A Fast Neural–Network Algorithm for Cell Placement

C. AYKANAT,[1] TEVFİK BULTAN[2] AND İSMAİL HARİTAOĞLU[2]

[1]Department of Computer Engineering and Information Science
Bilkent University
TR-06533 Bilkent, Ankara, Turkey

[2]Department of Computer Science
University of Maryland
College Park, MD 20742, USA

Corresponding author : Assoc. Prof. Cevdet Aykanat
Computer Engineering Department
Bilkent University
TR-06533 Bilkent
Ankara, Turkey
Phone : +90 312 266 4133
Fax : +90 312 266 4126
e-mail : aykanat@cs.bilkent.edu.tr

Running Title : Neural Algorithm for Cell Placement

A Fast Neural–Network Algorithm for Cell Placement

C. AYKANAT[1], TEVFİK BULTAN[2] AND İSMAİL HARİTAOĞLU[2]
[1]Bilkent University, [2]University of Maryland

Abstract —*Cell placement is an important phase of current VLSI circuit design styles as standard cell, gate array, and Field Programmable Gate Array (FPGA). Although nondeterministic algorithms such as Simulated Annealing (SA) have been successful in solving this problem, they are known to be slow. In this paper, we propose a neural network algorithm that produces solutions as good as SA in substantially less time. Our algorithm is based on Mean Field Annealing (MFA) technique, which has been successfully applied to various combinatorial optimization problems. We derive a MFA formulation for the cell placement problem that can easily be applied to all VLSI design styles. To demonstrate that the proposed algorithm is applicable to real world problems, we derive a detailed formulation for the FPGA design style, and generate the layouts of several benchmark circuits. The performance of the proposed cell placement algorithm is evaluated in comparison with commercial automated circuit design software Xilinx Automatic Place and Route (APR) which uses SA technique. Performance evaluation is performed using ACM/SIGDA Design Automation benchmark circuits. Experimental results indicate that the proposed MFA algorithm produces comparable results with APR. However, MFA is almost 20 times faster than APR on the average.*

Keywords—VLSI circuit design, Cell placement problem, Field programmable gate array, Mean field annealing, Neural–network algorithms.

1 INTRODUCTION

Cell placement is an important problem arising in various circuit design styles as standard cell, gate array and Field Programming Gate Array (FPGA). Given a circuit description, problem is to find a layout of the circuit while minimizing some cost function. Usually two closely related criteria are used to construct a cost function : minimization of the routing length and minimization of the chip area. In some design styles (e.g., standard cell), minimization of the area is equivalent to minimization of the routing length (Shahookar & Mazumder, 1991) whereas in some others area is fixed (e.g., FPGA). If the area is fixed, minimization of the routing length is necessary for the routability of the circuit using the available routing resources. Minimization of the routing length also minimizes the propagation delays of the circuit, hence increases its speed (Shahookar & Mazumder, 1991).

Although cell placement problem has different characteristics related to the technology used in different design styles, key features of the problem remain the same. This enables us to make a general definition for the cell placement problem which will be valid for all design styles. We will decompose the problem into two phases such that the first phase will be same for all design styles and the second phase will depend on the design style. An instance of the first phase of the cell placement problem consists of a hypergraph $\Omega(C, N)$ representing the circuit to be placed, and a rectangular grid of clusters with P rows and Q columns where the circuit will be placed. Hypergraph $\Omega(C, N)$ consists of a vertex set C representing cells of the circuit, a hyperedge set N representing the nets of the circuit, a cell weight function $w_{cell} : C \rightarrow \mathcal{N}$, and a net weight function $w_{net} : N \rightarrow \mathcal{N}$, where $\mathcal{N}$ represents the set of natural numbers. Question is to partition the vertex set C into $P \times Q$ clusters such that the routing cost is minimized and the weights of the clusters are nearly balanced. Weight of a cluster is the sum of the weights of the cells in that cluster. In general, cell weight function is used to encode the areas of cells, and net weight function is used to increase the importance of some nets which may be crucial for the performance of the circuit. The rectangular grid of clusters is used for estimating the final locations of the cells. We will discuss the computation of routing cost in detail in Section 2.

Fig. 1(a) illustrates an example circuit with 16 cells and 19 nets (Shahookar & Mazumder, 1991). The circuit has 3 input ($I1$, $I2$, $I3$) and 2 output ($O1$, $O2$) pads. Pads may be interpreted as cells which must be mapped to the boundaries of the cluster grid. The example circuit in Fig. 1(a) may be

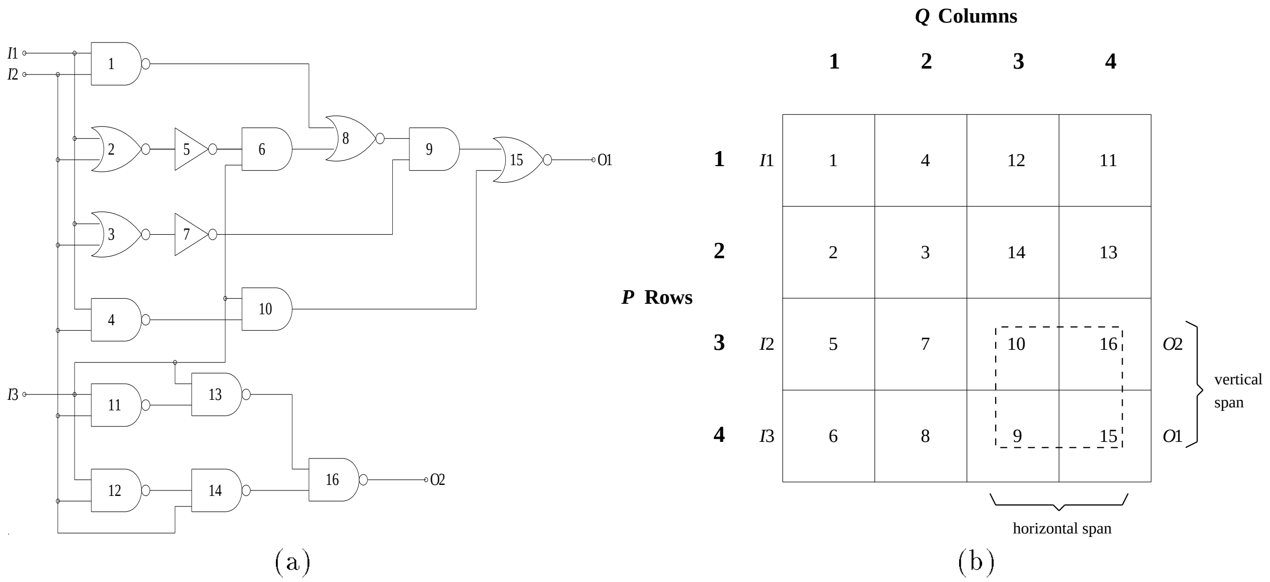


Figure 1: (a) A circuit with 16 cells, 19 nets and 5 pads, (b) a sample placement of the circuit to a 4×4 grid of 16 clusters. Bounding box and horizontal and vertical spans of the net $\{10, 15\}$ are shown in (b).

represented with a hypergraph $\Omega(C, N)$ according to the above definition as

$$
\begin{aligned}
C &= \{1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,I1,I2,I3,O1,O2\} \\
N &= \{\{I1,1,2,3,4\},\{I2,1,2,3,4,11,12\},\{I3,6,10,11,12,13\},\{1,8\},\{3,7\},\{11,13\},\{5,6\},\{8,9\}, \\
&\quad \{9,15\},\ \{13,16\},\ \{O1,15\},\ \{2,5\},\ \{4,10\},\ \{12,14\},\ \{6,8\},\ \{7,9\},\ \{10,15\},\{14,16\},\{O2,16\}\}
\end{aligned}
$$

Unit cell and net weights are assumed in this example. Figure 1(b) shows the placement of this circuit to a 4×4 grid of 16 clusters.

The second phase of the cell placement problem is the mapping of the cells in the clusters to their final locations in the layout. In standard cell design style, cells are used for constructing rows, and in gate array design style, cells are mapped to rows or grid locations according to the type of the gate array used (Sechen, 1988). Some gate arrays consist of modules forming a rectangular grid. For this type of gate arrays the second phase of the problem may be skipped by choosing the number of rows and columns of the cluster grid to be equal to the number of rows and columns of the module grid, respectively. Symmetrical FPGAs consist of logic blocks forming a rectangular grid (Rose et al., 1992, 1993). Hence, the second phase of the problem can be similarly skipped for symmetrical FPGAs. This two phase modeling enables us to develop heuristics for the first phase of the problem which are independent of the design style.

Since cell placement problem is NP-Hard (Lengauer, 1990) finding efficient placement heuristics is an important research issue. In last decade, neurocomputing approaches based on Hopfield model have been successfully applied to various combinatorial optimization problems such as traveling salesman problem (Peterson & Södeberg, 1989; Van den Bout & Miller, 1989; Takahashi, 1997), scheduling problem (Gislén et al., 1992), mapping problem (Bultan & Aykanat, 1992), knapsack problem (Ohlsson et al., 1993; Ohlsson & Pi, 1997), communication routing problem (Häkkinen et al., 1997), graph partitioning problem (Herault & Niez, 1989; Peterson & Södeberg, 1989; Van den Bout & Miller, 1990), graph layout problem (Cimikowski & Shope, 1996), circuit partitioning problem (Yih & Mazumder, 1990; Bultan & Aykanat, 1995). In this paper, we apply the *Mean Field Annealing* (MFA) technique to the cell placement problem. MFA is a new approach for solving combinatorial optimization problems (Peterson & Södeberg 1989; Van den Bout & Miller 1989, 1990; Gislen et al., 1992; Bultan & Aykanat, 1992, 1995; Ohlsson et al., 1993; Hakkinen et al., 1997; Ohlsson & Pi, 1997). MFA combines the collective computation property of *Hopfield neural networks* (Hopfield & Tank, 1985) with the annealing notion of *Simulated Annealing* (SA) (Kirkpatrick et al., 1983). In MFA, discrete variables called *spins* (or *neurons*) are used for encoding configurations of combinatorial optimization problems. An *energy* function written in terms of spins is used for representing the cost function of the problem. Then, using the expected values of these discrete variables, a nondeterministic gradient descent type relaxation scheme is used to find a configuration of the spins which minimizes the energy function associated with them.

In this paper, we propose a MFA–based cell placement algorithm. In order to show the performance of the proposed algorithm on concrete examples we derive our formulations for symmetrical-array FPGA design style. However, the MFA formulation proposed for FPGAs is general enough so that it can easily be applied to the first phase of the cell placement problem in other design styles with minor modifications.

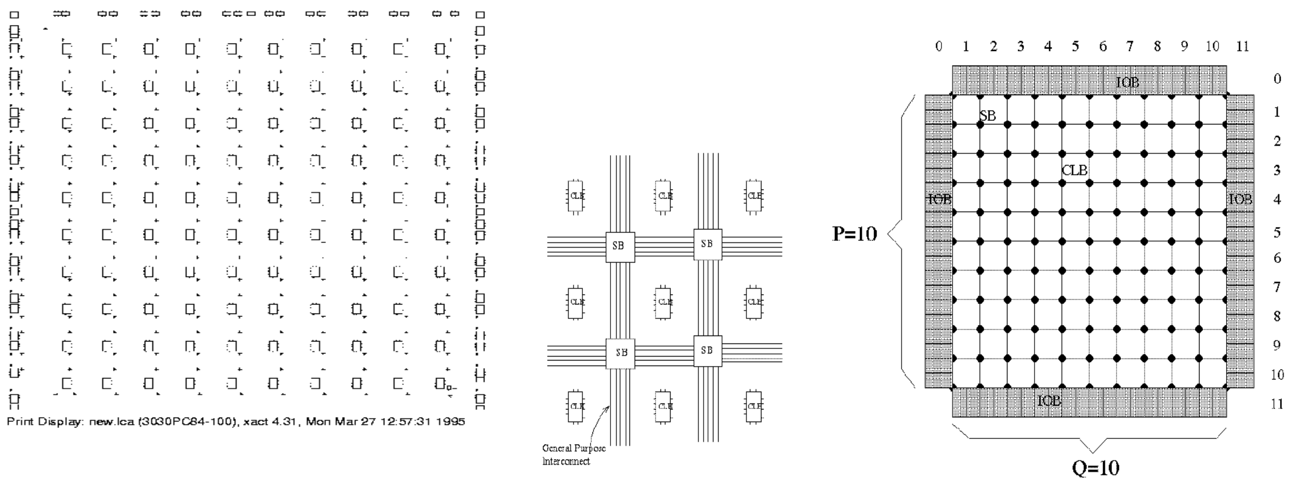
The organization of the paper is as follows. Section 2 discusses the method used for approximating the routing cost of the placement. FPGA design style is briefly summarized in Section 3. Section 4 begins with presentation of the general guidelines for applying MFA technique to combinatorial optimization problems. Then, the proposed formulation and implementation of the MFA algorithm for the cell placement problem following these guidelines are presented. The encoding scheme used in the proposed formulation is discussed in Section 4.1. The proposed energy function formulation and derivation of the mean field theory equations are presented in Sections 4.2 and 4.3, respectively. The parameter selection and cooling schedule are discussed in Section 4.4. Finally, experimental results which evaluate the relative performance of the proposed algorithm are discussed in Section 5.

2 ROUTING COST

Computation of the routing cost is the crucial part of the cell placement problem. In the first phase of the problem, cells are partitioned to $P \times Q$ clusters which form a rectangular grid. Fig. 1(b) shows the partitioning of the circuit in Fig. 1(a) to a 4×4 grid. Initially, we assume that all clusters have the same size, forming a uniform grid as in Fig. 1(b). After the cells are mapped to the clusters, areas of the clusters may be different, resulting with a nonuniform grid. If the clusters are balanced, difference between the uniform grid and the actual nonuniform grid is not significant.

In order to calculate the routing cost we must know the exact locations of the cells in the layout. We assume that each cell is placed to the center of the cluster to which it is mapped. During the placement it is not feasible to calculate the exact routing length for two reasons. First, a feasible placement is not available during the execution of some algorithms, e.g., (Dunlop & Kernighan, 1985), second, the computation of the exact routing cost necessitates the execution of the global and the detailed routing phases which are as hard as the placement phase. Hence, most of the placement heuristics use a method for approximating the routing cost. An efficient and commonly used approximation is the *semi-perimeter* method (Shahookar & Mazumder, 1991; Sherwani, 1993). In this method, the routing cost of a net is approximated by the semi-perimeter length of the smallest bounding rectangle (bounding box) enclosing all the cells connected to that net. Fig. 1(b) shows the bounding box of the net $\{10, 15\}$ with two cells. This method gives a good approximation to the *Steiner tree* which is the most efficient routing scheme (Shahookar & Mazumder, 1991). The shortest way to route a net is to find the minimum length Steiner tree of the cells connected to that net. Steiner trees can also be used as an approximation of the final routing length, but finding the minimum Steiner tree is an NP-Hard problem and its computation may not be feasible. Hence, semi-perimeter method is a good and efficient way of approximating the routing length.

Another way to view the semi-perimeter method is to define the vertical and the horizontal spans for each net (Sechen, 1988). The vertical and the horizontal spans of a net are the lengths of the vertical and the horizontal sides of its bounding rectangle, respectively. Fig. 1(b) shows the vertical and the horizontal spans of the net $\{10, 15\}$. Total routing cost can be computed by adding the vertical and the horizontal spans of all the nets. If vertical and horizontal routings have different costs, then the total routing cost can be approximated by multiplying the vertical and the horizontal spans of the nets by the appropriate unit costs.



(a) (b)

Figure 2: (a) A typical architecture of symmetrical FPGA (Xilinx XC3030 chip) (b) FPGA model used in the proposed MFA formulation

3 FPGA DESIGN STYLE

Field Programmable Gate Arrays (FPGAs) have been widely used in industry in recent years. As they provide cheap and flexible usage, fast manufacturing turnaround time and low prototype cost, many designers prefer to use them in their applications. Several types of FPGAs have been introduced over the last years, which differ from each other by their programming technologies, logic block architectures and routing network architectures (Rose et al., 1992). They can be classified into four main categories: symmetrical-array, row-based, hierarchical and sea-of-gates.

A typical symmetrical-array FPGA consists of a two-dimensional grid called *logic cell array* (LCA) which is interconnected with vertical and horizontal channels as shown in Fig. 2(a). Each point in this two-dimensional grid is called a *configurable logic block* (CLB). A CLB can implement a set of logic functions. In FPGA design style, CLBs are used to provide the functionality of the circuit by mapping the logic gates of the circuit to CLBs. Logic blocks at the boundaries of the LCA are called *input-output blocks* (IOBs). IOBs are used for external connections of the circuit. Routing network, which consists of vertical and horizontal channels placed in between CLBs, makes connections among CLBs and IOBs. *Switch blocks* (SBs) that connect wire segments in horizontal and vertical channels are also a part of the routing network. In commercial FPGAs routing resources are fixed and fairly limited (Xilinx, 1994). For example, there are only five tracks in each routing channel for Xilinx XC3000 series of FPGAs as in Fig. 2(a). The placement problem is especially important in designs using such devices, because fixed routing resources make it difficult to achieve 100% automatic routing.

Automated FPGA layout generation can be divided into four major phases, *partitioning*, *technology mapping*, *placement* and *routing* (Rose et al., 1993). Partitioning is used for very large logic circuits that require multiple FPGA chips. In technology mapping phase, a logic circuit is transformed to an optimized, generic logic input format that consists of CLBs and IOBs. In the placement phase, the circuit that is formed in the technology-mapping phase is assigned to specific CLBs and IOBs in the LCA. This phase of FPGA layout design is equivalent to the cell placement problem discussed earlier. Most commercial automated design tools for FPGAs use SA algorithm in the placement phase. SA technique provides high quality solutions but it is notably slow. In this paper, we propose a fast placement algorithm for symmetrical-array FPGAs that produces layouts which are as good as the ones produced by SA.

4 APPLYING MFA TO THE CELL PLACEMENT PROBLEM

MFA technique merges the collective computation and the annealing properties of Hopfield neural networks (Hopfield & Tank, 1985) and SA (Kirkpatrick et al., 1983), respectively, to obtain a general algorithm for solving combinatorial optimization problems. A combinatorial optimization problem consists of a set of configurations and a cost function. For example, for the cell placement problem the set of configurations corresponds to the set of all possible placements of the input circuit. Sometimes, configurations are also referred as solutions. Cost function assigns a cost to each configuration of the problem. For the cell placement problem, cost of each configuration (i.e., placement) is the routing length of that placement. Optimum solution of a combinatorial optimization problem is the configuration (i.e., solution) which has the minimum (maximum) cost if the problem is a minimization (maximization) problem. Hence, for the cell placement problem the optimum solution is the placement of the circuit which has the minimum routing length.

In the MFA technique (Peterson & Södeberg, 1989; Van den Bout & Miller, 1989, 1990), we use discrete variables called spins (or neurons) to encode the configurations of the problem. A configuration in the spin domain is a valuation of these discrete variables. We define an encoding which is a one-to-one mapping from the configurations of the problem to the configurations of the spins. Then we formulate the cost function of the problem in terms of spins. This function defines the energy of a configuration in the spin domain. MFA algorithm is a search algorithm in the spin domain which looks for the configuration with the minimum energy. To achieve this goal, expected values of the spins are updated iteratively using a nondeterministic gradient descent algorithm. In the following sections, we will describe the application of the MFA technique to the cell placement problem.

4.1 Encoding

The MFA algorithm is derived by analogy to *Ising* and *Potts* models which are used to estimate the state of a system of particles, called spins, in thermal equilibrium (Peterson & Södeberg, 1989; Van den Bout & Miller, 1989, 1990). In Ising model, spins can be in one of the two-states represented by 0 and 1, whereas in Potts model they can be in one of the K states. For the cell placement problem we use the Potts model for encoding the configurations of the problem.

In the K-state Potts model of S spins, the states of spins are represented using S K-dimensional vectors $\mathbf{S}_i = [s_{i1}, \ldots, s_{ik}, \ldots, s_{iK}]^t$, $1 \leq i \leq S$, where "t" denotes the vector transpose operation. The spin vector $\mathbf{S}_i$ is allowed to be equal to one of the principal unit vectors $\mathbf{e}_1, \ldots, \mathbf{e}_k, \ldots, \mathbf{e}_K$, and can not take any other value. Principal unit vector $\mathbf{e}_k$ is defined to be a vector which has all its entries equal to 0 except its kth entry which is equal to 1. Spin $\mathbf{S}_i$ is said to be in state k if it is equal to $\mathbf{e}_k$. Hence, a K-state Potts spin $\mathbf{S}_i$ is composed of K two-state variables $s_{i1}, \ldots, s_{ik}, \ldots, s_{iK}$, where $s_{ik} \in \{0, 1\}$, with the following constraint

$$\sum_{k=1}^{K} s_{ik} = 1, \qquad 1 \leq i \leq S \tag{1}$$

To encode the configuration space of the cell placement problem using these K-state Potts spins, we assign one spin to each cell of the circuit. Each state of a spin corresponds to a location in the layout, i.e., if a spin is in state k this means that the cell associated with that spin is placed to location k.

Two types of cells are considered in FPGA placement, namely L-cells and IO-cells. That is, in the circuit $\Omega(C, N)$, $C = C_L \cup C_{IO}$, where C_L and C_{IO} denote the sets of L-cells and IO-cells, respectively. Here, L-cells correspond to the logic cells of the circuit to be placed to CLBs in the LCA. IO-cells correspond to the input/output pads of the circuit to be placed to the IOBs on the boundaries of the LCA as shown in Fig. 2. Hence, we use two different encoding schemes for the L-cells and IO-cells.

4.1.1 Logic Cell Encoding

In order to encode the configuration space of the placement problem, one Potts spin is assigned to each L-cell $i \in C_L$ of the circuit $\Omega(C, N)$ to be placed. A (K=PQ)-dimensional Potts spin can be used to encode the location of each L-cell, where each state of the Potts spin corresponds to a location in the $P \times Q$ LCA. There will be a total of $|C_L|$ (PQ)-dimensional Potts spins in the system for encoding L-cells. Since, each Potts spin can be in one of the K states at a time, there is a one-to-one mapping between the configuration space of the problem domain and the spin domain. As each Potts spin consists of K two-state variables, a total of $|C_L|PQ$ two-state variables are needed for this encoding. However, a more efficient encoding is to represent the location of each L-cell i with two Potts spins with dimensions P and Q. Spins with dimension P are used to encode the rows, and spins with dimension Q are used to encode the columns of the LCA, respectively. Note that this encoding also constructs a one-to-one mapping between the configuration space of the problem domain and the spin domain. However, it is more efficient since it uses a total of $|C_L|(P+Q)$ two-state variables instead of $|C_L|PQ$ two-state variables of the previous encoding. Spins with dimensions P and Q are called row and column spins and labeled as $\mathbf{S}_i^r = [s_{i1}^r, \ldots, s_{ip}^r, \ldots, s_{iP}^r]^t$ and $\mathbf{S}_i^c = [s_{i1}^c, \ldots, s_{iq}^c, \ldots, s_{iQ}^c]^t$, $i \in C_L$, respectively. If a row (column) spin is in state p (q) we say that the corresponding L-cell is assigned to row p (column q). Hence, $s_{ip}^r = 1$ $(s_{iq}^c = 1)$ means that L-cell i is assigned to row p (column q) of LCA. That is, if $s_{ip}^r = 1$ and $s_{iq}^c = 1$, this means that L-cell i is assigned to the CLB at location pq. Here and hereafter, row and column spins of L-cells will be referred as L-row and L-column spins, respectively.

4.1.2 Input/Output Cell Encoding

In the Xilinx series of FPGAs, there are four IOBs, two on each side, at the boundaries of each row and column of the layout as shown in Fig. 2. Therefore, a $(P \times Q)$-dimensional FPGA has $M = 4(P+Q)$ IOBs. As in CLB encoding, one Potts spin is assigned to each IO-cell $b \in C_{IO}$ of the circuit $\Omega(C, N)$ to be placed. An M-dimensional Potts spin can be used to encode the position of each IO-cell, where each state of the Potts spin corresponds to a unique IOB location in the layout. There will be a total of $|C_{IO}|$ M-dimensional Potts spins in the system for encoding IO-cells. As each Potts spin consists of M two-state variables a total of $|C_{IO}|M$ two-state variables are needed for this encoding. Spins with dimension M are called IO spins and labeled as $\mathbf{S}_b^{io} = [s_{b1}^{io}, \ldots, s_{bm}^{io}, \ldots, s_{bM}^{io}]^t$, for $b \in C_{IO}$. If an IO spin is in state m we say that the corresponding IO-cell is assigned to IOB at location m in the layout. In order to simplify the encoding, we extend the FPGA model by adding two new boundary columns and two new boundary rows as shown in Fig. 2(b). The rows 0 and $P+1$ and columns 0 and $Q+1$ are allocated to IOBs. An L-cell can be assigned to any internal row p, $1 \leq p \leq P$, and any internal column q, $1 \leq q \leq Q$. An IO-cell can only be assigned to boundary rows 0 and $P+1$ or boundary columns 0 and

$Q+1$. IOB locations are numbered in clockwise direction starting from the upper left corner of the layout from 1 to $4P+4Q$. We define two new functions $row(m)$ and $col(m)$ to show the IOB location m in terms of its row and column locations. Using this numbering scheme, $s^{io}_{bm}=1$ means that IO-cell b is assigned to IOB at location m, that is IO-cell b is assigned to one of the two IOBs at the location pq of LCA where $p = row(m)$ and $q = col(m)$. Note that either $p \in \{0, P+1\}$ or $q \in \{0, Q+1\}$.

4.2 Energy Function Formulation

In the MFA algorithm, the aim is to find the spin values minimizing the energy function of the system. In order to achieve this goal, the average (expected) values of the spin vectors $\mathbf{S}^r_i$, $\mathbf{S}^c_i$ and $\mathbf{S}^{io}_b$ are iteratively updated using a nondeterministic gradient descent algorithm. Iterations continue until the system stabilizes at some fixed point. We define

$$\begin{aligned}
\mathbf{V}^r_i = \left[v^r_{i1}, \ldots, v^r_{ip}, \ldots, v^r_{iP}\right]^t &= \langle \mathbf{S}^r_i \rangle = \left[\langle s^r_{i1}\rangle, \ldots, \langle s^r_{ip}\rangle, \ldots, \langle s^r_{iP}\rangle\right]^t \\
\mathbf{V}^c_i = \left[v^c_{i1}, \ldots, v^c_{iq}, \ldots, v^c_{iQ}\right]^t &= \langle \mathbf{S}^c_i \rangle = \left[\langle s^c_{i1}\rangle, \ldots, \langle s^c_{iq}\rangle, \ldots, \langle s^c_{iQ}\rangle\right]^t \\
\mathbf{V}^{io}_b = \left[v^{io}_{b1}, \ldots, v^{io}_{bm}, \ldots, v^{io}_{bM}\right]^t &= \langle \mathbf{S}^{io}_b \rangle = \left[\langle s^{io}_{b1}\rangle, \ldots, \langle s^{io}_{bm}\rangle, \ldots, \langle s^{io}_{bM}\rangle\right]^t
\end{aligned}$$

where $\mathbf{V}^r_i$, $\mathbf{V}^c_i$ and $\mathbf{V}^{io}_b$ denote the expected values of the spins $\mathbf{S}^r_i$, $\mathbf{S}^c_i$ and $\mathbf{S}^{io}_b$, respectively. Note that $s^r_{ip}, s^c_{iq}, s^{io}_{bm} \in \{0, 1\}$, i.e., s^r_{ip}, s^c_{iq} and s^{io}_{bm} are discrete variables taking only two values 0 and 1, whereas $v^r_{ip}, v^c_{iq}, v^{io}_{bm} \in [0, 1]$, i.e., v^r_{ip}, v^c_{iq} and v^{io}_{bm} are continuous variables taking any real value between 0 and 1. As the system is a Potts glass we have the following constraints similar to (1)

$$\sum_{p=1}^{P} v^r_{ip} = 1; \qquad \sum_{q=1}^{Q} v^c_{iq} = 1; \qquad \sum_{m=1}^{M} v^{io}_{bm} = 1; \tag{2}$$

for all $i \in C_L$ and $b \in C_{IO}$. These constraints guarantee that each Potts spin $\mathbf{S}^r_i$, $\mathbf{S}^c_i$ and $\mathbf{S}^{io}_b$ is in one of the P, Q and M states at a time, respectively, and each L-cell is assigned to only one row (column) and each IO-cell is assigned to only one IOB for our encoding of the placement problem. Note that $v_{ip}=\langle s_{ip}\rangle$, i.e., v^r_{ip} is the expected value of s^r_{ip}. Hence,

$$v^r_{ip} = \mathcal{P}\{s^r_{ip} = 0\} \times 0 + \mathcal{P}\{s^r_{ip} = 1\} \times 1 = \mathcal{P}\{s^r_{ip} = 1\} = \mathcal{P}\{L\text{-cell } i \text{ is in row } p\}.$$

Similarly,

$$v^c_{iq} = \mathcal{P}\{L\text{-cell } i \text{ is in column } q\}; \qquad v^{io}_{bm} = \mathcal{P}\{IO\text{-cell } b \text{ is in IOB } m\};$$

That is, v^r_{ip} is the probability of finding L-cell i in one of the Q CLB locations at row p. Similarly, v^c_{iq} is the probability of finding L-cell i in one of the P CLB locations at column q, and v^{io}_{bm} is the probability of finding IO-cell b at IOB location m. Note that v^{io}_{bm} also denotes the probability of finding IO-cell b in one of the two IOB slots at location pq of the LCA, where $p = row(m)$ and $q = col(m)$. If $v^r_{ip} = 1$ and $v^c_{iq} = 1$, then corresponding configuration is $\mathbf{S}^r_i = \mathbf{e}_p$ and $\mathbf{S}^c_i = \mathbf{e}_q$, respectively, which means that the L-cell i is placed to the CLB at location pq of the LCA. Similarly, if $v^{io}_{bm} = 1$ then the corresponding configuration is $\mathbf{S}^{io}_b = \mathbf{e}_m$ which means that the IO-cell b is assigned to the IOB at location m. The latter case also means that the IO-cell b is assigned to one of the two IOBs at location pq of the LCA.

The encoding scheme defined above ensures that L-cells are assigned to the CLBs in the internal rows and columns of the LCA. Similarly, it ensures that IO-cells are assigned to the IOBs in the boundary rows and columns of the LCA. However, for the sake of both simplicity of presentation and the efficiency of implementation we maintain $P+2$ and $Q+2$ dimensional vectors for row and column spins, respectively, for each L-cell $i \in C_L$;

$$\mathbf{V}^r_i = \left[v^r_{i0}, v^r_{i1}, \ldots, v^r_{ip}, \ldots, v^r_{iP}, v^r_{i,P+1}\right]^t \qquad \mathbf{V}^c_i = \left[v^c_{i0}, v^c_{i1}, \ldots, v^c_{iq}, \ldots, v^c_{iQ}, v^c_{i,Q+1}\right]^t \tag{3}$$

Note that v^r_{i0}, $v^r_{i,P+1}$, v^c_{i0} and $v^c_{i,Q+1}$ are initialized to and remain as all 0's since L-cells cannot be assigned to the boundary rows and columns. Here, v^r_{ip} for $1 \le p \le P$ and v^c_{iq} for $1 \le q \le Q$ correspond to the actual spin variables iteratively updated during the MFA algorithm. For similar reasons, we maintain and update $P+2$ and $Q+2$ dimensional row and column vectors for each IO-cell $b \in C_{IO}$;

$$\mathbf{V}^r_b = \left[v^r_{b0}, v^r_{b1}, \ldots, v^r_{bp}, \ldots, v^r_{bP}, v^r_{b,P+1}\right]^t \qquad \mathbf{V}^c_b = \left[v^c_{b0}, v^c_{b1}, \ldots, v^c_{bq}, \ldots, v^c_{bQ}, v^c_{b,Q+1}\right]^t \tag{4}$$

where, v^r_{bp} (v^c_{bq}) corresponds to the probability of finding IO-cell b in an IOB location at row p (column q) of the LCA. Note that, there are $2P$ ($2Q$) IOBs in the boundary rows (columns) 0 and $P+1$ ($Q+1$). However, there are only 4 IOBs in each internal row p (column q) for $1 \le p \le P$ ($1 \le q \le Q$). The row vector $\mathbf{V}^r_b$ can easily be computed using actual IO-spin values as follows:

$$v^r_{b0} = \sum_{m=1}^{2P} v^{io}_{bm}; \qquad v^r_{b,P+1} = \sum_{m=2P+2Q+1}^{4P+2Q} v^{io}_{bm}; \tag{5}$$

$$v^r_{bp} = v^{io}_{bk} + v^{io}_{b,k+1} + v^{io}_{b\ell} + v^{io}_{b,\ell+1} \qquad \text{for} \qquad 1 \le p \le P \tag{6}$$

where, $k = 2P+(2p-1)$ and $\ell = M-(2p-1)$. The column vector $\mathbf{V}_b^c$ can be similarly computed as

$$v_{b0}^c = \sum_{m=4P+2Q+1}^{M} v_{bm}^{io}; \qquad v_{b,Q+1}^c = \sum_{m=2P+1}^{2P+2Q} v_{bm}^{io}; \tag{7}$$

$$v_{bq}^c = v_{bk}^{io} + v_{b,k+1}^{io} + v_{b\ell}^{io} + v_{b,\ell+1}^{io} \qquad \text{for} \qquad 1 \leq q \leq Q \tag{8}$$

where, $k = (2q-1)$ and $\ell = (M-2Q)-(2q-1)$. This representation scheme is chosen for IO-cells since IO-cells assigned to the IOBs in the same row and column of the LCA incur the same vertical and horizontal routing cost, respectively.

As mentioned earlier, energy function corresponds to formulation of the cost function of the cell placement problem in terms of spins. Since the MFA algorithm iterates on the expected values of the spins we formulate the expected value of the energy function. The gradient of the expected value of the energy function is used in the MFA algorithm to compute the direction of maximum energy decrease, and the expected values of the spins are updated accordingly. We derive the expected value of the energy function for the cell placement problem as follows. Using the expected values of the spin variables defined earlier we can compute the following probabilities

$$\begin{aligned}
\mathcal{P}\{\text{no cell of net } n \text{ is in row } p\} &= \prod_{i \in n} \mathcal{P}\{\text{cell } i \text{ is not in row } p\} \\
&= \prod_{i \in n} (1 - v_{ip}^r) \\
\mathcal{P}\{\text{one or more cells of net } n \text{ is in row } p\} &= 1 - \mathcal{P}\{\text{no cell of net } n \text{ is in row } p\} \\
&= 1 - \prod_{i \in n} (1 - v_{ip}^r)
\end{aligned}$$

where, $i \in n$ denotes the subset of cells connected to the net n. These values may be computed for columns of the LCA similarly. We define π_{np}^r as the probability of the event that no cell of net n is in row p and π_{nq}^c as the probability of the event that no cell of net n is in column q, i.e.,

$$\pi_{np}^r = \prod_{i \in n} (1 - v_{ip}^r); \qquad \pi_{nq}^c = \prod_{i \in n} (1 - v_{iq}^c); \tag{9}$$

Note that, if $i \in n$ is an L-cell then v_{ip}^r and v_{iq}^c correspond to the actual L-row and L-column spin variables, for $1 \leq p \leq P$ and $1 \leq q \leq Q$, respectively, and to dummy 0 variables for $p = 0, P+1$ and $q = 0, Q+1$, respectively, in our representation scheme. If otherwise $i \in n$ is an IO-cell, then these values correspond to the respective entries of the row and column vectors maintained for IO-spins as discussed earlier. The vertical and horizontal routing costs of a net n are defined as $w_v \times w_n \times$ (vertical span of net n) and $w_h \times w_n \times$ (horizontal span of net n), respectively. Here, w_v and w_h are the unit vertical and horizontal routing costs between two successive cell (cluster) locations on the same column and row, respectively. In FPGA design style, we use $w_v = w_h = 1$. Formulation of the vertical routing cost of net n as an energy term E_{vn} using these definitions is

$$\begin{aligned}
E_{vn} &= w_v w_n \sum_{k=0}^{P} \sum_{\ell=k+1}^{P+1} (\ell - k)\mathcal{P}\{\text{vertical span of net } n \text{ is between row } k \text{ and } \ell\ \} \\
&= w_v w_n \sum_{k=0}^{P} \sum_{\ell=k+1}^{P+1} (\ell - k)\mathcal{P}\{\text{net } n \text{ is in row } k\}\mathcal{P}\{\text{net } n \text{ is in row } \ell\} \\
&\qquad \times \mathcal{P}\{\text{net } n \text{ is not in first } k-1 \text{ rows}\}\mathcal{P}\{\text{net } n \text{ is not in last } P-(\ell+2) \text{ rows}\} \\
&= w_v w_n \sum_{k=0}^{P} \sum_{\ell=k+1}^{P+1} (\ell - k)\mathcal{P}\{\text{net } n \text{ is in row } k\}\mathcal{P}\{\text{net } n \text{ is in row } \ell\} \\
&\qquad \times \prod_{s=0}^{k-1} \mathcal{P}\{\text{net } n \text{ is not in row } s\} \prod_{t=\ell+1}^{P+1} \mathcal{P}\{\text{net } n \text{ is not in row } t\} \\
&= w_v w_n \sum_{k=0}^{P} \sum_{\ell=k+1}^{P+1} (\ell - k)(1 - \pi^r_{nk})(1 - \pi^r_{n\ell}) \prod_{s=0}^{k-1} \pi^r_{ns} \prod_{t=\ell+1}^{P+1} \pi^r_{nt} \qquad (10)
\end{aligned}$$

where, net n is in row k if and only if one or more cells of net n is in row k, otherwise net n is not in row k. Similarly, energy formulation for the horizontal routing cost of net n is

$$E_{hn} = w_h w_n \sum_{k=0}^{Q} \sum_{\ell=k+1}^{Q+1} (\ell - k)(1 - \pi^c_{nk})(1 - \pi^c_{n\ell}) \prod_{s=0}^{k-1} \pi^c_{ns} \prod_{t=\ell+1}^{Q+1} \pi^c_{nt} \qquad (11)$$

Total vertical and horizontal routing cost terms of the energy function (i.e., E_v and E_h) can be derived using the formulation given in (10) and (11) as

$$E_v = \sum_{n \in N} E_{vn} \qquad\qquad E_h = \sum_{n \in N} E_{hn} \qquad (12)$$

If we use the routing cost as the only factor in the cost function, the optimum solution is mapping all cells of the circuit to one location in the layout. This placement will reduce the routing cost to zero but obviously it is not feasible. Hence, we need a term in the cost function which will penalize the placements that put more than one cell to the same location. We call this term the overlap cost.

We formulate the energy term corresponding to the overlap cost for CLBs and IOBs as

$$\begin{aligned}
E_o^{clb} &= \frac{1}{2} \sum_{i \in C_L} \sum_{j \in C_L, j \neq i} w_i w_j \mathcal{P}\{L\text{-cells } i \text{ and } j \text{ are in the same CLB location}\} \\
&= \frac{1}{2} \sum_{i \in C_L} \sum_{j \in C_L, j \neq i} w_i w_j \sum_{p=1}^{P} \sum_{q=1}^{Q} \mathcal{P}\{L\text{-cell } i \text{ in CLB location } pq\} \mathcal{P}\{L\text{-cell } j \text{ in CLB location } pq\} \\
&= \frac{1}{2} \sum_{i \in C_L} \sum_{j \in C_L, j \neq i} w_i w_j \sum_{p=1}^{P} \sum_{q=1}^{Q} v_{ip}^{r} v_{iq}^{c} v_{jp}^{r} v_{jq}^{c} \qquad (13) \\
E_o^{iob} &= \frac{1}{2} \sum_{a \in C_{IO}} \sum_{b \in C_{IO}, b \neq a} w_a w_b \sum_{m=1}^{M} \mathcal{P}\{IO\text{-cells } a \text{ and } b \text{ are in the same IOB location } m\} \\
&= \frac{1}{2} \sum_{a \in C_{IO}} \sum_{b \in C_{IO}, b \neq a} w_a w_b \sum_{m=1}^{M} v_{am}^{io} v_{bm}^{io} \qquad (14)
\end{aligned}$$

Note that, this overlap cost term becomes equal to the sum of the inner products of the weights of the cells at each cell (cluster) location when the system converges. In general placement, this term is minimized when weights of all the clusters are equal. If there is an imbalance among the cluster weights, this term increases with the square of the amount of imbalance, penalizing imbalanced clusterings. In FPGA placement, all cell weights are equal to 1 and only one L-cell and one IO-cell can be placed to one CLB and one IOB location, respectively. Furthermore, we have $|C_L| \leq (P \times Q)$, $|C_{IO}| \leq M$. Hence, the overlap cost is minimized when either a single or no L-cell (IO-cell) is located to each CLB (IOB) location. If there is an overlap in a location, the overlap cost term increases with the square of the amount of overlap, penalizing the overlapped locations. Total energy term can be defined in terms of routing cost terms and the overlap cost term as

$$E = E_v + E_h + \beta \times E_o, \quad \text{where} \quad E_o = E_o^{clb} + E_o^{iob} \qquad (15)$$

Parameter β is used to balance the two conflicting objectives of the energy function: minimizing the routing cost and the overlap cost. Note that allocating all cells to the same location minimizes the routing cost while maximizing the overlap cost. Minimization of the above energy function corresponds to distributing the cells of the circuit to the locations in such a way that the semi-perimeter and overlap costs are minimized.

The derivation of the gradient of the energy function using the formulation discussed above results in substantially complex expressions. Hence, we simplify the total energy function given in (15) in order to get more suitable expressions for the gradient. Simplification of the E_v and E_h terms given in (12) is as follows. A close examination of (10) and (11) reveals the symmetry between E_{vn} and E_{hn} terms. In fact, expressions for E_{vn} and E_{hn} can be obtained from each other by interchanging "r" with "c", "P" with "Q" and "w_v" with "w_h". Hence, algebraic simplifications will only be discussed for the E_{vn} term. Similar steps can be followed for the E_{hn} term. We introduce the following notation for the sake of simplification of the routing cost terms.

$$F^r_{nk} = \prod_{s=0}^{k} \pi^r_{ns}, \quad L^r_{nk} = \prod_{s=k}^{P+1} \pi^r_{ns}, \quad F^c_{nk} = \prod_{s=0}^{k} \pi^c_{ns}, \quad L^c_{nk} = \prod_{s=k}^{Q+1} \pi^c_{ns}. \tag{16}$$

Here, F^r_{nk} and L^r_{nk} denote the probabilities that net n has no cells in the first $k+1$ rows (rows $0, 1, 2, \ldots, k$) and the last $P-k+2$ rows (rows $k, k+1, \ldots, P, P+1$), respectively. Similarly, F^c_{nk} and L^c_{nk} denote the probabilities that net n has no cells in the first $k+1$ and the last $Q-k+2$ columns, respectively. Using this notation, E_{vn} in (10) can be rewritten as

$$E_{vn} = w_v w_n \sum_{k=1}^{P+1} (1-\pi^r_{nk}) F^r_{n,k-1} \sum_{\ell=k+1}^{P+1} (\ell-k)(1-\pi^r_{n\ell}) L^r_{n,\ell+1} \tag{17}$$

Since,

$$\begin{aligned} (1-\pi^r_{nk}) \prod_{s=0}^{k-1} \pi^r_{ns} &= \prod_{s=0}^{k-1} \pi^r_{ns} - \prod_{s=0}^{k} \pi^r_{ns} = F^r_{n,k-1} - F^r_{nk} && (18) \\ (1-\pi^r_{n\ell}) \prod_{t=\ell+1}^{P} \pi^r_{nt} &= \prod_{t=\ell+1}^{P} \pi^r_{nt} - \prod_{t=\ell}^{P} \pi^r_{nt} = L^r_{n,\ell+1} - L^r_{n\ell} && (19) \end{aligned}$$

(17) becomes

$$E_{vn} = w_v w_n \sum_{k=1}^{P} (F^r_{n,k-1} - F^r_{nk}) \sum_{\ell=k+1}^{P+1} (\ell-k)(L^r_{n,\ell+1} - L^r_{n\ell}) \tag{20}$$

The innermost summation in (20) telescopes to

$$\sum_{\ell=k+1}^{P+1} (\ell-k)(L^r_{n,\ell+1} - L^r_{n\ell}) = \sum_{\ell=k+1}^{P+1} (1 - L^r_{n\ell}) \tag{21}$$

since $L_{n,P+2} = 1$. Substituting (21) into (20) we obtain

$$E_{vn} = w_v w_n \sum_{k=1}^{P} (F^r_{n,k-1} - F^r_{nk}) \sum_{\ell=k+1}^{P+1} (1 - L^r_{n\ell}) \tag{22}$$

After computing the telescoping outer sum in (22) and thru some algebraic manipulations, expression for E_{vn} simplifies to

$$E_{vn} = w_v w_n \sum_{k=0}^{P} (1 - F^r_{nk})(1 - L^r_{n,k+1}) \tag{23}$$

Similarly, the expression for E_{hn} in (11) simplifies to

$$E_{hn} = w_h w_n \sum_{k=0}^{Q} (1 - F^c_{nk})(1 - L^c_{n,k+1}) \tag{24}$$

Note that (23) and (24) compute the vertical and horizontal routing cost of net n, respectively, in an incremental manner. Hence, total energy function in (15) can be rewritten as

$$\begin{aligned} E \;=\; & w_v \sum_{n \in N} w_n \sum_{k=0}^{P} (1 - F^r_{nk})(1 - L^r_{n,k+1}) + w_h \sum_{n \in N} w_n \sum_{k=0}^{Q} (1 - F^c_{nk})(1 - L^c_{n,k+1}) \\ & + \frac{\beta}{2} \sum_{i \in C_L} \sum_{j \in C_L, j \neq i} w_i w_j \sum_{p=1}^{P} \sum_{q=1}^{Q} v^r_{ip} v^c_{iq} v^r_{jp} v^c_{jq} + \frac{\beta}{2} \sum_{a \in C_{IO}} \sum_{b \in C_{IO}, b \neq a} w_a w_b \sum_{m=1}^{M} v^{io}_{am} v^{io}_{bm} \end{aligned} \tag{25}$$

4.3 Derivation of the Mean Field Theory Equations

The expected values $\mathbf{V}^r_i$, $\mathbf{V}^c_j$ and $\mathbf{V}^{io}_b$ of each L-row, L-column and IO spins $\mathbf{S}^r_i$, $\mathbf{S}^c_j$ and $\mathbf{S}^{io}_b$ are iteratively updated using the Boltzmann distribution as

$$(a) \quad v^r_{ip} = \frac{e^{\phi^r_{ip}/T^r}}{\sum_{k=1}^{P} e^{\phi^r_{ik}/T^r}}, \qquad (b) \quad v^c_{jq} = \frac{e^{\phi^c_{jq}/T^c}}{\sum_{k=1}^{Q} e^{\phi^c_{jk}/T^c}}, \qquad (c) \quad v^{io}_{bm} = \frac{e^{\phi^{io}_{bm}/T^{io}}}{\sum_{k=1}^{M} e^{\phi^{io}_{bk}/T^{io}}}, \tag{26}$$

for $p = 1, 2, \ldots, P$, $q = 1, 2, \ldots, Q$ and $m = 1, 2, \ldots, M$. Here, ϕ^r_{ip}, ϕ^c_{ip} and ϕ^{io}_{bm} denote the elements of the mean field vectors corresponding to the variables v^r_i, v^c_i and v^{io}_{bm}, respectively. In (26), T^r, T^c and T^{io} denote the temperature parameters used for annealing the L-row, L-column, and IO spins, respectively. Recall that the number of states of the L-row, L-column and IO spins are different (P, Q and M, respectively) in the proposed encoding. As the convergence time and the temperature parameter of the system depend on the number of states of the spins, we interpret the L-row, L-column and IO spins as different systems. Note that (26.a), (26.b) and (26.c) enforce each L-row, L-column and IO spins $\mathbf{S}^r_i$, $\mathbf{S}^c_j$ and $\mathbf{S}^{io}_b$ to be in one of the P, Q and M states, respectively, when they converge. In the proposed MFA formulation, L-row, L-column, and IO spins are updated in an alternative manner, i.e., each L-row spin update is followed by an L-column spin update which is followed by an IO-spin update.

In the proposed formulation, L-row, L-column and IO mean field vectors $\mathbf{\Phi}_i^r$, $\mathbf{\Phi}_j^c$ and $\mathbf{\Phi}_b^{io}$ are to be computed in L-row, L-column and IO iterations, respectively. Each element ϕ_{ip}^r, ϕ_{jq}^c and ϕ_{bm}^{io} of the L-row, L-column and IO mean field vectors $\Phi_i^r = [\phi_{i1}^r, \ldots, \phi_{ip}^r, \ldots, \phi_{iP}^r]^t$, $\Phi_j^c = [\phi_{j1}^c, \ldots, \phi_{jq}^c, \ldots, \phi_{jQ}^c]^t$ and $\Phi_b^{io} = [\phi_{b1}^{io}, \ldots, \phi_{bm}^{io}, \ldots, \phi_{bM}^{io}]^t$ experienced by L-row, L-column and IO Potts spins denote the decrease in the energy function by assigning $\mathbf{S}_i^r$ to $\mathbf{e}_p$, $\mathbf{S}_j^c$ to $\mathbf{e}_q$ and $\mathbf{S}_b^{io}$ to $\mathbf{e}_m$, respectively. Hence, $-\phi_{ip}^r$, $-\phi_{jq}^c$ and $-\phi_{bm}^{io}$ may be interpreted as the decrease in the overall solution quality by placing L-cell i to row p, L-cell j to column q, and IO-cell b to the IOB location m, respectively. Then, in (26.a), (26.b) and (26.c), v_{ip}^r, v_{jq}^c and v_{bm}^{io} are updated such that the probabilities of placing L-cell i to row p, L-cell j to column q and IO-cell b to the IOB location m increase with increasing mean field values ϕ_{ip}^r, ϕ_{jq}^c and ϕ_{bm}^{io}, respectively. Using the simplified expression for the proposed energy function in (25) we derive

$$\begin{aligned} \phi_{ip}^r &= E(\mathbf{V}^r, \mathbf{V}^c, \mathbf{V}^{io})|_{\mathbf{V}_i^r=\mathbf{0}} - E(\mathbf{V}^r, \mathbf{V}^c, \mathbf{V}^{io})|_{\mathbf{V}_i^r=\mathbf{e}_p} \\ &= -w_v \sum_{n\in N_i} w_n Z_{np}^{ir} - \beta^r w_i \sum_{j\in C_L, j\neq i} w_j v_{jp}^r \sum_{q=1}^{Q} v_{iq}^c v_{jq}^c \quad \text{where} \qquad (27) \\ Z_{np}^{ir} &= \sum_{k=1}^{p} L_{nk}^{ir}(1 - F_{n,k-1}^{ir}) + \sum_{k=p}^{P} F_{nk}^{ir}(1 - L_{n,k+1}^{ir}), \qquad (28) \end{aligned}$$

$$\begin{aligned} \phi_{jq}^c &= E(\mathbf{V}^r, \mathbf{V}^c, \mathbf{V}^{io})|_{\mathbf{V}_j^c=\mathbf{0}} - E(\mathbf{V}^r, \mathbf{V}^c, \mathbf{V}^{io})|_{\mathbf{V}_j^c=\mathbf{e}_q} \\ &= -w_h \sum_{n\in N_j} w_n Z_{nq}^{jc} - \beta^c w_j \sum_{i\in C_L, i\neq j} w_i v_{iq}^c \sum_{p=1}^{P} v_{jp}^r v_{ip}^r, \quad \text{where} \qquad (29) \\ Z_{nq}^{jc} &= \sum_{k=1}^{q} L_{nk}^{jc}(1 - F_{n,k-1}^{jc}) + \sum_{k=q}^{Q} F_{nk}^{jc}(1 - L_{n,k+1}^{jc}), \quad \text{and} \qquad (30) \end{aligned}$$

$$\begin{aligned} \phi_{bm}^{io} &= E(\mathbf{V}^r, \mathbf{V}^c, \mathbf{V}^{io})|_{\mathbf{V}_b^{io}=\mathbf{0}} - E(\mathbf{V}^r, \mathbf{V}^c, \mathbf{V}^{io})|_{\mathbf{V}_b^{io}=\mathbf{e}_m} \\ &= -w_v \sum_{n\in N_b} w_n Z_{np}^{br} - w_h \sum_{n\in N_b} w_n Z_{nq}^{bc} - \beta^{io} w_b \sum_{a\in C_{IO}, a\neq b} w_a v_{am}^{io} \qquad (31) \end{aligned}$$

Here, N_i denotes the set of nets connected to cell i, and $p = row(m)$, $q = col(m)$. Note that we use different balance parameters β^r, β^c and β^{io} in (27), (29) and (31) since L-row, L-column and IO spins are treated as different systems. Here, F^{ir}_{nk}, L^{ir}_{nk}, F^{jc}_{nk} and L^{jc}_{nk} are defined as

$$F^{ir}_{nk} = \prod_{s=0}^{k} \pi^{ir}_{ns}, \quad L^{ir}_{nk} = \prod_{s=k}^{P+1} \pi^{ir}_{ns}, \quad F^{jc}_{nk} = \prod_{s=0}^{k} \pi^{jc}_{ns}, \quad L^{jc}_{nk} = \prod_{s=k}^{Q+1} \pi^{jc}_{ns} \quad \text{where} \tag{32}$$

$$\pi^{ir}_{ns} = \prod_{j \in n, j \neq i} (1 - v^r_{js}), \quad \pi^{jc}_{ns} = \prod_{i \in n, i \neq j} (1 - v^c_{is}) \tag{33}$$

In (28), Z^{ir}_{np} computes the increase in the vertical span of net n by assigning its L-cell i to row p (i.e., setting $\mathbf{V}^r_i$ to $\mathbf{e}_p$) in an incremental manner. Similarly, in (30), Z^{jc}_{nq} computes the increase in the horizontal span of a net n by assigning its L-cell j to column q (i.e., setting $\mathbf{V}^c_j$ to $\mathbf{e}_q$) in an incremental manner. In (31), Z^{br}_{np} and Z^{bc}_{nq} correspond to the increase in the vertical and horizontal spans of net n, respectively, by assigning its IO-cell b to one of the two IOBs at location pq (i.e., setting $\mathbf{V}^{io}_b$ to $\mathbf{e}_m$) where $p = row(m)$ and $q = col(m)$. The expressions for Z^{br}_{np} and Z^{bc}_{nq} can be obtained by replacing "i" and "j" with "b" in (27) and (29), respectively. Note that row (column) assignment of a cell does not affect the horizontal (vertical) spans of nets connected to that cell. The last summation terms in (27), (29) and (31) represent the increase in the overlap cost term by assigning L-cell i to row p, L-cell j to column q and IO-cell b to IOB location m, respectively.

Figure 3 illustrates the pseudo-code for the MFA algorithm proposed for the placement problem. At step 1, temperature parameters T^r, T^c and T^{io} are initialized to sufficiently high temperatures for the annealing of L-row, L-column and IO spins, respectively. At step 2, an initial high temperature spin average is assigned to each Potts spin. In general, each spin variable is initialized to $1/K$ plus a small disturbance term which varies between $-0.1/K$ and $+0.1/K$. Here, $K = P$, $K = Q$ and $K = M$ for L-row, L-column and IO spin variables, respectively. Note that v^r_{ip}, v^c_{jq} and v^{io}_{bm} spin variables updated according to (26) will approach to $1/P$, $1/Q$ and $1/M$ with $T^r \to \infty$, $T^c \to \infty$ and $T^{io} \to \infty$, respectively. Then, outermost *while-loop* (step 3) iterates while T^r, T^c and T^{io} are all in the cooling range. At each iteration of the innermost *repeat-loop* (step 3.1.2), the mean field vector effecting on a randomly selected L-row spin is computed (step 3.1.2.1), then the respective L-row spin average vector is updated (step 3.1.2.2). Similar operations are performed for randomly selected L-column and IO spins as shown in steps 3.1.2.3–3.1.2.6. These spin update operations are repeated for random sequences of L-row, L-column and IO spins as shown in the *repeat-loop* (step 3.1.2). The system is observed at the end of each *repeat-loop* in order to detect the convergence to an equilibrium state at the current temperature. If the average energy decrease due to the spin updates performed in

1. Compute the initial temperatures T_0^r, T_0^c, T_0^{io} and set $T^r = T_0^r$, $T^c = T_0^c$, $T^{io} = T_0^{io}$.
2. Initialize the spin averages $\mathbf{V}_i^r$, $\mathbf{V}_j^c$ and $\mathbf{V}_b^{io}$ where $i \in C_L$, $j \in C_{IO}$ and $b \in C_{IO}$.
3. **While** temperatures T^r, T^c and T^{io} are in the cooling range **do**
   - 3.1 **While** E is decreasing **do**
     - 3.1.1 Generate 3 random L-row, L-column and IO sequences corresponding to the random permutations of unconverged L-row, L-column and IO spins, respectively.
     - 3.1.2 **Repeat**
       - 3.1.2.1 Compute L-row mean field vector $\mathbf{\Phi}_i^r$ for the next L-row spin i in the L-row sequence using (27).
       - 3.1.2.2 Update the L-row spin average vector $\mathbf{V}_i^r$ using (26.a).
       - 3.1.2.3 Compute L-column mean field vector $\mathbf{\Phi}_j^c$ for the next L-column spin j in the L-column sequence using (29).
       - 3.1.2.4 Update the L-column spin average vector $\mathbf{V}_j^c$ using (26.b).
       - 3.1.2.5 Compute IO mean field vector $\mathbf{\Phi}_b^{io}$ for the next IO-spin b in the IO sequence using (31).
       - 3.1.2.6 Update the IO spin average vector $\mathbf{V}_b^{io}$ using (26.c).
     - 3.1.2 **Until** all sequences become empty.
   - 3.2 $T^r = \alpha \times T^r$, $T^c = \alpha \times T^c$ and $T^{io} = \alpha \times T^{io}$.

Figure 3: MFA algorithm proposed for the placement problem.

the *repeat-loop* is below a threshold value, this means that the system is stabilized for the current temperature. Then, T^r, T^c and T^{io} are decreased according to the cooling schedule (step 3.2) and the overall iterative process (step 3.1) is re-initiated.

As mentioned earlier, the proposed MFA algorithm is an iterative process. The complexity of MFA iterations is mainly due to the mean field computations. As seen in (27), (29) and (31) calculations of mean field values are computationally very intensive. We use an efficient implementation scheme which reduces the complexity of individual L-row, L-column and IO iterations to $\Theta(d_{avg}P+PQ)$, $\Theta(d_{avg}Q+PQ)$ and $\Theta(d_{avg}(P+Q)+M)$, respectively. Here, d_{avg} denotes the average cell degree, i.e., average number of nets connected to a cell. This scheme is based on the techniques developed in (Bultan & Aykanat, 1995) for circuit partitioning problem, and can be derived from the formulations in (Bultan & Aykanat, 1995). Therefore, we will not give its details here. Note that a sequence of L-row, L-column and IO spin updates can be considered as a single MFA iteration. Hence, a single MFA iteration takes $\Theta(d_{avg}(P+Q)+PQ+M) = \Theta(d_{avg}(P+Q)+PQ)$ time in our implementation scheme since $M=4(P+Q) \leq PQ$ for sufficiently large P and Q values.

4.4 Parameter Selection and Cooling Schedule

The parameters β^r, β^c, β^{io} used in mean filed computations and the initial temperatures T_0^r, T_0^c, T_0^{io} used in spin updates are estimated using initial random spin averages. Recall that parameter β in the energy function formulation (25) is introduced to determine a balance between the two conflicting optimization objectives of the placement problem. Also recall that we use different balance parameters β^r, β^c, β^{io} in the L-row, L-column and IO mean filed computations since L-row, L-column and IO spins are treated as different systems. For example, in the L-row mean field computations (27), β^r determines a balance between the terms

$$\phi_{ip}^{r(v)} = w_v \sum_{n \in N_i} w_n Z_{np}^{ir} \qquad \text{and} \qquad \phi_{ip}^{r(o)} = w_i \sum_{j \in C_L, j \neq i} w_j v_{jp}^r \sum_{q=1}^{Q} v_{iq}^c v_{jq}^c$$

where $\phi_{ip}^r = \phi_{ip}^{r(v)} + \beta^r \phi_{ip}^{r(o)}$. Note that $-\phi_{ip}^{r(v)}$ and $-\phi_{ip}^{r(o)}$ represent the increases in the vertical routing cost term and overlap cost term, respectively, by assigning L-cell i to row p. Then, we compute the averages

$$\langle \phi_{ip}^{r(v)} \rangle = \Big(\sum_{i \in C_L} \sum_{p=1}^{P} \phi_{ip}^{r(v)} \Big) / (|C_L| P); \qquad \langle \phi_{ip}^{r(o)} \rangle = \Big(\sum_{i \in C_L} \sum_{p=1}^{P} \phi_{ip}^{r(o)} \Big) / (|C_L| P);$$

of these two terms using the initial random spin averages and compute β^r as

$$\beta^r = \gamma \langle \phi_{ip}^{r(v)} \rangle / \langle \phi_{ip}^{r(o)} \rangle$$

where constant γ is chosen as 0.8. The parameters β^c and β^{io} are computed similarly. The same $\gamma = 0.8$ is used in these computations.

Selection of initial temperatures is crucial for obtaining good quality solutions. In previous applications of MFA (Peterson & Södeberg, 1989; Van den Bout & Miller, 1990), it is experimentally observed that spin averages tend to converge at a critical temperature. It is suitable to chose initial temperatures slightly greater than these critical temperatures. Although there are some methods proposed for the estimation of critical temperature (Peterson & Södeberg, 1989; Van den Bout & Miller, 1990), we prefer an experimental way of computing the initial temperatures. After the balance parameters β^r, β^c, β^{io} are fixed, average L-row, L-column and IO mean fields

$$\langle \phi_{ip}^r \rangle = \frac{\sum_{i \in C_L} \sum_{p=1}^{P} \phi_{ip}^r}{|C_L| P}; \qquad \langle \phi_{jq}^c \rangle = \frac{\sum_{j \in C_L} \sum_{q=1}^{Q} \phi_{jq}^c}{|C_L| Q}; \qquad \langle \phi_{bm}^{io} \rangle = \frac{\sum_{b \in C_{IO}} \sum_{m=1}^{M} \phi_{bm}^{io}}{|C_{IO}| M}; \tag{34}$$

are computed using initial random spin averages, respectively. Then, T_0^r, T_0^c, T_0^{io} are computed as

$$T_0^r = \sigma\langle\phi_{ip}^r\rangle/P; \qquad T_0^c = \sigma\langle\phi_{jq}^c\rangle/Q; \qquad T_0^{io} = \sigma\langle\phi_{bp}^{io}\rangle/M; \tag{35}$$

where σ is a constant. Our experiments indicate that it is suitable to chose the parameter σ as 100. Note that initial temperatures are inversely proportional to the dimensions of the respective Potts spins which is also observed for the critical temperature formulations presented in other implementations (Peterson & Södeberg, 1989; Van den Bout & Miller, 1990). The same cooling schedule is adopted for L-row, L-column and IO iterations. At each temperature level, L-row, L-column and IO iterations proceed in an alternate manner for randomly selected unconverged L-row, L-column and IO spin updates. Here, a temperature level corresponds to a particular set of T^r, T^c and T^{io} values. Spin variables are tested for convergence after each spin update. If kth variable (for any k, $1 \leq k \leq K$) of a spin is detected to be greater than 0.95, that spin is assumed to converge to state k. At the end of each random sequence of L-row, L-column and IO spin updates, the total decrease ΔE in the energy due to these spin updates is computed. Note that a random sequence of L-row, L-column and IO spin updates corresponds to a single iteration of the *repeat-loop* (step 3.1.2) in Fig. 3. For each iteration of the *repeat-loop* (step 3.1.2) the average energy decrease per spin update is $\Delta E/\Psi$ where Ψ is the total number of spin updates performed during the random sequence of L-row, L-column and IO spin updates. If $(\Delta E/\Psi) \leq \epsilon$ where ϵ is a small constant chosen as $\epsilon = 0.1$, we conclude that the energy is stabilized for the current temperature level, and we decrease the temperature values according to the cooling schedule.

The cooling process is realized in two phases, slow cooling followed by fast cooling, similar to the cooling schedules used for SA. In the slow cooling phase, temperatures are decreased using $\alpha = 0.95$ until $T < T_o/1.5$. Then, in the fast cooling phase, α is set to 0.85. The cooling process continues until either 90% of the spins are converged or T reduces below 0.01. At the end of this process, the variable with maximum value in each unconverged spin is set to 1 and all other variables are set to 0. Then, the result is decoded as described in Section 4.1 and the resulting placement is obtained.

The resulting placement may be infeasible, i.e., more than one L-cell or IO-cell may be allocated to the same CLB or IOB location, respectively. In such cases, spins causing infeasible allocations are re-initialized to the random initial values together with the set of unconverged spins at the end of the cooling process. Then, MFA algorithm is executed only for these spins starting from the initial

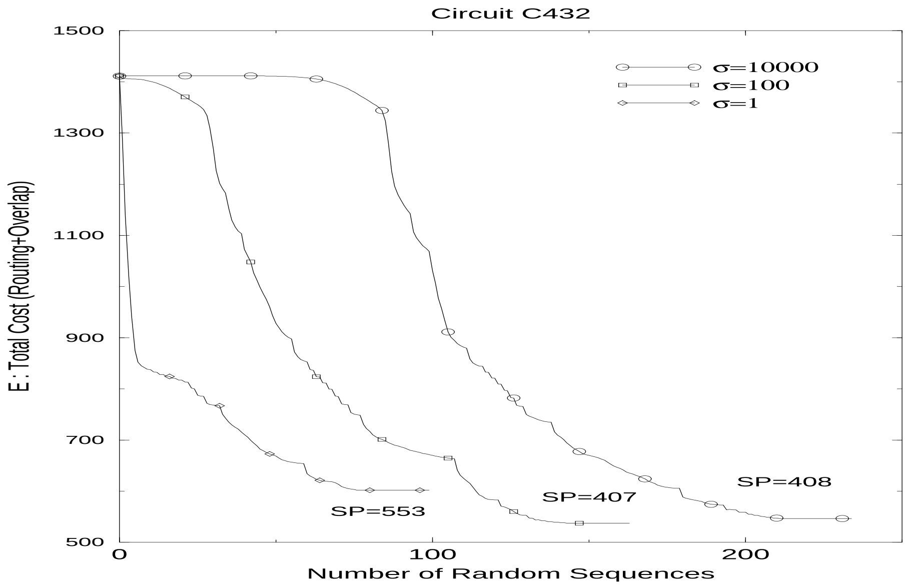


Figure 4: Evaluation of the total energy with MFA iterations for the placement of c432.

high temperatures according to the same cooling schedule. Note that converged spins are held in their decoded values during this re-heating process. This re-heating process is continued until a feasible placement is found.

Figure 4 illustrates the evolution of the energy corresponding to the total placement cost with MFA iterations for the placement of circuit *c432* onto a 10×10 FPGA. This figure is constructed by computing the total energy term (25) at the end of each random sequence of L-row, L-column and IO spin updates. Three curves in Fig 4 correspond to the evolution of the total placement cost for three different initial temperatures computed using $\sigma = 10000$, $\sigma = 100$ and $\sigma = 1$ in (35). In Fig 4, the major decrease in the energy terms for all three cases occur around the same temperature which corresponds to the critical temperature mentioned earlier. In this figure, $\sigma = 10000$ and $\sigma = 100$ correspond to initial temperatures which are significantly and slightly greater than the critical temperature, respectively. As seen in this figure, both initial temperatures yield almost the same solution quality. Note that initial temperatures corresponding to $\sigma = 10000$ and $\sigma = 100$ yield placement solutions with semi-perimeter costs of 408 and 407, respectively. On the other hand, $\sigma = 1$ corresponds to an initial temperature smaller than the critical temperature. This case results in a significantly worse solution quality with a semi-perimeter cost of 553. In general starting from initial temperatures which are slightly greater than the critical temperature is sufficient for obtaining good solutions.

Table 1: Properties of the MCNC benchmark circuits used in the experiments.

BENCHMARK CIRCUITS					
Circuit	# CLB	# IOB	# NET	$P \times Q$	Target FPGA
$c499$	66	73	107	10x10	XC3030PC84
$c1908$	116	58	191	12x12	XC3042CQ100
$c1355$	70	73	115	10x10	XC3030PC84
$c880$	84	86	187	16x20	XC3090PQ160
$c432$	50	43	111	10x10	XC3030PC84
$s1238$	158	30	251	16x20	XC3090PQ160
$c3540$	283	72	489	16x20	XC3090PQ160

5 EXPERIMENTAL RESULTS

This section presents experimental performance evaluation of the proposed MFA algorithm in comparison with *Xilinx Automated Placement and Routing* (*APR 3.30*) program which uses simulated annealing algorithm in placement. Our MFA algorithm was implemented in C language and run on Sun-4 ELC workstations. We used seven MCNC benchmark circuits to test the performance and efficiency of both programs. We used Xilinx 3000 series chips as the target FPGAs. The circuits were mapped into 3000 series logic blocks by Xilinx XACT tools and these mapping results were used as inputs to the placement programs.

Table 1 illustrates the properties of the benchmark circuits. The first two columns illustrate the number of CLBs and IOBs in the circuits to be placed. The third column shows the number of multi-pin nets. The last two column illustrates the $P \times Q$ dimensions of the FPGAs and the names of the target Xilinx chips used for placement.

The placement and routing results are displayed in Tables 2 and 3. Both MFA and *APR* programs were run 10 times for each problem instance. Table 2 displays the average placement costs and the average execution times of 10 runs for each placement instance. The placement results of both MFA and *APR* placement programs are used as inputs to the routing program of *Xilinx APR* tool. The average, the minimum and the maximum values for the maximum path delays obtained in 10 runs are displayed in Table 3. Table 3 also displays the average execution times of *Xilinx APR* tool for routing the placements produced by MFA and *APR* programs. Maximum path delay values were computed running *Xilinx XDelay* program for each routing result.

The *APR* routing program produced 100% routability for each placement result obtained by both

Table 2: Performance of the MFA and APR programs for the placement of MCNC circuits.

PLACEMENT RESULTS						
Circuit	Semi-Perimeter Cost		APR Cost		Execution Time(sec)	
	MFA	APR	MFA	APR	MFA	APR
*c*499	51.2	87.6	25625	22578	56	792
*c*1908	76.6	162.7	54346	49805	138	1845
*c*1355	52.2	92.5	23740	20816	32	639
*c*880	67.2	138.4	36126	27412	188	4828
*c*432	44.3	89.3	16461	15193	87	506
*s*1238	110.2	237.5	140128	117900	367	7843
*c*3540	160.3	401.8	196168	142522	435	16834

Table 3: Routing results obtained by *Xilinx APR* tool for placements produced by MFA and *APR* programs.

ROUTING RESULTS								
Circuit	Maximum Path Delay (ns)						Execution Time (sec)	
	MFA			APR				
	Avg.	Min.	Max.	Avg.	Min.	Max.	MFA	APR
*c*499	94.9	93.0	99.6	98.5	94.8	100.4	136	85
*c*1908	159.6	145.6	168.5	166.2	157.8	17 2.1	796	853
*c*1355	94.5	92.9	98.3	91.5	84.0	93.8	98	78
*c*880	151.2	141.1	164.6	139.1	137.2	14 2.6	187	266
*c*432	173.5	162.1	192.5	178.3	174.4	185. 8	202	314
*s*1238	198.3	184.5	214.5	165.3	154.7	174.7	428	986
*c*3540	243.5	239.6	264.4	238.5	221.9	269.5	4380	5726

placement programs for all circuits except the largest circuit *c3540*. The router fails to route all the nets in the placement of this circuit. We have not experienced any infeasibility due to the assignment of L-cells to the same CLB locations in our MFA runs. However, we have experienced infeasibility due to the assignment of IO-cells to the same IOB locations in some of our runs. However, a single re-heating pass was sufficient for obtaining feasible solutions in all these placement instances.

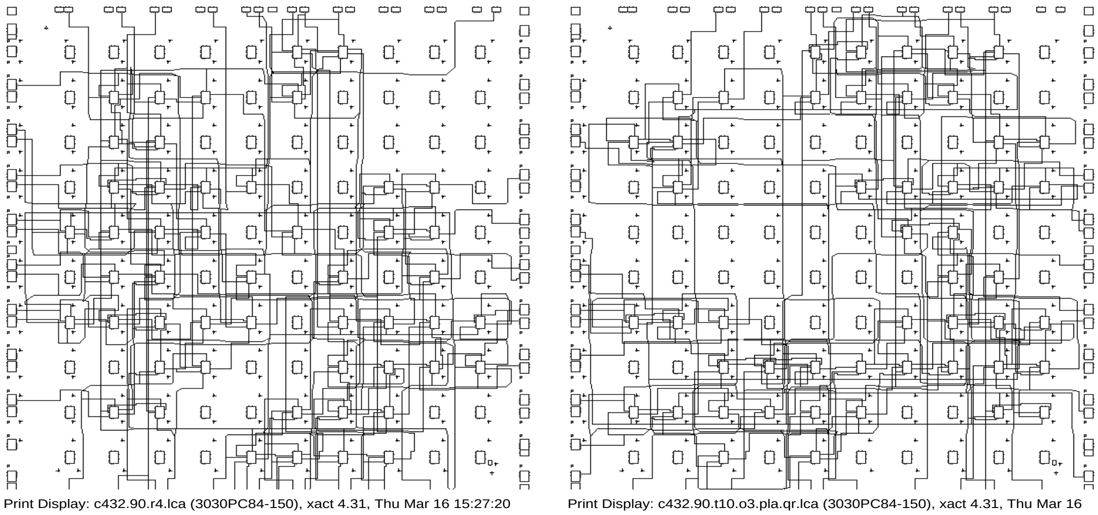
The semi-perimeter cost values displayed in Table 2 correspond to the average normalized semi-perimeter costs computed for the placement results of both programs as is described in Section 2. Here, normalization refers to assuming a unit square layout. That is, vertical and horizontal spans of the nets are normalized by multiplying them with $1/Q$ and $1/P$, respectively, during the computation of total semi-parameter cost values for Table 2. The *APR* cost values correspond to the average costs computed for the placement results of both programs according to *APR*'s placement cost definition.

Table 4: Normalized average performance measures for the placement results obtained by MFA and *APR*.

NORMALIZED RESULTS				
Circuits	Maximum Path Delay (ns)		Execution Time (sec)	
	MFA	APR	MFA	APR
$c499$	1.00	1.03	1.00	14.1
$c1908$	1.00	1.04	1.00	13.4
$c1355$	1.00	0.96	1.00	19.9
$c880$	1.00	0.91	1.00	25.6
$c432$	1.00	1.03	1.00	5.8
$s1238$	1.00	0.83	1.00	21.3
$c3540$	1.00	0.98	1.00	38.7
Avg	1.00	0.97	1.00	19.8

The semi-perimeter costs of the placement results obtained by the MFA program are 105% better than those of the *APR* program. However, APR-costs of the placement results obtained by the APR program are 16% better than those of the MFA program.

Table 4 illustrates the normalized relative performance results of the two placement programs. In this table, the averages of the maximum path delay values obtained by the *Xilinx XDelay* program after routing the placement results of *APR* placement program are normalized with respect to those of the MFA program. This table also illustrates the execution times of the *APR* placement program normalized with respect to those of the MFA program. As seen in this table, the MFA placements yield slightly better routing results in 3 circuits out of seven circuits. *APR* placements yield 3% better routing results on the overall average. However, as seen in Table 2 and Table 4, MFA placement program is significantly faster than the *APR* placement program in all instances. MFA placement program is 19.8 times faster than the *APR* placement program on the overall average. Fig 5 illustrates sample routing results of the circuit $c432$ for placements obtained by *APR* and MFA.



(a) (b)

Figure 5: Routing results of the circuit $c432$ for the placements obtained by (a) APR (b) MFA.

6 CONCLUSIONS

In this paper, we proposed a fast nondeterministic cell placement algorithm for VLSI design automation based on Mean Field Annealing (MFA). The performance of the proposed placement algorithm is evaluated in comparison with the commercial automated circuit design software *Xilinx Automatic Place and Route* (APR) tool for the placement of seven MCNC benchmark circuits. The results show that neurocomputing approaches as the MFA technique can be applied to real world problems and can compete with the commercially available tools successfully. Experimental results indicate that our algorithm achieves comparable placements with APR. However, our algorithm is significantly faster than APR.

Acknowledgment

The authors would like to thank Jonathan Rose for helpful discussions on FPGAs.

REFERENCES

Bultan T., & Aykanat C. (1992). A new mapping heuristic based on mean field annealing. *Journal of Parallel and Distributed Computing*, **16**, 292–305.

Bultan, T., & Aykanat, C. (1995). Circuit partitioning using mean field annealing. *Neurocomputing*, **8**, 171–194.

Cimikowski, R., & Shope, P. (1996). A neural-network algorithm for a graph layout problem. *IEEE Transactions on Neural Networks*, **7**(2), 341–345.

Dunlop, A. E., & Kernighan, B. W. (1985). A procedure for placement of standard-cell VLSI circuits, *IEEE Transactions on Computer-Aided Design*, **4**, 92–98.

Gislén, L., Peterson, C., & Söderberg, B. (1992). Complex scheduling with Potts neural networks. *Neural Computation*, **4**, 805–831.

Häkkinen, J., Lagerholm, M., Peterson, C., & Söderberg, B. (1997). *A Potts neuron approach to communication routing* (LU–TP Technical report No. 97–02) Sweden: Lund University, Department of Theoretical Physics II. (also submitted to *Neural Computation*).

Herault, L., & Niez, J. (1989). Neural networks and graph k-partitioning. *Complex Systems*, **3**, 531–575.

Hopfield, J. J., & Tank, D. W. (1985). Neural computation of decisions in optimization problems. *Biological Cybernetic*, **52**, 141–152.

Kirkpatrick, S., Gellat, C. D., & Vecchi, M. P. (1983). Optimization by simulated annealing. *Science*, **220**, 671–680.

Lengauer, T. (1990). Combinatorial Algorithms for Integrated Circuit Layout. John Wiley and Sons.

Ohlsson, M., Peterson, C. & Söderberg, B. (1993). Neural networks for optimization problems with inequality constraints — the knapsack problem. *Neural Computation*, **5**(2), 331–339.

Ohlsson, M., & Pi, H. (1997). A study of the mean field approach to knapsack problems. *Neural Networks*, **10**(2), 263–271.

Peterson, C., & Söderberg, B. (1989). A new method for mapping optimization problems onto neural networks. *International Journal of Neural Systems*, **1**(3), 3–22.

Rose, J., Francis, R. J., Brown, S., & Vranesic, Z. G. (1992). Field-Programmable Gate Arrays. MA: Kluwer Academic Publishers.

Rose, J., Elgamal, A. E., & Sangiovanni-Vincentelli, A. (1993). Architecture of field-programmable gate-array, *Proceedings of IEEE*, **81**, 1013–1029.

Sechen C. (1988). VLSI placement and global routing using simulated annealing. MA: Kluwer Academic Publishers.

Shahookar, K., & Mazumder, P. (1991). VLSI cell placement techniques, *ACM Computing Surveys* **23**(2), 142–220.

Sherwani, N. (1993). Algorithms for VLSI physical design automation. Kluwer Academic Publishers.

Takahashi, Y. (1997) Mathematical improvement of the Hopfield model for TSP feasible solutions by synapse dynamical systems. *Neurocomputing*, **15**(1), 15–43.

The Programmable Gate Array Data Book. (1994). Xilinx Inc.

Van den Bout, D. E., & Miller, T. K. (1989). Improving the performance of the Hopfield-Tank neural network through normalization and annealing. *Biological Cybernetics*, **62**, 129–139.

Van den Bout, D. E., & Miller, T. K. (1990). Graph partitioning using annealing neural networks. *IEEE Transaction on Neural Networks*, **1**(2), 192–203.

Yih, J. S., & Mazumder, P. (1990). A neural network design for circuit partitioning. *IEEE Transactions on Computer-Aided Design*. **9**, 1265–1271.