2	0.05	3920	0.07	4000	1.4x	1.02x
s349 [3]	4-Bit Multiplier	161	4	3	0.36	10,720	0.53	2960	1.39x	1.10x
adderfds [24]	Adder	223	33	17	4.13	35,964	6.99	40,500	1.62x	1.11x
alu2 [24]	ALU	335	10	6	44.06	89,832	57.78	89,832	1.31x	–
s1494 [3]	Generic controller	647	9	19	119.80	83,824	144.66	83,824	1.21x	–
C3540 [2]	ALU and Control	1669	50	22	338.5	275,840	620.4	279,040	1.83x	1.01x

As future work, we aim to extend our approach to encompass the Routing with the presence of a clocking scheme and minimize the wire crossing. We also intend to improve our pre-routing stage, especially the Placement phase and develop a highly efficient physical mapping.

Acknowledgements This study was financed in part by the Coordenação de Aperfeiçoamento de Pessoal de Nível Superior—Brasil (CAPES)—Finance Code 001, Fundação de Amparo à Pesquisa do Estado de Minas Gerais—FAPEMIG, and Conselho Nacional de Desenvolvimento Científico e Tecnológico—CNPq.

Funding This study was financed in part by the Coordenação de Aperfeiçoamento de Pessoal de Nível Superior—Brasil (CAPES)—Finance Code 001, Fundação de Amparo à Pesquisa do Estado de Minas Gerais—FAPEMIG, and Conselho Nacional de Desenvolvimento Científico e Tecnológico—CNPq.

Availability of data and material Not applicable.

Declarations

Conflict of interest The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Code availability Not applicable.

References

- Anderson, N.G., Bhanja, S.: *Field-Coupled Nanocomputing*, 1st edn. Springer, Berlin, Heidelberg (2014)
- Brglez, F.: A Neutral Netlist of 10 Combinational Benchmark Circuits and a Target Translator in FORTRAN (1985)
- Brglez, F., Bryan, D., Kozminski, K.: Combinational Profiles of Sequential Benchmark Circuits. *IEEE Int. Sympos. Circuits Syst.* **3**, 1929–1934 (1989)
- Cavin, R., Lugli, P., Zhirnov, V.V.: Science and engineering beyond Moore's law. *Proc. IEEE* **100**, 1720–1749 (2012)
- Cowburn, W.: Room temperature magnetic quantum cellular automata. *Science* **287**(5457), 1466–8 (2000)
- Dechter, R., Pearl, J.: Generalized best-first search strategies and the optimality of A*. *J. ACM* **32**(3), 505–536 (1985). <https://doi.org/10.1145/3828.3830>
- Fontes, G., Silva, P.A.R.L., Nacif, J.A.M., Vilela Neto, O.P., Ferreira, R.: Placement and routing by overlapping and merging QCA Gates. In: 2018 IEEE International Symposium on Circuits and Systems (ISCAS), pp. 1–5 (2018)
- Freitas, L., Neto, O.P.V., Rahmeier, J.G.N., Melo, L.G.C.: Nmlsim 2.0: A robust cad and simulation tool for in-plane nanomagnetic logic based on the Ilg equation. In: 2019 32nd Symposium on Integrated Circuits and Systems Design (SBCCI), pp. 1–6 (2019)
- Freitas, L.A.L., Rahmeier, J.G.N., Vilela Neto, O.P.: Shape engineering for custom nanomagnetic logic circuits in nmlsim 2.0. *IEEE Des. Test* (2020)
- Hart, P.E., Nilsson, N.J., Raphael, B.: A formal basis for the Heuristic determination of minimum cost paths. *IEEE Trans. Syst. Sci. Cybern.* **4**(2), 100–107 (1968)
- Imre, A., Csaba, G., Ji, L., Orlov, A., Bernstein, G., Prood, W.: Majority logic gate for magnetic quantum-dot cellular automata. *Science* **311**, 205–208 (2006)
- Liu, T., Kuhn, K.: Cmos and beyond : logic switches for terascale integrated circuits (2015)
- Liu, W., Lu, L., O'Neill, M., Swartzlander, E.E.: Design rules for quantum-dot cellular automata. In: 2011 IEEE International Symposium of Circuits and Systems (ISCAS), pp. 2361–2364. IEEE (2011)
- Luz, L.O., Nacif, J., Ferreira, R., Neto, O.P.V.: Nmlib: A nanomagnetic logic standard cell library. In: 2021 IEEE International Symposium on Circuits and Systems (ISCAS), pp. 1–5 (2021)
- Melo, L.G., Soares, T.R., Vilela Neto, O.P.: Analysis of the magnetostatic energy of chains of single-domain nanomagnets for logic gates. *IEEE Trans. Magn.* **53**(9), 1–10 (2017)
- Niemier, M., Kogge, P.: Problems in designing with qcas: layout = timing. *Int. J. Circuit Theory Appl.* **29**, 49–62 (2001)
- Riente, F., Turvani, G., Vacca, M., Roch, M.R., Zamboni, M., Graziano, M.: ToPoliNano: a CAD tool for nano magnetic logic. *IEEE Trans. Comput. Aided Des. Integr. Circuits Syst.* **36**(7), 1061–1074 (2017)
- Sill Torres, F., Silva, P.A., Fontes, G., Nacif, J.A., Santos Ferreira, R., Vilela Neto, O.P., Chaves, J., Drechsler, R.: Exploration of the synchronization constraint in quantum-dot cellular automata. In: 2018 21st Euromicro Conference on Digital System Design (DSD), pp. 642–648 (2018)
- Silva, P.A.R.L., Neto, O.P.V., Nacif, J.A.M.: Toward nanometric scale integration: an automatic routing approach for NML Circuits. In: 2019 32nd Symposium on integrated circuits and systems design (SBCCI), pp. 1–6 (2019)
- Soares, T.R., Rahmeier, J.G.N., De Lima, V.C., Lascasas, L., Melo, L.G.C., Vilela Neto, O.P.: Nmlsim: a nanomagnetic logic (nml) circuit designer and simulation tool. *J. Comput. Electron.* **17**(3), 1370–1381 (2018)
- Stuart, R., Peter, N.: *Artificial intelligence—a modern approach*. Berkeley, New York (2016)
- Trindade, A., Ferreira, R., Nacif, J.A.M., Sales, D., Neto, O.P.V.: A placement and routing algorithm for quantum-dot cellular automata. In: 2016 29th Symposium on Integrated Circuits and Systems Design (SBCCI), pp. 1–6 (2016). <https://doi.org/10.1109/SBCCI.2016.7724048>
- Walter, M., Wille, R., Große, D., Sill, F., Drechsler, R.: An exact method for design exploration of quantum-dot cellular automata. In: 2018 Design, Automation and Test in Europe Conference and Exhibition (DATE), pp. 503–508 (2018)
- Yang, S.: *Logic synthesis and optimization benchmarks user guide version 3.0* (1991)

Publisher's Note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.

Springer Nature or its licensor (e.g. a society or other partner) holds exclusive rights to this article under a publishing agreement with the author(s) or other rightsholder(s); author self-archiving of the accepted manuscript version of this article is solely governed by the terms of such publishing agreement and applicable law.