Two Combinatorial Problems on the Layout of Switching Lattices

Anna Bernasconi Antonio Boffa Fabrizio Luccio Linda Pagli
Dipartimento di Informatica, Università di Pisa, Italy
{anna.bernasconi, antonio.boffa, fabrizio.luccio, linda.pagli}@unipi.it

***Abstract*—A non classical approach to the logic synthesis of Boolean functions based on switching lattices is considered, for which deriving a feasible layout has not been previously studied. The problem presents new interesting combinatorial and algorithmic aspects. Our basic assumptions are that the positions of the switches in the lattice are fixed in the synthesis stage, and the layout for connecting the subsets of switches with the same input literal must be realized in superimposed planes through vias that take the same switch area. The overall goal is to minimize the number of layers needed. Since multiple choices of input literals are possible for each switch, we first study how to assign a single literal to each switch, to minimize the number of lattice portions of adjacent cells associated to the same literal (Problem 1). Then we study how to derive a feasible layout by building connections onto different layers, to minimize the number of layers (Problem 2). Problem 1 is NP-hard. Problem 2 seems to be also intractable, and exhibits limit instances that require an exceedingly number of layers or are even unsolvable. Heuristic algorithms are then developed for both problems and their encouraging performances are proved on a set of known benchmarks.**

***Index Terms*—Circuit Layout, Switching Lattices, Logic Synthesis, NP-Complete Problems**

I. Introduction

The logic synthesis of a Boolean function is the procedure that implements the function into an electronic circuit. The literature on this subject is extremely vast and large part of it is devoted to *two-level* logic synthesis, where the function is implemented in a circuit of maximal depth 2 [9]. In this paper, we focus on a different synthesis method, based on *switching lattices*. A switching lattice is a two-dimensional array of four-terminal switches implemented in its cells. Each switch is linked to the four neighbors and is connected with them when the switch is ON, or is disconnected when the switch is OFF. The idea of using regular two-dimensional arrays of switches to implement Boolean functions dates back to a seminal paper by Akers in 1972 [1]. Recently, with the advent of a variety of emerging nanoscale technologies based on regular arrays of switches, synthesis methods targeting lattices of multi-terminal switches have found a renewed interest [2], [3], [6].

A Boolean function can be implemented in a lattice with the following rules:

- each switch is controlled by a Boolean literal;
- if a literal takes the value 1 all corresponding switches are connected to their four neighbors, else they are not connected;

978-1-5386-4756-1/18/$31.00

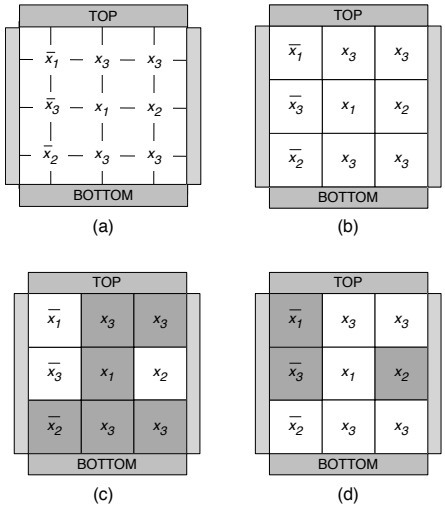

Fig. 1: A four terminal switching network implementing the function $f = \overline{x}_1\overline{x}_2\overline{x}_3 + x_1x_3 + x_2x_3$ (a); the corresponding lattice (b); the lattice evaluated on the assignments 1,0,1 (c) and 0, 1, 0 (d), with grey and white squares representing ON and OFF switches, respectively.

- the function evaluates to 1 for any input assignment that produces a connected path between two opposing edges of the lattice, e.g., the top and the bottom edges; the function evaluates to 0 for any input assignment that does not produce such a path.

For instance, the 3×3 network of switches in Fig. 1 (a) corresponds to the lattice form depicted in Fig. 1 (b), which implements the function $f = \overline{x}_1\overline{x}_2\overline{x}_3 + x_1x_3 + x_2x_3$. If we assign the values 1, 0, 1 to the variables x_1, x_2, x_3, respectively, we obtain paths of gray square connecting the top and the bottom edges of the lattices (Fig. 1 (c)), and f evaluates to 1. On the contrary, the assignment $x_1 = 0, x_2 = 1, x_3 = 0$, on which f evaluates to 0, does not produce any path from the top to the bottom edge (Fig. 1 (d)).

The synthesis problem on a lattice consists of finding an assignment of literals to switches implementing a given target function with a lattice of minimal size, measured as the number of switches in the lattice. In [2], [3], Altun and Riedel developed a synthesis method for switching lattices that assigns *at least* one literal to each lattice position, with the literal controlling the corresponding switch. If several literals are assigned to a switch the choice of the controlling literal is arbitrary.

Starting from the lattice obtained by the Altun-Riedel method we consider two problems related to the physical implementation of the circuit, both motivated by the following considerations and assumptions.

1) Equal literals must be connected together, and to an external terminal on one side (e.g. the top edge) of the lattice. This may require using different layers, and vias to connect cells of adjacent layers.
2) Connections can be laid out horizontally or vertically (but not diagonally) between adjacent cells.
3) Each cell can be occupied by a switch, or by a portion of a connecting wire, or by a via. No two such elements can share a cell on the same layer.
4) The overall target is designing a layout with the minimum number of layers. Since the problem is hard, it will be relaxed to finding a reasonable layout by heuristic techniques.

As a consequence the circuit will be generally built starting from the original $N \times M$ lattice and superimposing to it a certain number H of layers, to give rise to a multidimensional grid of size $N \times M \times H$. Note that the switches associated with the same literal cannot be generally connected all together on the same layer, so several subsets of these switches will be connected on different layers and then connected through vias. The degree of freedom arising from the multiple choices of literals is exploited to enlarge these subsets (Problem 1). Then the connection of the different subsets with the same literal by themselves and by the external lead is addressed (Problem 2).

As said, we first consider the problem of assigning one literal to each switch in case of different choices at the switch. Consider the $N \times M$ lattice as a non-directed graph $G = (V, E)$ whose vertices correspond to the switches (then $|V| = NM$) and whose edges correspond to the horizontal and vertical connections between adjacent switches (then $E = 2NM - N - M$). We shall refer indifferently to the lattice or to the graph; to switches or to vertices; and to connections or to edges. Occasionally a vertex will be indicated with v_i, with $1 \leq i \leq N \cdot M$, or with a pair of integers (h, k) denoting the row and the column of the lattice where the vertex lays, with $1 \leq h \leq N$ and $1 \leq k \leq M$. Obviously the vertices have degree 2 or 3 if they lay on the corners or on the borders of the of the lattice, and have degree 4 if they are internal to the lattice. Finally let L be the set of literals occurring in the Boolean function. Each vertex v_i is associated with a non-void subset L_i of L, from which one literal has to be eventually assigned to v_i. Once a single literal assignment has been done for each vertex, an *area* denotes a *maximal connected subgraph* of G (or connected portion of the lattice) where all vertices have the same literal assigned. Note that if two areas A_1, A_2 have the same literal they must be disjoint and no two vertices $a_1 \in A_1, a_2 \in A_2$ may be adjacent in G. We pose:

Problem 1. *Find a literal assignment that minimizes the number of areas.*

Any literal assignment solving Problem 1, and the corresponding family of areas, is called an *MPA* for *minimal partition assignment*. The problem has been studied in [8] for general graphs and for some of its variations, showing that is NP-hard on a lattice. Hence we will study how to solve it heuristically.

After Problem 1 is solved, we have to choose how to connect the different areas associated with the same literal and then connect them to the external input leads. To this end different layers are needed to attain all non-crossing connections. Formally we pose the following problem, whose complexity is discussed in Section III-B:

Problem 2: *Find a minimum number of layers allowing to connect together all areas with the same literal, and to connect them to the input leads, using non-crossing connections.*

The solution of Problem 1 gives the input for Problem 2. Since both problems are hard we solve them heuristically, showing experimentally that, for reasonable sizes of the lattice and of the number of variables, our heuristics allow to find efficient solutions.

II. Solving Problem 1

The solution of Problem 1 may be simplified if a preliminary examen of the lattice is performed with the attempt of reducing the number of literals contained in the subsets associated to the vertices. For this purpose the following Rule 1 may be tested and applied if possible.

Rule 1. Let v_j be a vertex; v_1, v_2, v_3, v_4 be the four vertices adjacent to v_i (if any); L_j, L_1, L_2, L_3, L_4 be the relative subsets of literals. Apply in sequence the following steps:

Step 1. Let $|L_j| > 1$. If a literal $x \in L_j$ does not appear in any of the setsL_i, for $1 \leq i \leq 4$, cancel x from L_j and repeat the step until at least one element remains in L_j.

Step 2. Let $|L_j| > 1$, and let $L_k \subset L_j$ with $k \in \{1, 2, 3, 4\}$. If a literal $x \in L_j$ appears in exactly one set L_h with $h \in \{1, 2, 3, 4\}$ and $h \neq k$, then cancel x from L_j and repeat the step until at least the literals of L_k remain in L_j.

The following proposition be easily proved:

Proposition 1. *The application of Rule 1 does not prevent finding an MPA.*

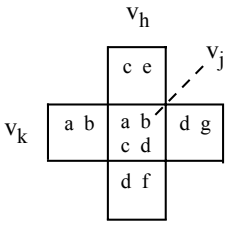

Fig. 2: Canceling a literal from a multiple choice using step 2 of Rule 1. Literals are denoted by a, b, c, d, e, f, g. Literal c in cells v_j, v_h is canceled from v_j.

An example of application of step 2 of Rule 1 is shown in Fig. 2. A literal cancelation from L_j may induce a further cancelation in an adjacent cell. In the example of Fig. 2, if all the cells adjacent to v_h except for v_j do not contain the literal c, the cancelation of c from L_j induces the cancelation of c from L_h if step 1 of Rule 1 is subsequently applied to v_h. Before running an algorithm for solving Problem 1, the sets L_i may be reduced using Rule 1 through a scanning of the lattice.

1	1	5	5	2	2
1	4	2	1	5	5
3	3	2	1	5	7
3	3	2	4	6	6
5	6	6	3	3	7

Fig. 3: Example of starting lattice.

Moreover, as we have seen several successive scans may be applied for further reduction until no change occurs in a whole scan, although this is likely to produce much less cancelations than the first scan. These operations constitutes the first phase of any algorithm. Then, as the problem is computationally intractable, an heuristic must be applied. The one proposed here is the simplest possible, namely:

- scan the lattice row-wise: for any vertex v_i reduce the associated set of literals L_i to just one of its elements chosen at random;
- for any vertex v_i not yet included in an area, build a tree T_i spanning the maximal connected subgraph whose vertices hold the same label of T_i and include the vertices of T_i in a new area.

The experimental results discussed in the last section are derived with this simplified approach. Better results would be possibly obtained with more skilled heuristics at the cost of a greater running time.

III. Solving Problem 2

In order to better understand the nature of the problem let us explain it with an example. The starting point for Problem 2 is a lattice of $N \times M$, positions, each associated to one of the $2n$ literals, corresponding to the n input variables and their complements; in practical applications we have $2n < N \times M$, hence there are positions assigned to the same literal. As we have mentioned before, all these positions must be connected together in order to be reached in parallel from outside.

Fig. 3 shows an example of starting lattice of size 5×6 and 7 literals, (in this case directed variables only) indicated by numbers and not by x_i for simplicity. Let us suppose that the side devoted to connections to outside is the top side of the lattice. In the first layer the only connections we can lay out are those of the areas on the top row with outside and those connecting positions inside the same area. The connections of the first layer are shown in Fig. 4 (a). The first layer can be implemented only in this way for this example. In the first layer there is no room for other connections, hence a new layer must be added. For the already connected areas, it is enough that a single position of the area goes to the next layer. Obviously the final solution can be affected also by the selection of the position we choose, however we do not consider this possibility and we arbitrary choose this position: in our example of Fig. 4 (a), the underlined literals indicate that the corresponding position is selected.

A possible implementation of the second layer is shown in Fig. 4 (b). Recall that connections cannot cross, hence not all areas can be connected. Note that areas already connected

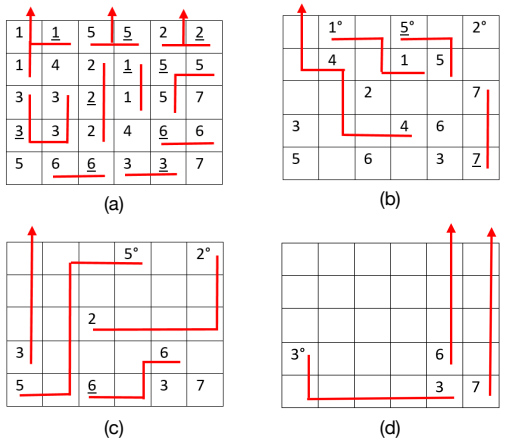

Fig. 4: (a) The first layer, (b) the second layer, (c) the third layer, and (d) the last layer.

	0	1	2
0	1	2	3
1	2	3	1
2	3	1	2

Fig. 5: A non solvable instance.

to the top side, even if not completely connected, don't need to be connected to the top side again. For example, the area associated to literal 1, already connected to outside in layer 1, is not connected to the top side in layer 2; areas associated to literals $3, 6, 7$ instead have still to be connected. Note also that all literals with label 4 have been connected in a single area and outside, therefore there is no need to connect this area to the next layer. The next layer, layer 3 is depicted in Fig. 4 (c), and the fourth and last layer is shown in Fig. 4 (d). The way of selecting the connections is also arbitrary for layer 3, while all the rest is connected in the last layer.

In principle we do not know if the given problem can be solved in less than 4 layers. In fact, as we discuss below, the problem is computationally quite difficult, if not at all impossible to be solved. For a formal approach we define a *legal wiring* an assignment of the connections to a layer such that all the rules $(1-4)$ introduced in Section I are respected. Then Problem 2 is formalized as follows:

Instance: An array of size $N \times M$ positions, containing integers in the range $1, \ldots k$, $k < N \times M$ or empty.

Goal: Connect together all integers with the same value and lead the connections to row 1, in a legal wiring, to obtain the layout with the minimal number H of layers.

A. Impossible Instances

It is not difficult to verify that not all possible configurations of the input lattice are solvable. For instance consider an array where each row contains a cyclic shift of the literals in the previous row, as in the example of Fig. 5. Since no cell can be connected with others with the same literal, no connections are possible in any layer. We now show that the vast majority of problem instances are theoretically solvable,

although some may require an exceedingly high number of layers to be practically solved. We have:

Proposition 2. *A problem instance cannot be solved if and only if in the initial literal assignment no two adjacent cells share a same literal and, no matter how the multiple assignments are resolved in Problem 1, each cell in row zero contains a literal that occurs also in another cell.*

Proof. *If part.* If no two adjacent cells share a same literal initially, in the assignment of Problem 1 all areas contain exactly one literal. Although the cells of row zero can be connected to the output, there is no way to connect them to the other cells with the same literal since all cells will be occupied by a connection in all layers.

Only-if part. If at least one of the conditions stated in the proposition does not hold, at least one cell in layer 2 is made available (or "free") for routing due to an area built in layer 1, or to a connection to the output in that layer. Once a free cell arises, it can be "moved" to any cell of the array by consecutive movements of adjacent literals as in the well known *15-slide game*, and any literal adjacent to the free cell can similarly be moved around to be brought adjacent to a cell with the same literal. Proceeding with this strategy all cells with a same literal can be linked together and brought to the output. □

Note that the strategy indicated in the only-if part of the above proof may require a very large number of layers if only a small number ν of free cells exist in a layer, as only ν movements can be done in that layer. In particular, if only a few cells are made free by the solution of Problem 1, i.e. if layer 1 contains a large number of small areas, the routing mechanism may not apply in practice. As we shall see, an answer must be left to simulations on significant examples.

B. Hardness of Problem 2

Solving Problem 2, the cells containing the same literal in any layer will be connected as trees (not as general subgraphs) to avoid useless occupation of free cells. The problem of minimizing the number of layers is related to the one of building the maximum number of such trees in any layer whose edges do not intersect. If a 15-slide movement of free cells is required the problem is NP-hard [10]. If such movements are not required the problem has strong similarities with other known NP-hard problems dealing with grid embedding of graphs, as for example determining the Steiner tree among k vertices on a grid [4], or determining the rectilinear crossing number of a graph [5], etc. We have not been able to prove that Problem 2 is NP-hard also in this simpler case, however for its solution we rely on a heuristic algorithm that produces satisfying results on a large class of benchmark instances. If no tree can be directly built in a layer, as discussed in the previous subsection, the heuristic stops declaring that routing is impossible. Otherwise we have:

Proposition 3. *Let α be the number of areas generated by Problem 1 and k be the number of literals. An upper and a lower bound to the number of layers are given by α and $\lceil k/M \rceil$, respectively.*

Proposition 3 is immediately proved by noting that at least one pair of cells holding the same literal are connected in each layer (upper bound), and k external leads must be reached from the M cells of the upper row of the grid (lower bound). In the example of Fig. 4 we have $\alpha = 15$, $k = 7$, and $M = 6$. The proposed layout with $H = 4$ layers is far from approaching the upper bound 15, while is reasonably close to the lower bound $\lceil 7/6 \rceil = 2$.

C. Heuristics for Problem 2

We propose a heuristic algorithm for solving Problem 2. In this first approach, we never "move" free cells of the array, as in a 15-slide game, as this strategy might lead to layouts with a very high number of layers; thus we consider impossible all instances whose solvability requires such moves. This limitation only slightly affects our experimental results on benchmark circuits, as our algorithm failed to find layout for theoretically solvable lattices only for about 4% of the lattices (see Section IV).

A general greedy heuristic for Problem 2, could consist in the following two main steps:

(i) Connect all adjacent cells with the same literal on the first layer and then connect them to the external input leads, whenever possible;

(ii) while there are literals still to be connected between them and/or to the outside:

- add a new layer
- try to connect each pair of cells assigned to the same literal
- try to connect each literal to outside, if not already connected to the external leads.

Step (i) can be implemented in a standard way, visiting the lattice to search and connect all positions inside an area of adjacent cells assigned to the same literal. Moreover, all areas with a cell on the top row can be connected to the external leads. Note that this initial step is optimal, i.e., an optimal minimization algorithm for the number of layers cannot do better on the first layer.

To implement the second and main step of the heuristic, we introduce the concept of *free area* and *boundary cells*, that will be exploited to check whether two cells with the same literal can be connected, and to search a path between them.

Definition 1. *A* free area *in a lattice is a subset of free adjacent cells. The* boundary cells *of a free area are the cells surrounding it.*

Free areas are computed through a scanning of the lattice, in time linear in its dimension. An example of lattice with three different free areas is shown in Fig. 6. Observe that all boundary cells assigned to a single literal facing the same free area can be connected, since we can use the free cells inside the area to lay out the connections. Of course, this holds only for the first literal of the boundary cell that will be processed, while the others boundary cells, assigned to different literals, can be connected only if the required connections do not cross those already laid out on that area, since different connections cannot share a position.

Our idea is to use free areas to avoid the search for connections that are impossible to implement: we limit the

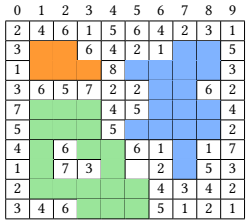

Fig. 6: A lattice with three free areas.

Algorithm for layout computation

Function *Thread (Free Area A)*
  for each *pair of boundary cells* c_1, c_2 *with the same literal*
    SearchHeuristic(c_1, c_2)

Function *main ()*
  Connect areas of adjacent cells with the same literal
  Connect cells on the top row to the external leads
  while *there are cells still to be connected*
    Select a single position in each area for the next layer
    Add a new layer
    Compute the free areas
    for each *(free area A)* StartNewThread *(Thread (A))*
    JoinThread()

Fig. 7: Layout computation.

search to the connections traversing a free area, leaving the search for connections between cells facing different areas to the next iterations. In this way we can save computational time, as cells around different areas cannot be directly connected (see for example the two cells $(1, 0)$ and $(2, 9)$ assigned to the literal 3 in the lattice in Fig. 6). Therefore, we structure step *(ii)* of the proposed heuristic as follows: *first compute all free areas, then try to connect all boundary cells assigned to the same literal facing the same free area.* Since free areas are mutually disjoint, the searches for connections can be performed in parallel creating a thread for each free area. The only portion of the lattice shared by multiple threads are the boundary cells facing different free areas. These situations are managed with appropriate lock variables, which force the threads to access those cells in mutual exclusion.

As already mentioned, a drawback of this strategy is that it does not completely eliminate the search for connections that are impossible to implement on the current layer. Indeed, as soon as two cells have been connected through a path on a free area, some other cells facing this same area become unreachable from one another, since the first connection divided the free area in separate subareas. We could solve this issue recomputing the free areas after each connection, but this approach is computationally very heavy. Therefore, we compute free areas only once, at the beginning of each iteration, and then apply *non-exhaustive search algorithms* within each area, in order to limit the search for non-existing connections, still guaranteeing that mutually reachable cells will be connected with high probability. The overall heuristic is described in Fig. 7.

Let us now briefly discuss the possible implementations of the search heuristics within each area. We are given a boundary cell c_1 that must be connected to a target cell c_2, through a path of distinct free cells in the free area A. This can be formalized as a state space search, where the state space is of size $O(4^{N\times M})$ as the number of cells in the area is $O(N\times M)$ and there are at most four possible moves from each cell. As a search in this space would be prohibitively expensive, we can use heuristics to find solutions of high quality as quickly as possible. We have considered *best-first, beam searches, greedy beam search, and hill-climbing* heuristics [7], [11]. These heuristics select the next cell to visit according to an evaluation function h that provides an estimate of the distance from the target cell, both under Euclidean and Manhattan distance. The first two heuristics provide better results, but are computationally very expensive (their time complexity is $O(4^{N\times M})$ and $O(2^{N\times M})$, respectively) and can be applied only to small size lattices. The last two have time complexity linear in the lattice size, but produce worse quality results, as they fail in connecting some mutually reachable cells on a given layer and the final layout may contain a very high number of layers. Depending on the lattice size and on the specific application, we can therefore select one of the four heuristics, trading-off quality of results vs. scalability.

IV. Experimental Results

In this section we report the experimental results related to the physical implementation of switching lattices, according to the rules (1-4) described in Section I. The aim of our experimentation is to determine whether the physical implementation of the lattices, shaped as a multidimensional grid of size $N\times M\times H$, where N and M are the number of rows and columns of the lattice, and H is the number of layers, could be considered technologically feasible. In our work we have considered the lattices obtained applying the Altun-Riedel method to the benchmarks taken from LGSynth93 [12], where each output has been treated as a separate Boolean function. Due to the limited space available, we report in the following only a significant subset of the functions as representative indicators of our experiments.

The experiments have been run on a IntelCore i7-4710HQ 2.50GHz CPU with 8 GB of main memory, running Linux Ubuntu 17.10. The algorithm for computing the number of layers has been implemented in C.

In this first experimental evaluation, we have analyzed lattices where the literals assigned to the switches have been chosen arbitrarily, in all cases of different choices at the switch. In Table I we report (a subset of) the results of this experimentations. The first column reports the name and the number of the separate output function of the benchmark circuit. The following two columns report the number of different literals occurring in the lattice and its dimension ($N\times M$). Finally, the last four columns report the number H of layers computed by the Algorithm in Fig. 7 with the *best-first* and with the *greedy beam search* heuristic, together with the corresponding running time (in seconds). The last row reports the sum of the values of the corresponding column. The cases

TABLE I: Number of layers for the lattice layout of a subset of standard benchmark circuits, for lattices with arbitrary selection of literals in all cases of different choices at the switch.

			Best-first		Greedy Beam	
Bench	lit	N×M	H	Time(s)	H	Time(s)
add6(5)	24	156×156	8	757.59	9	0.01
adr4(1)	16	36×36	21	0.85	24	0.08
alu2(2)	16	10×11	4	0.01	5	0.01
alu2(5)	20	13×14	5	0.01	5	0.01
alu3(0)	8	4×5	4	0.01	4	0.01
alu3(1)	12	7×8	5	0.01	5	0.01
b12(0)	7	6×4	4	0.01	4	0.01
b12(1)	9	5×7	5	0.01	5	0.01
b12(2)	10	6×7	7	0.01	7	0.01
bcc(5)	28	27×9	11	0.04	12	0.01
bcc(7)	29	31×11	19	0.08	26	0.02
bcc(8)	29	31×12	17	0.07	20	0.02
bcc(27)	28	39×19	18	0.21	20	0.06
bcc(43)	28	20×10	9	0.02	10	0.01
bench1(2)	18	45×24	22	0.43	29	0.15
bench1(3)	18	31×16	29	0.19	45	0.04
bench1(5)	18	50×27	20	0.51	25	0.20
bench1(6)	18	35×21	25	0.33	32	0.08
bench1(7)	18	43×21	19	0.24	22	0.11
bench1(8)	18	44×24	22	0.45	28	0.14
bench(6)	10	8×4	6	0.01	6	0.01
br2(4)	18	18×8	9	0.01	10	0.01
br2(5)	19	14×4	16	0.01	16	0.01
br2(6)	19	16×5	14	0.01	14	0.01
clpl(3)	11	6×6	2	0.01	2	0.01
clpl(4)	9	5×5	2	0.01	2	0.01
co14(0)	28	92×14	12	0.38	14	0.05
dc1(4)	7	5×4	6	0.01	6	0.01
dc2(4)	11	10×9	8	0.01	8	0.01
dc2(5)	9	6×6	7	0.01	7	0.01
dk17(1)	10	8×2	6	0.01	6	0.01
dk17(3)	11	11×3	9	0.01	9	0.01
dk17(4)	12	9×3	–	0.01	–	0.01
ex1010(0)	20	91×46	34	29.22	40	1.35
ex4(4)	13	17×6	7	0.01	7	0.01
ex4(5)	27	35×45	14	1.12	15	0.08
ex5(32)	14	4×10	8	0.01	8	0.01
ex5(36)	11	2×8	4	0.01	4	0.01
ex5(38)	13	4×9	5	0.01	5	0.01
ex5(40)	15	6×12	10	0.01	14	0.01
ex5(43)	15	8×14	11	0.01	15	0.01
exam(5)	13	11×6	8	0.01	10	0.01
exam(9)	20	59×30	27	2.19	33	0.43
max128(5)	14	14×17	11	0.01	13	0.01
max128(8)	13	5×10	10	0.01	12	0.01
max128(17)	14	26×25	20	0.18	19	0.04
max1024(5)	20	117×122	31	1087.12	33	13.83
mp2d(6)	14	10×6	10	0.01	10	0.01
mp2d(9)	14	6×8	4	0.01	4	0.01
mp2d(10)	10	6×3	–	0.01	–	0.01
sym10(0)	20	130 × 210	11	2938.34	13	11.57
tial(5)	28	181×181	21	5622.57	30	40.99
z4(0)	7	15×15	7	0.01	9	0.01
z4(1)	14	28×28	11	0.12	13	0.01
Z5xp1(2)	14	12×11	10	0.01	10	0.01
Z5xp1(3)	14	18×18	13	0.03	16	0.01
			658	10442,61	770	72,32

where the algorithm failed in finding a layout for theoretically unsolvable lattices are marked with –. Considering the whole set of benchmarks analyzed, the algorithm did not find a layout for about 4% of the lattices.

By comparing the results, the values show that, as expected, we obtain a better layout using the best-search heuristic, at the expense of the computational time. Moreover, we note that the increase in the number of layers computed with the faster greedy beam search heuristic appears quite limited on average, while it can be relevant on single lattices (see for example benchmarks $bench1(3)$ and $tial(5)$).

V. Concluding remarks

We have presented the first study on connection layout for two-dimensional switching lattices referring to the network implementation proposed by Altun and Riedel [3]. We have shown how to build a stack of consecutive layers where the connections between switches driven by the same variable can be laid without crossings, with the aim of minimizing the number of layers. Since the problem is computationally intractable we have designed a family of heuristics for finding satisfactory solutions, reporting only the results of the fastest and the slowest of the two on a standard subset of Boolean functions, for space reasons.

Countless improvements are open. For theoretical completeness, the NP-hardness of step 4 of our approach has to be proved to fully justify the use of heuristics. Better heuristics could be studied, and tested on larger data samples. The layout for other switching lattices should be considered. The layout rules should be possibly changed, in particular allowing more than one wire traversing a switch area in the higher layers. We are presently working on all these issues.

References

[1] S. B. Akers, "A Rectangular Logic Array," *IEEE Trans. Comput.*, vol. 21, no. 8, pp. 848–857, Aug. 1972.

[2] M. Altun and M. D. Riedel, "Lattice-Based Computation of Boolean Functions," in *Proceedings of the 47th Design Automation Conference, DAC 2010, Anaheim, California, USA, July 13-18, 2010*, 2010, pp. 609–612.

[3] ——, "Logic Synthesis for Switching Lattices," *IEEE Trans. Computers*, vol. 61, no. 11, pp. 1588–1600, 2012.

[4] C. C. N. Chu and Y. Wong, "FLUTE: fast lookup table based rectilinear steiner minimal tree algorithm for VLSI design," *IEEE Trans. on CAD of Integrated Circuits and Systems*, vol. 27, no. 1, pp. 70–83, 2008. [Online]. Available: https://doi.org/10.1109/TCAD.2007.907068

[5] J. Fox, J. Pach, and A. Suk, "Approximating the rectilinear crossing number," *CoRR*, vol. abs/1606.03753, 2016. [Online]. Available: http://arxiv.org/abs/1606.03753

[6] G. Gange, H. Søndergaard, and P. J. Stuckey, "Synthesizing Optimal Switching Lattices," *ACM Trans. Design Autom. Electr. Syst.*, vol. 20, no. 1, pp. 6:1–6:14, 2014.

[7] P. E. Hart, N. J. Nilsson, and B. Raphael, "A formal basis for the heuristic determination of minimum cost paths," *IEEE Trans. Systems Science and Cybernetics*, vol. 4, no. 2, pp. 100–107, 1968. [Online]. Available: https://doi.org/10.1109/TSSC.1968.300136

[8] F. Luccio and M. Xia, "The MPA graph problem: definition and basic properties." *Department of Informatics, University of Pisa. Technical Report.*, 2018.

[9] G. D. Micheli, *Synthesis and Optimization of Switching Theory*. McGrow Hill, 1994.

[10] D. Ratner and M. K. Warmuth, "Finding a shortest solution for the N × N extension of the 15-puzzle is intractable," in *Proceedings of the 5th National Conference on Artificial Intelligence. Philadelphia, PA, August 11-15, 1986. Volume 1: Science.*, 1986, pp. 168–172.

[11] S. J. Russell and P. Norvig, *Artificial intelligence - a modern approach, 2nd Edition*, ser. Prentice Hall series in artificial intelligence. Prentice Hall, 2003. [Online]. Available: http://www.worldcat.org/oclc/314283679

[12] S. Yang, "Logic Synthesis and Optimization Benchmarks User Guide Version 3.0," Microelectronic Center, User Guide, 1991.