The Connection Layout in a Lattice of Four-Terminal Switches

Anna Bernasconi, Antonio Boffa, Fabrizio Luccio, Linda Pagli

The Connection Layout in a Lattice of Four-Terminal Switches

Anna Bernasconi[1], Antonio Boffa[1], Fabrizio Luccio[1], and Linda Pagli[1]

Dipartimento di Informatica, Università di Pisa, Italy
{anna.bernasconi, 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. All switches controlled by the same literal must be connected together and to an input lead of the chip, and the layout of such connections must be realized in superimposed layers. Inter-layer connections are realized with vias, with the overall goal of minimizing the number of layers needed. The problem shows new interesting combinatorial and algorithmic aspects. Since the specific lattice cell where each switch is placed can be decided with a certain amount of freedom, and one literal among several may be assigned for controlling a switch, we first study a lattice rearrangement (Problem 1) and a literal assignment (Problem 2), to place in adjacent cells as many switches controlled by the same literal as possible. Then we study how to build a feasible layout of connections onto different layers using a minimum number of such layers (Problem 3). We prove that Problem 2 is NP-hard, and Problems 1 and 3 appear also intractable. Therefore we propose heuristic algorithms for the three phases that show an encouraging performance on a set of standard benchmarks.

1 Introduction

The logic synthesis of a Boolean function is the procedure for implementing 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 NAND or NOR circuit of maximal depth 2 [**?**]. In this paper, we focus on a different synthesis method based on a *switching lattice*, that 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 [**?**]. 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 [**?**,**?**,**?**]. A Boolean function can be implemented in a lattice with the following rules:

– each switch is controlled by a Boolean literal, i.e. by one of the input variables or by its complement;

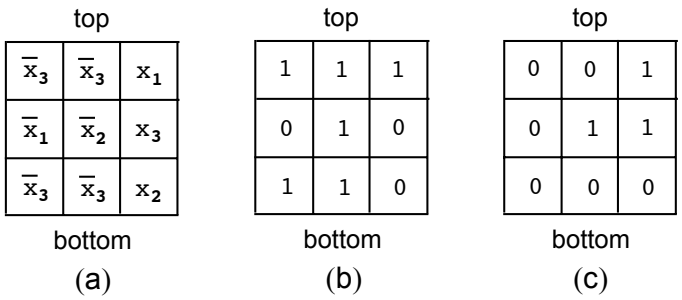

Fig. 1. A network of four terminal switches implementing the function $f = x_1x_2x_3 + \overline{x}_1\overline{x}_3 + \overline{x}_2\overline{x}_3$ (a); the lattice evaluated on the assignments 1,0,0 (b) and 1, 0, 1 (c), with 1's and 0's representing ON and OFF switches, respectively.

- if a literal takes the value 1 all corresponding switches are connected to their four neighbors, else they are not connected;
- 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 lattice of switches and corresponding literals in Figure 1-a implements the function $f = x_1x_2x_3+\overline{x}_1\overline{x}_3+\overline{x}_2\overline{x}_3$. If we assign the values 1, 0, 0 to the variables x_1, x_2, x_3, respectively, we obtain paths of 1's connecting the top and the bottom edges of the lattices (Figure 1-b), and f evaluates to 1. On the contrary, the assignment $x_1 = 1, x_2 = 0, x_3 = 1$, on which f evaluates to 0, does not produce any path from the top to the bottom edge (Figure 1-c). All the other input assignments can be similarly checked.

The synthesis of a function f on a lattice consists of finding an assignment of input literals to the switches, such that the top-bottom paths in the lattice implement f and the number of switches in the lattice is reduced as much as possible. Recalling that the *dual* f^D of a function f is such that $\overline{f^D}(\overline{x}_1, \ldots, \overline{x}_n) = f(x_1, \ldots, x_n)$, in [**?**,**?**] Altun and Riedel developed a synthesis method where the implicants (products) of the minimal irredundant SOP forms of the function f and of its dual f^D are respectively associated, in any order, to the columns and to the rows of the lattice. The literal assigned to a switch is chosen from the necessarily non-void intersection of the two subsets of literals appearing in the implicants corresponding to the intersecting column and row of the lattice. If several literals appear in the intersection anyone of them can be chosen. As an elementary example consider the function $f = x_1\overline{x}_3\overline{x}_4 + x_1x_2 + \overline{x}_1\overline{x}_3x_4$ and its dual $f^D = x_1\overline{x}_3 + x_1x_4 + x_2\overline{x}_3 + \overline{x}_1x_2\overline{x}_4$. Figure 2-a shows the lattice for f where just one multiple assignment $x_1, \overline{x}_3$ occurs.

Starting from the lattice obtained by the Altun-Riedel method we consider three problems related to the physical implementation of the circuit, that must be solved obeying the following assumptions.

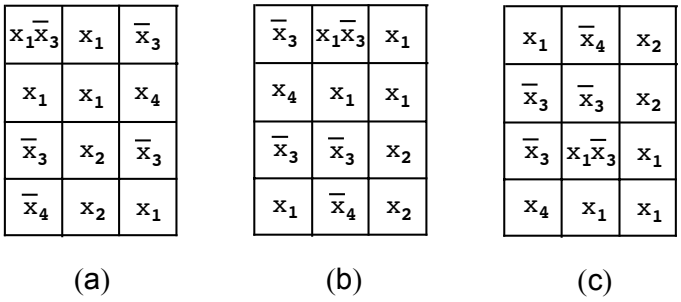

Fig. 2. A lattice implementing the function $x_1\overline{x}_3\overline{x}_4 + x_1x_2 + \overline{x}_1\overline{x}_3x_4$ (a), where a choice between x_1 and $\overline{x}_3$ must be performed in the first cell. The lattice after column permutation (b), and then row permutation (c), obtained with our method.

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. In particular the connections cannot cross on the same layer.
4. The overall target is designing a layout with the minimum number of layers. Since the problem is computationally-hard, it will be relaxed to finding a reasonable layout by heuristic techniques.

The circuit will be built starting from the original $N \times M$ lattice (*level 0*), and superimposing to it a certain number H of layers (*levels 1 to H*), to give rise to a three-dimensional grid of size $N \times M \times (H+1)$. Note that the switches associated with the same literal cannot be generally connected all together on the same layer, so one or more subsets of these switches will be connected on a layer and then made available through vias on the next layers to be connected to other subsets.

Two degrees of freedom remain after the function has been synthesized, to be used to reduce the number of layers in the layout. One is the possibility of permuting lattice columns and rows arbitrarily, as exploited in the following Problem 1. The other is the possibility of selecting any literal from a multiple choice for each switch, as exploited in the following Problem 2. Problems 1 and 2 apply at level 0, preparing the lattice for the actual layout design which takes place in the next layers 1 to H, where the inputs of the switches associated with the same literal are connected together and to the corresponding external lead, as treated in the following Problem 3.

Problems 1, 2, and 3 are solved one after the other, each producing the input for the next one. Since all of them are computationally-hard, we solve them heuristically, then show experimentally that our solutions are efficient for

standard benchmarks where the size of the lattice and the number of variables are reasonable. A preliminary version of this study limited to Problem 2 (without a proof on NP-hardness) and to Problem 3 has been presented in [**?**].

2 Rearranging the lattice

In principle a lattice can be seen as an array $A[N \times M]$, or as a non-directed graph $G = (V, E)$ whose vertices correspond to the lattice cells (then $|V| = NM$) and whose edges correspond to the horizontal and vertical connections between adjacent cells (then $|E| = 2NM - N - M$). We shall refer indifferently to a lattice cell $A[i, j]$, $1 \leq i \leq N, 1 \leq j \leq M$, or to graph vertex v_k, $1 \leq k \leq NM$. Obviously the vertices have degree 2, 3, or 4 if they respectively correspond to corner, border, or internal cells of the lattice.

Let $x_1, x_2, \ldots, x_n$ be the variables of the function and L be the set of all literals, $|L| = 2n$. After the Altun-Rieder synthesis is completed, each cell $A[i, j]$ is associated with a non-void subset $L_{i,j} \in L$, from which one literal has to be eventually assigned to the corresponding graph vertex. We pose:

Definition 1. *If a single variable is associated to each vertex, an* mc-area *is a maximal connected subgraph S of G in which all variables hold the same literal. Equivalently an mc-area is the portion of the lattice corresponding to S.*

Note that if two mc-areas A_1, A_2 hold the same literal, no two cells $c_1 \in A_1, c_2 \in A_2$ may be adjacent since A_1 and A_2 are maximal.

A basic task is minimizing the number of mc-areas, or equivalently make them as large as possible, since this will imply reducing the number of layers. As shown below the core of this problem is NP-hard, so we study how to solve it heuristically. As already said we proceed in two consecutive phases. The first phase is aimed at permuting columns and rows in order to increase the number of adjacent cells holding common literals, even though subsets of two or more literals may still be associated to each cell. The second phase consists of the selection of one literal in each of the multiple assignments at the cells, so the mc-areas can be built according to Definition 1. To implement the first phase we pose:

Definition 2. *The* weight *of a pair of cells $A[r, s], A[u, v]$ is given by: $w^{u,v}_{r,s} = |L_{r,s} \cap L_{u,v}|/(|L_{r,s}| \cdot |L_{u,v}|)$.*

For example for $L_{r,s} = \{a, b, c\}, L_{u,v} = \{b, c, d, e\}$ we have $L_{r,s} \cap L_{u,v} = \{b, c\}$, then: $w^{u,v}_{r,s} = 2/(3 \cdot 4) = 1/6$. Note that $w^{u,v}_{r,s}$ is the probability for $A[r, s], A[u, v]$ to share the same literal if one literal is randomly chosen from $L_{r,s}$ and $L_{u,v}$.

The weight is relevant in our case for pairs of adjacent cells. As we are interested in building large mc-areas, we pose:

Problem 1. *Find a column permutation and a row permutation of A, to maximize the sum of weights of the pairs of adjacent cells.*

Once Problem 1 has been solved, one literal must be selected in each subset $L_{i,j}$ as stated in the following:

Problem 2. *Find a literal assignment for each cell $A[i,j]$ that minimizes the number of mc-areas.*

2.1 Solving Problem 1

To ease the task of Problem 1, an elementary observation is in order:

Observation 1 *Two cells can be made adjacent by column or row permutation if and only if they lay on the same row or on the same column, respectively.*

By Observation 1, column and row permutations can be independently performed since the result of one does not affect the result of the other. In fact the problem will be solved by a column permutation followed by a row permutation. We pose:

Definition 3. *The* weight $C_{s,v}$ *of a pair of columns s, v of the lattice is the sum of weights of all the pairs of cells lying in the columns s, v and in the same row, that is: $C_{s,v} = \sum_{i=1}^{N} w_{i,s}^{i,v}$.*

And, simmetrically:

Definition 4. *The* weight $R_{r,u}$ *of a pair of rows r, u of the lattice is the sum of weights of all the pairs of cells lying in the rows r, u and in the same column, that is: $R_{r,u} = \sum_{j=1}^{M} w_{r,j}^{u,j}$.*

For building large mc-areas we are interested in bringing adjacent pairs of columns and rows with higher weights, via permutations. To this end we adopt the following heuristic on the columns, then on the rows.

1. compute the weights $C_{s,v}$ of all pairs of columns;
2. build a stack S whose elements contain the pairs (s, v) with non-zero weights, ordered for decreasing value of such weights;
3. start with M subsets of adjacent columns, each containing exactly one of them;
4. pop one by one the pairs (s, v) from S: for the pair currently extracted decide a column re-arrangement (without actually performing it), merging two subsets of adjacent columns, one containing column s and the other containing column v, if the two columns can be made adjacent without breaking the subsets already built. The columns in each subset are kept ordered for increasing value of their index. This step 4 is performed according to the scheme COLUMN-PERM shown below;
5. once the stack S is empty, permute the groups of columns as indicated in the corresponding subsets previously determined. These groups can then be arranged in the lattice in any order.

COLUMN-PERMUTE

1. define a vector P of M elements, whose values are:
   $P[j] = 1$ if column j forms a subset of one column;
   $P[j] = 2$ if column j is the first one in a subset of more than one column;
   $P[j] = 3$ if column j is the last one in a subset of more than one column;
   $P[j] = 4$ if column j is neither the first one nor the last one in a subset of more than two columns;
2. initialize P as $P[j] = 1$ for $1 \leq j \leq M$;
   // all the initial subsets of columns contain one element:
   // recall that S is a stack of pairs of columns;
3. **while** S is not empty
   pop a pair (s, v) from S;
   if ($P[s] = 4$ OR $P[v] = 4$) discard (s, v)
   else according to the values $P[s], P[v]$ the subsets of s and v are merged bringing s adjacent to v and the values of $P[s], P[v]$ are updated accordingly;
   // nine combinations of values $P[s], P[v]$ are possible:
   // for $P[s] = P[v] = 2$ and $P[s] = P[v] = 3$ the order
   // of the columns in one of the subsets must be inverted;
4. the process ends with one or more subsets each corresponding to the permutation of a group of columns.

After column permutation, the rows are permuted with a procedure perfectly symmetrical to the one given for columns, using row weights $R_{r,u}$.

In the lattice of Figure 2 (a), the pairs of columns in S with non-zero weights, ordered for decreasing values of the corresponding weights are:

(1,2) $C_{1,2} = 3/2$, (1,3) $C_{1,3} = 3/2$

producing the subsets of adjacent columns $\{1, 2\}, \{3\}$ and then $\{3, 1, 2\}$, from which the column permutation of Figure 2 (b) is built. The ordered pairs of rows in S with non-zero weights are:

(1,2) $R_{1,2} = 3/2$, (1,3) $R_{1,3} = 3/2$, (3,4) $R_{3,4} = 1$

producing the subsets of adjacent rows $\{1, 2\}, \{3\}, \{4\}$, $\{3, 1, 2\}, \{4\}$, and then $\{4, 3, 1, 2\}$, from which the final row permutation of Figure 2 (c) is built.

A global measure W of the amount of adjacent equal variables in the whole lattice, called the *lattice weight*, is naturally given by the sum of weights of all pairs of adjacent cells, or equivalently by the sum of weights of all the pairs of adjacent columns and rows, namely: $W = \sum_{j=1}^{M-1} C_{j,j+1} + \sum_{i=1}^{N-1} R_{i,i+1}$. In fact an increase of the value of W can be seen as an indicator of the advantage deriving from a column-row permutation. In the example of Figure 2 the lattice

weight increases from $W = 4$ (part (a)) to $W = 7$ (part (c) after column and row permutation). In the simulations discussed in the last section such value roughly doubles on the average after the heuristic for Problem 1 is applied.

Implemented with standard data structures, the time required by the above heuristic is $O(NM^2n + MN^2n)$ versus $O(NMn)$ of the input size (recall that n is the number of variables).

2.2 Hardness of Problem 1

Although our heuristic for Problem 1 shows an encouraging experimental value, a weakness derives from the decision of not restructuring a subset of columns/rows after it is built, aside from possibly inverting the order of its elements. Merging two subsets in a strictly optimal way would lead to an exponential explosion of the time needed with a possibly minor improvement of the result. Up to now we have not been able to decide the time complexity of the problem, although we believe that is NP-hard. We simply pose the question of its precise hardness as a challenging open problem.

2.3 Solving Problem 2

Problem 2 is of high computational interest for two reasons. The first is that the number of prime implicants in f and f^D strongly increases with the number of variables of the function, so that the elementary examples that we have given so far do not show the real entity of the phenomenon. In particular the possibility of having multiple assignments of many literals in the lattice cells is substantially high, making Problem 2 a crucial part of lattice rearrangement. The second reason is that Problem 2 is NP-hard as we will prove in the next subsection, so a clever heuristic must be devised for its solution. To this end a preliminary exam of the lattice is performed by applying the following Rule 1, with the attempt of reducing the number of literals contained in the subsets associated to the vertices.

Rule 1. Let v_j be a vertex; v_1, v_2, v_3, v_4 be the four vertices adjacent to v_j (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 sets L_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.

We have:

Proposition 1. *The application of Rule 1 does not prevent finding a literal assignment that minimizes the number of mc-areas.*

Proof. Assume that a literal x canceled from v_j by Step 1 or Step 2 of the rule would instead be assigned to v_j in the final assignment.

Step 1. An mc-area containing only v_j with literal x would result. If, in a minimal solution, a different literal y is assigned to v_j and is not assigned to v_1, v_2, v_3, or v_4, the number of mc-areas remains the same. If instead y is assigned to one or more of these vertices the number of mc-areas decreases.

Step 2. If x is not assigned to v_h an mc-area containing only v_j with literal x would result, otherwise an mc-area containing v_j, v_h, and possibly other vertices with literal x would result. Since one of the literals $y \in L_k$ must be assigned to v_k in any minimal assignment, and $L_k \subset L_j$, then y could be assigned to v_j instead of x and the number of mc-areas would remain the same, or would decrease if y is assigned also to other neighbors of v_j. □

An example of application of step 2 of Rule 1 is shown in Figure 3. A literal cancelation from L_j may induce a further cancelation in an adjacent cell. In the example of Figure 3, 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.

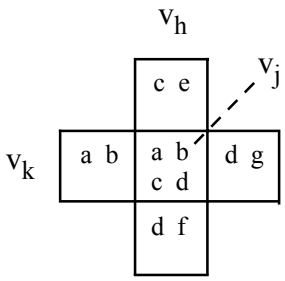

Fig. 3. Canceling a literal from a multiple choice using step 2 of Rule 1. Literals are denoted by a, b, c, d, e, f, g. Literal c in cells v_j, v_h is canceled from v_j.

Before running an algorithm for solving Problem 2, the sets L_i may be reduced using Rule 1 through a scanning of the lattice. This is a simple task, although a clever data structure must be devised for reducing the time required. Moreover several successive lattice 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, a heuristic must be applied. The following algorithm MC-AREAS proposed here, and already experimented in [**?**], builds each mc-area as a BFS tree, looking for a subset of adjacent vertices that share a same literal and assigning that literal to them. The lattice is treated as a tree.

MC-AREAS

1. start from a vertex v_i and reduce the associated set of literals L_i to just one of its elements l_i chosen at random;
   v_i will be the root of a tree T_i under construction for the current mc-area;
2. traverse the lattice from v_i in BFS form:
   forany vertex v_j encountered such that v_j does not belong to a BFS tree already built:
   if $(l_i \in L_j)$ assign l_i to v_j and insert v_j into T_i
   else insert v_j into a queue Q;
   continue the traversal for T_i;
3. **if** (Q is not empy) extract a new vertex v_i from Q and repeat steps 1 and 2 to build a new tree;

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

2.4 Hardness of Problem 2

To prove that Problem 2 is NP-hard we formulate it in graph form as done in [**?**] where the proof appeared. Let $G = (V, E)$ be an undirected graph and let C be a set of k colors such that each vertex $v_i \in V$ is associated to (or "contains") a non-void subset C_i of C. The graph problem equivalent to ours, indicated as *MPA* for *minimal partition (color) assignment*, is the one of assigning to each vertex v_i of G a single color from among the ones in C_i such that the number σ of maximal connected subgraphs of $G_1, \ldots, G_\sigma$ of G whose vertices have the same color is minimal. Dealing with a lattice, as required in our Problem 2, it is sufficient to start with *PMPA*, that is the MPA problem where G is planar. We have:

Proposition 2. *The PMPA Problem is NP-hard.*

Proof. Reduction from planar graph 3-coloring. Let H be an arbitrary planar graph to be 3-colored with colors $1, 2, 3$, and let G be a corresponding planar graph whose MPA must be built. The set of colors of G is $\{1, 2, 3, 4, 5, 6, 7, 8, 9\}$. For each edge $e = (u, v)$ of H there are nine vertices $u, e, v, x_1, x_2, x_3, y_1, y_2, y_3$ in G connected as shown in Figure 4, with subsets of colors:

$$C_u = \{1, 2, 3\},\ C_v = \{1, 2, 3\},\ C_e = \{4, 5, 6, 7, 8, 9\},$$

$$C_{x_1} = \{1, 6, 7, 8, 9\},\ C_{x_2} = \{2, 4, 5, 8, 9\},\ C_{x_3}\{3, 4, 5, 6, 7\},$$

$$C_{y_1} = \{1, 4, 5, 6, 9\},\ C_{y_2} = \{2, 5, 6, 7, 8\},\ C_{y_3}\{3, 4, 7, 8, 9\}.$$

Consider a minimal collection of monochromatic connected subgraphs $G_1, \ldots, G_\sigma$ of G. We have: (1) the vertices u, v must belong to two distinct subgraphs

G_i, G_j and the vertex e cannot belong to G_i or to G_j because $C_e \cap C_u = \emptyset$ and $C_e \cap C_v = \emptyset$; (2) at most one of the vertices x_1, x_2, x_3 may belong to G_i and at most one of the vertices y_1, y_2, y_3 may belong to G_j due to their colors; (3) at most two of the vertices x_1, x_2, x_3 and at most two of the vertices y_1, y_2, y_3 may belong to the same subgraph of e implying that the colors assigned to u and v must be different due to the colors of all the vertices involved. Letting $G = (V, E)$, from the points 1, 2, and 3 we have $\sigma \geq |V| + |E|$ and equality is met if and only if different colors can be assigned to u and v, depending on the color constraint imposed by the other vertices to which u and v are adjacent in H. That is, H can be 3-colored if and only if MPA can be solved on G with $\sigma = |V| + |E|$. □

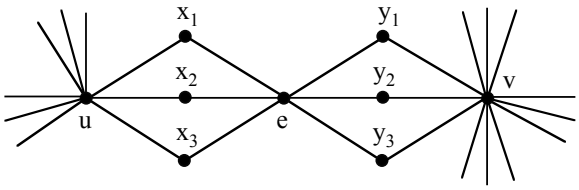

Fig. 4. Portion of graph G corresponding to the edge $e = (u, v)$ of graph H.

Starting from Theorem 2 we now prove that the result holds true even for planar grid-graphs *GMPA* as in the case of our lattice.

Proposition 3. *The GMPA Problem is NP-hard.*

Proof. We proceed in two steps. First we prove the result holds true for planar graphs of bounded vertex degree $d \geq 3$ by reduction from PMPA. Then we pass from bounded degree planar graphs to GMPA.

1. Reduction from PMPA, by insertion of new vertices to reduce all vertex degrees to at most 3. If edges $(a, b), (a, c), (a, d), (a, e)$ and possibly other edges (a, x) exist, i.e. $\deg(a) > 3$, insert a new vertex z with $C_z = C_a \cup C_b \cup C_c$, delete edges $(a, b), (a, c)$, and insert new edges $(a, z), (z, b), (z, c)$. Note that the degree of a decreases by 1, the degrees of b, c are unchanged, and z has degree 3. Continue until each vertex has degree ≤ 3. The solution for the new graph, i.e. the connected mono-colored subgraphs, coincides with a solution for PMPA if the new vertices z are deleted and the original edges are restored. Note that the graph resulting after the transformation is planar.
2. A result of Leslie G. Valiant (Theorem 2 of [**?**]), states that a planar graph G of n vertices with degree at most four admits a planar embedding in an $O(n \times n)$ grid Γ. Of the $O(n^2)$ cells of Γ, obviously only n are used in the embedding for the vertices of G, while many of the others are used

for embedding the edges of G as non intersecting sequences of cells in i, j directions. In [**?**] was then shown that one such embedding can be built where all edges are just straight line segments.

3. Build the embedding on Γ, and extend the grid to a new grid Γ' as follows. If two horizontal sequences of cells representing two edges of G lie in two rows $i, i+1$ and part of these sequences share the same columns (i.e. the two sequences are partly adjacent), insert a new empty row between i and $i+1$, using its cells where needed to fix vertical sequences possibly interrupted by the new row. Repeat the operation for any pair of partly adjacent sequences. Repeat the process on the columns, inserting new columns until no vertical sequences are partly adjacent. Note that the construction of Γ' has been done in time and space polynomial in n.
4. If two adjacent vertices a, b of G are embedded in two non adjacent cells of Γ', assign the set of colors $C_a \cap C_b$ to the cells of the sequence representing the edge (a, b). Repeat for all pairs of adjacent vertices. Assign a new color $c \notin C$ to all the grid cells not corresponding to the vertices and to the edges of G.
5. Solve GMPA on Γ' considering all the cells as vertices of a new larger graph. Discard the subsets of cells with color c, and in any other subset take only the cells corresponding to original vertices of G. These subsets constitute a solution for a bounded degree PMPA. □

3 Solving Problem 3

After Problem 2 is solved, we have to choose how to connect the different mc-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:

Problem 3: *Find a minimum number of layers allowing to connect together all the mc-areas with the same literal, and to connect them to the input leads, obeying the assumptions 1 to 4 of Section 1.*

The solution has to be constructive, that is, the actual layout must be shown.

In order to better understand the problem let us discuss an example. We start from a lattice of $N \times M$ cells each associated to one of the $2n$ input literals. Figure 5 shows a 5×6 lattice with 7 literals indicated for simplicity with the numbers 1 to 7. In practical applications we usually have $2n < N \times M$, hence there are cells assigned to the same literal that must be connected together to be reached in parallel from outside. Recall that these literals may be grouped in the mc-areas deriving from the previous lattice rearrangement. The starting lattice constitutes layer 0 of the layout, above which the connections are to be built in successive layers.

Suppose that the input leads to the circuit are in the top edge of the lattice. In layer 1 the only connections we can lay out are those of the mc-areas, and the mc-areas with cells in the top row may also be connected outside, see Figure 6 (a).

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. 5. Example of starting lattice.

Note that in layer 1 there is no room for other connections, hence a new layer 2 must be added. For each of the mc-areas connected in layer 1 and holding a literal that appears also in other mc-areas, or not yet connected to the outside, a single cell is connected through a *via* to the next layer. The final number of layers may be affected by the specific selection of these cells, however, in our heuristic we do not consider this point and choose the positions of the vias arbitrarily. In the layers of Figure 6 the underlined literals indicate where the corresponding vias start.

A possible implementation of the second layer is shown in Figure 6 (b). Each surviving mc-area is now represented by the arrival of a via. Recall that connections cannot cross, hence in general not all areas can be connected. Note that areas already connected to the top edge don't need to be connected to the this edge again, even though they are not completely connected among each other. This is the case of the area associated with literal 1, while areas associated with literals 3, 6, and 7 have still to be connected to the top edge. Note also that all literals with label 4 can be connected in a single area and outside, therefore there is no need to connect this area to the next layer.

The next layer 3 is depicted in Figure 6 (c) where one on the two areas of literal 3 can be connected to the top edge, but these two areas are still to be connected to one another. In this layer the connections for literals 2 and 5 are completed, while vias for literals 3, 6, and 7 are arbitrarily chosen. In the last layer 4 the layout is completed as shown in Figure 6 (d).

3.1 Impossible instances

To address Problem 3 formally we must start with a crucial observation, namely not all instances of it are solvable no matter how many layers are used. As an introduction consider the lattice of Figure 7 where each row contains a cyclic shift of the literals in the previous row and all these shift are different. It is easy to see that no cells with the same literal can be connected together independently of any column/row permutation. 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 4. *A problem instance cannot be solved if and only if in the initial literal assignment no two adjacent cells share a same literal after any column/row*

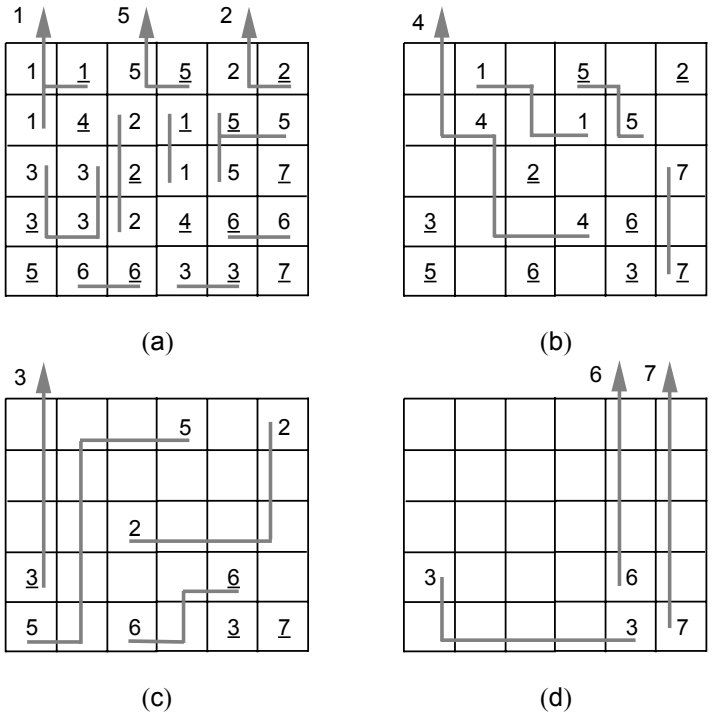

Fig. 6. (a) Layer 1 with the first connections. (b) Layer 2. (c) Layer 3. (d) Layer 4. Arrows indicate the connections to the input leads on the top edge. Underlined literals indicate the starting cells of the vias.

permutation (Problem 1) and, no matter how the multiple assignments are resolved (Problem 2), each cell in row zero contains a literal that occurs also in another cell.

Proof. If part. Assume that no two adjacent cells share a same literal after any solution of Problems 1 and 2. Although the cells of row zero can be connected to the output, there will be no way to connect them to the other cells with the same literal since all cells will be occupied by a via 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

1	2	3
2	3	1
3	1	2

Fig. 7. A non solvable instance.

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 exists 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 Problems 1 and 2, i.e. if layer 1 contains a large number of small mc-areas, the routing mechanism could require so many layers not to apply in practice. A decision on building or not such a layout must be taken after the simulation on significant examples.

3.2 Hardness of Problem 3

In the solution of Problem 3, the cells containing the same literal in any layer are connected, in the best case, as trees (and not as general subgraphs) to minimize the occupation of free cells, see next Subsection 3.3. 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 [**?**]. 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 [**?**], or determining the rectilinear crossing number of a graph [**?**], etc.

We have not been able to prove that Problem 3 is NP-hard, and leave it as a challenging open problem. For its solution we rely on a heuristic algorithm that produces satisfying results on a large class of benchmark instances as shown in the last section. 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 5. *Let α be the number of mc-areas generated by Problems 1 and 2, h be the number of literals involved in the lattice, and k be the number of literals appearing in the cells of the top edge. An upper and a lower bound to the number H of layers are given by $\alpha + \lceil (h-k)/M \rceil$ and $\lceil h/M \rceil$, respectively.*

Proof. Upper bound. First note that in layer 1 all the cells of any mc-area are connected together, and in each of the successive layers at least one pair of cells having a via from the previous layer, and holding the same literal, are connected. Then at most α layers are needed to connect all the cells holding the same literal. In addition all the h literals must be connected to the corresponding inputs leads on the top edge of the lattice. For the k literals already in this edge the connections can take place in layer 1. The remaining $h-k$ literals, in the worst case, may be brought to a further layer $\alpha+1$ by vias and connected to the input leads in $\lceil (h-k)/M \rceil$ layers.

Lower bound. Observe that h external leads must be displayed on the top edge of the lattice, possibly in different layers, and this edge contains M cells. □

In the example of Figure 6 we have $\alpha = 15$ (there are 15 mc-areas in layer 0), $h = 7, k = 3$, and $M = 6$. The proposed layout with $H = 4$ layers is far from approaching the upper bound $15 + \lceil 4/6 \rceil = 16$, while is closer to the lower bound $\lceil 7/6 \rceil = 2$.

3.3 Heuristics for Problem 3

Let us now discuss possible greedy heuristics to solve Problem 3 in a reasonable amount of time.

Independently of the lattices that cannot be solved according to the conditions stated in Proposition 4, other cases may require an exceedingly large number of layers if the "15-slide game" moves are required, as indicated in the proof of the same Proposition 4. We do not accept such moves, treating a lattice requiring them as unsolvable. This limitation shows a minor importance in practice since our algorithms failed very rarely to find a layout for a theoretically solvable lattice, out of a very large number of cases, see Section 4.

The general structure of our heuristics consists of the following two main steps:

1. Starting with layer 0 resulting from a re-arrangement of the lattice done in Problems 1 and 2, build the connecting trees for all mc-areas in layer 1, and connect the literals of the top side to the external leads.
2. While there are trees with the same literals still to be connected between them and/or to the outside:
   (a) place a via on a cell chosen at random of each such a tree;
   (b) add a new layer to receive the vias;
   (c) try to connect together as many vias as possible, associated to the same literal;
   (d) try to connect each group of cells containing a literal to the corresponding external lead, if not already done.

Step 1 can be implemented in a standard way, in a lattice traversal. Note that this initial step is optimal, i.e., no algorithm for the minimization of the number of layers can do better on the first layer.

To implement the second and main step of the heuristics we introduce the concept of *free area* where a connection between cells with the same literal can be displayed, namely:

Definition 5. *In the layers from 2 on, a* free cell *is one not containing a via; a* free area *is a maximal connected subset of free cells; the* boundary cells *of a free area are the ones surrounding it.*

For example layer 2 in Figure 6 (b) contains seven free areas, as shown in Figure 8. Using proper coding and data structures, free areas and their boundary cells can be easily computed through a scanning of the lattice in optimal time $O(N \times M)$. We have:

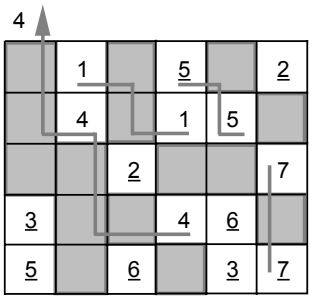

Fig. 8. Layer 2 of Figure 6 (b) with the seven free areas shown in grey.

Proposition 6. *In any given layer from 2 on, cells holding the same literal can be connected together through free cells if and only if they are boundary cells of the same free area.*

Proof. If part. Any subset of cells holding the same literal and bounding the same free area can be connected by a tree of connections laid out inside the area.
Only if part. Cells not bounding any free area are completely surrounded by vias holding different literals and cannot be connected to any other cell in that layer. For cells holding the same literal but not bounding the same area, any connecting path would inevitably meet another via. □

Some considerations are in order. If a set of cells holding the same literal and facing the same free area are connected using free cells inside the area, other boundary cells of the same area can be connected only if the required connections do not cross those already laid out on that area. If two cells holding the same literal bound different areas they can still be connected in the layer through a path that meets only vias with their same literal, if any. If these conditions are not met, the cells must be connected in a next layer.

In the example of Figure 8 four free areas can be used to connect the two 1's in cells (1,2), (2,4); the two 5's in cells (1,4), (2,5); the two 7's in cells (3,6), (5,6); and the two 4's in cells (2,2), (4,4), the latter also connected to the external lead, as already shown in Figure 6 (b). The two 5's in cells (1,4) and (5,1) cannot be connected in this layer since they do not bound the same free area. However, if literal 2 in cell (3,3) were a 5, that connection would have been possible passing through cell (3,3), thus merging connections laid out in two different free areas. The still missing connections among different 5's, 2's, 6's, and 3's, will be done in the free areas of layers 3 and 4 as shown in Figure 6.

On these grounds we execute the crucial points c, d in step 2 of our heuristic first computing all the free areas in the layer, then trying to connect all boundary cells assigned to the same literal and facing the same free area. A relevant feature of this process is that free areas are mutually disjoint, then the searches for connections can be performed in parallel creating a thread for each free area.

The only portion of the layer shared by multiple threads are the boundary cells facing different free areas, that can be managed through lock variables that force the threads to access those cells in mutual exclusion.

The cells to be connected are treated pairwise, however, a subset of more than two cells holding the same literal may bound a free area and as many as possible of them must be connected in tree form. Therefore as soon as two of them are connected, the couple is treated as a single cell identified as the central cell of the connecting path holding the literal of the two, and the process continues looking for other cells to be connected to them.

Clearly a connecting path cuts the free area in two parts and other cells facing this same area may become unreachable from one another. We could solve this issue by recomputing the free areas after a new connection is done, but this approach is computationally very heavy. Therefore, we compute free areas only once in each layer, and then apply a proper *non-exhaustive search algorithm* to limit the search for non-existing connections, still guaranteeing that mutually reachable cells will be connected with high probability.

Let us now briefly discuss the possible implementations of this search within each free area. The main point is considering a boundary cell c_1 that must be connected to a target cell c_2 through a path of distinct free cells in a free area. 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 use heuristics to find solutions of high quality as quickly as possible. We have tested several heuristics proposed in the literature, while for space reasons the results reported in Section 4 are limited to *Best-first* and *Greedy-beam* (see [**?**,**?**]) that select the next cell to visit according to an estimate of the Manhattan distance from the target cell. The first heuristic provides better results but its time complexity $O(4^{N\times M})$ is very high and can be applied only to small size lattices. The time complexity of *Greedy-beam* is instead linear in the lattice size, but produce worse quality results. In fact it may fail in connecting some mutually reachable cells on a given layer, so the final layout may contain a high number of layers. Depending on the lattice size and on the specific application, we can therefore select one of the two heuristics (or, for that matter, other known ones), trading quality of results vs. scalability.

4 Experimental Results

In this section we report the experimental results related to the physical implementation of switching lattices $N \times M$ according to the assumptions 1 to 4 reported in Section 1. The physical implementation of a lattice is a 3-dimensional grid $N \times M \times (H + 1)$, where H is the number of layers needed to route all the connections among switches controlled by the same input literal. The aim of our experimentation is to determine whether the proposed implementation can be considered technologically feasible. To this ends we have considered the lattices obtained applying the Altun-Riedel method to the benchmark functions taken

Table 1. Number of layers for the lattice layout of a subset of standard benchmark circuits, built along the lines indicated in Problems 1,2, and 3.

			Best-first		Greedy Beam	
Bench	lit	N×M	H	Time(s)	H	Time(s)
add6(5)	24	156×156	7	733.21	8	0.43
adr4(1)	16	36×36	6	0.19	8	0.02
alu2(2)	16	10×11	4	0.01	4	0.01
alu2(5)	20	13×14	4	0.01	4	0.01
alu3(0)	8	4×5	3	0.01	3	0.01
alu3(1)	12	7×8	4	0.01	4	0.01
b12(0)	7	6×4	4	0.01	4	0.01
b12(1)	9	5×7	4	0.01	4	0.01
b12(2)	10	6×7	3	0.01	3	0.01
bcc(5)	28	27×9	10	0.02	9	0.01
bcc(7)	29	31×11	10	0.03	11	0.01
bcc(8)	29	31×12	10	0.04	9	0.01
bcc(27)	28	39×19	10	0.13	12	0.02
bcc(43)	28	20×10	6	0.02	6	0.01
bench1(2)	18	45×24	10	0.26	11	0.04
bench1(3)	18	31×16	8	0.03	9	0.01
bench1(5)	18	50×27	9	0.31	9	0.04
bench1(6)	18	35×21	9	0.09	12	0.03
bench1(7)	18	43×21	9	0.12	12	0.02
bench1(8)	18	44×24	9	0.19	10	0.04
bench(6)	10	8×4	5	0.01	5	0.01
br2(4)	18	18×8	6	0.01	6	0.01
br2(5)	19	14×4	6	0.01	6	0.01
br2(6)	19	16×5	6	0.01	6	0.01
clpl(3)	11	6×6	3	0.01	3	0.01
clpl(4)	9	5×5	3	0.01	3	0.01
co14(0)	28	92×14	11	0.29	12	0.04
dc1(4)	7	5×4	–	0.01	–	0.01
dc2(4)	11	10×9	5	0.01	5	0.01
dc2(5)	9	6×6	4	0.01	4	0.01
dk17(1)	10	8×2	6	0.01	6	0.01
dk17(3)	11	11×3	–	0.01	–	0.01
dk17(4)	12	9×3	5	0.01	5	0.01
ex1010(0)	20	91×46	11	8.42	13	0.24
ex4(4)	13	17×6	7	0.01	7	0.01
ex4(5)	27	35×45	7	0.51	8	0.02
ex5(32)	14	4×10	3	0.01	3	0.01
ex5(36)	11	2×8	2	0.01	2	0.01
ex5(38)	13	4×9	3	0.01	3	0.01
ex5(40)	15	6×12	5	0.01	5	0.01
ex5(43)	15	8×14	6	0.01	6	0.01
exam(5)	13	11×6	4	0.01	4	0.01
exam(9)	20	59×30	9	0.75	12	0.04
max128(5)	14	14×17	6	0.01	6	0.01
max128(8)	13	5×10	5	0.01	5	0.01
max128(17)	14	26×25	7	0.06	8	0.01
max1024(5)	20	117×122	10	191.33	14	0.89
mp2d(6)	14	10×6	5	0.01	5	0.01
mp2d(9)	14	6×8	3	0.01	3	0.01
mp2d(10)	10	6×3	4	0.01	4	0.01
sym10(0)	20	130 × 210	9	1571.93	10	2.14
tial(5)	28	181×181	10	1491.45	12	1.19
z4(0)	7	15×15	5	0.01	6	0.01
z4(1)	14	28×28	7	0.08	7	0.01
Z5xp1(2)	14	12×11	4	0.01	4	0.01
Z5xp1(3)	14	18×18	5	0.01	7	0.01
			336	3999,8	367	5,61

from LGSynth93 [**?**], where each output has been treated as a separate Boolean function. For space reasons, in the following Table 1 we report only a significant subset of these functions as representative indicators of our experiments.[1].

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 algorithms have been implemented in C according to the lines indicated in the paper for the solution of Problems 1, 2, and 3.

Table 1 is organized as follows. The first column reports the name and the number of the separate output functions of the benchmark circuit. The following two columns report the number of different literals occurring in the lattice and its dimension $N \times M$. The last four columns report the number H of layers computed with the *Best-first* and the *Greedy-beam* heuristics, together with the corresponding running time. The last row reports the sum of the values of the corresponding column. The cases where the algorithm failed in finding a layout (see Subsection 3.3) are marked with a hyphen.

As expected, we have obtained layouts with a smaller number of layers using the *Best-first* heuristic at the expense of a higher computation time. However we note that the increase in the number of layers computed with the faster *Greedy-beam* heuristic appears quite limited.

Finally, these simulations have shown the effectiveness of the heuristic for Problem 1: indeed, the number of layers computed applying only the heuristics for Problems 2 and 3, without first permuting rows and columns, increases on average of about 35% using *Best-first*, and of about 43% using *Greedy-beam*. Moreover, running both the heuristic for Problem 1 and the algorithm MC-AREAS for Problem 2, we have obtained a considerable reduction of the number of layers, when compared with the results published in [**?**].

5 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 [**?**]. 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 H of layers. Since the problem is computationally intractable we have designed a family of heuristics for finding satisfactory solutions, then applied to a very large set of standard Boolean functions to validate our approach. For space reasons we have presented only the results obtained with the fastest and the slowest heuristics, and only for a subset of the functions analyzed taken as representative of the work done.

The overall design consists of three main phases, studied as Problems 1, 2, and 3. The first two are aimed at rearranging the switch positions and their literal assignment of the starting lattice, in order to place in adjacent cells as

[1] Experimental results on a much larger set of benchmark functions may be requested to the present authors

many switches controlled by the same literal as possible. The third phase then builds the actual connections on the different layers of the chip.

Countless improvements are open. While the NP-hardness of Problem 2 has been proved, for theoretical completeness also the NP-hardness of Problems 1 and 3 has to be proved to fully justify the use of heuristics. Better algorithms could be studied, and tested on larger data samples. The layout for other switching lattices should be considered. The layout rules should possibly be changed, in particular allowing more than one wire traversing a switch area in the higher layers. We are presently working on all these issues.