Proceedings of the Federated Conference on Computer Science and Information Systems pp. 147–156
DOI: 10.15439/2019F91
ISSN 2300-5963 ACSIS, Vol. 18

Best Response Dynamics for VLSI Physical Design Placement

Michael Rapoport
School of Computer Science
The Interdisciplinary Center
Herzliya, Israel
Email: mishkarapoport@gmail.com

Tami Tamir
School of Computer Science
The Interdisciplinary Center
Herzliya, Israel
Email: tami@idc.ac.il

***Abstract*—The physical design placement problem is one of the hardest and most important problems in micro chips production. The placement defines how to place the electrical components on the chip. We consider the problem as a combinatorial optimization problem, whose instance is defined by a set of 2-dimensional rectangles, with various sizes and wire connectivity requirements. We focus on minimizing the placement area and the total wire-length.**

We propose a local-search method for coping with the problem, based on natural dynamics common in game theory. Specifically, we suggest to perform variants of *Best-Response Dynamics* (BRD). In our method, we assume that every component is controlled by a selfish agent, who aim at minimizing his individual cost, which depends on his own location and the wire-length of his connections.

We suggest several BRD methods, based on selfish migrations of a single or a cooperative of components. We performed a comprehensive experimental study on various test-benches, and compared our results with commonly known algorithms, in particular, with simulated annealing. The results show that selfish local-search, especially when applied with cooperatives of components, may be beneficial for the placement problem.

I. Introduction

PHYSICAL DESIGN is a field in Electrical Engineering which deals with *very large scale integration* (VLSI). Specifically, physical design is the main step in the creation of *Integrated Circuit* (IC). The basic question is *how to place the electrical components on the chip*. This fundamental question became relevant with the invention of ICs in 1958 [19], and remains critical our days with the development of micro-electricity. Recent developments in micro-electricity enables transistors to reach the size of nanometers, thus a single chip can accommodate thousands of components of different sizes and dispersed connectivity. Bad layout of electrical components leads to expensive production and poor performance. Figure 1 presents the Intel i7 processor [10], and demonstrates how efficient design is crucial in enabling the accommodation of many components on a small area.

The research was supported by THE ISRAEL SCIENCE FOUNDATION (grant No. 1036/17).

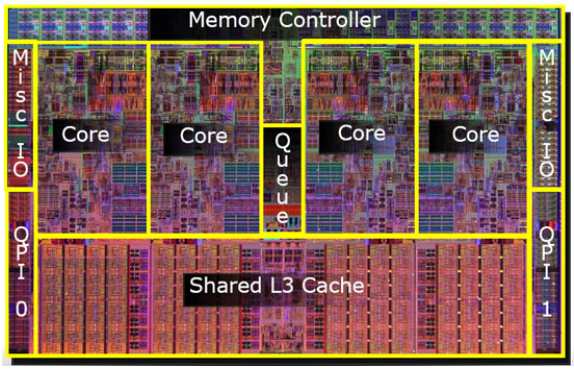

Fig. 1. Intel i7 processor placement.

The complexity of *VLSI physical design* led to the establishment of a design process, in which the problem is divided into several steps, each is an independent NP-complete problem. The most fundamental steps are: (i) *Floorplan:* choose the area of the chip and decide the positions of the building blocks of the chip, (i.e., in Intel processor: cores, graphic processor, cache, memory controller). (ii) *Placement:* Each of the above mentioned building blocks consists of several components. These components should be placed in a way that minimizes area and wire-length. (iii) *Signal and Clock Routing:* route the wires via components white space, which is an extra area assigned for wiring.

In this work, we focus on the *Placement* problem. The floorplan is usually performed manually, and the signal and clock routing is more of a production engineering problem which is tackled using different tools.

Several common methods for coping with the Placement problem are based on local-search. We propose a new class of local-search algorithms that consider the problem as a game played among the components, where each component corresponds to a selfish player who tries to maximize his own welfare. Our algorithms are different from other algorithms based on local search in the way they explore the solution space. Every solution is associated with a global cost, and every component is associated with its individual cost, which is based on its own placement and connections. We move from one solution to another if this move is selfishly beneficial for

a single component or for a small cooperative of components, without considering the effect on the global cost.

In this paper we first review the placement problem, and survey some of the existing techniques to tackle it, in particular local-search algorithms and Simulated Annealing. We then describe our new method of performing selfish Best-Response Dynamics (BRD). In order to evaluate this method, we performed an extensive experimental study in which we simulated and tested several variants of BRD on various test-benches. Our results show that BRD may produce a quick and high quality solution.

A. The Placement Problem

The Placement process determines the location of the various circuit components within the chip's core area. The problem, and even simple subclasses of it, were shown to be NP-complete by reductions to the *bin packing* and the *rectangle packing* problems [14], [15]. Moreover, a reduction to the *Quadratic assignment problem* shows that achieving even a constant approximation is NP-hard [9].

Bad placement not only reduces the chip's performance, but might also make it non-manufacturable by forcing very high wire-length, lack of space, or failing timing/power constraints. As demonstrated in Figure 2, good placement involves an optimization of several objectives that together ensure the circuit meets its performance demand [4], [17]:

1) *Area:* Minimizing the total area used to accommodate the components reduces the cost of the chip and is crucial for the production.
2) *Total wire-length:* Minimizing the total length of the wires connecting the components is the primary objective of the physical design. Long wires require the insertion of additional buffering, to insure synchronization between the components. Short wires decrease the power consumption and the system's leakage.
3) *Wire intersection:* Our days, wire intersection is allowed as long as a single wire does not have more than a predefined number of intersections. The manufacturing process enables several routing layers. Nevertheless, a good layout avoids unnecessary intersections.
4) *Timing:* The timing cycle of a chip (clock frequency) is determined by the delay induced by the longest wire, usually referred to as the critical path.

Our work considers the initial placement calculation, denoted *global placement*. This stage is followed by the *detailed placement* stage, in which the global placement results are put into use and the cells are actually placed on the die. The detailed placement stage includes small changes to solve local issues such as wire congestion spots, remaining overlaps, layout constrains (such as via locations), connecting to the die pinout, etc.

In the global placement stage, several parameters are optimized. We focus on the total wire-length and placement area. By adjusting the cost function associated with each configuration, our method enables considering additional parameters such as wire congestion, critical path length, and more.

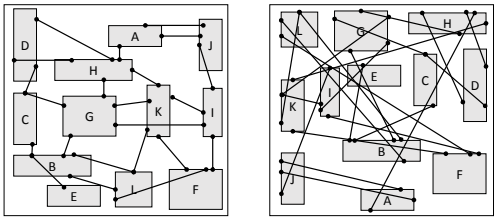

Fig. 2. An example of a good placement (left) v.s a bad placement (right). In the good placement the wires are shorter and there is almost no congestion. In this example, the area of both placements is the same.

B. Formal Description of the Placement Problem

We describe the placement problem as a combinatorial optimization problem. The components composing the problem are represented by 2-dimensional rectangles denoted *blocks*. In the placement, they can be rotated by $90°$, $180°$ or $270°$, but not mirrored. The sides of the assigned blocks must be parallel to each other, and to the bounding area. Whenever we refer to a location in a block, we let $(0, 0)$ be the bottom-left corner, and every other point in the block is given by its (x, y) coordinates with respect to this corner.

Formally, an instance of the problem is defined by:

1) A set of n blocks $\{B_1, B_2, .., B_n\}$ to be placed on the chip. Every block $1 \leq i \leq n$, has associated width w_i and height h_i.
2) A list of required connections between the blocks, $\{N_1, N_2, .., N_m\}$. Every connection is given by a pair of blocks, and the locations in which these blocks should be connected. Formally $N_j = \langle B_j^1, x_j^1, y_j^1, B_j^2, x_j^2, y_j^2 \rangle$, for $0 \leq x_j^1 \leq w_j^1, 0 \leq y_j^1 \leq h_j^1$ and $0 \leq x_j^2 \leq w_j^2, 0 \leq y_j^2 \leq h_j^2$, corresponds to a request to connect blocks B_j^1 and B_j^2, such that the wire is connected to coordinate (x_j^1, y_j^1) in B_j^1 and to coordinate (x_j^2, y_j^2) in B_j^2.

The output of the problem is a placement F given by the locations of the blocks on the plane $\{L_1, L_2, .., L_n\}$, such that for every $1 \leq i \leq n$, $L_i = (x_i, y_i, r_i)$. The parameter $r_i \in \{0, 1, 2, 3\}$ specifies how block B_i is rotated corresponding to $\{0, 90, 180, 270\}$ degrees. Note that a rotation by $180°$ is not equivalent to not rotating at all, since the location of the required connections is also rotated. Formally, block B_i is placed in the rectangle whose diagonal endpoints are (x_i, y_i) (this corner is independent of the value of r_i), and (x_i+w_i, y_i+h_i) if $r_i = 0$, or (x_i+h_i, y_i+w_i) if $r_i = 1$, or (x_i-w_i, y_i-h_i) if $r_i = 2$, or $(x_i - h_i, y_i + w_i)$ if $r_3 = 1$.

A placement is legal if no two blocks overlap, that is, the rectangles induced by L_{i_1} and L_{i_2} are disjoint for all $i_1 \neq i_2$.

This condition may be relaxed a bit in the global placement stage, and allow small percentage of overlaps area. These overlaps are resolved later during the detailed placement stage.

The *bounding box* of a Placement F, is the minimum axis-aligned rectangle which contains all the blocks. The *area* of a placement F is the area of the bounding box, and is denoted $A(F)$.

The blocks' location, together with the required connections, induce the wire-length of a placement. Formally, assume

that blocks B_1 and B_2 are located in L_1 and L_2, respectively, and let $N_j = \langle B_j^1, x_j^1, y_j^1, B_j^2, x_j^2, y_j^2 \rangle$. We first calculate the actual coordinates of the connection points, based on L_1, L_2, and the values of $\langle x_j^1, y_j^1 \rangle$ and $\langle x_j^2, y_j^2 \rangle$. Let $\langle \hat{x_j^1}, \hat{y_j^1} \rangle$ and $\langle \hat{x_j^2}, \hat{y_j^2} \rangle$ be the points we need to connect. The wire-length associated with N_j, denoted $Len(N_j)$, is calculated in a way that fits the actual production process, in which all wires are parallel to the blocks and to the bounding area, that is, $Len(N_j) = \Delta X + \Delta Y = |\hat{x_j^1} - \hat{x_j^2}| + |\hat{y_j^1} - \hat{y_j^2}|$. The total wire-length of a Placement F is denoted $L(F)$, and is given by $L(F) = \sum_{j=1}^{m} Len(N_j(F))$. An example of wire-length calculation is given in Figure 3.

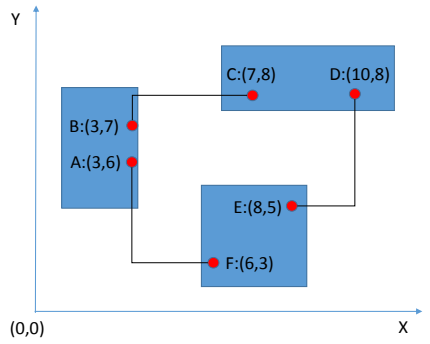

Fig. 3. An Example of wire-length calculation. There are three connections between the pairs of points $\{A, F\}, \{D, E\}$ and $\{B, C\}$. The total wire-length is $(|X_A - X_F| + |Y_A - Y_F|) + (|X_E - X_D| + |Y_E - Y_D|) + (|X_B - X_C| + |Y_B - Y_C|) = (3 + 3) + (2 + 3) + (4 + 1) = 16$.

The goal in the placement problem is to minimize $\alpha L(F) + (1 - \alpha)A(F)$ where the parameter $0 \leq \alpha \leq 1$ weight the importance of the two objectives. These days (as the number of components per chip rises) it is a very common practice to focus on the wire-length of the placement and only when finished optimizing the wire-length, perform small changes in order to gain better area result, with a minimal harm of the achieved wire-length. Thus, in our experiments (to be described in Section III), we give a substantially higher weight to the wire-length.

C. Current Techniques for Efficient Placement

We now overview the common disciplines to handle the Placement problem. Some algorithms are tailored for simplified classes of instances. Specifically,

1) *Standard cell:* Components may have different width, but they all have the same height and are placed in rows. With over-cell routing the goal is to minimize the width of the widest row and the total wire-length.
2) *Gate array / FPGA:* The area is discretized to equally sized squares where each square is a possible component location. All the components have the same size and shape but different connections between them, the goal is to minimize the total wire-length.

Both classes induce simplified problems, which are still NP-hard, but can be approximately solved using Linear Programming [22], [7], Greedy Algorithms [25], [2], Slicing Tree representation [3], or by Divide and Conquer algorithms that allows temporal block overlaps [2], [7].

A different solution approach is to develop heuristics, usually with strong randomness involved. Most heuristics have no assumptions on the problem thus able coping with general instances. Heuristics have no performance guarantee but perform well in practice. Some heuristics were tailored for *Standard cell* and *Gate array* instances [8], [16], [24]. The most commonly used algorithm concept for placement is *simulated annealing* (SA) [23], [20]. Modern algorithms of our days are always compared against it and many of them are based on its concept. While SA is unlikely to find an optimal solution, it can often find a very good one. The name simulated annealing come from annealing in metallurgy, a technique involving heating and controlled cooling of a material to increase the size of its crystals and reduce their defects. Both are attributes of the material that depend on its thermodynamic free energy. Heating and cooling the material affects both the temperature and the thermodynamic free energy. The simulation of annealing as an approach for minimization of a function of large number of variables was first formulated in [13]. Many modern algorithms are based on the concepts of simulated annealing.

Additional widely used placement methods include (i) Force Directed Placement, in which the problem is transformed into a classical mechanics problem of a system of objects attached to springs [21], (ii) Placement by Partitioning, in which the circuit is recursively partitioned into smaller groups [2], [7], (iii) Numerical Optimization Techniques, based on equation solving and eigenvalue calculations [25], [18], and (iv) Placement by Genetic Algorithm, that emulates the natural process of evolution as a means of progressing toward optimum, [25]. Some of these methods are only suited for *Standard cell* or *Gate array* instances, and some are general. A survey of the above and of additional algorithms for placement can be found in [1], [24], [11].

II. Our Local-Search Method for Solving the Placement Problem

The main challenges involved in solving the Placement problem are the need to optimize several objectives simultaneously, and to achieve even a good approximate solution in reasonable time. Optimizing even a single objective is an NP-hard problem. Naturally, combining several objectives, that may be conflicting, makes the problem more challenging.

Our proposed method not only performs a good placement in a relatively short time, but also copes with the multiple objective challenge.

A. The Placement Problem as a Game

We propose to tackle the problem by a local-search algorithm, using natural dynamics common in *game theory*. Specifically, we suggest to perform variants of *Best-Response Dynamics* (BRD), assuming the components correspond to strategic selfish agents who strive to optimize their own welfare. In a BRD process, every agent (player) in turn, selects

his best strategy given the strategies of the other players. In our game, the strategy space of a player consists of all the locations his component can be placed in, given the location of the other components. Players keep changing strategies until a Nash equilibrium of the game is reached. A Nash equilibrium is a strategy profile in which no player can benefit from changing his strategy [12]. A lot of attention is given to best-response dynamics in the analysis of non-cooperative games, as this is the natural method by which players proceed toward a NE. The common research questions are whether BRD converges to a NE, the convergence time, and the quality of the solution (e.g. [5], [6]).

BRD can also be performed with coordinated deviations. That is, in each step, a group of players, denoted a *cooperative*, moves simultaneously, such that their total cost is reduced. Note that in a coordinated deviation of a *cooperative*, unlike a *coalition*, some members of the cooperative may be hurt. The deviation is beneficial if the total members' cost is reduced.

In order to consider the placement problem as a game played by selfish agents, we need to associate a value, or cost, for each player in each possible configuration of the game. In our setting, players correspond to blocks and configurations correspond to placements. The BRD process is define with respect to a cost function that depends on the wire-length connected to the player's block, and the total placement area. The individual cost function is calculated for each block or cooperative, and is relevant only to the currently playing block or cooperative.

Recall that for a configuration F, the global cost of F is

$$Global\_cost(F) = \alpha L(F) + (1-\alpha)A(F),$$

where $L(F)$ is the total wire-length, $A(F)$ is the bounding box area, and the parameter α is used to weight these two components of the cost function. In our algorithms, the global cost function, is used only to evaluate the final configuration - in order to compare different methods and to analyze the progress of the algorithms.

The *individual cost function* is used to evaluate the possible deviations of the currently playing block. For a single block B_i, let $L_{B_i}(F)$ denote the total wire-length of B_i's connections. By definition, $L(F) = \frac{1}{2}\sum_{1\leq i\leq n} L_{B_i}(F)$. The total individual wire-length is divided by 2, since every wire is counted in the individual wire-length of its two endpoints. For a configuration F and a block B_i, the individual cost of B_i in F is defined as follows:

$$Ind\_cost(B_i, F) = \alpha \cdot (L_{B_i}(F))^2 + (1-\alpha)\cdot A(F).$$

Note that in the individual cost function, the corresponding wire-length is squared – for normalization with the area component.

Let Γ be a subset of the blocks. In order to evaluate configurations that are a result of a coordinated deviation, we define, for a cooperative Γ in a configuration F, the individual cost of Γ in F:

$$Ind\_cost(\Gamma, F) = \alpha \cdot \sum_{B_i\in\Gamma} (L_{B_i}(F))^2 + (1-\alpha)\cdot A(F).$$

Since finding the best response is NP-hard in most scenarios and particularly for coordinated deviation, we perform a better-response move, in which the player (or a cooperative) benefits, but not necessarily in the optimal way. In practice, we perform the best response move in a restricted search space. Also, in some algorithms, when there is no local improving step, we may perform a move which harms the cost function. Such moves result in a temporary worse state and are used in order to allow the algorithm to escape from local minima.

In our experiments, we compared our results with those achieved by *Simulated Annealing* (SA), *Greedy Local Search* (GLS) algorithm, based on hill climbing, and *Fast Local Search* (FLS) algorithm (faster version of the greedy local search). For each test-bench we run our algorithms as well as these algorithms, and compared the results. In this paper we only provide the comparison with SA, as it outperformed the other two local-search methods.

B. Search for a Solution over the Solution Space

Before presenting our algorithms we give an overview of the local search technique, and explain how the search for the solution is performed. A local-search algorithm performs a search over the solution space. Every possible solution (placement F) is associated with a score ($Global\_cost(F)$). The global cost function defines a placement-cost multidimensional complex, on which the algorithm advances. Each point on the complex is a placement and the complex includes all possible placements.

Every local-search algorithm moves on the placement-cost complex searching for a point corresponding to a placement having minimum cost. The local-search paradigm implies that the movement along the complex is almost continues. When the algorithm encounters a heap on the complex which it cannot pass, it may try to bypass it in order to continue the search in that direction.

The main challenge when applying such algorithms, is how to pick the next point to explore and how to decide when to stop the search. As demonstrated in Figure 4, we can continue to search up to some point of worse cost but we do not know what awaits us further down the path of the search. We may attempt to remember each minimum we visit during the search and traverse different search paths from each local minimum detected. However, such methods perform a brute force search, which in turn results in exponential running time. Finding the global minimum means we have found an optimal solution for an NP-hard problem. Hence, such algorithms must have exponential running time (regardless of the algorithm's logic) unless P = NP.

The main difference between our algorithms and previous algorithms based on local search (in particular SA, GLS, FLS), is the way we evaluate each solution in the solution space, and the way we advance to the next solution in the search process. Previous algorithms calculate the cost function for the entire placement, while our algorithms base their progress on the individual cost of a block (or of a cooperative of blocks). The main goal of this work is to examine the quality of local-search algorithms for the placement problem, in which

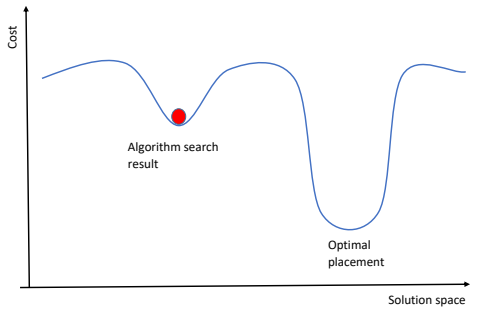

Fig. 4. A general description of a local-search method. The algorithm traverses the cost-placement multidimensional complex. Local search may end-up in a local minimum, unable to escape, thus also unable to find an optimal placement.

the search on the solution space is determined by only a single block (or a small cooperative), in a selfish manner, according to its individual cost-over-time curve. Our method does not use a global cost function; we present the global objective parameters (wire-length, total area, etc.) only for the analysis. As detailed in Section II-C, some of our algorithms accept moves that harm a bit the individual cost function (thus breaking the selfishness to some extent). This possibility allows the algorithms to escape local minimum and the search becomes much more versatile.

C. Algorithms Based on Best-Response Dynamics

The BRD process proceeds as follows: Every block corresponds to a player. In every step a player or a cooperative of players have a chance to change their current location, in a way that reduces their individual cost in the resulting configuration. In some of our algorithms, a step increasing the individual cost may be accepted with some probability.

Every block can perform one or more of the following moves: {Up, Down, Left, Right, Rotate $90°$, Rotate $180°$, Rotate $270°$}. A step is legal as long as the resulting configuration is legal.

We use three search methods for the block migrations:

1. *Best Response (BRD)*: Each block is controlled by a selfish player. Each player can perform one move per turn, the move is the best local move the block can perform to reduce its individual cost. The algorithm advances in rounds, where in each round, every player gets an opportunity to migrate. Players are allowed to perform only legal moves (no block-overlaps are created).

2. *Constant Perturbations (BRD-ConstPerb)*: In This variant of BRD, when a player does not have a legal improving move to perform, he may, with some non-negligible probability, choose a step which harms its individual cost. In our experiments we found 0.3 to be a good probability for accepting a worse state. It is small enough not to harm the selfishness on one hand, and allows the placement to escape local minima on the other.

3. *Relaxed Search (BRD-RlxSrch)*: This algorithm is another variant of BRD. The difference is that players can select illegal locations - that involve block overlaps. While blocks are not allowed to overlap beyond a reasonable limit in the final configuration, temporal overlaps may be fruitful. Our relaxed search allows overlaps with varying fines on the area of the overlap. The overlaps fine are added to the block's individual cost. The fines are increased every round - to encourage convergence to a final placement with hardly any overlaps. The Global placement stage can tolerate small overlaps, so the output is accepted if the final placement does include some overlaps.

Each of the above algorithms is ran in two variations: without and with *swap moves*. A swap move is a move in which the active block swaps places with some other block if the swap is legal and reduces the active block's individual cost, as well as the global cost function (this ensures we avoid recurrent swaps between a pair of blocks). Swap moves break the locality of the search and allows another method with which to escape local minima. Instead of attempting to escape a local minimum by accepting a worse state, the algorithm can escape a local minimum by jumping to a better, yet not local neighbor, state. Swap moves do not break the selfishness of our algorithms but rather only the locality, and only to some extent. As our experiments reveal, enabling swap moves improves the quality of the solution.

D. Coordinated Deviation of a Cooperative

Unlike a unilateral deviation, a coordinated deviation is initiated by a group of players, denoted a *cooperative*, who migrate simultaneously. Such a migration may harm the individual cost of some cooperative members (for example, if they give up good spots for other members), however, the total cost of the cooperative members is strictly reduced. When applying a coordinated deviation, we first determine the cooperative size and then the blocks composing it. A coordinated deviation is therefore defined by (i) the search method, (ii) the cooperative size method, and (iii) the cooperative member selection method.

We simulated three different methods for determining the cooperative's size:

1) *Increasing*: Starting from $k = 1$, after converging to a k-NE profile, which is stable against deviations of cooperatives of size k, we increase the active cooperative size to $k + 1$. We keep increasing the cooperative size up to a predefined limit.
2) *Iterating*: Each round has a different cooperative size, the sizes are incremented after each round, when the size reaches a predefined limit we reset the size to a single block.
3) *Random*: Each cooperative has a random size, the size is uniformly distributed between a single block and a predefined limit.

The cooperative's members are selected in the following way: we iterate over all the blocks, selecting a different *head block* of the cooperative in each round. The head block constructs a cooperative according to one of the following methods:

1) *Closest Connected blocks*: in every iteration we add to the cooperative a block with the shortest wire-length to some other block already in the cooperative.
2) *Farthest Connected blocks*: in every iteration we add to the cooperative a block with the longest wire-length to some other block already in the cooperative.
3) *Closest Geometrically blocks*: in every iteration we add to the cooperative a block with the smallest closest geometrical distance to the head block of the cooperative.
4) *Farthest Geometrically blocks*: in every iteration we add to the cooperative a block with the highest geometrical distance to the head block of the cooperative.
5) *Random*: Random set of blocks. The cooperative is built by uniformly adding blocks one by one, until the cooperative size is reached.

In our experiments, we run and compared various combinations of search algorithms with cooperative size and formation methods. The algorithms advance as follows: once the cooperative has been formed, all the feasible permutations of possible moves for the cooperative (depending on the search algorithm) are calculated. For each permutation we calculate the individual cost of the cooperative in the resulting placement. The permutation that minimizes this cost is chosen. Only the cooperative cost is taken into account, and we ignore the global cost as well as the internal distribution of the cost among the blocks composing it.

E. Expected Algorithms' Progress

In this section we review our algorithms by describing their progress in general. A typical cost-over-time progress of BRD-ConstPerb is depicted in Fig 5. Since players can choose a step which harms their individual cost, we expect the algorithm to be able to escape local minima by moving to a more expensive placement and improving it by a sequence of cost-reducing moves, which hopefully lead to a better local minimum.

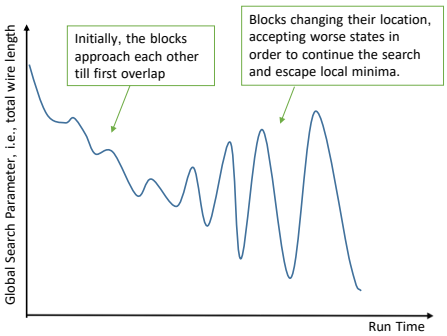

Fig. 5. Expected progress of the *BRD-ConstPerb* search method with unilateral deviations.

The progress of our BRD-RlxSrch method depends heavily on the fines for overlaps. Recall that these fines increase with the run time. As illustrated in Figure 6, this enables the algorithm to explore more areas in the solution space. Once the fines are above some threshold, the algorithm explores feasible or almost feasible solutions whose cost may be higher than former non-feasible solutions explored earlier.

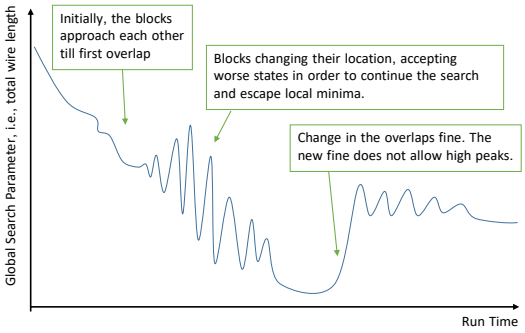

Fig. 6. Expected progress of the *BRD-RlxSrch* method with unilateral deviations. The slope has peaks, the cost function is monotonically decreasing via game of tradeoffs between the search parameters and the overlaps.

Finally, Figure 7 illustrates the typical cost-over-time progress of BRD-RlxSrch and BRD-ConstPerb when coordinated deviations are allowed. The possibility to accept worse or unfeasible solutions enables the algorithm to escape local minima and to advance in the search space towards a better solution.

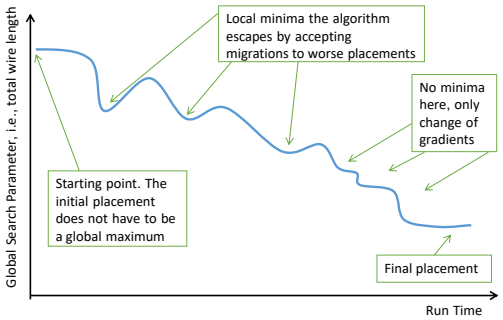

Fig. 7. Expected progress of *BRD-ConstPerb* and *BRD-RlxSrch* with coordinated deviation.

In the above figures we present the slopes as monotonically nicely curved lines, in reality this is not the case. The real lines have various gradient changes, and they are far from being nicely curved over the monotonic movement sections. The figures present the tendency of the algorithm and the overall progress.

III. Experimental Results

In this section we present our experimental study. Our experiments simulate the global placement stage. This stage is followed by the detailed placement stage, in which the global placement results are applied and the blocks are actually placed on the die. Usually at the detailed placement stage, small changes occur in order to solve some local issues such as wire congestion spots, remaining overlaps, layout constraints, connecting to the die pinout, etc.

We first demonstrate our concept by presenting the results of the unilateral deviation algorithms. Next, we compare some of

our heuristics with the *Simulated annealing* algorithm. Finally, we study coordinated deviations and analyze the effects of the cooperative size and structure. We explore how coordinated deviations improves the results obtained by unilateral deviations, regardless of the selected method for the search algorithm, and also consider algorithms that combine unilateral and coordinated deviations.

A. Experiments Setup

All algorithms are ran on the same machine with similar conditions. We sample various parameters during the algorithms run, in order to study not only the final outcome but also the search process. Time measurement is conducted and counted by the algorithms context timers, thus if a context switch occurs the timer pauses. While the time values themselves vary on different machines, the progress of the algorithms and comparison between them is valid and independent of the machine.

Our experiments were performed on 6 different test-benches, $T_{30}^4, T_{30}^6, T_{30}^8, T_{40}^4, T_{40}^4$ and T_{40}^8, where T_n^c corresponds to an instance of n blocks, with c connections-per-block. In all instances, the block sizes are randomly distributed, height and width being a random equally distributed number in the range $[30, 80]$ (pixels). The different connections-per-block parameter enables a good comparison and allows us to isolate and emphasize various aspects of the algorithms.

Recall that the Individual cost of a block B_i in a placement F is defined to be $Ind\_cost(B_i, F) = \alpha \cdot (L_{B_i}(F))^2 + (1 - \alpha) \cdot A(F)$. We run the search algorithms with various values for α, and found out that the wire-length component should get much higher weight. Thus, all the results described in this section were obtained with $\alpha = 0.9$. This fits the common practice these days to focus on minimizing the wire-length of the placement and only when done optimizing the wire-length, perform small changes in order to gain better area result.

As detailed in Section II-C, we used three local search method: BRD – only legal profitable moves, BRD-ConstPerb – legal but maybe harmful moves, and BRD-RlxSrch – profitable but maybe non-legal moves (overlaps associated with fine). These search methods are constructed into algorithms, combining unilateral players and coordinated deviation players.

Each of these local search methods run in two different variations, without or with *swap moves*. Note that a swap move differs from a cooperative of size 2. The two members of a cooperative may swap places as long as it improves their total cost. However, a swap move is initiated by a single block and may hurt significantly the individual cost of the second block involved.

In order to better evaluate the algorithms, we performed each experiment several times. Specifically, we ran the algorithms on the same instance with 5 different random initial placements. While the initial placement has a strong impact on the results, the final result depends on the progress of the algorithm as much as on the initial placement. If for any test bench one algorithm is better than the other, then it is almost certain better for any initial placement. The variation between the results of different algorithms is consistent for most of the initial placements. In order to compare the algorithms we look at the average results over all initial placements.

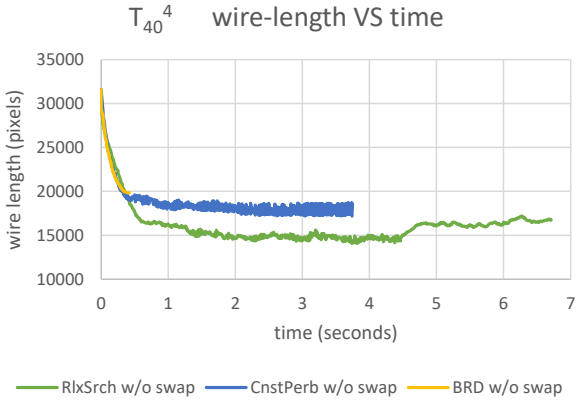

Fig. 8. Progress of total wire-length. No swap moves.

B. Results for Unilateral Deviations

The first experiment we present is a comparison of the three search methods, when applied without and with swap moves. We run each of these variants on our test-bench T_{40}^4, that is, an instance consisting of 40 blocks each with 4 connections. All the algorithms were applied starting from the same initial placement. Figures 8 and 9 present the progress of wire-length over running-time without and with swaps, respectively. Figures 10 and 11 present the progress of area over running time with and without swaps, respectively. We can see that the algorithms reach their stopping criteria at different times and have different progress gradient.

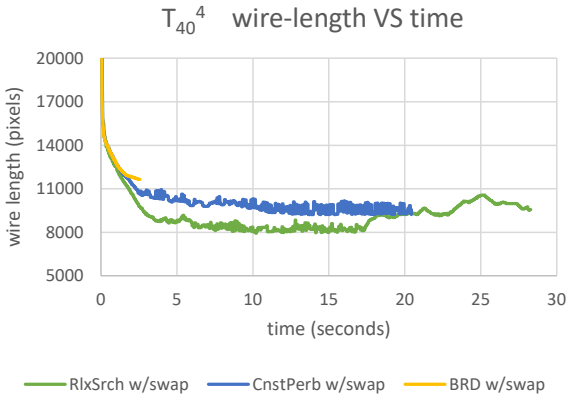

Fig. 9. Progress of total wire-length. Swap moves allowed.

The BRD algorithm is the first to finish - its local search is more restricted and thus, the stopping criteria is reached earlier. Each algorithm has a different progress curve. The gradient of the changes in the cost function according to the time is different. Nevertheless an obvious pattern can be observed: the algorithms that can progress to a worse state

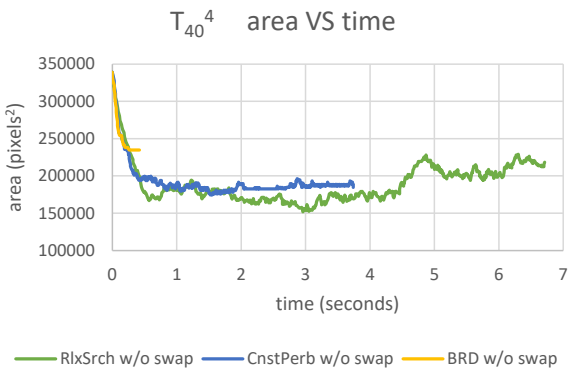

Fig. 10. Progress of Bounding-box area. No swap moves.

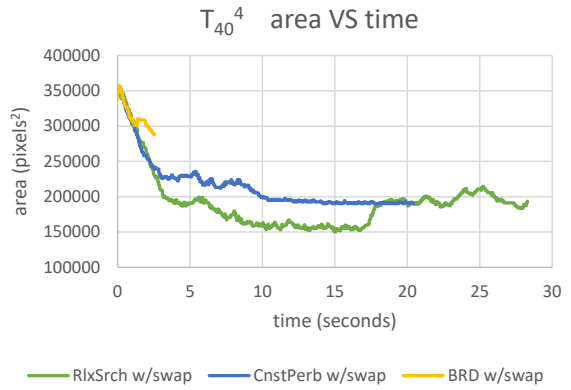

Fig. 11. Progress of Bounding-box area. No swap moves.

tend to continue and improve their result as the algorithm progresses. Such algorithms are able to escape local minima and continue the search, thus obviously the progress gradient is much more moderate. Moreover, we also witness the major influence of swaps. Allowing swaps increases the running time but improves the result. Such behavior leads to having a more moderate gradient of change, but due to the increase in run time, we eventually reach a lower level and a better result.

For BRD-RlxSrch and BRD-ConstPerb we can see peaks and drops in the performance – corresponding to reaching and escaping local minima. In BRD the spikes are limited due to the search method, which always chooses an improving step. Still, the curve is not monotonically decreasing as the improvement are with respect to the deviating block's individual cost, that may conflict with the global cost.

Due to space constraints we do not present the plot presenting the progress of the overlap area in the BRD-RlxSrch algorithm. In both applications, with or without swaps, the overlap area is not increasing or decreasing monotonically. Initially, the algorithm explores non-feasible solutions, that have low wire-length and bounding box area; however, as the fine for overlaps is increased, the placements become more and more overlap-free. The final placement achieved by the relaxed search algorithm is feasible, and its quality is more or less equivalent to the quality achieved by BRD-ConstPerb.

C. *Comparison with the* Simulated Annealing *Algorithm*

In this section we present a comparison of our unilateral deviation algorithms with the Simulated Annealing (SA) algorithm. We present the results by normalizing the SA result to 1. We ran SA and each of our algorithm on all six test-benches, and normalized the result with respect to the corresponding SA result. For example, if on some instance the SA algorithm produces a placement whose area is 12000 pixel^2, and our algorithm produces a placement whose area is 9600 pixel^2, then the result of our algorithm is presented as $9600/12000 = 0.8$.

When presenting the results, we distinguish between two groups of algorithms. The first group includes algorithms that are more run-time oriented than result-oriented, while the other aim to achieve a good result. The first group consists of BRD with and without swaps, and BRD-ConstPerb without swaps, while the second group consists of BRD-RlxSrch with and without swaps, and BRD-ConstPerb with swaps.

Figures 12 and 13 present the results for the total wire-length, and Figures 14 and 15 present the results for the placement area. In all the figures, the horizontal black line represents the result achieved by the SA algorithm.

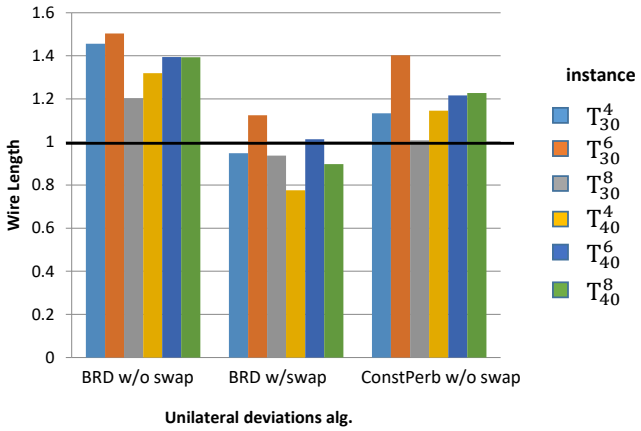

Fig. 12. Wire-length normalized to SA performance – Group I

Figures 16 and 17 present the comparison between the run-times of the algorithms. The algorithm of the first group are indeed much faster than SA. Also, all algorithms when run without swaps are at least 5 times faster than SA.

The experiments reveal that we can achieve better results of both wire-length and placement area while paying with a slightly worse running time. As well as the other way around, that is, slightly worse result can be achieved with a fraction of the running time. We also witness certain algorithms, which on the majority of the test benches, have succeeded to achieve a better result in a lower running time.

D. *Coordinated Deviation*

As detailed in Section II-D, coordinated deviations are performed by a cooperative of blocks. In this section we analyze the results achieved by our search algorithms when

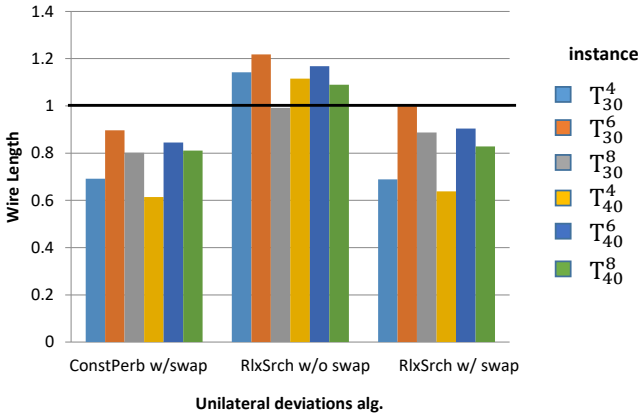

Fig. 13. Wire-length normalized to SA performance – Group II

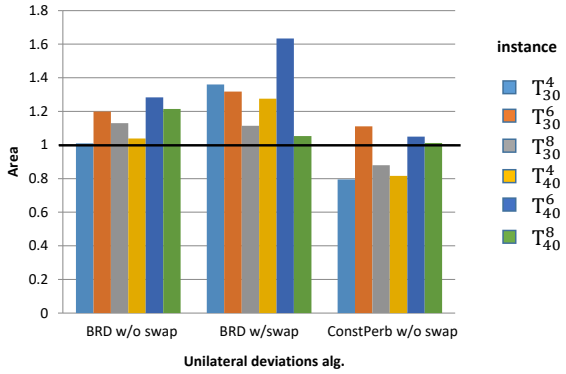

Fig. 14. Area normalized to SA performance – Group I

applied with coordinated deviations. Recall that a deviation of a cooperative is beneficial if the total cost of the cooperative members is strictly reduced. In addition to the local-search method (BRD, BRD-ConstPerb and BRD-RlxSrch), the algorithms are different in the way they determine the active cooperative size and formation. In all the experiments with coordinated deviations, the algorithms were performed on the same instance and the same initial placements.

Due to space constraints, we do not present the results

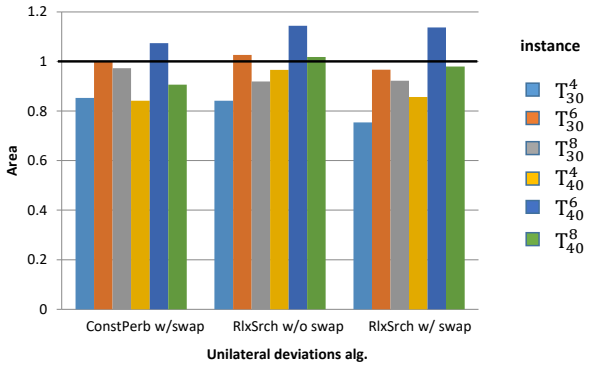

Fig. 15. Area normalized to SA performance – Group II

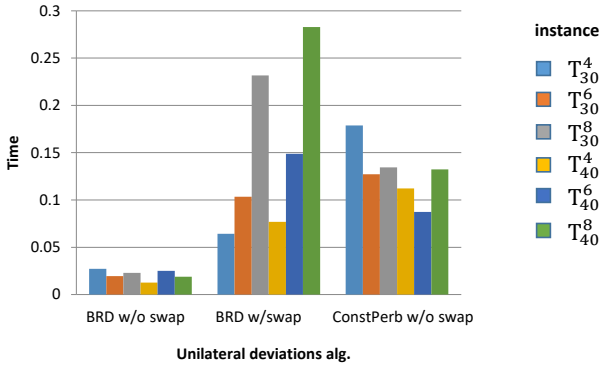

Fig. 16. Running time normalized to SA performance – Group I. The line corresponding to SA (Time = 1) is not shown, as it is way above the shown bars.

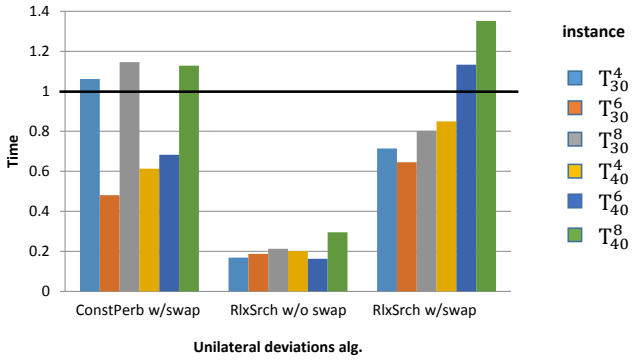

Fig. 17. Running time normalized to SA performance – Group II

using plots, and only summarize our conclusions. Our results show that the cooperative size has the largest influence on the results. The method for determining the cooperative size in each round is not crucial as the predefined limit for the maximal cooperative size. The higher this limit, the better the results. In addition, the experiments do not crown a specific method for selecting the cooperative members - the results vary and depend heavily on the initial placement.

We witnessed a major improvement in the results already with cooperatives of two blocks (compared with unilateral deviations). Further increase in the cooperative size do tend to improve the result, however it involves an exponential increase in the running time. Therefore, the best is to run the algorithm initially with cooperatives of size two, and allow non-frequent rounds in which larger cooperatives are activated. Such executions converge to a final placement much faster than SA, and if performed multiple times, with different initial placements, are expected to produce at least one excellent outcome.

We believe that this unique algorithm, that combines our search methods with a mixture of coordinated and unilateral deviations, is the main result of this work.

IV. Summary and Conclusions

In this work we examined the performance of local search algorithms for the global placement problem in VLSI physical design. Our algorithms are different from common local search algorithms in the way they explore the solution space. Every solution is associated with a global cost (based on its bounding-box area, the total wire-length, and possibly additional parameters), and every component is associated with its individual cost (based on its own placement and connections). We explore the solution space by moving from one solution to another if this move is selfishly beneficial for a single or for a cooperative of components, without considering the effect on the global cost. Best-response dynamics (BRD) is performed until no component has a beneficial migration. We suggested several methods for selecting the component(s) initiating the next step, and for selecting their migration. In order to evaluate our algorithms, we have tested them on various test-benches, and each test-bench was ran with various initial placements.

Based on our experiments, we can distinguish between two approaches for handling the problem. The first approach is to use algorithms with high run-time that also tend to supply good results. Due to their high run-time, these algorithms can only be ran a small number of times (with several different initial placements). The second approach is to use fast algorithms and ran them many times. We expect to get at least one good output. The first approach rely on the algorithms' ability to consider multiple local minima, thanks to their ability to escape local minimum. In the second approach the algorithms tend to stop at the first local minimum they found, however, this is compensated by the high number of runs, with many different initial placements.

Our algorithms, even for instances on which they perform poorly, achieve results not far from SA with only a fraction of its running time. This feature obviously can be very handy when one tries to get a quick estimation of the results achievable for a given instance. We believe that this work has proved the concept of selfish local search to be valid and efficient. Moreover, this concept may be useful in solving additional optimization problems arising in real-life applications.

References

[1] S. N. Adya and I. L. Markov. Combinatorial techniques for mixed-Size placement. *ACM Transactions on Design Automation of Electronic Systems*, Vol. 10(1), DOI:10.1145/1044111.1044116, 2005.

[2] C.J. Alpert, D.P. Mehta, and S. S. Sapatnekar. *Handbook of Algorithms for Physical Design Automation*, Auerbach Publications, 2008.

[3] Y.C. Chang, Y. W. Chang, G. M. Wu, and S. W. Wu. B*-Trees: a new representation for non-slicing floorplans. *Proc. of the 37th Annual Design Automation Conference*, pp. 458-463, DOI: 10.1145/337292.337541, 2000.

[4] C. Chu. Electronic Design Automation: Synthesis, Verification, and Test. Chapter 11: Placement. pp. 635–685. *Springer*, 2009.

[5] E. Even-Dar and Y. Mansour. Fast Convergence of Selfish Rerouting. In *Proc. of SODA*, pp. 772–781, DOI: 10.1145/1070432.1070541, 2005.

[6] M. Feldman, Y. Snappir, and T. Tamir. The Efficiency of Best-Response Dynamics. *The 10th International Symposium on Algorithmic Game Theory (SAGT)*, DOI: 10.1007/978-3-319-66700-3-15, 2017.

[7] S. H. Gerez. Algorithms for VLSI Design Automation. *John Wiley & Sons, Inc.*, NY, USA. 1999.

[8] P. Ghosal and T. Samanta. Thermal-aware Placement of Standard Cells and Gate Arrays: Studies and Observations, *Symposium on VLSI. IEEE Computer Society Annual*, DOI: 10.1109/ISVLSI.2008.37, 2008.

[9] E. Huang and R. E. Korf. Optimal Rectangle Packing: An Absolute Placement Approach. *Journal of Artificial Intelligence Research*, vol. 46:47–87, 2012.

[10] Intel Core $i7$ processors. www.intel.com/content/www/us/en/products/processors/core/i7-processors.html, 2008.

[11] Z. Jiang, H. Chen, T. Chen and Y. Chang. Challenges and Solutions in Modern VLSI Placement, *International Symposium on VLSI Design, Automation and Test (VLSI-DAT)*, DOI: 10.1109/VDAT.2007.373223, 2007.

[12] J. Kleinberg and E. Tardos. Chapter 12: Local Search. *Algorithm Design*. Addison-Wesley, pp. 690–700, 2005.

[13] A. Khachaturyan, S. Semenovskaya, and B. Vainshtein. Statistical-Thermodynamic Approach to Determination of Structure Amplitude Phases, Sov. Phys. Crystallography. Vol. 24(5):519–524, 1979

[14] A. B. Kahng, J. Lienig, I. L. Markov, and J. Hu. VLSI Physical Design: From Graph Partitioning to Timing Closure. *Springer*, DOI: 10.1007/978-90-481-9591-6, 2011.

[15] T. Lengauer. Combinatorial Algorithms for Integrated Circuits. *Wiley-Teubner*, DOI: 10.1007/978-3-322-92106-2, 1990.

[16] Y. Lin, B. Yu, X. Xu, J. Gao, N. Viswanathan, W. Liu, Z. Li, C. J. Alpert, ad D. Z. Pan. MrDP: Multiple-row Detailed Placement of Heterogeneous-sized Cells for Advanced Nodes. *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, DOI: 10.1109/TCAD.2017.2748025, 2016.

[17] I. L Markov, J. Hu, and M. Kim. Progress and Challenges in VLSI Placement Research. *Computer-Aided Design (ICCAD),IEEE/ACM International Conference*, DOI:10.1145/2429384.2429441, 2012.

[18] H. Murata and E. S. Kuh. Sequence-pair based placement method for hard/soft/pre-placed modules. *Proc. of the international symposium on Physical design*, pp. 167–172, DOI:10.1145/274535.274560, 1998.

[19] R. Norman, J. Last, and I. Haas. Solid-state Micrologic Elements, *Solid-State Circuits Conference. Digest of Technical Papers*. pp. 82–83, DOI: 10.1109/ISSCC.1960.1157264, 1960.

[20] S. Pattanaik, S. P. Bhoi, and R. Mohanty. Simulated Annealing Based Placement Algorithms and Research Challenges. *Journal of Global Research in Computer Science*. Vol. 3(6), 2012.

[21] N. Quinn and M. Breuer. A forced directed component placement procedure for printed circuit boards. *IEEE Transactions on Circuits and Systems*, Vol. 26(6), 1979.

[22] S. Reda and A. Chowdhary. Effective linear programming based placement methods. *Proc. of the international symposium on Physical design*, pp. 186–191, DOI:10.1.1.83.2416, 2006.

[23] R. A. Rutenbar. Simulated Annealing Algorithms: An Overview. *IEEE Circuits and Devices Magazine*, Vol.5(1):19 - 26, 1989.

[24] K. Shahookar and P. Mazumder. VLSI Cell Placement Techniques. *ACM Computing Surveys*, Vol. 23(2):143-220, 1991.

[25] Z. Yang and S. Areibi. Global Placement Techniques for VLSI Physical Design Automation. *Proc. of the 15th Intl. Conf. on Computer Applications in Industry and Engineering*, 2002.