MINIMIZATION OF VLSI FLOORPLAN USING HYBRID CUCKOO SEARCH AND PSO

Dr S.Kishore Reddy,
Associate Professor, Department of Electrical & Computer Engineering,
School of Engineering, Adama Science & Technology University, Ethiopia.

Abstract: Floorplanning is a major issue in the very large-scale integrated (VLSI) circuit design automation. It helps to determine the size, reliability, performance and area of the chip. The main target of VLSI floorplanning is to minimize both area and wire length simultaneously. This paper proposes a new floorplan representation called Corner List (CL) representation for non-slicing floorplan. Since VLSI floorplanning is the Non-deterministic Polynomial-time Hard Problem (NP-Hard), a hybrid of Cuckoo Search (CS) and Particle Swarm Optimization (PSO) is used to solve this problem. Corner list representation is used to represent the initial floorplan and utilized proposed algorithm to find out the optimal placement solution. Experimental results on Microelectronics Centre of North Carolina (MCNC) and Gigascale Systems Research Center (GSRC) benchmark circuits show that the proposed algorithm is effective and well optimized floorplan is obtained.

Keywords: Corner List representation, Cuckoo Search algorithm, Non-deterministic Polynomial-time hard problem, Non-slicing floorplan, Particle Swarm Optimization, VLSI floorplanning.

1. Introduction

Floorplanning is becoming more important for very large-scale integration (VLSI) physical design. Floorplanning is the process of planning the arrangement of modules in such a manner that area and interconnection wirelength should be minimized. According to Moore’s Law, the number of transistors gets doubled for every eighteen months. This results in increasing the complexity of the circuits. The feature size of the IC is considerably scaled down, which results in an increase in search complexity. This makes the IC design more complex and corresponds to the NP Hard Problem.

Floorplanning can be represented by using a different approach such as graph-based, tree based or sequence-based representation. VLSI floorplanning is an NP-Hard problem and it can be solved using heuristic and meta-heuristic algorithm. Genetic algorithm has been proved to be an efficient method to solve VLSI floorplanning. Another heuristic algorithm, Particle Swarm Optimization can also be applied to solve NP-Hard Problem and VLSI floorplan experimental results prove that PSO has better searching ability than Simulated Annealing (SA). The advantage of the PSO algorithm is that it avoids falling into local minimum and exhibits a rapid convergence rate. The main drawback of the PSO algorithm is that it has weak local search ability. In order to overcome this drawback, PSO algorithm is hybridized with Cuckoo search algorithm.

Cuckoo Search (CS), originally introduced by Yang and Deb (2009) in 2009, is a new meta-heuristic search algorithm inspired by the behaviour of the bird cuckoo. CS uses Lévy flights rather than simple isotropic random walks. The advantage of CS is that it has fewer parameters to be fine-tuned when compared to genetic algorithm.

The paper is structured as follows. Section 2 describes the problem statement. Section 3 depicts the Corner list representation. Section 4 reviews the mechanism of Hybrid PSOCS. Section 5 discusses experimental results. Finally, Section 6 concludes the paper.

2. Problem statement

Let M be the set of modules represented by M= \{m_1, m_2, ..., m_N\}, where N is the number of modules. Each module m_i is represented by (W_i, H_i), where 1≤i≤N, W_i is width of the module m_i and H_i is the height of the module m_i. The aspect ratio of m_i is defined as H_i / W_i. The area A_i of the module m_i is given by W_i * H_i. There are three different kinds of rectangular modules namely soft modules, hard modules and pre-placed modules. The soft modules have variable aspect ratio within specified range and fixed area. In hard modules, both area and aspect ratio are fixed structure. The detailed description is given as follows:

2.1 Slicing floorplan: A slicing floorplan is obtained by cutting the floorplan either horizontally or vertically repetitively. Fig.1(a) represents slicing floorplan. A slicing tree is a binary tree. The pre-placed module is a one in which modules coordinates are given by the floorplanner. Let H denotes set of hard modules, S
denotes set of soft modules and P denotes set of preplaced modules. Let M be the union of these three sets of modules. The representation of floorplanning can be done in two layout forms, namely the slicing structure and non-slicing which is used to represent a slicing floorplan. Generally, there are two cut types, H and V. The H (V) represents floorplan horizontal (vertical) cut. Fig.1(b) shows a slicing tree of Fig.1(a).

![Fig.1 Slicing floorplan and its slicing tree](image)

**2.2 Non slicing floorplan:** Non slicing floorplan is more common than slicing floorplan. All the children of the given cell cannot be obtained by bisecting the floorplan. This is called non-slicing floorplan. Horizontal constraint graph and vertical constraint graph can be used to model a non-slicing floorplan. In a constraint graph, a node represents a module.

![Fig.2 Non-slicing floorplan and its constraint graphs](image)

**3. Corner list representation**

Corner list (CL) is a sequence-based representation that is used to represent the initial floorplan. It is an efficient and effective method for non slicing floorplan representation. CL can be able to perform transformation between floorplan layout and representation of floorplan is n times faster than moving block sequence. Time complexity and search space of corner sequence is similar to corner list, but CL has more corners than that of corner sequence. So, CL representation is better when compared to moving block sequence and corner sequence. The technical term used in CL algorithm is given as follows

**Definition 1:** Consider two dummy modules mb and ml. These modules are considered to be at bottom and left boundaries of floorplan, having dimensions of \([\infty, 0]\) and \([0, \infty]\) respectively. Set of modules is represented by M, and arbitrary corner in the set of corner list is given by C.

**Definition 2:** Consider a set of modules S, where S\(\subset M\), and any arbitrary corner in the set of corner list C, represented by \([C_b, C_l]\), then \(C = \{C_b, C_l\} \subset (S \cup \{m_b, m_l\})\). Here l and b represents left and bottom respectively.

**Definition 3:** Consider an arbitrary corner \([C_b, C_l]\), the co-ordinates of corners \(C_l\) and \(C_b\) are given as \((xTR-l, yTR-l)\) and \((xTR-b, yTR-b)\) respectively, then, the corner is realizable under the conditions of \(xTR-b \geq xTR-l\) and \(yTR-l \geq yTR-b\). Here TR represents Top-Right.

**Definition 4:** Corner list generally has two tuples: first tuple \(\Omega\) is used for representing the order of modules chosen. Second tuple \(\chi\) is used to denote the corner list array. The corner list array consists of the sequence of the corners chosen by the corresponding modules.

\[
\text{CL} = \left\{ \Omega, \chi \right\} \tag{1}
\]

\(\Omega = \{\Omega_1, \Omega_2, \ldots, \Omega_n\} : \Omega_i = m_i \in M, \quad \Omega_i \neq \Omega_j, \quad 0 \leq i, j \leq n,\)

\(\chi = \{x_1, x_2, \ldots, x_n\} : 0 \leq i, j \leq n,\)

\(x_i = \{m_{\text{left}}, m_{\text{bottom}}\} : i = 1\)

\(x_i = \{c_{\text{left}} - i, c_{\text{bottom}} - i\} : 2 \leq i \leq n\)

**Corner list placement:**

![Corner list placement](image)
Two dummy modules mb and ml are located at the bottom and left boundaries respectively, corresponding to two lists: one for module and another for corners. Consider the module sequence (m4, m1, m2, m3), where module m4 is placed on the corner (mb, ml). The placement of the first module generates two corners (m4, ml) and (mb, m4) which is shown in Fig.3(a). Corner list array is updated containing two corners. Placing of the second module is done by randomly selecting any one of the corner from the list. Fig.3(b) shows that m1 is placed on the corner (m4, ml) and the selected corner is deleted from the array. Corner list is updated for every module placement. This process will be repeated for remaining modules placement as shown in Fig.3(c) and Fig.3(d).

CL representation has more corners for selection than corner sequence representation. The property and theorems of CL are given below:

### 3.1. Properties of CL

- Consider module 1 and 2 having same width and height are placed adjacent to each other, in such a way that module 2 uses the bottom corner of module 1, then top left corner of module 2 is not considered.
- If the numbers of modules placed onto the floorplan are n, there will be (n+1) possible corners in the corner list if and only if all modules width and height are not same. If two modules of equal height are placed adjacent to each other, then it is considered to be single module.

### 3.2. Theorems of CL

**Theorem 1:** If the number of modules placed onto the floorplan are n, then the solution space of the corner list will be bounded by $O((n!)^2)$.

**Proof:** Consider that there are n modules to be placed onto the floorplan, hence there is $n!$ of permutations in the placement sequence. By using property, there are always $(n + 1)$ possible corners in the corner list when the number of modules placed onto the floorplan is n. The solution search space of the CL is bounded by:

$$\frac{n!}{(2)}$$

**Theorem 2:** If the number of modules onto the floorplan are n, the evaluation time of corner list is bounded by O(n) time.

**Proof:** Consider that there are n modules in corner sequence, the time complexity to insert the modules onto the floorplan is O(n). Deletion of corner from corner list takes less time than O(n). So, the time complexity of corner list placement is O(n).

### 4. Proposed algorithm

#### 4.1. Cuckoo Search Algorithm:

Cuckoo search algorithm is a new nature-inspired meta-heuristic algorithm, developed by Xin-She Yang and Suash Deb in 2009. Normally, cuckoo birds laid their eggs in the nest of other host birds, because cuckoo never builds...
their own nests. Once host bird recognises cuckoo egg, the host bird either destroys the cuckoo eggs out, or host bird simply abandons its own nest and builds a new nest. Cuckoo bird choose a nest where host bird laid their eggs just before. Cuckoo egg hatches earlier as compared to that of host egg. After the cuckoo egg hatches, cuckoo chick evict host egg, which increases cuckoo’s share of food provided by the host bird.

Cuckoo search algorithm is based on the following three idealized rules.

1. Each cuckoo lays one egg at a time, and dumps it in a randomly chosen nest.
2. The best nest with high quality of eggs or solution will carry over to the next generation.
3. The number of available hosts nests is fixed, and a host can discover an alien egg with a probability \( p_a \epsilon (0,1) \).

Let us consider new solution generated be \( x_i(t+1) \) and \( i \) represents cuckoo. Lévy flight is performed by using the following formula,

\[
x_i(t+1) = x_i(t) + \alpha \cdot \text{Lévy}(\lambda) \tag{3}
\]

Here, \( \alpha \) is the step size and it should be greater than 0. The Lévy flight helps to provide a random walk whose random step length is calculated from the Lévy distribution,

\[
\text{Lévy} \sim u = t^{-\lambda} \tag{4}
\]

Some of the representations of the cuckoo search algorithm are each egg in a nest represents a solution and cuckoo can lay only one egg and to use the new and potentially better solutions to replace a bad solution in the nests. For simplicity purpose, each nest has only one egg. It uses a balanced combination of a local random walk and the global explorative random walk, controlled by a switching parameter \( p_a \).

4.2. Particle Swarm Optimization (PSO): PSO is a population based optimization method and it can be used to optimize a problem by several iterations in order to improve a candidate solution with regard to quality. PSO optimizes a problem by using a population of candidate solutions (from corner list). Each particle represents an individual in the population. By adopting PSO algorithm, the particles are moved around in the search-space according to the particle’s position and velocity. The particle’s best position so far is said to be pBest. The best of pBest value among all the particles is considered as gBest. The velocity and position of each particle is updated according to the following equations.

\[
V_{i+1} = W_i \cdot V_i + C_1 \cdot \text{rand} ( ) \cdot (p\text{Best} - X_i) + C_2 \cdot \text{rand} ( ) \cdot (g\text{Best} - X_i) \tag{5}
\]

\[
X_{i+1} = X_i + V_{i+1} \tag{6}
\]

where \( t \) is the iteration index, \( W \) is the inertia weight, \( C_1 \) and \( C_2 \) are two positive constants, called acceleration constants, \( \text{rand} ( ) \) is random function within the range \([0,1]\). The new velocity of each particle is calculated by using equation (5) based on its previous velocity and the personal best location and the global best location of the population. The particle’s position is updated by using equation (6).

Acceleration coefficient of each particle is calculated using equation (7) and (8).

\[
c_1 = c_{Iter} (c_{1e} - c_{1s}) / \text{MAXITER} + c_{1s} \tag{7}
\]

\[
c_2 = c_{Iter} (c_{2e} - c_{2s}) / \text{MAXITER} + c_{2s} \tag{8}
\]

where \( c_{Iter} \) is the current iteration number and \( \text{MAXITER} \) is the maximum number of allowable iteration, \( c_{1e}, c_{2e} \) represent the final value of the \( c_1 \) and \( c_2 \) and \( c_{1s}, c_{2s} \) represent the initial values of the \( c_1 \) and \( c_2 \).

The pseudo code of the PSO is as follows:

Create and initialize a corner list and particle
Repeat until maximum iterations
For each particle
Calculate fitness value from corner list
If fitness value is better than pBest
Set fitness value as pBest
Choose gBest from pBest of all particles
Calculate the particle velocity by using equation (5)
Update the particle position by using equation (6)
End

The flowchart of PSO algorithm is given below:

4.3 Hybrid PSOCS Algorithm Description
STEP 1: Load the modules input and initialize the parameter of the PSO algorithm.

STEP 2: Generate an initial population, initialize the position and velocity of each particle, and calculate the Pbest of each particle and the Gbest population.

STEP 3: Calculate the fitness value of each particle by equation (5) and (6).

STEP 4: Check each particle, if its fitness value is better than Pbest, update its Pbest with the fitness value.

STEP 5: Check each particle, if its fitness value is better than population’s Gbest, update its Gbest with the fitness value.

STEP 6: Adjust the position and velocity of each particle according to equations (1) and (2).

STEP 7: If termination condition is satisfied, the algorithm stops; otherwise, go to step 3.

Fig. 7 Floorplanning based on Hybrid CS/PSO

5. Experimental results

5.1 Parameter settings for Algorithm: All the modules are considered as hard modules. The parameters of the PSO algorithm are set as follows: W=0.95, c1=2.1 and c2=2.1, number of particles will be in the range of 10 to 40 and it can be 10, which is large enough to get good results. The parameters of CS algorithm are set as follows: the probability or discovery rate of cuckoo egg p_a=0.2 yields good result in terms of convergence rate. The number of host nest, which is also known as population size, n=15 is efficient for most optimization problems.

5.2 Performance of Algorithm: The MCNC benchmark circuits namely apte, xerox, hp, ami33 and ami49 are considered. Each benchmark circuit has number of modules, nets, I/O pad, pins and total area of all the modules as shown in Table 1.

<table>
<thead>
<tr>
<th>CIRCUIT</th>
<th>MODULE S</th>
<th>NET S</th>
<th>I/O PAD</th>
<th>PIN S</th>
<th>TOTAL AREA</th>
</tr>
</thead>
<tbody>
<tr>
<td>apte</td>
<td>9</td>
<td>97</td>
<td>73</td>
<td>287</td>
<td>46.561</td>
</tr>
<tr>
<td>xerox</td>
<td>10</td>
<td>203</td>
<td>2</td>
<td>698</td>
<td>19.350</td>
</tr>
<tr>
<td>hp</td>
<td>11</td>
<td>83</td>
<td>45</td>
<td>309</td>
<td>8.8306</td>
</tr>
<tr>
<td>ami33</td>
<td>33</td>
<td>123</td>
<td>42</td>
<td>522</td>
<td>1.1564</td>
</tr>
<tr>
<td>ami49</td>
<td>49</td>
<td>408</td>
<td>22</td>
<td>953</td>
<td>35.445</td>
</tr>
</tbody>
</table>

5.3 Area and wirelength minimization: For multi-objective optimization, both area and interconnection wirelength should be minimized simultaneously. The fitness function for both area and wirelength optimization is given by,

$$y(x) = \alpha \cdot f(x) + \beta \cdot g(x)$$

(9)

where \(\alpha\) and \(\beta\) are constant weight values in the range of 0 to 1, \(f(x)\) represents the area of the particle \(x\), \(g(x)\) represents the wirelength of particle \(x\). The wirelength of particle \(x\) is calculated by using Half-Perimeter Wire Length (HPWL). The HPWL of the net \(k\) is calculated by using the following formula:

$$L_p = (X_{max} - X_{min}) + (Y_{max} - Y_{min})$$

(10)

The optimized results are compared with the results reported by (Chen 2011) which is shown in the Table 2. The simulation results of optimized area and wirelength of MCNC Benchmark circuits are shown in the Fig.8, 9, 10, 11 respectively. From Table 2, it can be seen that, in terms of area, the proposed algorithm performs better result than PSO for all MCNC benchmark circuits. In terms of area, the proposed algorithm gives better result when compared with enhanced o-tree for MCNC apte, xerox and ami33. But, enhanced o-tree produces good optimal result for
MCNC ami49 when compared with the proposed algorithm. Table 3 shows the comparison of proposed hybrid CS/PSO algorithm with other algorithms in terms of area and wirelength for MCNC benchmark circuits.

![Simulation result on MCNC apte](image1)
![Simulation result on MCNC xerox](image2)

Table 2. Comparison of results on MCNC benchmark circuits with $\alpha=1$ and $\beta=0$

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>apte</th>
<th>Area (mm²)</th>
<th>DS(%)</th>
<th>ami33</th>
<th>Area (mm²)</th>
<th>DS(%)</th>
<th>ami49</th>
<th>Area (mm²)</th>
<th>DS(%)</th>
<th>xerox</th>
<th>Area (mm²)</th>
<th>DS(%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Enhanced O-tree</td>
<td>46.92</td>
<td>0.77</td>
<td>1.20</td>
<td>3.77</td>
<td>37.73</td>
<td>6.45</td>
<td>20.21</td>
<td>4.44</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SA with B* Tree</td>
<td>47.30</td>
<td>1.58</td>
<td>1.36</td>
<td>17.6</td>
<td>43.34</td>
<td>22.27</td>
<td>20.47</td>
<td>5.78</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PSO</td>
<td>46.92</td>
<td>0.77</td>
<td>1.28</td>
<td>10.69</td>
<td>41.01</td>
<td>15.69</td>
<td>20.47</td>
<td>5.78</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>HPSO/CS with CL</td>
<td>46.83</td>
<td>0.58</td>
<td>1.19</td>
<td>2.59</td>
<td>40.98</td>
<td>15.30</td>
<td>20.14</td>
<td>4.08</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Note: DS(%) - percentage of deadspace in the total layout area

Table 3. Comparison of results of MCNC benchmark circuits with $\alpha=0.5$ and $\beta=0.5$

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>apte</th>
<th>Area (mm²)</th>
<th>Wire (m)</th>
<th>ami33</th>
<th>Area (mm²)</th>
<th>Wire (m)</th>
<th>ami49</th>
<th>Area (mm²)</th>
<th>Wire (m)</th>
<th>xerox</th>
<th>Area (mm²)</th>
<th>Wire (m)</th>
</tr>
</thead>
<tbody>
<tr>
<td>O-tree</td>
<td>51.9</td>
<td>321</td>
<td>1.28</td>
<td>51</td>
<td>39.6</td>
<td>689</td>
<td>20.4</td>
<td>477</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Enhanced O-</td>
<td>52.0</td>
<td>321</td>
<td>1.30</td>
<td>52</td>
<td>39.9</td>
<td>703</td>
<td>20.4</td>
<td>381</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
6. Conclusion
In this paper, we have proposed hybrid of cuckoo search and PSO algorithm in order to solve non-slicing floorplan in efficient manner. Corner list representation is a new floorplan representation and it is used to represent non-slicing floorplan. The experimental results for MCNC benchmark circuits demonstrated that the proposed algorithm can achieve the optimal result for hard modules placement. The future work will focus on the parameter related to some constraints.

REFERENCES