CHAPTER 3
MULTI-OBJECTIVE OPTIMIZATION IN VLSI
FLOORPLANNING WITH CONSTRAINTS

3.1 INTRODUCTION

Entering into the nanometer regime era, the designer will take more care of providing extra options to place the modules in the final packing for various reasons. The problem of floorplanning with irregular boundaries can also be solved by treating the protruding parts along the boundaries as preplaced module. If there are many interconnections between the modules or align vertically in the middle of the chip for bus-based routing, the VLSI circuit designers is to limit the separation between two modules. This will also happen in design re-use in which a designer would like to keep the positions of some modules unchanged in the new floorplan. Therefore, it is desirable to find an efficient and effective way to handle the floorplanning problem with the placement constraints (Chang-Tzu et al 2002 and Valenzuela & Wang 2002).

Based on various floorplan representations, researchers have proposed many methods to process the placement constraints for practical uses such as boundary, abutment, preplaced, range, clustering, timing constraints and so forth. Abutment constraints and alignment constraints are used to have several dedicated modules in a circuit abut one after another to support the transmission of data between them. Boundary constraints are used to minimize the off-chip connections in the chip boundary. In addition to the conventional area and interconnection constraints, present VLSI floorplanners must also handle different placement constraints efficiently.
3.2 FLOORPLANNING WITH PLACEMENT CONSTRAINTS

Integrated Circuit design and IP module reuse design methodology is widely used for floorplanning in nanometer regime. This development makes floorplanning more and more important. Floorplanning is to decide the positions of circuit blocks or IP blocks on a chip subject to various constraints. Additional constraints are imposed on a subset of blocks that have to be located inside the chip in the real world floorplanning problem. Therefore it is need to allow users for specifying placement constraints during the floorplanning process. The basic concepts of various placement constraints have been discussed in the following sections.

Pre-Placement Constraints

The preplaced constraint is preferable to place blocks at pre-specified coordinates of the layout in some cases. In the mixed-signal layout generation problem, pre-placed constraints play a vital role. If the floorplan has obstacles, it can be solved by treating the obstacles as pre-placed blocks (Figure 3.1). The pre-placed constraint can be formally defined as follows.

**Definition 3.1:** Given a block \( b \) with fixed orientation, and a point \((x_b, y_b)\) \( b \) must be placed with its lower-left corner at \((x_b, y_b)\) in the final packing.

![Figure 3.1 Block b has a Pre-placed Constraint at Location \((x_b,y_b)\)](image-url)
Range Constraint

It requires that blocks can be placed within the pre-specified region of the floorplan. Range constraint problem is a general problem of pre-placed constraint because any pre-placed constraint can be written as a range constraint by specifying the rectangular region to be the same size as the block itself (Figure 3.2). The range constraint can be formally defined as follows.

**Definition 3.2:** Given a block \( b \) and rectangular region \( R = \{(x, y) \mid x_1 \leq x \leq x_2, y_1 \leq y \leq y_2\} \), block \( b \) must be placed inside \( R \) in the final packing.

![Figure 3.2 Block b has a Range Constraint](image)

Boundary Constraint

Another common situation is that I/O blocks must be placed at the periphery of the chip which is specified by boundary constraint. Floorplanning is usually done hierarchically in which blocks are grouped into different units and it can be done independently. If some blocks are forced to be packed in the boundaries of the unit so that they can abut with some others in the neighboring units, then the boundary constraint is applied to the blocks. The boundary constraint is shown in Figure 3.3, and defined as follows.
**Definition 3.3:** Given a block \( b \), it must be placed on one of the four sides: on the left, on the right, at the bottom or at the top of the final packing.

![Diagram](image)

**Figure 3.3** Blocks \( a, b, c \) and \( d \) are Placed on the Boundaries of the Chip Respectively

**Abutment Constraints**

With the advent of deep submicron technology and new packing schemes, integrated circuit components are often not rectangular. Non-rectangular shaped blocks are introduced to facilitate the usage of chip area and improve the blocks connectivity. Abutment constraint specifies that two blocks are abutted with each other horizontally or vertically. As an example, Figure 3.4 illustrates two blocks are abutted horizontally or vertically.

An abutment is classified into H-abutment (Figure 3.4(a)) and V-abutment (Figure 3.4(b)) according to the orientation, horizontal and vertical respectively. Of course, by flipping the orientation of the two blocks, H-abutment will become a V-abutment, and vice versa.
Figure 3.4 Two Blocks are Abutted Horizontally or Vertically

Alignment Constraints

With bus structure or in a pipeline, it is better that the blocks are aligned in a row, abutting with each other. The blocks can be aligned horizontally or vertically. Alignment constraint specifies that several blocks can be aligned in a row within a range. It is different from abutment constraint, because several abutted blocks may not be aligned. The techniques operating on the constraint graph (Fujiyoshi & Murata 1999) can be applied to alignment constraint, but the complexity is high (O (n^3)).

Figure 3.5 4 Blocks are Aligned Horizontally. l is the Alignment Range
Alignment constraint specifies that several blocks are aligned in a range (e.g. Bus width), abutting one by one horizontally or vertically. It is used to facilitate data transfer in bus structure or a pipeline. As an example, Figure 3.5 illustrates that four blocks are aligned horizontally. Aligned blocks are abutted with each other in a row. But alignment is different from abutment in the sense that several abutted blocks may not be aligned, as shown in Figure 3.6. Moreover, it has more freedom in positioning than rectilinear shape because the relative positions are not fixed.

![Diagram of aligned blocks](image)

**Figure 3.6 4 Blocks are Abutted One by One but Not Aligned**

Alignment constraint is classified into H-alignment and V-alignment according to the orientation, horizontal and vertical respectively. H-alignment can be formally defined as follows:

**Definition 3.4:** Given a range \( l \) and \( k \) blocks, \( b_i, i = 1, 2, \ldots, k \), with dimension \( w_i \times h_i \) and coordinates \( (x_i, y_i) \) referring to the lower-left corner, \( i = 1, 2, \ldots, k \) respectively, the \( k \) blocks are H-aligned iff \( x_i + w_i = x_{i+1}, 1 \leq i \leq k - 1 \) (abutting one by one) and \( y_{\text{max}} + l \leq y_i + h_i, 1 \leq i \leq k \), where \( y_{\text{max}} = \max \{y_i \mid i = 1, 2, \ldots, k\} \) (aligning horizontally).
A pre-condition for $l$ is $l \leq \min \{ h_i \mid i = 1, 2\ldots k \}$. Otherwise the $k$ blocks cannot be H-aligned. Notably, H-abutment is a special case of H-alignment as long as we let $k = 2$ and $l = \min (h_1, h_2)$. Similarly, we can define V-alignment.

**Definition 3.5** Given a range $l$ and $k$ blocks, $b_i$, $i = 1, 2,\ldots,k$ with dimension $w_i \times h_i$ and coordinates $(x_i, y_i)$, $i = 1, 2,\ldots,k$ respectively, the $k$ blocks are V-aligned iff $y_i + h_i = y_{i+1}$, $1 \leq i \leq k - 1$ (abutting one by one) and $x_{\max} + l \leq x_i + w_i$, $1 \leq i \leq k$, where $x_{\max} = \max \{ x_i \mid i = 1, 2,\ldots,k \}$ (aligning vertically).

**Performance Constraints**

In deep submicron VLSI design technology, the interconnect delay dominates the circuit performance due to increasing IC package. Traditional floorplanners consider the total wirelength only. It cannot guarantee bounded delay for critical nets. However, it is desirable to minimize the critical net delay to optimize circuit performance. The performance constraint is intended for this purpose. This constraint requires designated nets (blocks) to be placed within a predefined bounding box net (Figure 3.7). The performance constraint is used to optimize not only the circuit delay, but also the total wire length.

![Diagram](image)

**Figure 3.7 The Bounding Box of the Farthest Pin Locations in a Net**
Definition 3 (Performance constraint) Given a set of blocks and the performance constraints, minimize the total wire length and place designated blocks in a bounding box such that the critical net delay satisfy performance constraints at the same time.

3.3 LITERATURE REVIEW

VLSI layout engineers have proposed many methods to process placement constraints based on various floorplan representations and optimization algorithms. Young et al (1998) handled the preplaced modules with a well known slicing floorplanner. The packing is reasonably tight for the given constraints of the preplaced modules resulting in short runtime. Then, Young et al (2001) proposed an algorithm to handle abutment constraint based on slicing structure. They have used the two sets of modules such as rectangular and L/ T shaped modules. To expand all the L and T shaped modules it is required to scan the polish expression four times that takes O (n) time. Therefore, the total time taken for each iteration of the annealing process is O (np) in the worst case and O (n + p) on the average. Next, Chang et al (2005) also presented the slicing floorplanning with abutment and fixed outline constraints using evolutionary algorithm. This method gives feasible solution for sliceable floorplanning with these constraints and not for general floorplan representation.

helpful in reducing critical path. Then, Meng-Chen Wu & Yao-Wen Chang (2004) proposed the placement problem with alignment constraints using B*-tree representation. However, it does not guarantee to produce an optimal floorplan with fixed-outline constraint.

Song Chen et al (2006) proposed VLSI block placement using alignment constraints. They used the Corner Block List (CBL) representation to handle the alignment constraints. These methods have the packing problem of aligning the blocks in the room based structures. Then, Wan-Ping Lee et al (2009) developed a dynamic-programming based voltage scaling algorithm and a timing driven floorplanning with the B*-tree representation using Multi Supply Voltage (MSV) design. More stringent is the timing constraint, more timing slack is reserved for delay optimization during floorplanning. Next, Bus driven VLSI floorplanning with thermal consideration was proposed by the Po-Hsun Wu & Tsung-Yi Ho (2012). It is based on sequence pair representation with the SA algorithm. It effectively separates hotspots and reduces the chip temperature in the VLSI floorplanning. The main drawback of this method is having a larger solution space and time complexity to construct the admissible VLSI floorplan for bus based routing. Table 3.1 shows the comparison between various Constraints in the VLSI floorplanning representations.

This thesis work is considered as the alignment and performance constraints for fixed frame floorplanning based on B*-tree representation using DE algorithm to facilitate bus based routing and sequential I/O operations.
<table>
<thead>
<tr>
<th>Year</th>
<th>Author</th>
<th>Representation With Constraints</th>
<th>Discussion/Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>2004</td>
<td>Meng-Chen Wu &amp; Yao-Wen Chang</td>
<td>Non slicing (B*-tree) representation with alignment and performance constraints</td>
<td>It takes only amortized linear time to find the optimal solution. But the solution space is large.</td>
</tr>
<tr>
<td>2004</td>
<td>Evangeline F.Y. Young, Chris C.N. Chu, and M.L. Ho</td>
<td>Non slicing (Sequence pair) representation with all constraints</td>
<td>They were taken ami 33 and ami49 with number of placement constraints</td>
</tr>
<tr>
<td>2005</td>
<td>Rong Liu, Sheqin Dong, Xianlong Hong and Yoji Kajitani</td>
<td>Non slicing sequence pair representation with boundary and pre-placement constraints</td>
<td>Experiment results shown that the Instance augmentation algorithm was quite promising in fixed-outline Floorplanning.</td>
</tr>
<tr>
<td>2005</td>
<td>Chang-Tzu Lin, De-Sheng Chen, Yi-Wen Wang and Hsin-Hsien Ho</td>
<td>Slicing (binary tree) representation with abutment constraints</td>
<td>It was based on the slicing floorplan and evolutionary Algorithm. It can guarantee a feasible floorplan with abutment constraint and produce optimal solution</td>
</tr>
<tr>
<td>2006</td>
<td>Song Chen, Sheqin Dong, Xianlong Hong, Yuchun Ma, and C. K. Cheng</td>
<td>Nonslicing CBL (Corner block List) representation with alignment Constraints</td>
<td>Experimental results shown that the area usages are little higher compared with the previous algorithm.</td>
</tr>
<tr>
<td>2006</td>
<td>Chang-Tzu Lin, De-Sheng Chen, and Yi-Wen Wang</td>
<td>Slicing (Generalised Polish expression) Representation, Boundary Constraints</td>
<td>They were taken only ami33 and ami49 with boundary constraints. Simulated Annealing algorithm was used.</td>
</tr>
<tr>
<td>2009</td>
<td>Wan-Ping Lee, Hung-Yi Liu, and Yao-Wen Chang</td>
<td>Non slicing (B*-tree)representation With voltage alignment algorithm</td>
<td>They have proposed a dynamic-programming based voltage scaling algorithm and a PN-FP for the MSV design.</td>
</tr>
<tr>
<td>Year</td>
<td>Authors</td>
<td>Representation</td>
<td>Description</td>
</tr>
<tr>
<td>------</td>
<td>---------</td>
<td>----------------</td>
<td>-------------</td>
</tr>
<tr>
<td>2012</td>
<td>Po-Hsun Wu, Tsung-Yi Ho</td>
<td>Non slicing (sequence pair) representation with bus driven and thermal driven floorplanning</td>
<td>This paper can effectively separate hotspots and reduce the chip temperature in the VLSI Floorplanning.</td>
</tr>
<tr>
<td>2013</td>
<td>Wenxu Sheng and Sheqin Dong</td>
<td>Less Flexibility First (LFF) algorithm is used to represent the VLSI Floorplan with Multi-bend bus driven VLSI Floorplanning.</td>
<td>This algorithm is to reach as feasible and effective a solution as possible within a limited period of time.</td>
</tr>
</tbody>
</table>

### 3.4 PROPOSED METHODOLOGY

This section defines the feasibility conditions for B*-tree representation with alignment and performance constraints and then formulate the placement problem based on DE Optimization Algorithm. The flowchart of our proposed algorithm is given in Figure 3.8.

#### 3.4.1 Problem Description

Let \( B = \{b_1, b_2, ..., b_n\} \) will be a set of ‘n’ rectangular blocks whose respective width, height, and area are denoted by the \( w_i, h_i, \) and \( a_i, 1 \leq i \leq n \). And \((x_i, y_i)\) denote the coordinate of the bottom-left corner of block \( b_i, 1 \leq i \leq n \) which is on a chip. When a placement \( P \) with the alignment constraints is an assignment of the coordinate \((x_i, y_i)\) for each block is \( b_i, 1 \leq i \leq n \), such that no two blocks can overlap and the given constraints are satisfied. The goal of floorplanning/placement is to optimize a predefined cost metric, such as the area (the minimum bounding rectangle of \( P \)) is induced by the assignment of \( b_i \) which is on the chip.
Figure 3.8 Proposed Flowchart for VLSI Floorplanning based on B*-tree with Constraints
3.4.2 B*-tree with Alignment Constraints

The alignment blocks have the following two properties:

i) Alignment blocks must abut one by one;

ii) These blocks have to be located in an alignment range.

First, the solution can be applied to abutment placement. For a B*-tree, the left child $n_j$ of the node $n_i$ representing the lowest adjacent block $b_j$ which is right to block $b_i$ (i.e. $x_j = x_i | w_j$). Hence, blocks can be adjacent to one by one if their corresponding nodes form a left-skewed subtree in a B*-tree.

![Alignment Range Diagram](image)

**Figure 3.9** (a) An Infeasible Placement with Blocks Falling Out of the Alignment Range (b) Inserting Dummy Blocks

**Property 1** In a B*-tree, the nodes in a left-skewed sub-tree may correspond to a set of abutment blocks.

After packing, the blocks are compacted to the bottom left corner (Meng-Chen et al 2004). The blocks which are associated with a left-skew sub-tree of a B*-tree may be aligned together if there is no block falling
during packing. To solve the falling down problem, we should introduce \textit{dummy blocks} to fix it.

When a dummy block comes from an alignment range $1$, the dummy block will have the same x-coordinate with the alignment block and right below it. The width of the dummy block is equal to its corresponding alignment block. Its height can be adjusted to make a displaced alignment block which will be shifted into the right alignment range. As illustrated in Figure 3.9 (a), the blocks $b_2$ and $b_4$ will fall out of the alignment range. The heights of the two dummy blocks will be adjusted to shift the displaced alignment blocks into the correct alignment range in Figure 3.9 (b). After adjusting the heights of the dummy blocks, the alignment constraints will be assured for the feasible placement problem.

\textbf{Property 2} \textit{Inserting a dummy block of an appropriate height, we can guarantee a feasible placement of the corresponding alignment block.}

\textbf{Feasibility Condition for Alignment Constraints}

The properties mentioned in the preceding section provide the way to develop the feasibility condition for a B*-tree with the alignment constraints. Consequently, the advantage of the feasibility condition will be taken to transform an infeasible to a feasible placement of the alignment blocks.

The node representing an alignment block in the B*-tree is known as an alignment node. For each alignment node, a \textit{dummy node} is introduced in the B*-tree to make the alignment node into the right child of its corresponding dummy node. According to the definition of the B*-tree, this will make a dummy block right under its corresponding alignment block. After that the height of the dummy block can be adjusted to change the y-
coordinate of the alignment block, if needed. The cluster node is known as a set of an alignment node and its corresponding dummy node.

![Diagram of the alignment shape in a B*-tree](image)

**Figure 3.10** (a) The Alignment Shape in a B*-tree (b) The Corresponding Placement

As an example shown in Figure 3.10 (a), the three-cluster nodes \( n_3 \), \( n_4 \) and \( n_5 \) will form a left-skewed sub-tree in an alignment shape, for which the corresponding placement of the alignment blocks \( b_3 \), \( b_4 \), and \( b_5 \) will be one by one as shown in Figure 3.10 (b). In order to provide the way to develop the feasibility condition of a B*-tree with the alignment constraints, the following theorem will be used here (Tang & Wong 2002, Meng-Chen et al 2004).

**Theorem**

If the alignment nodes in a B*-tree form an alignment shape, there is a feasible placement with alignment constraints will exist.
To deal with the alignment constraints, two passes are needed to pack blocks correctly. In the first pass, the coordinates for each non-dummy block (a regular block or an alignment block) is computed. Then, validate every alignment block is in the alignment range or not. If there is any violation of the alignment constraints, compute in the second pass which is the minimum movement (height) for the corresponding alignment (dummy) block to shift into the alignment range. If \( k \) alignment blocks and the alignment range \( l \) are given, the equation for computing the minimum movement (height) \( \Delta_i \) for the alignment block \( b_i \), \( i=1, 2, \ldots, k \) in H-alignment is as follows:

\[
\Delta_i = \begin{cases} 
  y_{\text{max}} + 1 \leq y_i + h_i & \text{if } 1 \leq i \leq k \\
  0 & \text{otherwise}
\end{cases} 
\]  

(3.1)

where \( y_{\text{max}} = \max \{ y_i \mid i=1, 2, 3\ldots k \} \)

After getting the minimum movement for each alignment block, the height of the corresponding dummy block is set to \( \Delta_i \) for shifting the alignment block towards the alignment range. Then, guarantee that the final placement will be feasible without violating any alignment constraint by using such a two-pass packing scheme.

As it is shown in Figure 3.11 (a), the alignment blocks \( b_2, b_3, b_4, \) and \( b_5 \) abut one by one, but the blocks \( b_2 \) and \( b_5 \) will fall out of the alignment range after the 1st-pass packing. Then, we should compute the minimum movement (height \( \Delta_i \)) for each alignment (dummy) blocks. And, \( b_2 \) and \( b_5 \) blocks should be shifted upward by \( \Delta_2 \) and \( \Delta_5 \) respectively. Figure 3.11 (b) gives a feasible placement after the adjustment.
3.4.3 Floorplanning with Performance Constraints

VLSI floorplanners try to minimize the total wire length but cannot guarantee that critical nets meet the delay constraint. In order to make critical net delay satisfy the delay constraint, place the performance blocks are needed to connect by critical nets near to each other. The delay $D_{s,t}$ of a two-pin net from the source at $(x_s, y_s)$ to a sink at $(x_t, y_t)$ at the floorplanning stage can be approximated by the following equation:

$$D_{s,t} = \delta(x_t - x_s) + |y_t - y_s|$$  \hspace{1cm} (3.2)

where $\delta$ is a constant to scale the distance to timing. The above linear function is examined to estimate the delay because the actual delay is close to linear to
the source-sink distance with appropriate buffer insertions. (Of course, the more sophisticated approximation can also be used for this purpose by trading off the running time). From the above equation and the given delay bound, $D_{\min}$, the distance from the sources $(s)$ to the sink $(t)$, $I_{s,t}$ must satisfy the following inequality to meet the performance constraint:

$$I_{s,t} = |x_t - x_s| + |y_t - y_s| = \frac{D_{s,t}}{\delta} \leq \frac{D_{\text{max}}}{\delta}$$

(3.3)

The popular approximation method is used for a net that the distance of pins is given by half of the perimeter of the minimum bounding box of the blocks connected to the net. To meet the performance constraints, place the constrained blocks in a bounding box whose half of the perimeter is smaller than the distance with the delay bound.

![Figure 3.12](image)

**Figure 3.12** (a) A Feasible Placement for Performance Constraints. (b) An Infeasible Placement with the Blocks and the Delay Bound Given in (a). (c) Cluster the Feasible Placement in (a)

In Figure 3.12 (a), the bounding box (dotted lines) of the blocks is smaller than the bounding box (dash lines) with the delay bound, so the placement is feasible for the given performance constraint. The placement of
Figure 3.12 (b) is infeasible because the bounding box of the blocks is greater than the delay bound. In Figure 3.12 (c), a feasible placement is obtained with the performance constraint from Figure 3.12 (a). Then, the blocks are clustered as a rectilinear super block and the shape of the rectilinear super block is fixed. The performance constraint will be satisfied afterwards. Then repartition of the rectilinear super block into a set of new blocks for further processing with other blocks is performed.

Let $I_{\text{bound}}$ denote the bounding distance, which is half of the perimeter of the maximum bounding box of blocks connected by the net, and $I_{\text{max}}$ denote the maximum bounding distance, which is the distance of the delay bound. We have the following property:

**Property 3** We can get a feasible placement with performance constraints by placing performance blocks in the bounding box whose bounding distance is smaller than or equal to the maximum distance bound.

**Feasibility Conditions for Performance Constraints**

Given a set of blocks and performance constraints, the nodes representing performance blocks as performance nodes in a B*-tree. To meet the performance constraint, the performance blocks shall be located near each other. Given a placement $P$ of $k$ performance blocks whose areas are $A_i$, $i = 1, 2, ..., k$, the width $w$, the height $h$, and the dead space $S_{\text{perf}}$ of the placement, the sub-placement of the performance blocks must satisfy the following inequality:

$$w+h=I_{\text{bound}} \leq I_{\text{max}}$$

(3.4)

If the bounding distance of sub-placement is greater than the distance bound, the sub-placement cannot meet the performance constraints
and thus the placement with the sub-placement either. Thus, the
sub-placement is modified until it is smaller than the distance bound. By
doing so, a set of feasible sub-placements are obtained for the performance
blocks. Among these sub-placements, pick the one with the minimum
\[ S_{per} = w \times h - \sum_{i=1}^{k} A_i. \]
And treat the sub-placement as a rectilinear block.

Then the rectilinear block is fixed (and thus fix the delay) for
further processing with other blocks. By clustering performance blocks into
an appropriate rectilinear block and fixing its shape, it can guarantee that the
performance constraint will be satisfied throughout the remaining processing.
For example, repartition the rectilinear super block of Figure 3.12 (a) as the
new sub-blocks shown in Figure 3.12 (c) and fix the relation between the new
sub-blocks. By maintaining such a rectilinear block, a feasible placement with
the performance constraints after the whole process is guaranteed.

**Theorem 2** By pre-processing the performance blocks into the rectilinear
blocks and keeping their shape, we can guarantee to generate feasible
placements with performance constraints.

### 3.4.4 Differential Evolutionary Algorithm for VLSI floorplanning with
Constraints

**B*-tree Perturbation Operation**

During the optimization algorithm, perturbation operation can be
done in the B*tree. The following operations are required to perturb a B*-tree.

- **Op1**: Rotate a block.
- **Op2**: Flip a block.
- **Op3**: Move a block to another place.
- **Op4**: Swap two blocks.
- **Op5**: Move a set of alignment blocks to another place.
The first four operations are very similar to the previous chapter and the last one is designed for alignment constraints. An operation 1 is used to rotate a block. Without changing the relationship between any two nodes except performance blocks, this action can be performed. The corresponding rectilinear blocks are needed to rotate together with the performance constraints. Then, the blocks are flipped in the second operation. The correct relations between the rectilinear blocks should be maintained in the first operation.

Operation 3 and operation 4 changes the relations of blocks to get a different placement. These two operations cannot apply for alignment blocks. The performance blocks still need to move its corresponding rectilinear block to another place. An operation 5 is specially used for alignment constraints. In operation 5, a set of alignment blocks is moved to another place. The positions of the first pair of a dummy node and an alignment node change in the alignment shape. Then, other pairs of dummy node and alignment nodes have attached to the correct positions to maintain their shape.

**Fitness Function**

The VLSI floorplanning objective is to minimize the area and the wire length.

\[
\cos t(F) = ar + \omega_1 \frac{area}{area} + \omega_2 \frac{wirelength}{wirelength}
\]  

(3.5)

In the above equation, area is the area of the smallest rectangle enclosing all the modules with the constraints block. Wire length which represents the interconnection costs is calculated by the semi-perimeter method. Area* and wirelength* represents the minimal area and the interconnection costs, respectively. ‘ar’ is the aspect ratio to control the floorplan operations. Since their values do not know into practice, estimated
values are used. \( \omega_1 \) and \( \omega_2 \) are the weights assigned to the area minimization objective and the interconnection minimization objective, respectively, were 0 \( \leq \omega_1 \), \( \omega_2 \leq 1 \), and \( \omega_1 + \omega_2 = 1 \).

The interconnection cost is the total wire length of all the nets, and the wire length of a net is calculated by the half perimeter of the minimal rectangle enclosing the centers of the modules that have a terminal of the net on it. The constraints function will be included in the objective function of this algorithm. The fitness of an individual (Floorplan) is defined as follows

\[
Fitness(F) = \frac{1}{cost(F)} + \text{Constraint function} \tag{3.6}
\]

Where \( F \) is the corresponding floorplan and the cost (F) is defined in Equation (3.5).

**Initial Population**

An individual in the initial population is a B*-tree, which representing an admissible VLSI floorplan. A constructive algorithm is designed to construct an admissible B*-tree.

The algorithm starts with defining an evaluation function (V) for each rectangular module. Then, it inserts the modules into an initially empty B*-tree \( B \) according the value of V. When inserting a module into \( B \), it checks a point such that the new module can be packed in the feasible region with its bottom-left corner without violating alignment and performance constraints rule. Also, check the alignment blocks fall out the required area. If not, adjust the height of the dummy blocks to fix the alignment violations. The constructive algorithm is invoked iteratively to generate an initial population of individuals.
DE Algorithm

The DE algorithm can be classified as a floating-point encoded evolutionary optimization algorithm. There are several variants is used in DE (Price 1999). The particular version used throughout this analysis was the DE/rand/1/bin scheme. Thus the working algorithm is explained below. The ‘rand’ means that perturbation can be done by randomly chosen vector. The perturbation is done with a ‘single’ vector difference, out of the three distinct randomly chosen vectors, the weighted vector differential of any two vectors is added to the third one. Next ‘bin’ represents the binomial crossover. The crossover is performed on each of the D variables whenever a randomly picked number between 0 and 1 is within the CR value. So for high values of CR, the exponential and binomial crossovers yield similar results.

Generally, a multi-constrained nonlinear optimization problem can be expressed as follows:

\[ X = (x_1, \ldots, x_D) \quad (3.7) \]

To minimize an objective function

\[ f(X) \]

Subject to constraint functions

\[ g_j(X) \leq 0 \quad j = 1, \ldots, m \]

And subject to boundary constraints

\[ x_i^{(L)} \leq x_i \leq x_i^{(U)} \quad i = 1, \ldots, D \]

Thus, the optimization goal is to find a feasible vector \( X \), composed of \( D \) parameters, to minimize the objective function \( f(X) \).
DE has been operating on a population \((P_G)\) of candidate solutions, not just a single solution. These candidate solutions are the individuals of the population. In particular, DE maintains a population of constant size that consists of \(NP\), real-valued vectors \((X_{i,G})\). Where \(i\) indexes the population and \(G\) is the generation to which the population belongs.

\[
P_G = (X_{1,G}, \ldots, X_{NP,G}) \quad G = 0, \ldots, G_{\text{max}}
\]

(3.8)

Additionally, each vector contains \(D\) real parameters:

\[
X_{i,G} = (X_{1,j,G}, \ldots, X_{D,j,G}) \quad i = 1, \ldots, NP, \quad G = 0, \ldots, G_{\text{max}}
\]

(3.9)

There is no more knowledge available about the location of a global optimum than the limits of the problem variables. Then a natural way to seed the initial population \((P_{G,0})\) is chosen from the given boundaries:

\[
X_{j,i,0} = \text{rand} \left[ 0,1 \right] \cdot (X_j^{(u)} - X_j^{(L)}) + X_j^{(L)}
\]

(3.10)

Where, \(i=1\ldots NP, j=1\ldots D\) and \(\text{rand} \left[ 0, 1 \right]\) denotes a uniformly distributed random value within the range: \([0.0, 1.0]\) that is chosen anew for each \(j\).

DE’s reproduction scheme is different from other evolutionary algorithms. From the 1\(^{\text{st}}\) generation, vectors in the current population \((P_G)\) are randomly sampled and combined to create candidate vectors for the subsequent generation, \(P_{G+1}\). The population of candidate or “trial” vectors \(P_{G+1} = U_{i,G+1} = U_{j,i,G+1}\) (where \(i = 1, \ldots, NP, j = 1, \ldots, D\)) is generated as follows:

\[
u_{j,i,G+1} = \begin{cases} v_{j,i,G+1} & \text{if } \text{rand} \left[ 0,1 \right] \leq CR \lor j = k \\ x_{j,i,G} & \text{otherwise} \end{cases}
\]

(3.11)

where, \(i=1\ldots NP, j=1, \ldots, D\)
\[ k_i \in \{1, \ldots, D\} \text{ random parameter index} \]

\[ r_1, r_2, r_3 \in \{1, \ldots, NP\} \text{ randomly selected} \]

Expect: \( r_1 \neq r_2 \neq r_3 \neq i \)

\[ CR \in [0,1], \quad F \in (0,1) \]

The randomly chosen indexes, \( r_1, r_2 \) and \( r_3 \) are different from each other and also different from the running index \( i \). New, random integer values for \( r_1, r_2 \) and \( r_3 \) are chosen for each value of the index \( i \), i.e., For each individual. The index \( k \) refers to a randomly chosen chromosome which is used to ensure that each individual trial vector, \( U_{i,G+1} \), differs from its counterpart in the previous generation, \( X_{i,G} \), by at least one parameter. A new, random, integer value is assigned to \( k \) prior to the construction of each trial vector, i.e., for each value of the index \( i \). \( F \) and \( CR \) are DE control parameters.

Like \( NP \), both values remain constant during the search process. \( F \) is a real-valued factor in the range \((0.0, 1.0)\) that scales the differential variations, and therefore controls mutation amplifications. Respectively, \( CR \) is a real-valued crossover factor in the range \([0.0, 1.0]\) that controls the probability that a trial vector parameter will come from the randomly chosen, mutated vector, \( V_{j, i, G+1} \), instead of from the current vector, \( X_{j, i, G} \). Usually, suitable values for \( F, CR \) and \( NP \) can be found by trial-and-error after a few tests using different values.

DE’s replacement scheme also differs from other evolutionary algorithms. The population for the next generation, \( P_{G+1} \), is selected from the current population, \( P_G \), and the child population, according to the following rule:
\[ X_{i,G-1} = \begin{cases} U_{i,G-1} \quad \text{iff} \ (U_{i,G-1}) \leq f(X_{i,G}) \\ X_{i,G} \quad \text{otherwise} \end{cases} \] (3.12)

Hence, each individual of the temporary population is compared with its counterpart in the current population. It is assumed that the vector with the lower value wins a place in the next generation’s population. The interesting point concerning DE’s replacement scheme is that a trial vector is only comparable to one individual, not to all the individuals in the current population. The pseudo code for DE algorithm with constraints is given in Figure 3.13.

```
Algorithm: Placement with Alignment Constraints (blocks, constraints)
Input: A set of blocks and alignment and performance constraints.
Output: A placement without violating the given constraints.
1. Generate the rectilinear blocks for performance blocks
2. Initialize a B*-tree for the input blocks and constraints;
3. Differential Evolutionary Process;
4. Begin
5. Initialize the parameters
6. Evaluate the cost of the vector
7. perturb();
8. first-packing();
7. adjust y-coordinates of the sub-blocks for rectilinear blocks.
8. if alignment blocks fall out of the required area
9. then adjust heights of dummy blocks to fix alignment violations
10. final-packing();
10. Evaluate the B*-tree cost; Obtain the best solution.
11. until criteria met;
12. End
13. Return the best solution;
```

**Figure 3.13 Pseudo Code for DE Algorithm with Constraints**
3.5 EXPERIMENTAL RESULTS AND DISCUSSION

The experiment employed MCNC benchmarks for the VLSI floorplanning using B*-tree representation with alignment and performance constraints. Table 3.1 shows the various constraints with different algorithms and representations used in previous literature review. From the table, it is clear that the same cases used in Meng-Chen Wu & Yao-Wen Chang (2004) and our proposed method. So our proposed DE algorithm have compared with SA framework as shown in Table 3.2. All experiments were run on the same machine. The proposed approach adopts a DE algorithm without resorting to floorplan representations, and the placement constraints of the buses are satisfied during the block-packing process.

Table 3.2 Area Estimation with Alignment Constraints

<table>
<thead>
<tr>
<th>Circuit</th>
<th>Blocks</th>
<th>Constrained Blocks</th>
<th>Simulated Annealing Algorithm</th>
<th>Ours; Differential Evolution</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Alignment</td>
<td>Area (mm²)</td>
<td>Area (mm²)</td>
</tr>
<tr>
<td>apte</td>
<td>9</td>
<td>4</td>
<td>46.92</td>
<td>45.639</td>
</tr>
<tr>
<td>xerox-1</td>
<td>10</td>
<td>4</td>
<td>20.08</td>
<td>19.19</td>
</tr>
<tr>
<td>xerox-2</td>
<td>10</td>
<td>4</td>
<td>20.08</td>
<td>19.98</td>
</tr>
<tr>
<td>hp-1</td>
<td>11</td>
<td>4</td>
<td>9.20</td>
<td>8.95</td>
</tr>
<tr>
<td>hp-2</td>
<td>11</td>
<td>4</td>
<td>9.349</td>
<td>9.00</td>
</tr>
<tr>
<td>ami 33-1</td>
<td>33</td>
<td>4</td>
<td>1.180</td>
<td>1.15</td>
</tr>
<tr>
<td>ami 33-2</td>
<td>33</td>
<td>4</td>
<td>1.181</td>
<td>1.22</td>
</tr>
<tr>
<td>ami 49-1</td>
<td>49</td>
<td>5</td>
<td>36.60</td>
<td>35.56</td>
</tr>
<tr>
<td>ami 49-2</td>
<td>49</td>
<td>4</td>
<td>36.56</td>
<td>35.55</td>
</tr>
<tr>
<td>ami 49-3</td>
<td>49</td>
<td>4</td>
<td>36.64</td>
<td>35.60</td>
</tr>
</tbody>
</table>
Usually, suitable values for Scaling Factor (F), Crossover (CR) and Number of Population (NP) can be found by trial-and-error after a few tests using different values. In our experiment, we have obtained better optimal results by setting NP is 100, the probabilities for the CR is 0.9 and F is 0.5 compared with the SA. The experimental results are shown in Table 3.1 to Table 3.5.

**Table 3.3 Run Time Estimation with Alignment Constraints**

<table>
<thead>
<tr>
<th>Circuit</th>
<th>Blocks</th>
<th>Constrained Blocks</th>
<th>Simulated Annealing Algorithm</th>
<th>Ours; Differential Evolution</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>Alignment</td>
<td>Time(sec)</td>
</tr>
<tr>
<td>apte</td>
<td>9</td>
<td>4</td>
<td>3.6</td>
<td>1.48</td>
</tr>
<tr>
<td>xerox-1</td>
<td>10</td>
<td>4</td>
<td>5.8</td>
<td>4.09</td>
</tr>
<tr>
<td>xerox-2</td>
<td>10</td>
<td>4</td>
<td>6.4</td>
<td>5.12</td>
</tr>
<tr>
<td>hp-1</td>
<td>11</td>
<td>4</td>
<td>5.9</td>
<td>3.98</td>
</tr>
<tr>
<td>hp-2</td>
<td>11</td>
<td>4</td>
<td>6.1</td>
<td>4.04</td>
</tr>
<tr>
<td>ami 33-1</td>
<td>33</td>
<td>4</td>
<td>35.4</td>
<td>20.98</td>
</tr>
<tr>
<td>ami 33-2</td>
<td>33</td>
<td>4</td>
<td>52.6</td>
<td>25.78</td>
</tr>
<tr>
<td>ami 49-1</td>
<td>49</td>
<td>5</td>
<td>132.7</td>
<td>87.98</td>
</tr>
<tr>
<td>ami 49-2</td>
<td>49</td>
<td>4</td>
<td>97.9</td>
<td>78.56</td>
</tr>
<tr>
<td>ami 49-3</td>
<td>49</td>
<td>4</td>
<td>109.2</td>
<td>80.09</td>
</tr>
</tbody>
</table>

Table 3.1 and 3.2 shows the area and runtime comparison between SA and our proposed DE algorithm based on VLSI floorplanning with alignment constraints only. The alignment constraint gives the optimal area (2.82%) and reduces the time (45.98%) complexity.
Table 3.4 Area Estimation with Alignment and Performance Constraints

<table>
<thead>
<tr>
<th>Circuit</th>
<th>Blocks</th>
<th>Constrained blocks</th>
<th>Simulated Annealing</th>
<th>Ours; Differential Evolution</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Align</td>
<td>Perf</td>
<td>Area (mm²)</td>
</tr>
<tr>
<td>apte</td>
<td>9</td>
<td>4</td>
<td>0</td>
<td>46.92</td>
</tr>
<tr>
<td>xerox-1</td>
<td>10</td>
<td>4</td>
<td>0</td>
<td>20.08</td>
</tr>
<tr>
<td>xerox-2</td>
<td>10</td>
<td>4</td>
<td>2</td>
<td>20.08</td>
</tr>
<tr>
<td>hp-1</td>
<td>11</td>
<td>4</td>
<td>0</td>
<td>9.20</td>
</tr>
<tr>
<td>hp-2</td>
<td>11</td>
<td>4</td>
<td>2</td>
<td>9.349</td>
</tr>
<tr>
<td>ami 33-1</td>
<td>33</td>
<td>4</td>
<td>0</td>
<td>1.180</td>
</tr>
<tr>
<td>ami 33-2</td>
<td>33</td>
<td>4</td>
<td>3</td>
<td>1.181</td>
</tr>
<tr>
<td>ami 49-1</td>
<td>49</td>
<td>5</td>
<td>0</td>
<td>36.60</td>
</tr>
<tr>
<td>ami 49-2</td>
<td>49</td>
<td>4</td>
<td>3</td>
<td>36.56</td>
</tr>
<tr>
<td>ami 49-3</td>
<td>49</td>
<td>4</td>
<td>6</td>
<td>36.64</td>
</tr>
</tbody>
</table>

Table 3.5 Runtime Estimation with Alignment and Performance Constraints

<table>
<thead>
<tr>
<th>Circuit</th>
<th>Blocks</th>
<th>Constrained blocks</th>
<th>Simulated Annealing</th>
<th>Ours; Differential Evolution</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Align</td>
<td>Perf</td>
<td>Run Time(sec)</td>
</tr>
<tr>
<td>Apte</td>
<td>9</td>
<td>4</td>
<td>0</td>
<td>3.6</td>
</tr>
<tr>
<td>xerox-1</td>
<td>10</td>
<td>4</td>
<td>0</td>
<td>5.8</td>
</tr>
<tr>
<td>xerox-2</td>
<td>10</td>
<td>4</td>
<td>2</td>
<td>6.4</td>
</tr>
<tr>
<td>hp-1</td>
<td>11</td>
<td>4</td>
<td>0</td>
<td>5.9</td>
</tr>
<tr>
<td>hp-2</td>
<td>11</td>
<td>4</td>
<td>2</td>
<td>6.1</td>
</tr>
<tr>
<td>ami 33-1</td>
<td>33</td>
<td>4</td>
<td>0</td>
<td>35.4</td>
</tr>
<tr>
<td>ami 33-2</td>
<td>33</td>
<td>4</td>
<td>3</td>
<td>52.6</td>
</tr>
<tr>
<td>ami 49-1</td>
<td>49</td>
<td>5</td>
<td>0</td>
<td>132.7</td>
</tr>
<tr>
<td>ami 49-2</td>
<td>49</td>
<td>4</td>
<td>3</td>
<td>97.9</td>
</tr>
<tr>
<td>ami 49-3</td>
<td>49</td>
<td>4</td>
<td>6</td>
<td>109.2</td>
</tr>
</tbody>
</table>
Table 3.6 Wirelength Estimation with Alignment and Performance Constraints

<table>
<thead>
<tr>
<th>Circuit</th>
<th>Blocks</th>
<th>Constrained blocks</th>
<th>Ours: Differential Evolution</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Align</td>
<td>Perf</td>
</tr>
<tr>
<td>Apte</td>
<td>9</td>
<td>4</td>
<td>0</td>
</tr>
<tr>
<td>xerox-1</td>
<td>10</td>
<td>4</td>
<td>0</td>
</tr>
<tr>
<td>xerox-2</td>
<td>10</td>
<td>4</td>
<td>2</td>
</tr>
<tr>
<td>hp-1</td>
<td>11</td>
<td>4</td>
<td>0</td>
</tr>
<tr>
<td>hp-2</td>
<td>11</td>
<td>4</td>
<td>2</td>
</tr>
<tr>
<td>ami 33-1</td>
<td>33</td>
<td>4</td>
<td>0</td>
</tr>
<tr>
<td>ami 33-2</td>
<td>33</td>
<td>4</td>
<td>3</td>
</tr>
<tr>
<td>ami 49-1</td>
<td>49</td>
<td>5</td>
<td>0</td>
</tr>
<tr>
<td>ami 49-2</td>
<td>49</td>
<td>4</td>
<td>3</td>
</tr>
<tr>
<td>ami 49-3</td>
<td>49</td>
<td>4</td>
<td>6</td>
</tr>
</tbody>
</table>

The following notations will be used in the benchmark circuits: xerox-1 and hp-1 contains four alignment blocks only. Xerox-2 and hp-2 contains four alignment blocks and two performance blocks. ami33-1 and ami49-1 consists of four and five alignment blocks ami33-2 and ami49-2 contains four alignments and three performance blocks. Table 3.3 and 3.4 compares area (2.05%) and runtime (43.99%) between SA and DE with alignment and performance constraints. Table 3.5 represents the wire length estimation of the algorithm. When the performance blocks are used, it reduces the critical net delay and it leads to decrease in the total wire length of the module.
Figure 3.14  The Result Packing of ami33-2. Block 1, 2, 3 and 4 are to be Aligned, and Block 5, 6 and 7 are Performance-constrained in a Critical Net

Figure 3.15  The result Packing of ami49-3. Block 1, 2, 3 and 4 are to be Aligned, and there are two Performance Constraints (block 5, 6 and 7 are Connected in a Critical Net, and Block 30, 34 and 44 are Belonging to Another Critical Net)
3.6 CONCLUSION

The proposed method uses an efficient and effective DE optimization algorithm for floorplan structure with constraints based on B*-tree representation. The proposed DE algorithm guarantees a feasible placement solution with alignment constraints and generates a good placement with performance constraints during each operation for giving support to sequential data transfer (bus or pipeline signals). MCNC benchmark circuit’s data are used for testing and the results are very promising. Our approach generates a good placement solution with an optimum area and a short run time. Another major advantage using DE is the reduction of computation time by proposing a trial solution which eliminates the need to evaluate the objective function every time. Also, efficient packing is obtained with alignment and performance constraints.