Chapter IV

Development of Algorithms based on Evolutionary Approach

4.1 Introduction
In this work, five evolutionary approaches namely Hybrid Artificial Bee Colony Optimization with Simulated Annealing for Circuit Partitioning (HABCSACP), Non Revisited Evolutionary Approach for Circuit Partitioning (NRECP) algorithm, Extended-Non Revisited Evolutionary Approach for Circuit Partitioning algorithm (E-NRECP) , Soft Computing Algorithm for Partitioning (SCAP) Approach, DNA Based Approach for Circuit Partitioning (DBACP), for the optimization of VLSI netlist bi-Partitioning are proposed. These approaches are based on soft computing, DNA computing, simulated annealing, and artificial bee colony and trie data structures. For simulation work a set of spp-benchmark series is used to evaluate the efficiency of the algorithms.

4.2 HABCSACP (Hybrid Artificial Bee Colony Optimization with Simulated Annealing for Circuit Partitioning) Approach
The Artificial Bee Colony (ABC) algorithm proposed by Karaboga( Karaboga, D., 2005) is one of the swarm intelligence algorithm inspired by the intelligent food foraging behavior of honey bees. Due to the superior performance of ABC and its variants in comparison to many other existing evolutionary and swarm intelligence algorithms, this algorithm has raised great interest amongst researchers in recent years (Karaboga, D., Basturk, B., 2008). Over the last few years, ABC has been successfully applied to wide and diverse range of problems, such as numerical optimization, discrete optimization, multi-objective optimization, software testing and so on (D. Karaboga, B. Akay, 2009). However according to Zhu, and Kwong, the basic ABC algorithm is good at exploration but poor at exploitation (Zhu, G. and Kwong, S., 2010). Exploration and exploitation are necessary for the population-based optimization algorithms. In practice, the exploration and exploitation contradict each other, and in order to achieve good optimization performance, the two abilities should be perfectly balanced.
Simulated annealing (SA) algorithm is a general-purpose stochastic optimization method that has proven to be a practical method for solving combinatorially large optimization problems (Kirkpatrick et al., 1983). Simulated annealing is able to explore the full solution space and can escape from the local optima. In Simulated annealing, uphill movements fulfill the exploration capability and downhill ones play the role of the exploiter.

In this work a hybrid the HABCSACP (Hybrid Artificial Bee Colony Optimization with Simulated Annealing for Circuit Partitioning) approach is proposed which uses Artificial Bee Colony algorithm having good exploration capabilities in searching optimal solution and simulated annealing for local improvement of solution. The exploration/exploitation balancing strategy of Simulated Annealing incorporated into the original ABC algorithm helps to improve the quality of solution. The algorithm begins with random generation of colony of bees. The count of number of Employed bees (Nemployed) and Onlooker bees (Nonlooker) , both is set equal to the population size (figure 4.1).

**Initialize pop_size** - Number of solutions in the population

**maxCycle** - Total no. of iterations

**limit** - Total number of trials after which the solution is rejected,

**Ichrom** - Total number of modules/vertices of circuit i.e. length of solution

```plaintext
Initialise the parameters :pop_size, maxcycle, limit, lchrom
Set Nemployed =Nonlooker= pop_size
Set limit, maxcycle
Initialize (pop_size,lchrom);
for iter=1:maxcycle
Sendemployedbees(pop_size,lchrom);
Calculateprobabilities(pop_size);
Sendonlooker(pop_size,lchrom);
[Globalparam,globalmin]=Memorizebestsource(pop_size);
Local_improve_SA(Globalparam,globalmin);
SendScoutBees(pop_size,limit,lchrom);
End
Write : Globalparam, Globalmin //resultant solution
End.
```

**Figure 4.1: Pseudo code of the Proposed Algorithm**
4.2.1 Detailed steps of HABCSACP Algorithm

**Step 1:** *Initialize* \((pop\_size, lchrom)\): Randomly generate an initial population \(P\) of size \(pop\_size\) with set of feasible solutions (i.e. balanced partition w.r.t weight of vertices). Read the input files and convert it into netlist format. Calculate the fitness value of each solution in the population using the net cut evaluation mechanism. For a net cut evaluation a multiword mask of size of the chromosome is pre computed for each net. If a cell is connected to net, the corresponding bit position is set.

**Step 2:** Repeat the following steps for specified number of iterations i.e. maxcycle.

**Step 3:** *Sendemployedbees* \((pop\_size, lchrom)\): For each solution in the population produce a modification on the one existing. If the fitness value of new solution is higher than that of the previous one then memorizes the new solution and forget the old one. Otherwise keep the position the previous one (figure 4.2)

![Algorithm Steps](image)

**Figure 4.2: Steps of Sendemployedbees()**

Every employed bee in the population chooses a neighborhood solution. The neighborhood solution is generated by Inversion operator which has a strong selective pressure with an adaptive operator. The features of this adaptive operator are the number of inversions applied to a single individual and the segment to be inverted is determined by another (randomly selected) individual. The probability \(p\) of generating random inversion, and the number of iterations in the termination condition
Step 4: Calculate probabilities(pop_size); The probability value is calculated for each solution is calculated using the following formula $S_i.p$

$$S_i.p = \frac{f_i}{\sum_{i=1}^{\text{pop_size}} f_i}$$

where pop_size - is the population size and $f_i$ is the fitness value of $i^{th}$ solution

Step 5: SendOnlookerbees(pop_size,lchrom): At the third stage, an onlooker prefers a solution depending on the probability value of the solution(figure 4.3).

```
For each solution Si for i ≠ [1: Nemployed]
If(Rand() < Si.p )
    then
        SNi = Inversion operator (Si )  //Generate neighbourhood solution SNI.
        Calculate the fitness value of SNI.
        If fitness(SNi)>fitness(Si)
            Si=SNi
            Si.trial=0
        Else
            Si.trial= Si.trial+1
        End of if statement
    End of for statement
```

Figure 4.3: Steps of SendOnlookerbees() 

Step 6: Local_improve_SA(Globalparam,globalmin) :Simulated annealing being a simulated annealing algorithm proposed by Kirkpatrick et al. is incorporated into the algorithm where the resultant solution is further improved by finding nearby solutions. The simulated annealing algorithm gives optimal results for circuit partitioning by locally improving the solution.

Step 7: SendScoutBees(pop_size,limit,lchrom): If the solution trial reaches the limit value then it is abandoned and new solution is generated randomly and replaced with the abandoned one (figure 4.4).
index = 1;

For each solution Si for i ∈ [2: pop_size]
    if ( Si.trial > Sindex. trial )
        index= i;
    end of if statement
end of for statement

if (Sindex. trial >=limit)
    popnext=generate_new_solution (index,lchrom);
end of if statement
end of for statement

Figure 4.4: Steps of SendScoutbees() 

After specified number of iterations the final solution is printed and runtime is calculated for multiple number of partitioning instance groups in each size range of test circuit instances.

4.3 Non Revisited Evolutionary Approach for Partitioning (NRECP)

Various Evolutionary approaches suffer from a major problem of revisiting the already evaluated solutions which may lead to premature convergence. This in particular occurs when the selection pressure is high, the population size is moderate, or the variation operators do not introduce much improvement. In the worst case, the population's diversity drops sharply and the evolutionary approach gets stuck by quickly filling the population with clones of the better fit individuals, making the algorithm difficult to escape from local optima (Zaharie and Zamfirache, 2006). Various techniques like Binary search trees, Binary space partition trees, heap, hash tables and cache memory have been used to memorize the search points or solutions to the problem that it visits in its lifetime (Yuen and Chow, 2007). Though all these duplicate removal method reduce the run time of a GA, they require huge data structures and a significant amount of time is spent for memory look ups. In this work a Non Revisited Evolutionary Approach for Circuit Partitioning (NRECP) algorithm is proposed that maintains the diversity through adaptive mutation, avoids revisits, without any additional memory overheads.
In the NRECP approach the archive is consulted each time after a new solution is generated by crossover. The following describes the various components of the algorithm. Let $K$ be the number of sub circuits into which the circuit with graph $G$ is divided, and let $n$ ($n=|P|$) be the number of logic gates of the original circuit; then each solution is represented by an array $S$ of $n$ elements as $S = p_1, p_2, p_3 \ldots \ldots p_n$ with $S = p_i \in [0, k - 1]$ where the $p_i$ element in array $S$ represents the subcircuit to which the logic gate $i$ belongs to. In this proposed algorithm the value of $K$ is 2. The circuit graph is traversed in a breadth-first way for ordering of vertices.

4.3.1 Pseudo code of the algorithm

The various steps of the algorithm are:

**Step 1:** Randomly generate an initial population $P$ with set of feasible solutions (reading .are files).

**Step 2:** Insert the population in the binary tries

**Step 3:** Read the input file (.net files) and convert it into netlist format. Calculate the fitness value of each solution in the population using the net cut evaluation mechanism. For a net cut evaluation a multiword mask of size of the chromosome is pre computed for each net. If a cell is connected to net, the corresponding bit position is set.

$$M_{ij} = \begin{cases} 1 & \text{if } C_j \in N_i \\ 0 & \text{otherwise} \end{cases}$$

Where $C_j$ is the $j^{th}$ cell in order.

$M_{ij}$ is the mask for $N_i$ net and $j^{th}$ bit position of $M_i$.

The value of $CM_i$ and $\bar{C}M_i$ is evaluated. If both values are nonzero then net is present in both partitions, hence a cut. Otherwise no cut.
**Step 4:** After evaluating all the population with the fitness function, the individuals of the next generation will be chosen by a proportional criterion, called roulette proportional criterion, which will guarantee that the best individuals of the current generation have more possibilities of passing to the next one.

**Step 5:** The cyclic crossover operation is applied at random points on the selected individuals to generate two offsprings.

**Step 6:** Check the feasibility of new solutions by examining whether the solutions satisfy the balance constraints. If not then repeatedly mutate the solution by inverting the bit positions at random point until getting feasible solution. In NRECP algorithm, the operators are based on the moves of gates between neighborhood partitions. The variants are deterministic, pseudo-random and random.

**Step 7:** Check the presence of the newly generated offsprings in solution archive. If any of the newly generated solution already present in archive then solution archive generates a new unvisited solution by level order traversal of binary trie structure and also insert that solution into solution archive. Otherwise insert the new offspring into archive.

**Step 8:** If the new solution is given by archive then check the feasibility of new solutions. If satisfying the balance constraint then accept the solution. Otherwise again retrieve the new solution from the archive. This process is repeated until the solution satisfy the balance constraint.

**Step 9:** The algorithm is repeated for some set of generations.

The trie structure has been pruned to cut out the number of comparisons for searching and transforming the solution, which further keep a check on maximum comparisons for searches.
4.4 E-NRECP (Extended-Non Revisited Evolutionary Approach for Circuit Partitioning algorithm)

The work proposed Extended-Non Revisited Evolutionary Approach for Circuit Partitioning algorithm (E-NRECP) is a variant of the NRECP approach where the solution archive provides random solution which is further repaired and newly transformed solution is again checked for revisit in the trie. The whole procedure is repeated until the solution obtained is unconsidered feasible solution.

The proposed algorithm uses a smart solution archive which stores efficiently all visited solutions during the evolutionary process by pruning the trie and at same time intelligently transforming the revisited solutions into yet unvisited feasible solutions giving balanced weighted partitions. The trie implementation in the proposed algorithm stores additional information regarding the sum of weights of the nodes till that node. The following describes the various components of the algorithm.

**Step 1:** In the E-NRECP algorithm, the population is encoded as a set of binary strings of zeros and ones. Let \( n \) be the number of logic gates of the original circuit; then each solution is represented by an array \( S \) of \( n \) elements as \( S=p_1, p_2, p_3, \ldots, p_n \) with \( p_i \in [0, 1] \) where the \( p_i \) element in array \( S \) represents the subcircuit to which the logic gate \( i \) belongs to. The algorithm begins with random generation of an initial population \( P \) which satisfies the balance constraints generating a set of feasible solutions by reading .are files.

**Step 2:** The population is inserted in the binary trie

**Step 3:** Calculate the fitness value of each solution in the population using the net cut evaluation mechanism by reading the input file (.net files) and convert it into netlist format.

**Step 4:** The crossover operation is applied at random points on the two worst selected individuals (based on fitness) to generate two offsprings.
Step 5: The newly generated offsprings are made feasible by satisfying the balance constraints, if required.

Step 6: Check for revisit of newly generated offsprings in trie. If there is any revisit, then solution archive generates a new unconsidered feasible solution by using level order traversal of binary trie structure and using additional information regarding the sum of weights of the nodes till that node. That new unconsidered solution is also inserted into the trie. The trie is also pruned during insertion process to efficiently utilize the memory space and to reduce the number of search operations during the search operation in trie.

Step 7: The algorithm is repeated for some set of generations.

In the E-NRECP algorithm the trie structure always generates a new feasible solution which prevents the algorithm for getting stuck into local optimum and at the same time allowing the algorithm to explore the search space of only feasible solutions.

4.5 SCAP (Soft Computing Approach to circuit Partitioning algorithm)

As the general partitioning problem is NP complete, various heuristic algorithms for partitioning have been developed. Kernighan and Lin proposed a graph bisection algorithm for a graph having the time complexity of $O(n^3)$, which is considered too high even for moderate size problems (Kernighan, B. W. and Lin, S, 1970). However, the similar bisecting approach can be used much more effectively by applying the SCA computation to circuit partitioning problem. This paper presents such an approach, which makes use of SCA computing to solve the instance of partitioning problem. The advantage of the proposed method is that output values are computed and stored parallel.

To solve the instance of Partitioning problem with $G = (V, E)$ ($|V| = n$) start with $2^n$ identical single stranded SCA memory strands each with $n+m$ bit regions. The first $n$ bit regions will represent the presence/absence of vertex in the first partition and the rest $m$ bit regions will represent the presence/absence of an edge crossing the partition.
The algorithm begins with an initialization phase during which each individual in the population is defined by a uniformly distributed random point in the solution space. Iteration of the SCA begins evaluating the fitness of the current generation (Kim, Y. H. & Moon, B. R., 2004). The application of crossover and mutation to the individuals creates a new generation of offspring, Solution representation. Let $K$ be the number of sub circuits into which the circuit with graph $G$ is divided, and let $n$ ($n=|P|$) be the number of logic gates of the original circuit; then each solution is represented by an array $S$ of $n$ elements as $S=A_1, A_2, A_3,\ldots, A_n$ with $Ai$ in $[1,\ldots, K]$ where the $Ai$ element in array $S$ represents the subcircuit to which the logic gate $i$ belongs to, Initial solution. Let $N$ be the population size, the algorithm has been run using two different initial solutions, $s_1$ or $s_2$. They have been obtained using fast algorithms that assign $n/K$ strongly connected nodes to each partition.

Partitioning is done in which the circuit graph is traversed in a depth-first way starting from the inputs, obtains the initial solution $s_1$. The fitness function is obtained directly from the cost function, Selection criteria. After evaluating all the population with the fitness function, the individuals of the next generation will be chosen by a proportional criterion, called roulette proportional criterion, which will guarantee that the best individuals of the current generation have more possibilities of passing to the next one.

In the SCA proposed here, the operators are based in moves of gates between neighborhood partitions. The variants are deterministic, pseudo-random and random. When the current generation is equal to this input, the algorithm finishes. Other condition to finish the evolutionary process occurs when the solutions don’t improvement. Thus, when the solutions in the last $N$ generations are very similar (almost $n$ improvement results) the algorithm finishes. Parameter $N$ depends of the circuit, because in a large circuit there are more possibilities of escaping from a local minimum than in a smaller one.
4.5.1 Explanation of SCAP algorithm

The SCAP approach (Master process) parallelizes the evolutionary steps to execute multiple copies of the same evolutionary approach one on each processor as slave process. There is a number of relatively independent populations each assigned a different processor. Each of the slave process starts with a different initial subpopulation and evolves and stops independently. There are no inter-processor communications among any of these independent genetic evolutions at any time during the slave process execution. The flow of execution of SCAP approach in parallel computing environment is shown in Figure 4.5.

Such subpopulation diversity reduces the chance that all slave processes prematurely converge to the same poor quality solution. The fact that there are many independent populations makes the global population more diverse. This approach is equivalent to simply taking the best solution after multiple executions of the evolutionary approach on different initial populations.
4.6 DBACP (DNA based computing Approach to Circuit Partitioning)

In recent works for high-performance computing, computation with DNA molecules has been paid considerable attention as one of non-silicon based computing, which has the potential to solve an NP-complete problem in a polynomial number of steps with DNA molecules. Adleman (1994) was the pioneer for using DNA computing to solve the Hamiltonian path problem of size n in O(n) steps using DNA molecules. In literature DNA computing has been used on a wide range of problems (Frisco, 2002; Fujiwara et al., 2004; Guarnieri et al., 1996; Gupta et al., 1997; Hug and Schuler, 2001; Kamio et al., 2003)

Motivated by the above mentioned work, this approach presents a parallel algorithm for figuring out solutions of the circuit partitioning problem based on a combination of Adleman Lipton model and the sticker model. The proposed algorithm requires a time cost and a DNA strand length that are linearly proportional to the instance size.

The basic idea behind the proposed algorithm (DBACP) for the minimum balanced bisection problem is as follows:

Step 1: Construct set T of all possible $2^n$ bipartition solutions of given circuit with n gates;

Step 2: Eliminate the solutions from the set T that don’t satisfy balance constraint (reading weighted file), thereby generating a set T of all possible feasible solutions

Step 3: According to T and incidence matrix M of graph G, determine the cut edge set C of the partitions for all solutions in T.

Step 4: Pick out a solution in T that has minimum edge cut.
4.6.1 Encoding of solution

To solve the instance of Partitioning problem with $G=(V, E)$ ($|V|=n, |E|=e$) start with $2^n$ identical single stranded DNA memory strands each with $(n+e)$ bit regions. The first $n$ bit regions will represent the presence/absence of vertex in the first partition and the rest $n$ bit regions will represent the presence/absence of an edge crossing the partition.

![Bit regions for 1st partition vertices and crossing edges](image)

Figure 4.6: Solution encoding in DBACP approach

As shown in Figure 4.6 the $2n$ bit regions of DNA strands are represented by $V1, V2...Vn, E1, E2...Em-1, Em$. $Vi$ is 1 if the vertex $i$ is present in the first partition and otherwise it is zero. Similarly $Ei$ is 1 if the edge is crossing the partition otherwise it is zero. The summation of $E1, E2...Em$ will represent the total number of edges crossing the partition. The DBACP (A DNA Based Approach for Circuit Partitioning) algorithm for partitioning the circuit (representing by graph G) in two equal sets is given below.

4.6.2 Steps of the algorithm

Following are the steps of the DBACP algorithm are:

Step 1: Initialization of memory strands

Prepare $(2n,n)$ library (Design $2^n$ DNA strands, each with $n+m$ bit regions) and initialize $(2n,n)$ library prepared in first step into the Tube $T$. (Initialize $(T,n)$)

Parallel separate the sample n times to yield $2^n$ data tubes, one for each value of the n-bit input(figure 4.7). This requires $2^{i-1}$ separation operator tube for the $i^{th}$ parallel separation (a total of $2^{n-1}$)
**Procedure**: initialize \((T, i)\)

*Input*: \(T\) : tube;
\(i\) : positive integer;

*Output*: \(T\) : a tube of all possible \((2^n)\) strands with no stickers annealed to the last \(n\) bit regions.

*begin*
1. if \(i=0\) return
2. \((T0,T1) := \text{separate}(T)\);
3. set\((T0,i)\);
4. initialize \((T0,i-1)\);
5. initialize \((T1,i-1)\);
6. \(T=\text{merge}(T0,T1)\);
*End*

**Figure 4.7**: Steps of initialize ( ) operation

**Step 2:** Read the weights file and separate the strands that don’t satisfy the balance constraints such that the tube \(T\) contains only strands of \(k\) feasible solutions i.e. a nonempty tube of \(k\) strands.

**Step 3:** Find the edge cut costs of every strand by using incidence matrix \(M\) of graph \(G\), for all solutions in \(T\) by calling edge_cut() function(figure 4.8).

**Procedure**: edge_cut\((T)\)

*Begin*

*for every strand in a tube of feasible solution*

*for* \(k:=1\) to \(n\)

*If* \(\text{get}(k)=0\)

*for* \(e:=1\) to \(m\)

*Sum*\((e)=m[k][e]+\text{sum}(e)\);

*for* \(e:=1\) to \(m\)

*If* \(\text{sum}(e)==1\)

*Set*\((\text{edgebit}(e))=1\)

*Else*

*Set*\((\text{edgebit}(e))=0\)

*End of for statement*

*End of for statement*

*End*

**Figure 4.8**: Steps of edge_cut( ) operation

**Step 4:**

bool := detect\((T)\) : Given a tube \(T\), get a Boolean variable bool, which assumes yes or no according as there is at least one DNA strand in \(T\) or not.

b := read\((T)\) : Given a tube \(T\) containing at least one strand, get randomly one strand \(s\) in \(T\).
**Procedure** $s := dna\_opt(T )$

**Input:** $T : a$ nonempty tube of $(k)$ strands.

**Output:** $s : a$ strand in $T$ such that $s$ contains a minimum edge cut cost.

**begin**
1. for $i = 1$ to $k$
2. $(T1, T0 ) := extract(T , n+ i );$
3. if detect$(T0 ) = yes, then$
4. $T :=rename(T0 );$
5. else $T :=rename(T1 );$
6. $s :=read(T );$
**end.**

**Figure 4.9:** Steps of $dna\_opt( )$ operation

### 4.6.3 Example of DBACP algorithm

Consider the following graph

![Graph](image)

**Step 1:** Prepare $(2n,n)$ library (Design $2^n$ DNA strands, each with $2n$ bit regions) and initialize $(2n,n)$ library prepared in first step into the Tube $T$. (Initialize $(T,n)$)

If the circuit has 4 nodes, 3 edges then the tube strands will be of $(4+3)$ bit length each and no. of strands in the tube will be $2^n$ nodes.

**Initialise the tube contents**

<p>| | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
</table>

-92-
Step 2: Create $2^n$ nodes strands with possible $2^n$ nodes permutations as shown in Figure 4.10.

Figure 4.10: shows the method of initializing the test tubes for a circuit with 4 nodes.
Creating $2^n$ strands in tube

<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>1</th>
<th>1</th>
<th>1</th>
<th>0</th>
<th>0</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>6</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>7</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>8</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>9</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>10</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>11</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>12</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>13</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>14</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>15</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Step 3: Read the weight file and separate the strands that don’t satisfy the balance constraints (with 10% tolerance i.e. within the range of 45% to 55%) such that the tube $T$ contains only strands of $k$ feasible solutions i.e. a nonempty tube of $k$ strands.

For the sake of simplicity let us assume that every

Let weight(1)=6, weight(2)=4, weight(3)=5, weight(4)=7

Tube after discarding non feasible strands based on balance constraints

<table>
<thead>
<tr>
<th>0</th>
<th>0</th>
<th>0</th>
<th>1</th>
<th>1</th>
<th>0</th>
<th>0</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Step 4: Reading the incidence matrix and to set the edge cut bits

Tube after filling the edge cut

<table>
<thead>
<tr>
<th>0</th>
<th>0</th>
<th>0</th>
<th>1</th>
<th>1</th>
<th>0</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

Step 5: Optimize the tube by only keeping the strands with minimal edge cut

Final tube with optimal solution

<table>
<thead>
<tr>
<th>0</th>
<th>0</th>
<th>0</th>
<th>1</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

i.e. Bipartitioning result includes {1,2} in one partition and {3,4} in other partition with minimal edge cost of 1.
4.7 Summary of Chapter

With current trends, the hypergraph partitioning with balance constraints, in addition to minimum cut objective is in vogue for digital circuit layout. In this work, five evolutionary approaches namely Soft Computing Algorithm for Partitioning (SCAP) Approach, Non Revisited Evolutionary Approach for Circuit Partitioning (NRECP), Enhanced Non Revisited Evolutionary Approach for Circuit Partitioning (E-NRECP), Hybrid Artificial Bee Colony Optimization with Simulated Annealing for Circuit Partitioning (HABCSACP) and A DNA based Approach for Circuit Partitioning (DBACP) for the optimization of VLSI netlist bi-Partitioning are proposed. These approaches are based on DNA computing, non-revisited evolutionary approach, simulated annealing, and artificial bee colony algorithm. For simulation work a set of benchmarks UCLA SPP series benchmark circuits and are used for evaluating the efficiency of the algorithms.

Paper Published


