Force-Directed Partitioning Technique for 3D IC

Fakheri Tabrizi, Aysa

UNIVERSITY OF CALGARY

Force-Directed Partitioning Technique for 3D IC

by

Aysa Fakheri Tabrizi

A THESIS

SUBMITTED TO THE FACULTY OF GRADUATE STUDIES

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE

DEGREE OF MASTER OF SCIENCE

DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING

CALGARY, ALBERTA

January, 2013

Abstract

3D IC design is one of the challenging problems of today. 3D partitioning solutions can significantly impact manufacturability and performance of a circuit. In this work, a 3D partitioning technique is developed that reduces the number of TSVs by using force directed placement technique. A circuit is partitioned into several layers and a force directed placement problem is solved to find the optimal locations of the partitions. This partitioning solution is improved by using a proposed force-based simulated annealing technique. The proposed technique is tested on ISPD04 circuits, and shows up to 20% reduction in the number of TSVs.

Acknowledgements

I would like to express my profound gratitude to my supervisor Dr. Laleh Behjat for her invaluable guidance, infinite patience, and constant support. I would also like to show my greatest appreciation to Dr. Bill Swartz for his invaluable advices during this research. My gratitude also goes to my committee members for their insightful comments. In addition, I would like to thank all my teachers in the past for their motivations and encouragement and all the knowledge that I learned.

I would like to acknowledge the Department of Electrical and Computer Engineering wonderful staff for keeping things running smoothly.

This work would not have been possible without the support from colleagues at the lab: Dr. Logan Rakai, Amin, Bardia, Delaram, Yangyang, and Emily. Special thanks go to Dr. Logan Rakai, Amin and Bardia for constructive discussions. I would like to thank my friends in Calgary for all their support and all the fun we have had and my friends all over the world for being always beside me despite the distance. I owe special thanks to Mohammad and Benyamin for their assistance in writing this thesis. I would like to thank Mohammad for his support and keeping me encouraged during writing this thesis.

Last but not least, I would like to thank my family members, my mom and dad and my brother Arash for their unconditional love and support.

Table of Contents

List of Tables

List of Figures and Illustrations

List of Symbols, Abbreviations and Nomenclature

Symbol	Definition
1D	One Dimensional
2D	Two Dimensional
3D	Three Dimensional
AMG	Algebraic Multigrid
CAD	Computer Aided Design
EC	Edge Coarsening
FM	Fiduccia-Mattheyses
FSA	Force-based Simulated Annealing
HEC	Hyperedge Coarsening
I/O	Input/Output
IC	Integrated Circuit
ISPD	International Symposium on Physical Design
KL	Kernighan-Lin
LOP	Linear Ordering Problem
MFSA	Modified Force-based Simulated Annealing
MHEC	Modified Hyperedge Coarsening
NP-hard	Non-deterministic Polynomial-time hard
SA	Simulated Annealing
TPR	Three Dimensional Place and Route
TSV	Through Silicon Via

Via	Vertical interconnect access
VLSI	Very Large Scaled Integration

Chapter 1

Introduction

In Very Large Scaled Integration (VLSI), millions of circuit components are integrated in a single chip. For example, Intel's Xeon X7460 released in 2008 contains more than 1.9 billion transistors [1]. As Moore predicted, in the past 50 years, the number of transistors have been increasing at exponential rates [2]. Therefore, innovative techniques had and still have to be designed to manage the complexity of the circuits. One of these techniques is to implement circuits in a Three Dimensional (3D) platform where several layers of silicon are stacked on top of each other [3, 4, 5, 6]. In this thesis, the partitioning problem for the physical design of 3D Integrated Circuits (ICs) is considered.

The rest of this chapter is organized as follows: In Section 1.1, the main stages of physical design are explained. In Section 1.2, the motivations of the thesis are discussed. Thesis contributions are briefly presented in Section 1.3. Finally, the organization of the thesis is described in Section 1.4.

1.1 Physical Design

Physical design is one of the main steps in IC design where the physical properties such as the exact locations of components and wires of the circuit are determined. Physical design includes several major stages: partitioning, floor planning, placement and routing [7, 8, 9, 10].

In the partitioning stage, the circuit is divided into several partitions that are relatively independent, i.e. as few wires as possible connect them. These partitions then can be designed individually or in parallel. As partitioning is the focus of this thesis, a more comprehensive introduction of this stage is presented in Chapter 2.

The arrangement of the partitions and their dimensions are determined in floor planning step. In the placement stage, the exact locations of the circuit components are determined. The main objective of placement is minimizing the total wire length while ensuring routability. Placement usually takes place in two phases: global placement and detailed placement. In global placement the approximate locations of the cells are determined. In this stage cells are allowed to overlap. In detailed placement, the exact locations to the cells are obtained and any overlaps are removed. Once all cell locations are determined, the paths for all of the wires of the circuit are determined in the routing stage.

1.2 Motivations

3D design has become a viable solution to reduce wire length and enhance integration in circuits [11, 12, 13, 14]. One of the biggest issues to be addressed during 3D IC integration is the communication between tiers using Through-Silicon Vias (TSVs). Minimizing the total number of TSVs is of upmost importance, as they are expensive, bulky and unreliable [15].

One of the most effective ways to reduce the number of TSVs is to use partitioning. Partitioning of Two Dimensional (2D) circuits is a well developed field and is usually performed by multi-level algorithms such as hMetis [16, 17]. However, 2D partitioning techniques do not consider the number of times a net is cut, where net refers to the interconnection between circuit elements, or how many partitions exist between the end points of a net. In a 3D circuit, partitions are stacked and any connection between two non-adjacent partitions results in using more than one TSV. For example, considering a 3D IC with 5 tiers, if a terminal of a net is placed in the top tier and another terminal in the bottom tier, the net is effectively cut 4 times and 4 TSVs are required to complete one single connection. Therefor, during 3D partitioning a designer has two main optimization criteria: reduce the number of nets between partitions, and determine the optimal locations of tiers. In this thesis it is proposed to consider floor planning during 3D partitioning to reduce the number of TSVs and obtain better integration for 3D ICs.

Another motivation of this thesis is to show that different heuristics should be used for different situations to obtain best results. For example, multilevel partitioning techniques are very effective in finding a good partitioning solution in a small amount of time, however, they do not give any indication on where partitions should be placed or how and in which direction cells have to move in order to further improve their results. Therefore, it is proposed to combine a multilevel technique with other techniques such as force directed placement and simulated annealing to obtain best results. In other words, in this thesis it is proposed to use different partitioning techniques in different stages of the design in order to harness the strengths of each method.

1.3 Thesis Contributions

In this work different optimization methods are combined to perform the 3D IC partitioning. These methods are used in different stages of IC physical design such as 2D partitioning, floor planning, and placement. Since these methods are developed targeting different applications they are not suitable for 3D IC partitioning in their current state. The main contributions of this thesis are as follows:

- Development of the a force directed partitioning technique.
- Development of force-based simulated annealing technique.
- Up to 20% reduction in the number of interconnections between Tiers.

A brief description of these contributions is given in the following.

Development of the a force directed partitioning technique:

In this thesis it is proposed to use state-of-the-art technique for 2D circuit partitioning which minimizes the number of connections between all partitions. However in a 3D IC, objective function is different from that of 2D IC and the goal is to minimize the number of vias between partitions that are stacked on one dimension. In this thesis, it is proposed to use a one dimensional placement to find a suitable ordering for the partitions. Ordering the initial partitions results in locating the partitions that have more connections close to each other and consequently reduce the length of connections and number of long connections.

Development of force-directed simulated annealing technique:

After the initial 3D partitioning solution is obtained an iterative improvement technique can be used to improve the results. Simulated Annealing (SA) [18] is shown to be effective to improve the partitioning solutions. In original SA, the cells to be moved and the direction of their move between the partitions are selected randomly. In this thesis, a variation is proposed to SA where random moves are replaced with moves that are directed by forces applied to cells. This method is combined with other methods in solving the particular 3D IC partitioning problem; however, it can be used in many other optimization problems. Unlike the original SA in which the current solution is replaced by a random neighbor solution, in the developed FSA, the selection of the new solution is probabilistic, i.e., not only the acceptance of a move, but also selecting the new solution is based on a probability. This probability of selection is based on the system information. In 3D partitioning problem this information is proposed to be the forces that connected cells impose on each other. Finally, numerical results on benchmarks released by IBM show that the proposed techniques outperform the existing 3D partitioning solutions for most cases.

1.4 Thesis Structure

This thesis is structured as follows:

- **Chapter 2:** In this chapter the background for partitioning, force-directed placement, simulated annealing and 3D IC design are given.

- **Chapter 3:** In this chapter one of the main contributions of this thesis, combining of several partitioning and floor planning techniques to obtain better 3D partitions, is given. In addition, the experimental results of the 3D partitioning are given.

- **Chapter 4:** In this chapter, a novel force-directed iterative improvement technique is introduced to further improve the partitioning solution based on the tier location of the cells. In addition, experimental results obtained for 3D partitioning are given and compared with other 3D partitioning techniques.

- **Chapter 5:** Finally, a summary of the contributions of the thesis and future work are given in this chapter.

Chapter 2

Background

2.1 Introduction

This chapter provides background on circuit partitioning and 3D ICs. Moreover, the techniques used in this thesis, simulated annealing and force directed placement, are reviewed.

This chapter is organized as follows: In Section 2.2, the partitioning problem is defined and existing partitioning algorithms are identified. Then, 3D IC, 3D IC partitioning problem and the existing 3D IC partitioning techniques are presented in Section 2.3 . In Section 2.4, SA meta-heuristic is described. In Section 2.5, the force directed placement method is reviewed.

2.2 Partitioning

The modern integrated circuits consist of millions of transistors [15]. Due to this large and ever increasing scale of modern circuits, the design of such circuits has become a complex task. A common strategy to handle the complexity and to perform a computationally feasible design is to implement a divide and conquer strategy where circuits are partitioned into smaller sub-circuits which can be designed independently. Because of the large number of transistors, it is very difficult to make a full-chip layout. Hence, it is required to divide the circuit into smaller sub-circuits which can be designed individually or in parallel. The procedure of dividing a circuit into smaller sub-circuits is called partitioning.

The wires that connect two different sub-circuits can be much more costly than those that are confined to a single sub-circuit. Moreover, as the design of each sub-circuit is done independently and without considering the other partitions in most cases, it is desired to generate partitions as independently as possible. Therefore, minimizing the connections between two sub-circuits is of great significance.

2.2.1 Terminology

A circuit is a collection of logic elements, referred to in this thesis as cells, that are connected to each other. The connection between the cells are called nets. Figure 2.1 illustrates a simple circuit schematic. There are also contact points that connect some cells to the outside of a circuit. These contact points are called Input/Output Pins (I/O Pins).

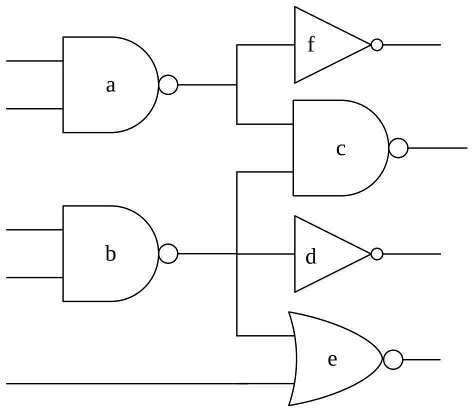


Figure 2.1: An example of a simple circuit schematic

Connectivity information of cells can be represented in different ways such as hypergraph/graph representation, netlist representation and connectivity matrix [7].

Each one of the representations is described in further detail here.

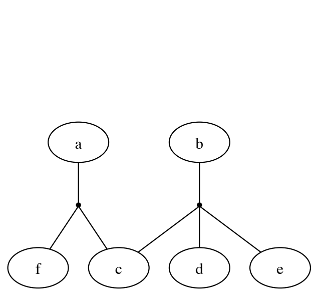
Hypergraph/Graph representation

A circuit can be represented by a graph $G(V, E)$ where V is the set of vertices or nodes that includes the cells and set E represents the set of edges that are the connection between cells, i.e. nets [19].

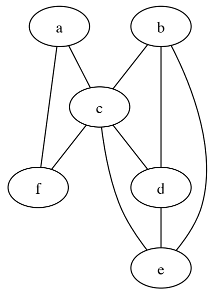
In the graph representation each edge connects to only two cells. Since in practical circuit design, there are lots of nets that are connected to more than two cells, a graph representation is not an adequate representation of a circuit and hypergraph representation is mostly used. A hypergraph is generalization of graph in which one edge can connect two or more than two vertices. The nets that connect more than two cells are called hyperedges [19].

In Figure 2.2, a hypergraph representation of the circuit with five cells, shown in Figure 2.1, and its corresponding graph representation are shown. In this figure vertices a to e represent the logic elements a to e shown in Figure 2.1 and the the edges represent the connection between logic elements. In this example, in converting a hypergraph to a graph, all hyperedges are converted to a set of edges that connect each pair of the nodes connected to a given hyperedge.

A multigraph is a graph that can have more than one edge between a pair of nodes. A multigraph can be used to represent the nets with different weights. The weight, $w(e)$, for an edge, $e \in E$, represents the weight of the corresponding edge. Finally, in a circuit, each logic element has a certain area. In this thesis, $area(v)$, for each $v \in V$ represents the area of each cell.



(a) Hypergraph



(b) Graph

Figure 2.2: Hypergraph and graph representation of the circuit shown in Figure 2.1

Netlist representation

A netlist contains the names of the nets followed by the names of all the cells that are connected to a given net. An example of a netlist representation of the circuit shown in Figure 2.1 is given in Figure 2.3.

```
node:   6
nets:   2

net     0:      a       f       c
net     1:      b       c       d       e
```

Figure 2.3: Netlist representation of the schematic given in Figure 2.1

Connectivity matrix

A connectivity matrix, C, is an $n \times n$ symmetric matrix where n is equal to the number of all cells including the I/Os. The value of element $C(i, j), i \neq j$ is equal to the weight of the connection between cells i and j. Element $C(i, j), i \neq j$ of C is zero if cell i is not connected to cell j and non-zero otherwise. The diagonal elements of the matrix are equal to the sum of the weights of the nets that are connected to the cell. Note that the connectivity matrix doesn't show the information of nets i.e. which cells are connected to a particular net. An example of a connectivity matrix for the schematic of Figure 2.1 is shown in Figure 2.4.

$$\mathbf{C} = \begin{array}{c} \\ a \\ b \\ c \\ d \\ e \\ f \end{array} \begin{array}{c} \begin{array}{cccccc} a & b & c & d & e & f \end{array} \\ \begin{bmatrix} 2 & 0 & 1 & 0 & 0 & 1 \\ 0 & 3 & 1 & 1 & 1 & 0 \\ 1 & 1 & 5 & 1 & 1 & 1 \\ 0 & 1 & 1 & 3 & 1 & 0 \\ 0 & 1 & 1 & 1 & 3 & 0 \\ 1 & 0 & 1 & 0 & 0 & 2 \end{bmatrix} \end{array}$$

Figure 2.4: Connectivity matrix of the schematic given in Figure 2.1

2.2.2 2D Partitioning Problem

The k-way partitioning divides a circuit into k partitions and is a combinatorial optimization problem[1]where the goal is to divide a circuit into several partitions that have as few wires as possible connecting them [8].

The first goal of partitioning is to divide the circuit in a way that the number of connections between the sub-circuits is minimized. The constraint of the problem is normally to keep balance between the size of the partitions, or in other worlds it is desired to produce partitions that are roughly from the same sizes. The mathematical formulation can be stated as [8, 9]:

$$\begin{aligned} \min \quad & \sum_{e \in \Phi} w(e), \\ s.t. \quad & \sum_{v \in V_i} area(v) \leq \frac{1}{k} \sum_{v \in V} area(v) = \frac{1}{k} area(V), \end{aligned}$$

[1]A combinatorial optimization problem is a problem of finding the optimal solution from the finite number of possible solutions [20].

where Φ denotes the set of cut edges and $w(e)$ is the weight of each cut edge. $V_i \subseteq V, i = 1, ..., k$ and k is the number of partitions. Since partitioning is an Non-deterministic Polynomial-time hard (NP-hard) problem [8], no deterministic algorithm is known which can find the optimal solution in polynomial time. However, several heuristics [21, 22, 23, 24] such as Kernighan-Lin (KL) algorithm [25] and Fidducia-Mattheyses (FM) algorithm [26] are developed that can improve the quality of a given partition. These algorithms are described in detail in the following sections. In these algorithms the partitioning problem is formulated as a graph partitioning problem. The vertices of the graph represent the circuit's nodes and the edges represent the interconnections between the cells or the nets.

With the increase in the number of circuit element, partitioning of 2D circuits is now performed by multi-level hypergraph partitioning algorithms such as hMetis [27, 17]. These multi-level algorithms on their own are based on iterative improvement techniques such as FM [26].

2.2.3 Kernighan Lin (KL) Algorithm

Kernighan-Lin heuristic, proposed by B. W. Kernighan and S. Lin in 1970, is one of the earliest works in solving the partitioning problem [25]. The original algorithm was developed to perform 2-way partitioning where the cell areas are equal. KL algorithm divides a graph $G(V, E)$ with $|V| = 2n$ nodes to two sets with n nodes in a way that the two sets have the minimum number of connections with each other.

In KL an initial partitioning solution is improved by swapping two nodes from different partitions. The algorithm selects the pair of nodes that their exchange results in highest net cut reduction.

KL is an iterative algorithm. The first iteration starts with arbitrary initial partitions i.e. the nodes are divided to two equal size sets A and B where each set consists of n nodes. Then a pair of nodes from different partitions with maximum gain is selected and swapped. Once the nodes are swapped, they are fixed and are not allowed to be selected for other swaps until all nodes are fixed. After every swap the gains of free nodes are updated and the pair selection and the procedure is repeated until all nodes are fixed. At this point all nodes become free and the algorithm starts the next iteration and the pair swapping is repeated. The algorithm terminates if no improvement can be made during a given iteration.

The algorithm is described step by step after gain calculation and related concepts are presented. To calculate the gain of a pair of nodes, first the cost of moving a node is calculated. The following equation shows the cost of moving a node that belongs to partition A to the partition B:

$$D(v) = |E_{AB}(v)| - |E_A(v)|, \tag{2.1}$$

where $E_{AB}(v)$ is the set of edges connected to node v that also has nodes in partition B and $E_A(v)$ is the set of edges connected to v that only has nodes in partition A. In other words, $E_{AB}(v)$ are the set of edges connected to node v that are cut by cut line and $E_A(v)$ are the set of edges that are not cut by the cut line. The value of $D(v)$ can be positive or negative. The high $D(v)$ value indicates that the benefit of moving node v is high.

The gain of swapping two nodes from different partitions is calculated by adding the cost of the move of each node and twice subtracting the connection weight between the pair. Notice that subtracting the connection weight between the nodes is necessary because the cost of each node is calculated with the assumption of not moving the other node. So the connected edge that was cut before the swap is counted in both cost calculations as moving only one node changes the status of the given edge to uncut, where it remains cut by swapping the pair. The gain of the x^{th} swap in a given iteration $g_x(i,j)$, is calculated by:

$$g_x(i,j) = D(i) + D(j) - 2C(i,j), \tag{2.2}$$

where D(i) and D(j) are the cost of moving node i and j respectively and C(i,j) is the connectivity weight between node i and node j.

The value of gain shows the reduction in cut size when the given pair is swapped. A positive gain indicates that the swap improves the quality of partitioning and a negative gain means the swap increases the cut size and lowers the quality of partitions. G_x is the cumulative gain that is calculated after each swap in a given iteration.

$$\begin{aligned} G_x &= \sum_{i=1}^{x} g_i \\ G_m &= max(G_x) \end{aligned}$$

The swap with maximum G_x indicates that the partitioned graph after performing the given swap has the highest quality among all configurations of iteration. Therefore the moves are only applied up to swap number m. Notice that all the moves that are found until all the nodes are fixed were in order to find G_m and the swaps are only applied up to swap number m.

KL Algorithm Description

The KL partitioning algorithm (Algorithm 1) starts with dividing the graph to two random equal size initial partitions (line 1-2). G_m gets the initial value of ∞ and iteration number is set to 1 (line 3-4). During each iteration, while G_m is greater than zero (line 5) all the nodes are set as free and the cost of all nodes are calculated (line 6-9). x is set to 1 to indicate the first swap (line 10). While there exist free nodes in the graph (line 11) the gain of swapping each pair of free nodes are calculated and the pair with maximum gain is swapped and fixed (line 12-18). The cumulative gain is calculated (line 19) and the cost of free nodes that are connected to swapped nodes are updated (line 20-22). The algorithm moves to next swap of the iteration (line 23). When all the nodes are fixed (line 24) the swap number with maximum cumulative gain is determined (line 25) and swaps up to a determined swap number are confirmed (line 26-28). The algorithm goes to next iteration if G_m is greater than zero (line 29-30) and terminates otherwise (line 31).

Example 2.2.1 *An example of a given circuit represented as a graph and the initial partitions are shown in Figure 2.5. In this figure, nodes a to e represent the cells and edges of the graph represent the nets between the cells. The dashed line divides circuit into two initial partitions. The initial cut size is 8. The cost and the gain of each node can be calculated as follows:*

$D(a) = 1 - 0 = 1$, $D(b) = 2 - 1 = 1$, $D(c) = 2 - 1 = 1$, $D(d) = 3 - 2 = 1$

$D(e) = 1 - 1 = 0$, $D(f) = 3 - 1 = 2$, $D(g) = 1 - 1 = 0$, $D(h) = 2 - 0 = 2$

$g_1(a, h) = 1 + 2 - 2 \times 0 = 3$

Algorithm 1 KL partitioning Algorithm

Input: Graph $G(V, E)$

Output: Partitioned graph $G(V, E)$

```
1: Begin
2: $A, B \leftarrow$ random initial partitions
3: $G_m \leftarrow \infty$
4: $iteration \leftarrow 1$
5: while $G_m > 0$ do
6:     for all $v \in V$ do
7:         $S(v) \leftarrow free$
8:         Compute $D(v)$
9:     end for
10:    $x \leftarrow 1$
11:    while there are free nodes do
12:        for all $i \in A, S(i) = free$ do
13:            for all $j \in B, S(j) = free$ do
14:                $g_x(i, j) \leftarrow$ gain of swapping $i$ and $j$
15:            end for
16:        end for
17:        swap $a$,$b$ with $max(g_x)$
18:        $S(a), S(b) \leftarrow fixed$
19:        Compute $G_x$
20:        for all free $v$ connected to $a$ or $b$ do
21:            Update $D(v)$
22:        end for
23:        $x \leftarrow x + 1$
24:    end while
25:    $G_m = max(G_x)$
26:    if $G_m > 0$ then
27:        Confirm swaps $x = 1, .., m$
28:    end if
29:    $iteration \leftarrow iteration + 1$
30: end while
31: End.
```

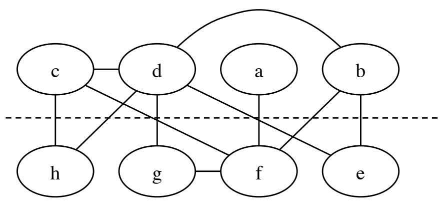


Figure 2.5: Initial partitions of an example circuit

The maximum gain belongs to a and h. Nodes a and h are swapped and fixed and is shown in Figure 2.6(a). $D(v)$ is updated for all free nodes that are connected to swapped nodes, a and h, which are c, d, and f.

$$D(c) = 1 - 2 = -1,\ D(d) = 2 - 3 = -1,\ D(f) = 2 - 2 = 0$$

$$g_2(b, g) = 1 + 0 - 2 \times 0 = 1$$

Nodes b and g are swapped and fixed shown in Figure 2.6(b).

$D(v)$ is updated for all free nodes that are connected to swapped nodes b and g which are d, e, and f.

$$D(d) = 2 - 3 = -1, D(e) = 1 - 1 = 0, D(f) = 2 - 2 = 0$$

$$g_3(c, e) = -1 + 0 - 2 \times 0 = -1$$

Nodes c and e are swapped and fixed.

$D(v)$ is updated for all free nodes that are connected to swapped nodes c and e which are d, and f.

$$D(d) = 2 - 3 = -1,\ D(f) = 2 - 2 = 0$$

$$g_4(d, f) = -1 + 0 - 2 \times 0 = -1$$

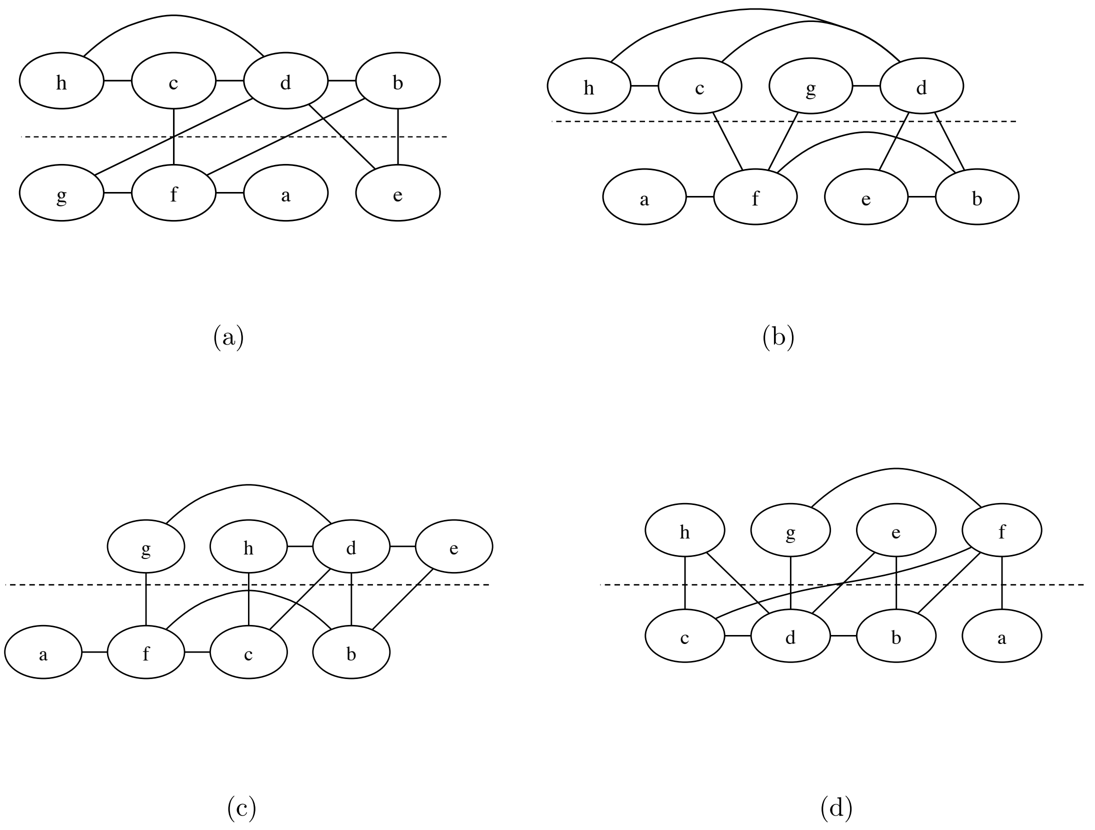


Figure 2.6: Steps of KL algorithm in the first iteration for the example in Figure 2.5

Nodes d and f are swapped and fixed.

Cumulative gain is computed.

$$G_1 = g_1 = 3$$

$$G_2 = g_1 + g_2 = 4$$

$$G_3 = g_1 + g_2 + g_3 = 3$$

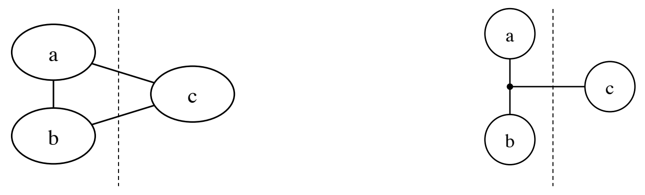
$$G_4 = g_1 + g_2 + g_3 + g_4 = 2$$

The maximum positive cumulative gain G_m is equal to 4 with $m = 2$. Therefore the first 2 swaps are performed and the algorithm goes to next iteration (Figure 2.6(b)). The cut size is reduced to 4 from 8 in first iteration.

2.2.4 Fiduccia-Mattheyses (FM) Algorithm

Feduccia-Mattheyeses is a partitioning heuristic introduced by Feduccia and Mattheyeses in 1982 [26] . FM divides the graph $G(V, E)$ into partitions so as minimize the total cut nets.

Unlike KL that swaps a pair of cells between two partitions, FM moves cells independently which makes the algorithm more flexible in solving unequal partition sizes. Another advantage of FM over KL is that hypergraphs can be used in FM. While KL minimizes the number of edges, FM minimizes the number of nets cut. That means if one net is connected to more than two cells and the net is cut, KL might count the cut more than once, while FM counts that only once which is practical. Figure 2.7 shows the shortcoming of partitioning methods using graph representation vs. hypergraph representation. Another advantage of FM is that it is applicable on graphs with different node areas.



(a) Grapgh representation (b) Hypergraph representation

Figure 2.7: Difference between the calculation of cuts in a graph, [a], and hypergraph, [b], representation of a circuit

Definitions used in FM

Cut - a net is cut if it has cells in more than one partition.

Uncut - a net is uncut if all its connected cells are located in one partition.

Gain - gain of a cell is the change in the number of cuts when cell is moved to other partition. The gain is defined as

$$g(v) = FS(v) - TE(v)$$

where $FS(v)$ is the number of cut nets that are only connected to v in v's partition. $TE(v)$ are the nets, connected to v, that all their other cells are in v's partition .i.e the uncut nets that are connected to v.

There are 3 type of nets connected to a node v. The first type is the nets that are cut and have other cells connected in the same partition that v is located. Moving v to the other partition doesn't affect the state of these kind of nets i.e. they remain cut. The second types are the nets that are cut and are only connected to v in v's partition. Moving v to the other partition changes the state of these nets to uncut, since all their connected cells are in the partition that v is moved to. These are the nets that are counted in calculating $FS(v)$. The third type of nets are the nets that are uncut. So moving cell v to the other partition changes the state of these nets to cut. These are the nets that are counted in calculating $TE(v)$.

Cumulative gain - G_s is the cumulative gain after s^{th} move in an iteration. As in KL, after each move in an iteration, the cumulative gain is calculated and at the end of the iteration the maximum cumulative gain G_m and its corresponding move number m is determined. Then, the moves are performed up to m^{th} move.

$$\begin{aligned} G_s &= \sum_{i=1}^{s} g_i \\ (G_m, m) &= (max(G_s), s) \end{aligned}$$

Critical nets - A critical net is a net that is either uncut or has exactly one cell in one partition and the rest of its cells in the other partition. These are the only nets that moving their connected cells might change their state from cut to uncut or vice versa.

Critical cells - Critical cells are the cells that are connected to the critical nets.

Ratio factor- The ratio factor is a parameter that shows the relation of the size of a partition to the size of graph.

$$r = \frac{|A|}{|A| + |B|}$$

where $|A|$ and $|B|$ are the sizes of the partition A and B respectively i.e. the total respective area of the all cells in partition A and partition B and

$$|A| + |B| = |V|,$$

where, V is the total area of all cells.

To set the balance criterion the size of the largest node must be considered. The partitioning is balanced if

$$r.|V| - |v|_{max} \leq |A| \leq r.|V| + |v|_{max},$$

where, $|v|_{max}$ is the size of the largest node in graph.

FM Algorithm description

The Algorithm 2 presents the pseudocode for FM algorithm. FM starts with computing the balance criterion (line 1-2) and arbitrarily divides the graph into two subgraphs or partitions (line 3). The value of G_m is set to ∞ and the first iteration starts (line 4-5). All the nodes are set as free and FS, TE, and g are calculated for all (line 7-13). A free cell that maximizes gain while satisfying the balance criterion is selected, moved to the opposite partition and fixed, then the track of moved cell is kept (line 15-19). The *criticalnets* and *criticalcells* connected to the moved cell are determined (line 20-21). The gain is updated for the critical cells that are connected to the moved cell i.e. the cells that are connected to the moved cell by critical nets (line 22-26). The cumulative gain is calculated (line 27). Then algorithm continues to the next move (line 28). When all the nodes are fixed (line 29) the move number with maximum cumulative gain is determined (line 30) and moves up to the determined move number are confirmed (line 31-33). The algorithm goes to next iteration if G_m is greater than zero (line 34-35) and terminates otherwise (line 36).

Algorithm 2 FM partitioning Algorithm

Input: Graph $G(V,E)$

Output: Partitioned graph $G(V,E)$

```
1: Begin
2: Balance criterion
3: initial partition
4: $G_m \leftarrow \infty$
5: $iteration \leftarrow 1$
6: while $G_m > 0$ do
7:     $G_s \leftarrow 0$
8:     for all $v \in V$ do
9:         $S(v) \leftarrow free$
10:        $FS(v) \leftarrow$ number of nets connected to $v$ and cut
11:        $TE(v) \leftarrow$ number of nets connected to $v$ but not cut
12:        $g_x(i,j) \leftarrow FS(v) - TE(v)$
13:    end for
14:    $x \leftarrow 1$
15:    while there are free nodes do
16:        $a \leftarrow$ cell with max gain
17:        $order \leftarrow \{order, a\}$
18:        move $a$ to opposite partition
19:        $S(a) \leftarrow fixed$
20:        $criticalnets \leftarrow$ nets connected to $a$
21:        $criticalcells \leftarrow$ cells connected to $criticalnets$
22:        for all $c \in criticalcells$ connected to $a$ do
23:            if thenS(c) == free
24:                Update $g_x(c)$
25:            end if
26:        end for
27:        $G_s \leftarrow G_s + g_x(a)$
28:        $x \leftarrow x + 1$
29:    end while
30:    $G_m = max(G_s)$
31:    if $G_m > 0$ then
32:        Confirm swaps $order(1:m)$
33:    end if
34:    $iteration \leftarrow iteration + 1$
35: end while
36: End.
```

Example 2.2.2 *The weighted cells and the nets of a circuit and the ratio factor for partitioning are given. The initial partitions are as shown in Figure 2.8.*

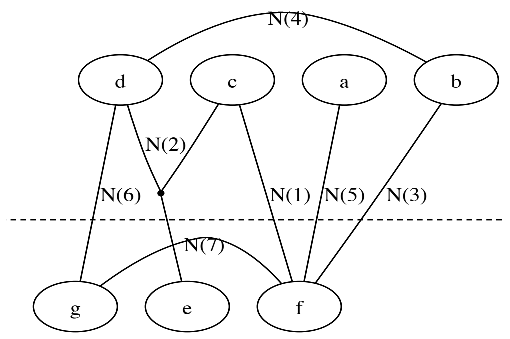


Figure 2.8: An example of FM partitioning with initial partitions

Cell areas:

$$area(a) = 4,\ area(b) = 2,\ area(c) = 2,\ area(d) = 5,$$

$$area(e) = 1,\ area(f) = 5,\ area(g) = 3.$$

The nets are as follows:

$$N(1) = (c, f),\ N(2) = (c, d, e),\ N(3) = (b, f),$$

$$N(4) = (b, d),\ N(5) = (a, f),\ N(6) = (d, g).$$

And the balance factor is $r = 0.4$*. Tie breaking rule: alphabetical order. Perform the* *first iteration of FM algorithm. Balance criterion can be calculated as follows:*

$$r.|V| - |v|_{max} \leq |A| \leq r.|V| + |v|_{max}$$

$$r.|V| - |v|_{max} = 0.4 \cdot 22 - 5 = 3.8$$

$$r.|V| - |v|_{max} = 0.4 \cdot 22 + 5 = 13.8$$

$$3.8 \leq |A| \leq 13.8$$

Initial partition sizes: $|A| = 13, |B| = 9$

$$FS(a) = 1,\ TE(a) = 0,\ g_1(a) = 1$$

$$FS(b) = 1,\ TE(b) = 1,\ g_1(b) = 0$$

$$FS(c) = 1,\ TE(c) = 0,\ g_1(c) = 1$$

$$FS(d) = 1,\ TE(d) = 1,\ g_1(d) = 0$$

$$FS(e) = 1,\ TE(e) = 0,\ g_1(e) = 1$$

$$FS(f) = 3,\ TE(f) = 1,\ g_1(f) = 2$$

$$FS(g) = 1,\ TE(g) = 1,\ g_1(g) = 0$$

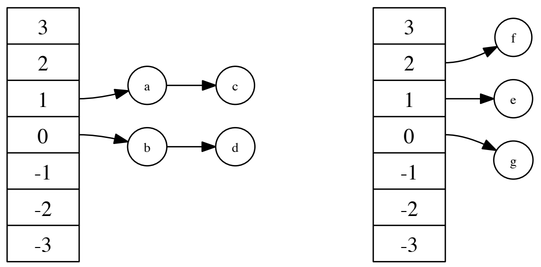


Figure 2.9: Gain list of cells in the example

The best gain is 2 which can be obtained by moving f.

Balance criterion after moving f *is* $|A| = 18, |B| = 4$ *which doesn't satisfy the balance criterion. The next maximum gain is selected. Possible cells are* $a, c,$ *and* e *with the gain of 1.*

Balance criterion after moving a *is* $|A| = 9, |B| = 13$

Balance criterion after moving c *is* $|A| = 11, |B| = 11$

Balance criterion after moving e *is* $|A| = 14, |B| = 8$

All three moves satisfy the balance criterion, the a *is selected based on alphabetical order.*

Now the critical nets connected to a *and the corresponding critical cells are determined and the gain of critical cells are updated.*

$g_1 = 1$

criticalnets connected to a: $N(5)$

criticalcells: f

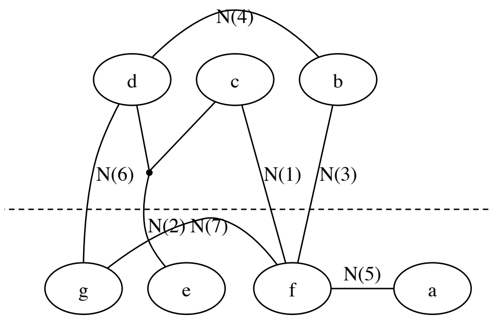


Figure 2.10: Graph representing the circuit after the first move

The best gain is 1 which can be obtained by moving c *or* e.

Balance criterion after moving c *is* $|A| = 7, |B| = 15$

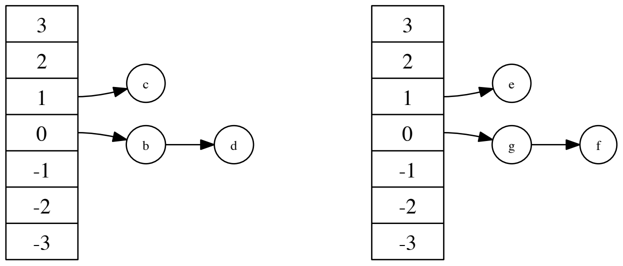


Figure 2.11: Updated gain list of cells in the example

Balance criterion after moving e is $|A| = 10, |B| = 12$

Moving c doesn't satisfy the balance criterion and cell e is selected to move.

$g_2 = 1$

criticalnets connected to e: $N(6)$

criticalcells: c, d

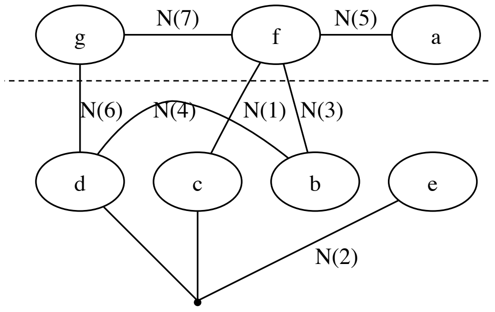


Figure 2.12: Graph representing the circuit after the second move

The best gain is 0 which can be obtained by moving b, c, f*, or* g*.*

Balance criterion after moving b is $|A| = 8, |B| = 14$

Balance criterion after moving c is $|A| = 8, |B| = 14$

Balance criterion after moving f *is* $|A| = 15, |B| = 7$

Balance criterion after moving g *is* $|A| = 13, |B| = 9$

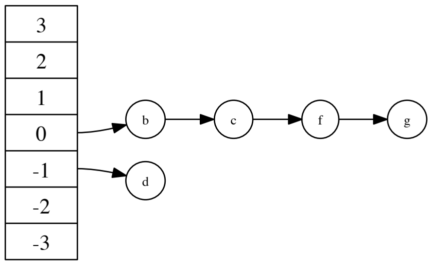


Figure 2.13: Updated gain list of cells in the example

Only moving g satisfies the balance criterion, so cell g is selected to move.

$g_3 = 0$

criticalnets connected to g: $N(6), N(7)$

criticalcells: d, f

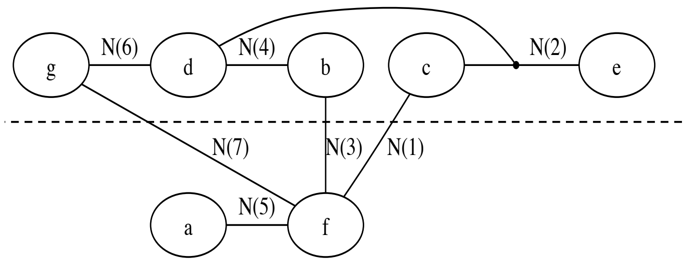


Figure 2.14: Graph representing the circuit after the third move

The best gain is 2 which can be obtained by moving f; however it doesn't satisfy the balance criterion. and b with the next best gain of 0 *is selected.*

Balance criterion after moving b is $|A| = 11, |B| = 11$

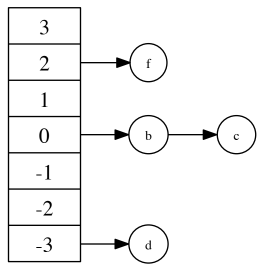


Figure 2.15: Updated gain list of cells in the example

$g_4 = 0$

criticalnets connected to b: $N(3), N(4)$

criticalcells: d, f

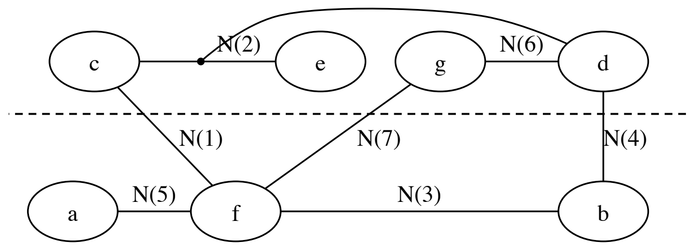


Figure 2.16: Graph representing the circuit after the fourth move

The next cell to be moved is c with the gain of 0 *and satisfying the balance criterion.*

Balance criterion after moving c is $|A| = 9, |B| = 13$

$g_5 = 0$

criticalnets connected to c: $N(1), N(2)$

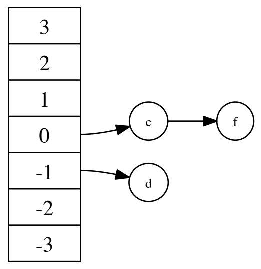


Figure 2.17: Updated gain list of cells in the example

criticalcells: f, d

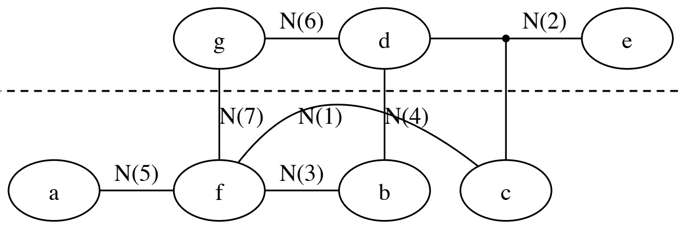


Figure 2.18: Graph representing the circuit after the fifth move

The best gain belongs to d however it doesn't satisfy the balance criterion. Therefore, f is moved.

Balance criterion after moving f is $|A| = 14, |B| = 8$

$g_6 = -2$

criticalnets connected to c: $N(1), N(3), N(5), N(7)$

No criticalcells.

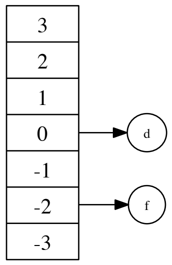


Figure 2.19: Updated gain list of cells in the example

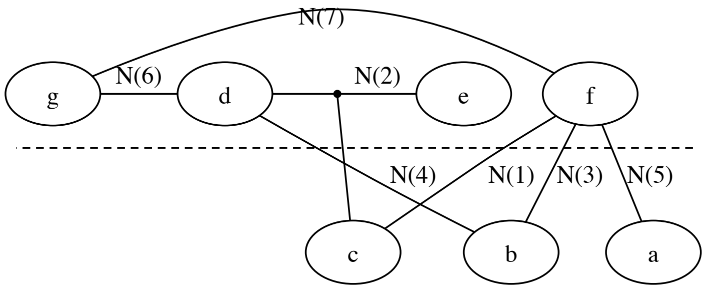


Figure 2.20: Graph representing the circuit after the sixth move

Cell e is moved.

$g_7 = 0$

In order to find the best move sequence the cumulative gains are calculated.

$G_1 = g_1 = 1$

$G_2 = g_1 + g_2 = 1 + 1 = 2$

$G_3 = g_1 + g_2 + g_3 = 1 + 1 + 0 = 2$

$G_4 = g_1 + g_2 + g_3 + g_4 = 1 + 1 + 0 + 0 = 2$

$G_5 = g_1 + g_2 + g_3 + g_4 + g_5 = 1 + 1 + 0 + 0 + 0 = 2$

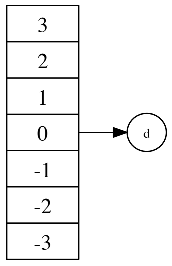


Figure 2.21: Updated gain list of cells in the example

$$G_6 = g_1 + g_2 + g_3 + g_4 + g_5 + g_6 = 1 + 1 + 0 + 0 + 0 + (-2) = 0$$

$$G_7 = g_1 + g_2 + g_3 + g_4 + g_5 + g_6 + g_7 = 1 + 1 + 0 + 0 + 0 + (-2) + 0 = 0$$

2.2.5 Multilevel Partitioning

As circuit sizes have been growing exponentially, new techniques were needed to deal with this increase. In multilevel partitioning methods first the size of a graph is reduced using clustering techniques in the coarsening phase. Then, the coarsened circuit is partitioned using partitioning techniques in the initial partitioning phase. The last step of the algorithm is uncoarsening and refinement phase [27, 16, 17]. In Figure 2.22 the three phases of multilevel hypergraph partitioning are shown. Each one of these phases is described in the following.

Coarsening phase

During this phase the smaller hypergraphs are constructed by clustering groups of vertices with common hyperedges. The clustered vertices form a single vertices and the hyperedges inside a cluster are removed from the reduced hypergraph. In Figure 2.22 two levels of coarsening is shown. In this figure the original hypergraph consists

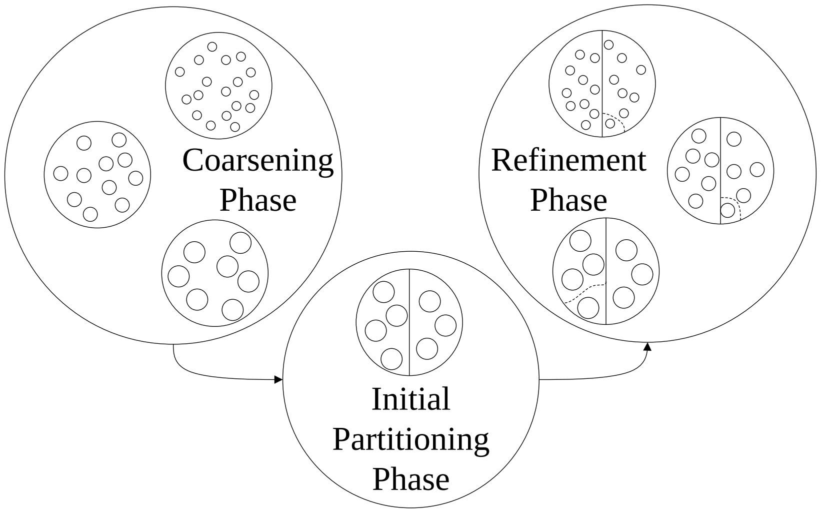


Figure 2.22: Multilevel partitioning flowchart

of 18 vertices and the number of vertices are reduced to 12 after one level coarsening and it is reduced to 7 vertices after two level coarsening. There are different methods to cluster the vertices such as Edge Coarsening (EC), Hyperedge Coarsening (HEC), Modified Hyperedge Coarsening (MHEC) [27, 16, 17], best choice [28] and Netcluster [29].

Initial partitioning phase

At this phase, the coarsest hypergraph is partitioned such that it has the smallest number of net cut, and the sizes of the partitions are almost equal. FM algorithm [26] is mainly used to perform this step. The coarsest hypergraph typically contains 75-200 nodes for which the FM algorithm performance is nearly optimal.

Uncoarsening and refinement phase

In this phase at each level the coarse hypergraph is uncoarsened to one level finer hypergraph and every sub-cluster is assigned to the partition to which its parent cluster was located and a refinement algorithm is used to reduce the number of net cuts by moving the vertices between the partitions while maintaining the balance criteria. The most common algorithms that is used for the refinement is FM algorithm. Uncoarsening and refinement continue until the original graph is partitioned.

2.3 3D IC

A 3D IC is a single chip in which the circuit elements are integrated in multiple vertical layers, called tiers or dies [3, 4, 5, 6, 30]. The interconnection between the tiers are known as 3D-Vias or TSVs of a 3D IC is shown in Figure 2.23.

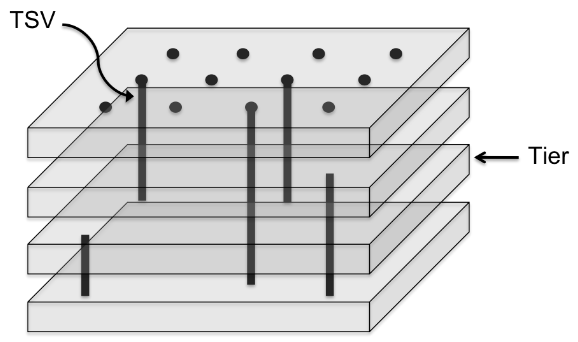


Figure 2.23: An example of a 3D circuit with TSV

3D technology is a promising technology that aims to provide higher integration and reduced wire length. The higher integration results in smaller footprint and the reduced wire length reduces the power consumption of the circuit which leads to longer battery life and less operational cost. Also, reducing the wire length by 3D technology, especially reducing the length of long wires can decrease circuit delay.

3D ICs are not widely used yet; however among the notable 3D circuits, the 3D version of Pentium 4 CPU, presented by Intel in 2004, can be mentioned which demonstrates 15% power saving and 15% performance improvement compared to its 2D version [1].

With the advances of 3D ICs, many new problems are added to the domain of physical design that are unique to these circuits. That's because the optimal design for 3D circuit is different from conventional 2D circuit design. Unlike the conventional 2D circuits, the circuit components lie on multiple tiers. One of the motivations of 3D technology is reducing the connection lengths.

The example illustrated in Figure 2.24 shows how 3D technology reduces the wire length. For a given circuit, partitioning the circuit to four partitions as a 2D circuit, the longest connection would be between the point a and point b that would be $4l$. However, 3D technology reduce the length of longest connection to $2l$ as shown in figure. Notice that TSV are used for the vertical connection and the thickness is not added to wire length. Reducing the wire length results in reduction of circuit delay as circuit delay is directly proportional to wire length. Also reduced wire length results in a better wire distribution and less congestion and less power consumption.

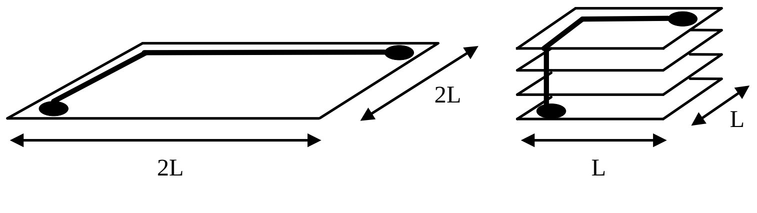


Figure 2.24: An example that shows how the 3D technology reduces the wire length

2.3.1 3D partitioning

In 3D IC partitioning, each partition forms a tier and the TSVs are used for the connections between layers [31, 32, 33]. The connection between two adjacent layer consumes 1 TSV but connection between two non-adjacent layer consumes $m - n$ TSVs, where m and n are the numbers of the given layers, $m > n$. For example in a 4 tier circuit, a connection between tier 2 and tier 4 needs two TSVs. Therefore a 3D partitioning problem can be formulated as:

$$\min \sum_{e_{m,n} \in \Phi} w(e_{m,n})|m - n|, \tag{2.3}$$

$$s.t. \quad \sum_{v \in V_i} area(v) \leq \frac{1}{k} \sum_{v \in V} area(v).$$

2.3.2 3D Partitioning Techniques

Several 3D computer-aided design tools exist. Three Dimensional Place and Route (TPR) is CAD tool that performs 3D physical design [34]. This tool uses hMetis algorithm to perform partitioning and divides the netlist to the number of partitions equal to the number of desired tiers. After dividing a circuit to tiers, the cells are placed and routed in each tier.

The partitioning step of TPR is performed using hMetis algorithm. Then, the partitions are ordered using a linear placement technique to minimize the total number of nets that are cut between two adjacent layers.

I/O pins method

In I/O pin method [35, 11], 3D partitioning is performed by first partitioning and placing the I/O pins and then the cells based on the I/O locations. The reason that this algorithm balances the I/Os is that the analytical placers use the I/O locations as the starting point.

There are different versions of I/O pins partitioning: *Alternate pins, Unlocked pins, I/O Pins method* and *Refinement I/O Pins method.*

The *I/O Pins method* algorithm computes the logical distance between I/O pins and considers the distance between each pair as the net weight in the first step. In order to find the logical distance between a pair of I/O pins the shortest paths between them are found. The algorithm uses hMetis tool to partition the I/O pins. Then the cells are partitioned based on the I/O pin locations and uses SA to stack the partitions.

The *Alternate pins* assigns the pins to the partitions alternatively and randomly. This algorithm replaces the first step of I/O pin method by assigning I/O nodes alternatively to different tiers. This algorithm provides a solution with balanced I/O pins.

The *Unlocked pins* partitions the whole netlist using hMetis and uses SA to stack the partitions. In this method I/O pins are not spread evenly.

Refinement I/O Pins method performs a refinement on solutions obtained from The *I/O Pins method.* In its refinement stage, the algorithm generates random perturbations of a solution and accepts or rejects the new solution based on cost variation. The new solution that decreases the cost is accepted and the solution that increases the cost is rejected.

The perform perturbation on a solution two alternatives are considered and one is selected each time with the probability of 50%. Either a cell or a I/O pin is randomly moved to a random partition, or a pair of random cells or I/O pins are exchanged.

The cost function is composed of three factors: Number of 3D vias, area balance and I/O pin balance. The cost function is represented by following equation:

$$Cost = \frac{(w_v \times v)}{v_1} + \frac{(w_a \times a)}{a_1} + \frac{(w_p \times p)}{p_1}$$

where v, a, and p are normalized values for usage of 3D vias, area and I/O pin balance respectively. v is is the sum of square via number of each net. a is the subtraction of area of the largest tier by smallest one. And p is sum of I/O pin area of the tier with largest I/O pin area subtracted by sum of I/O pin area of the tier with smallest I/O pin area. v, a, and p are divided by their initial value before any perturbation v_1, a_1, and p_1 to be normalized. w_v, w_a, and w_p are the weights imposed to tune the cost.

2.4 Simulated Annealing (SA)

SA is an iterative meta-heuristic optimization method introduced in [18, 36], based on Metroplis algorithm presented in [37]. This method simulates the process of annealing of metal, in which the metal is heated up to the melting temperature and then gradually cooled to allow the molecules (atomic structure) to get to an optimum energy state. Heating the metal allows the atoms to move from their initial position, wander randomly and move to higher energy states. Cooling gradually moves the atoms to lower energy states and finds better configuration than the initial configuration. That means the material which is in a certain energy level, changes its state to higher energy by heating and then goes to a lower energy level by being cooled gradually.

In simulated annealing, the cost of function is analogous to the energy level of the material. The objective of an optimization problem is to minimize the cost function[2](i.e. maximize the fitness function[3]), where the objective of annealing in metallurgy is to minimize the energy level. A high value for cost function of the problem is analogous to high energy state of a metal and a low value for cost function of the problem is analogous to low energy state of a metal.

The search space is simulated to the possible configurations of atoms. Each variable of the problem acts as an atom and possible values for each variable are analogous to possible positions of an atom. A candidate solution corresponds to a configuration of material.

The temperature in each step of annealing is simulated to a parameter T in SA. T is a parameter to be set according to the problem. Setting the initial T and the ratio of its decreasing depends on the problem. i.e. a certain value for initial T could be too high in solving one problem where the same value for initial T might be very low in solving another problem depending on the ratio of change in cost by a small change in the value of variables. This can be described by referring to annealing process. e.g. in metallurgy the material is heated to the stress-relief point which is called annealing temperature or annealing point. The annealing point and cooling rate for different materials are different e.g. for the glass it depends on thickness, thermal conductivity, heat capacity, and other features of the glass. In Table 2.1, the

[2]Cost function is the objective function of a minimization problem.
[3]Fitness function is the objective function of a maximization problem.

analogies between annealing and SA are described.

Table 2.1: Annealing analogy

Analogy	Metallurgy	Energy Level
	General Optimization Problem	Cost of Function
	Specific 3D Partitioning Problem	Number of TSVs
Analogy	Metallurgy	Possible configuration of atoms
	General Optimization Problem	Search space
	Specific 3D Partitioning Problem	Position of cells in different tiers
Analogy	Metallurgy	Atom
	General Optimization Problem	Variable
	Specific 3D Partitioning Problem	Cell
Analogy	Metallurgy	Position of an atom
	General Optimization Problem	Value of a variable
	Specific 3D Partitioning Problem	Location of a cell
Analogy	Metallurgy	Configuration of Material
	General Optimization Problem	Candidate solution
	Specific 3D Partitioning Problem	Partitioning Solution
Analogy	Matalurgy	Temperature
	General Optimization Problem	T Parameter
	Specific 3D Partitioning Problem	T Parameter
Analogy	Matalurgy	New Configuration with small change in atoms position
	General Optimization Problem	Neighbor solution
	Specific 3D Partitioning Problem	New partitioning Solution by moving a few cells between partitions

SA starts from an initial solution in the feasible space. This initial solution can be either random or a semi optimal solution obtained by another optimization method. Unlike hill climbing algorithms[4] that reject new solutions with the cost more than the cost of the current solution, SA accepts solutions with higher cost with a certain probability. At each iteration, a new candidate solution is selected randomly in the neighbourhood of the current solution. If the cost of the new solution is lower than the

[4]Hill climbing algorithms are iterative local search algorithms that start with an arbitrary solution and generate new solution by a change in the current solution. If the change produces a better solution, the current solution is replaced with the new solution repeating until no further improvements can be found.

cost of the current solution, the new solution is accepted. Otherwise the probability of its acceptance depends on the difference of the cost of the current solution and the candidate solution and the value of T. The higher the value of T, the higher is the chance of accepting the solution with higher cost. At the early stages of the algorithm, the T is set to be large and the probability of acceptance of high cost solutions is close to one. Therefore the algorithm gives chance for both uphill and downhill moves. Uphill moves are the moves that increase the cost of solution and downhill moves are the moves that decrease the cost of solution. As the temperature is decreased during the algorithm operation the probability of accepting high cost solutions goes down.

SA is an iterative method. In each iteration, algorithm considers a neighbour solution s' and decides between moving to the neighbour solution s' or staying at the current solution s. The decision is based on the quality of neighbour solution or a probability. If the neighbour solution s' is better than current solution s, i.e. The value of function at s' is smaller than the value of function at s, then s is replaced by s'. Otherwise, if s' results in a higher cost, the algorithm accepts the s' with a probability. This probability depends on the difference of costs between current solution and the candidate solution and also on the varying parameter T. This probability is specified by the acceptance probability function P which is based on Boltzmann acceptance criterion [18]:

$$P = e^{-\Delta cost/T} \tag{2.4}$$

$$\Delta cost = cost_{new} - cost_{current}$$

The neighbour solution is a state in search space that is produced by a small perturbation to the current solution i.e. a small change in value of one or more variable. The action of changing the state is called a "move". For a particular problem different moves can be considered.

Algorithm 3 Simulated Annealing

Input: Initial Solution s_{init}
Output: Optimized Solution

```
 1: Begin
 2: $T \leftarrow T_0$
 3: $i \leftarrow 0$
 4: $s \leftarrow s_{init}$
 5: $cost_{current} \leftarrow Cost(s)$
 6: while $T > T_{min}$ do
 7:     while stopping criteria do
 8:         $i \leftarrow i + 1$
 9:         $s' \leftarrow$ neighbor($s$)
10:         $cost_{new} \leftarrow Cost(s')$
11:         $\Delta cost = cost_{new} - cost_{current}$
12:         if $\Delta cost < 0$ then
13:             $s \leftarrow s'$
14:             $cost_{current} \leftarrow cost_{new}$
15:         else
16:             $r \leftarrow Random(0, 1)$
17:             if $r < e^{-\Delta cost/T}$ then
18:                 $s \leftarrow s'$
19:                 $cost_{current} \leftarrow cost_{new}$
20:             end if
21:         end if
22:     end while
23:     $T \leftarrow \alpha.T$
24: end while
25: End.
```

2.5 Force Directed Placement

In recent years, many placement algorithms have been proposed to minimize a measure of total wire length [38, 39, 40]. Analytical placers are designed based on the global optimization algorithms and have recently received much consideration from both academia and industry. These placers usually spread the cells over the placement area and minimize the total wire length simultaneously. Force directed placement is a type of analytical placement during which cells and their connections are modelled as a mass-spring system [41]. The connected cells attract each other and the magnitude of attraction is directly proportional to the distance between the cells or the connection length. In mass-spring system, the cells, which are free to move, will move in the direction of their forces and will settle in a configuration with the minimum energy. In the force directed placement, cells that are free to move are allowed to move in the direction of the forces applied to them. Therefore, the placement problem is reduced to solving an optimization problem.

The most used wire length measure is the squared Euclidean distance as it is a continuously twice differentiable function. Therefore, the placement problem formulation is as follows:

$$\min L = \sum_{i=1,j=1}^{n} c(i,j)((x_i - x_j)^2 + (y_i - y_j)^2), \tag{2.5}$$

where n represents the total number of cells and $c(i,j)$ is the corresponding connection cost between the cells i and j if they are connected and 0, otherwise. During global placement, each dimension can be considered independently. Therefore, the formulation can be divided into two separate optimization problems as follows:

$$\min L_x = \sum_{i=1,j=1}^{n} c(i,j)(x_i - x_j)^2,$$
$$\min L_y = \sum_{i=1,j=1}^{n} c(i,j)(y_i - y_j)^2.$$

This type of force directed technique is in a new capacity to perform partitioning in this thesis.

Chapter 3

Constructive 3D Partitioning

3.1 Introduction

In 3D partitioning, the quality of the solution does not only depend on the quality of the partitioning stage, but also on the quality of the floor planning where the locations of different tiers are determined.

A major contribution of this thesis is combining several partitioning and floor planning techniques so as to obtain better 3D partitions. The proposed technique reduces the number of nets between partitions and the number of times that a net is cut, and consequently, reduces the TSV usage by both providing high quality partitions and reducing the number of long connections.

First, different 2D partitioning methods are studied and the best technique for obtaining initial partitions is determined. In addition, experiments are performed to determine the best trade-off between the number of initial partitions and run time of the algorithm. The 1D force-directed placement is developed and solved for the partitioning problem to obtain the best locations of the partitions. Finally, it is proposed to obtain an initial 3D partitioning solution by merging layers together.

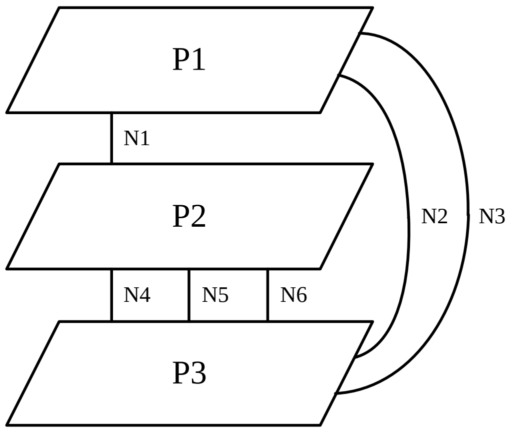
The rest of this chapter is organized as fallows. The proposed constructive 3D partitioning algorithm is introduced and described in Section 3.3. In Section 3.4, the results of the proposed 3D partitioning are given. Finally, a summary of the chapter is given in 3.5

3.2 Problem Statement

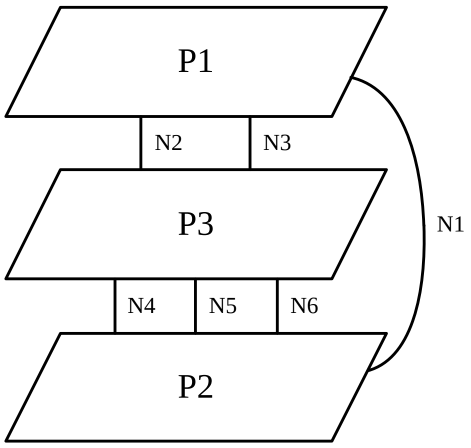
In a 3D circuit, partitions are stacked on top of each other and any connection between two non-adjacent partitions results in using more than one TSV. For example, considering a 3D IC with 5 tiers, if one terminal of a net is placed in the top tier and the other terminal in the bottom tier, the net is effectively cut 4 times and 4 TSVs will be required to complete one single connection. The example in Figure 3.1 shows how the location of tiers affects the number of TSVs. Stacking the partitions in the order shown in 3.1(a) results in usage of 8 TSVs, while changing the order of partitions to the order shown in 3.1(b) reduces the number of TSV usage to 7.

On the other hand, the produced partitions targeting 2D partitioning are not always the optimal partitions targeting 3D partitioning, even if they are stacked in their best possible order. This fact is shown using an example given in Figure 3.2. In 3.2(a) the best 2D partitioning solution of a circuit is given, where the number of 2D net cuts is equal to 6. Theses partitions are stacked in their best order in 3.2(a) and the number of 3D net cuts is equal to 8. However another partitioning solution for the same circuit, which is given in 3.2(b), results in 7 2D net cuts and 7 3D net cuts in its best stacking order. This is an example showing that the optimal partitions for 3D circuits may be different from the optimal partitions targeting 2D partitioning.

In this thesis, the goal of 3D partitioning is set to be *to minimize the number of nets that are cut by any horizontal line drawn between any pair of adjacent tiers.*

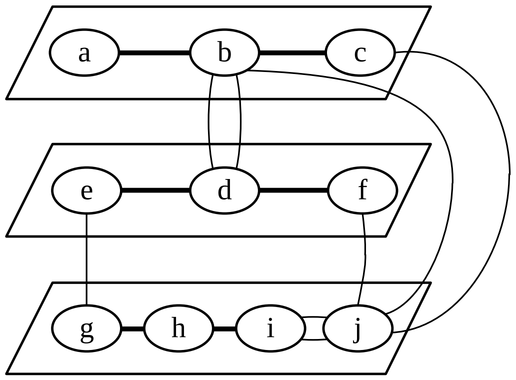


(a) Net cuts in 3D partitioning = 8

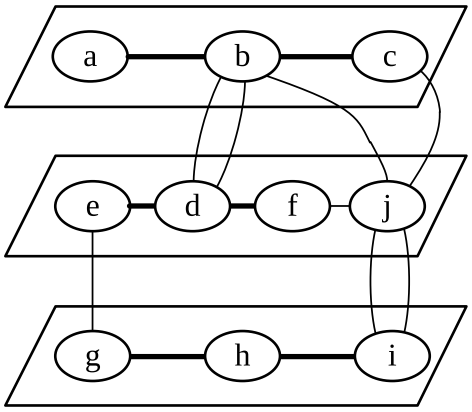


(b) Net cuts in 3D partitioning = 7

Figure 3.1: The effect of stacking the partitions in proper order in reducing the number of vias.



(a) Optimal partitioning solution targeting 2D circuit



(b) Optimal partitioning solution targeting 3D circuit

Figure 3.2: An example that shows the optimal 2D partitioning solution is not always the optimal 3D partitioning solution. The weight of thick edges is equal to 20 and the weight of narrow edges is equal to 1.

This means that, during partitioning, two optimization criteria are considered: form the optimal tiers, and determine the optimal locations of different tiers. These two optimization can not be done in series, since it is not possible to verify the optimal partitions before they are stacked. In this work, a technique that considers both of the mentioned criteria is introduced.

In developing the proposed constructive 3D partitioning algorithm, partitioning methods are combined with floor planning methods and the partitioning and floor planning stages are performed simultaneously.

3.3 Proposed Constructive 3D Partitioning

In this work a new approach is utilized to form the initial tiers i.e it is proposed to divide the circuit to several partitions (3 to 5 times of the number of desired tiers) instead of traditionally dividing the circuit to the exact number of desired tiers and form the tiers by finding the best combination of initial partitions while determining the best location of partitions at the same time.

Among the different partitioning methods, multilevel hypergraph partitioning is selected to perform initial partitioning. hMetis tool is used to perform this task, which provides high quality 2D partitions and renders an efficient run time of the algorithm. Another technique used in the proposed algorithm is the linear ordering, which is mainly used in floor planning stage of the physical design. The algorithm arranges the previously generated partitions in a single row in a way that minimizes the length of connections and consequently the number of net cuts. Ordering the initial partitions rather than stacking them randomly results in locating the partitions that have more connections close to each other. This consequently reduces the length of connections and the number of long connections and gives the semi optimal combination of the partitions to form the tiers.

The flowchart of proposed 3D constructive partitioning algorithm is shown in Figure 3.3. This algorithm has three main phases: k-way partitioning, stacking and merging. The steps of the algorithm are described in this section. First, the netlist and the desired number of tiers are fed to the algorithm. Then, an initial k-way partitioning is performed. A detailed description of this step can be found in Subsection 3.3.1. At this point and after the initial partitioning, each initial partition is selected as a seed cell (parameter S) and forms the bottom layer and the following procedure is implemented: (1) Stacking, through solving a linear ordering problem, for which the detailed explanation can be found in Subsection 3.3.2. The ordered partitions form the initial layers of the circuit. At this point, each cell belongs to a layer which has a location on the z-axes assigned to it. (2) merging, whose details are given elsewhere (refer to Subsection 3.3.3). Via merging, depending on the desired number of tiers, the initial layers are combined and form the initial tiers. Afterwards, the number of TSVs is counted to determine the suitable order that generates the minimum number of TSVs.

The inputs of the algorithm shown in Figure 3.3 are the netlist and the desired number of tiers, m. The output of the algorithm is the 3D partitioned netlist i.e. the tiers and their order and the number of TSVs needed. First, the netlist is divided into n partitions using multilevel hypergraph partitioning technique, where n is a

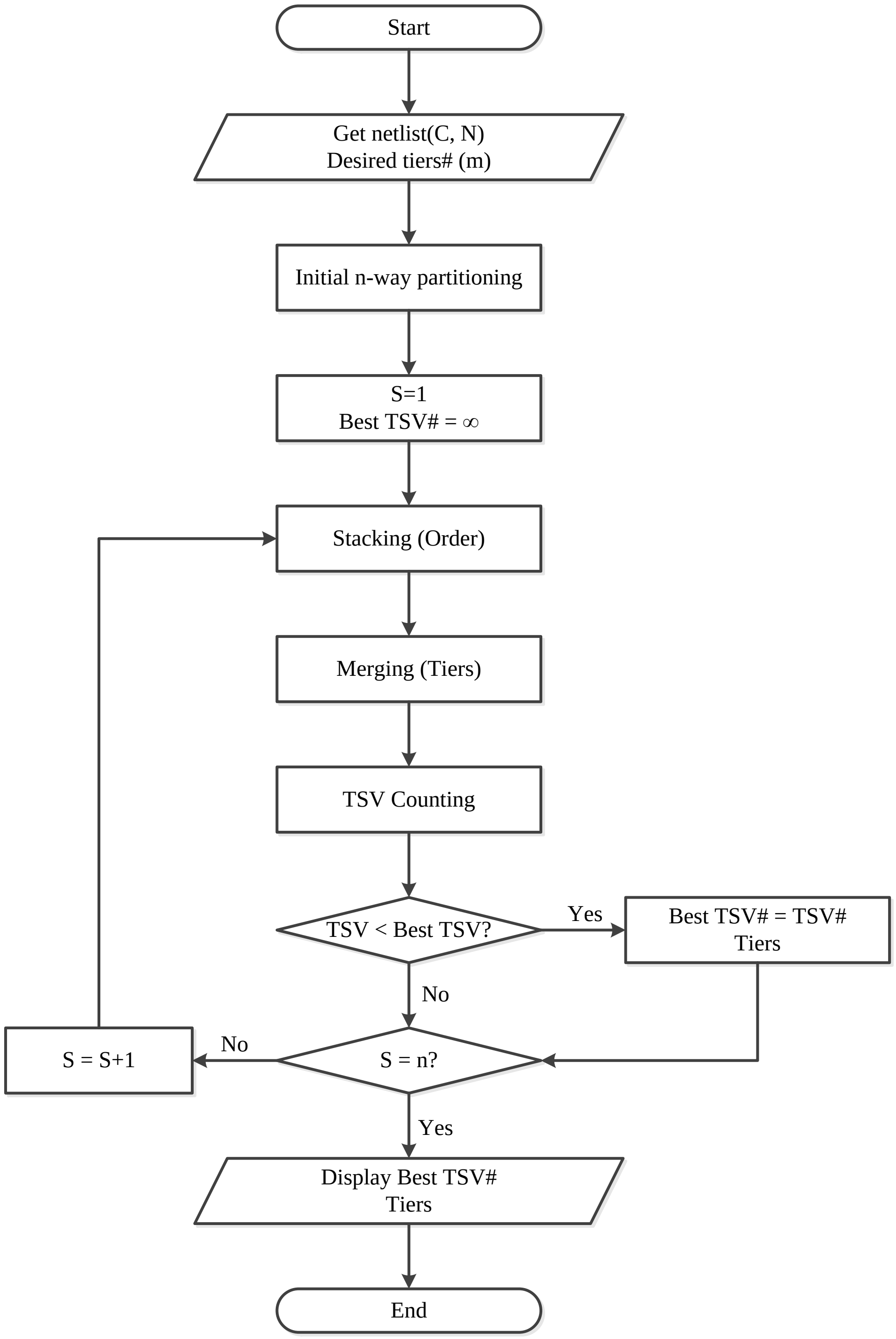


Figure 3.3: Proposed constructive algorithm flowchart

multiple of m. Then the generated partitions are stacked using a linear ordering heuristic. Every n/m successive layers are merged and form the 3D tiers.

In Section 3.3.1 the initial partitioning step is described. The linear ordering stage and merging the layers are described in Sections 3.3.2 and 3.3.3 respectively.

3.3.1 Multilevel Partitioning

In the first phase of the algorithm, the netlist is divided into several partitions using the method of multilevel hypergraph partitioning. Normally, the number of partitions is several times larger than the desired number of tiers. hMetis tool is used to form these partitions. The partitions are produced in a way that they have minimum numbers of connections with each other. These partitions provide a good initial solution in a reasonable time; however, this solution is not tailor made for 3D ICs. This algorithm divides netlist into partitions in a way that they have minimum interconnections with each other, but doesn't consider the number of times that a net is cut when the partitions are stacked.

3.3.2 Linear Ordering

The partitions obtained in the previous step have the minimum number of connections with each other; however, stacking them in different orders can result in different number of vias. The example circuit given in Figure 3.1 shows how the number of cuts changes when the partitions are reordered.

As the number of TSVs are equal to the number of nets that are cut by any horizontal line drawn between any two adjacent tiers, at this point it's desired to stack the initial partitions in an order that the number of nets that are cut by any horizontal line drawn between any two partitions is minimized.

In a general linear ordering problem, there are p objects to place in order [42]. This problem can be modeled as an integer programming.

$$\begin{aligned} \min \quad & \sum_{i=1}^{p} \sum_{\substack{j=1 \\ i \neq j}}^{p} g_{ij} x_{ij} \\ \text{s.t.} \quad & x \textit{ is the incidence vector of a tournament}^1 \\ & x_{ij} + x_{ji} = 1, \qquad \forall\, 1 \leq i, j \leq p \in \ N \\ & x \in \{0, 1\}, \end{aligned}$$

where x_{ij} for $i, j = 1, ..., p$ are the variables for the relative locations of partitions i and j. The value 1 for x_{ij} indicates that i is before j, otherwise x_{ij} is 0. g_{ij} is the cost of placing i before j. p is the total number of partitions.

This problem can be restated as:

$$\begin{aligned} \min \quad & \sum_{i=1}^{p-1} \sum_{j=i+1}^{p} c_{ij} x_{ij} \\ \text{s.t.} \quad & x_{ij} + x_{jk} - x_{ik} \leq 1, && 1 \leq i < j < k \leq p \\ & -x_{ij} - x_{jk} + x_{ik} \leq 0, && 1 \leq i < j < k \leq p \\ & x \in \{0, 1\} \end{aligned}$$

where $c_{ij} = g_{ij} - g_{ji}$.

[1]Tournament is a complete directed acyclic graph.

The problem of finding the best order of partitions is called Linear Ordering Problem (LOP) which is also an NP-hard optimization problem. In [42] a heuristic is developed in order to solve the linear ordering problem effectively. This algorithm is given in Algorithm 4. This method divides the nets connected to each partition to three different categories, i.e. terminating nets, new nets and continuing nets. Terminating nets are the nets that are terminated by placing the given partition. That means the terminating net is not connected to any other partition that are unplaced. New nets are the nets that start by placing the given partition. That means the new net is not connected to any partition that are placed in order. Continuing nets are the nets that are connected to at least one partition among ordered partitions and at least one unplaced partition. The algorithm starts with choosing a seed partition. Then, the partition that has the highest difference between the number of terminating nets and new nets is placed on top of the seed cell. In case of tie, the partition with the highest number of terminating nets is selected. In case of another tie, the partition that has the highest number of continuing nets is selected. If there are still multiple partitions that possess the same number of continuing nets, the partition with the lowest connection is selected. The algorithm continues by selecting the next partitions and stacking until no other partitions are left. The algorithm for the linear ordering is given in Algorithm 4.

3.3.3 Layer Merging

At this point, the layers that have a large number of connections are located close to each other and the number of long connections are reduced, so every m/n successive

Algorithm 4 Linear Ordering

Input: Set of initial partitions

Output: Order of initial partitions

```
1: Begin
2: select $Seed$ partition
3: $Order \leftarrow Seed$
4: $Set \leftarrow Set - Seed$
5: while $Set \neq \{\}$ do
6:     select the partition with highest (Terminating nets - New nets)
7:     if there is a tie then
8:         Select the partition with largest (Terminating nets)
9:         if there is a tie then
10:            Select the partition with most continuing nets
11:            if there is a tie then
12:                Select the partition with the fewest connected cells
13:                if there is a tie then
14:                    Select randomly
15:                end if
16:            end if
17:        end if
18:    end if
19:    Add the selected partition to $Order$
20:    Remove the selected partition from $Set$
21: end while
22: End.
```

layers are merged to form n initial Tiers. Therefore, in this procedure, each of $n-1$ cut lines is assumed as a border that separates each n tier from one another. Alternatively, we could have moved the cut lines back and forth provided the balance criteria is not disturbed in order to obtain a smaller number of TSVs. This is attributed to the fact that at a later stage (Chapter 4), where the SA is utilized for improving the solution, the simulation will not be that efficient if it is started from the lower temperatures.

3.4 Numerical Results

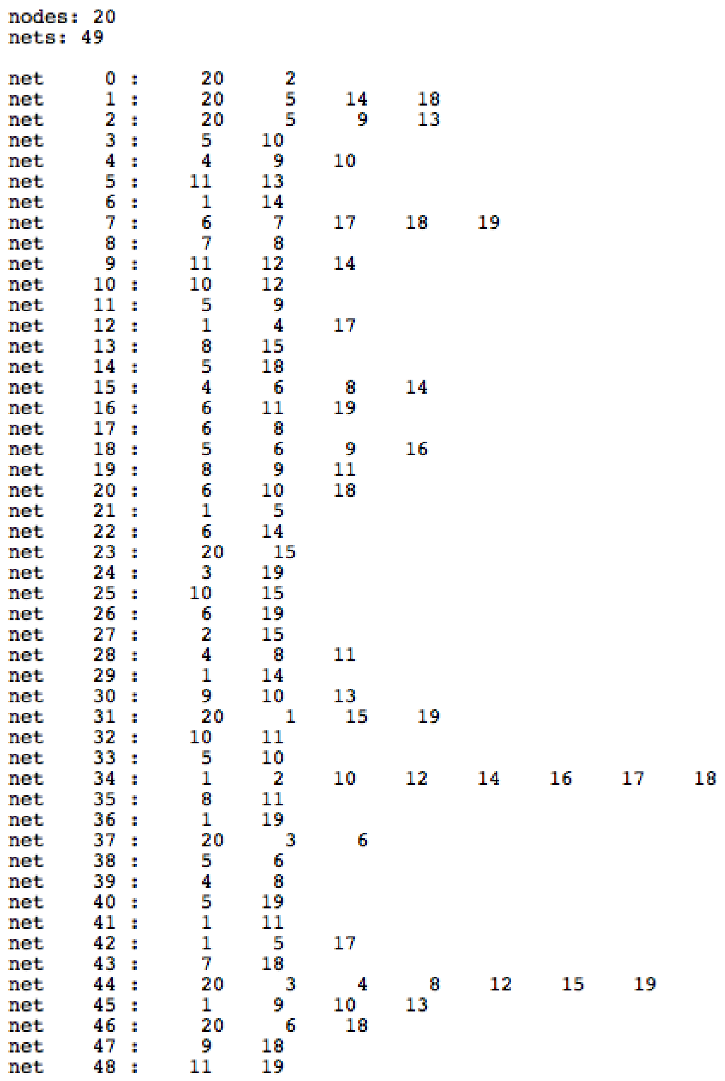
The effectiveness of the proposed partitioning technique is verified via empirical tests on **ISPD04** benchmark circuits [43]. All algorithms are written in Matlab. In addition, the steps of the proposed algorithm are explained by using a simple netlist consisting of 20 nodes and 49 edges and hyperedges. Explaining the algorithm with this simple circuit makes the algorithm to be understood better and it helps to have a profound insight into the goal and procedure of the algorithm. The netlist and graph representation of this circuit are shown in Figure 3.4 and Figure 3.5. In Figure 3.5 there are 20 circles that represent 20 cells of the circuit and the lines show the nets. The simple circuit shown in this figure illustrates the complexity of a circuit and gives an insight that how complex a circuit with millions of cells can be.

3.4.1 Benchmarks

In Table 3.1, the characteristics of the benchmark circuits including the number of cells, I/Os and nets [43] are shown.

3.4.2 Initial Partitioning Results

The algorithm starts by dividing the netlist to several initial partitions. these initial partitions are combined after linear ordering and form the initial tiers. One of the challenges in developing this algorithm was to determine the number of initial partitions. It is proposed to set the number of initial partitions equal to or greater than the number of desired tiers to be able to merge and form the initial tiers. In order to find the best number of initial partitions, empirical experiments were performed



```
nodes: 20
nets: 49

net      0 :      20      2
net      1 :      20      5     14     18
net      2 :      20      5      9     13
net      3 :      5     10
net      4 :      4      9     10
net      5 :     11     13
net      6 :      1     14
net      7 :      6      7     17     18     19
net      8 :      7      8
net      9 :     11     12     14
net     10 :     10     12
net     11 :      5      9
net     12 :      1      4     17
net     13 :      8     15
net     14 :      5     18
net     15 :      4      6      8     14
net     16 :      6     11     19
net     17 :      6      8
net     18 :      5      6      9     16
net     19 :      8      9     11
net     20 :      6     10     18
net     21 :      1      5
net     22 :      6     14
net     23 :     20     15
net     24 :      3     19
net     25 :     10     15
net     26 :      6     19
net     27 :      2     15
net     28 :      4      8     11
net     29 :      1     14
net     30 :      9     10     13
net     31 :     20      1     15     19
net     32 :     10     11
net     33 :      5     10
net     34 :      1      2     10     12     14     16     17     18
net     35 :      8     11
net     36 :      1     19
net     37 :     20      3      6
net     38 :      5      6
net     39 :      4      8
net     40 :      5     19
net     41 :      1     11
net     42 :      1      5     17
net     43 :      7     18
net     44 :     20      3      4      8     12     15     19
net     45 :      1      9     10     13
net     46 :     20      6     18
net     47 :      9     18
net     48 :     11     19
```

Figure 3.4: The net list representation of the example circuit

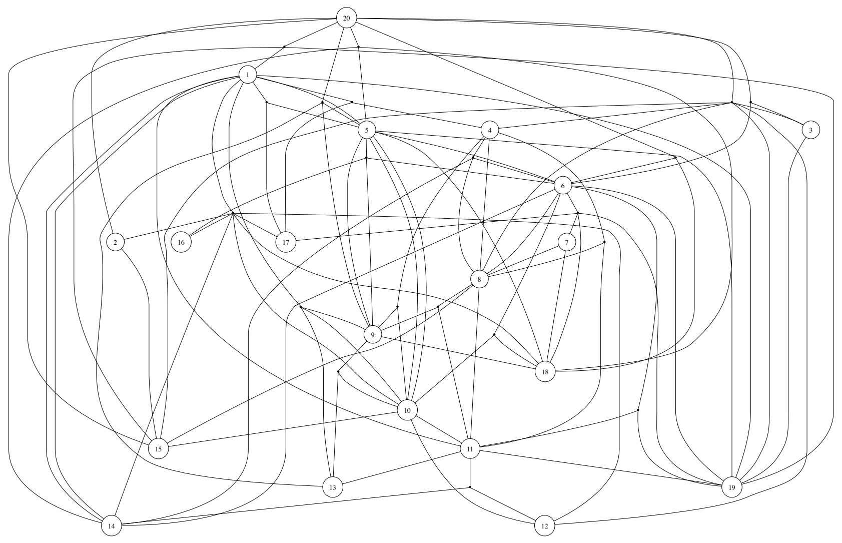


Figure 3.5: The graph representation of the example circuit

which is described in the following.

To avoid confusion, the initial partitions are referred to as partitions in multi-way partition phase, layers after linear ordering, initial tiers after layer merging, and tiers after refinement.

In these experiments, different numbers of initial partitions are assigned to each benchmark circuit targeting 3, 4, and 5 tier circuits.

The number of final net cuts and the run time for linear ordering step, for different numbers of initial partitions, are compared. Table 3.2 shows the results of the experiments targeting 3 tier circuit by assigning 3, 15, and 30 initial partitions. Table 3.3 shows the results of the experiments targeting 4 tier circuit by assigning 4, 16, and 32 initial partitions. And Table 3.4 shows the results of the experiments targeting 5 tier circuit by assigning 5, 15, and 30 initial partitions. Balance criterion is set to a minimum allowable imbalance (%1) in performing hMetis partitioning. The initial partitions are obtained by choosing the best solution of 50 runs of hMetis.

Table 3.1: ISPD 2004 Benchmarks

	Cells	I/Os	Nets
ibm01	12506	246	14111
ibm02	19342	259	19584
ibm03	22853	283	27401
ibm04	27220	287	31970
ibm05	28146	1201	2844 6
ibm06	32332	166	34826
ibm07	45639	287	48117
ibm08	51023	286	50513
ibm09	53110	285	60902
ibm10	68685	744	75196
ibm11	70152	406	81454
ibm12	70439	637	77240
ibm13	83709	490	99666
ibm14	147088	517	152772
ibm15	161187	383	186608
ibm16	182980	504	190048

The aim is to obtain the favorable initial partition number for different cases of tier (i.e., 3, 4 and 5). To do so, for each case, different initial partitions are selected. Then, net cut and run time are determined for each individual initial partition number. Tables 3.2, 3.3, and 3.4 show the effect of different initial partition number on net cut and run time targeting 3 tier IC, 4 tier IC, and 5 tier IC, respectively. Table 3.5 shows these effects on example circuit in Figure 3.5 and Figure 3.4. It was expected to witness a decrease in net cut and an increase in run time by an increase in the initial partition number. A favorable initial partition number is therefore the one at which the net cut reaches a sufficiently good value where the run time value is reasonably small. On the contrary, the results indicate that for up to a certain initial partition number, net cut decreases consistently after which point net cut increases. It is conjectured that this behavioral trend in the system stems from the shortcomings of the linear ordering heuristic. Considering this, for 3 tier IC, 4 tier IC, and 5 tier

IC, the best initial partition number is chosen as 15, 16, and 15, respectively.

Table 3.2: The effect of different initial partition number on net cut and run time targeting 3 tier IC

Cicuit	Net cut			Run time		
	Initial partition numbers			Initial partition numbers		
	3	15	30	3	15	30
ibm01	982	391	780	1	81	502
ibm02	1075	402	867	1	88	632
ibm03	2296	1976	2308	3	90	566
ibm04	1578	1298	1775	2	107	655
ibm05	4372	3581	4464	3	281	1749
ibm06	2407	1540	2059	2	118	707
ibm07	2762	1892	2433	2	167	1093
ibm08	2881	2557	3051	2	168	1131
ibm09	2114	1740	2378	1	105	650
ibm10	3363	2404	3869	3	235	1728
ibm11	3460	2026	2987	1	203	1220
ibm12	4015	3573	4889	3	407	2553
ibm13	2713	1553	2610	2	199	1435
ibm14	3773	3547	4876	3	344	3143
ibm15	6041	4622	6159	3	542	4237
ibm16	5809	3918	6022	3	634	5251
Average	3103	2314	3220	2	236	1703

The experiments show that increasing the number of initial partitions up to some point, starting from a number equal to the number of desired tiers, reduces the number of TSVs and improves the quality of solution; however, after some point the number of TSVs increases.

By increasing the number of initial partitions, more permutations are added without loosing any of pervious permutations. However, choosing a large value for initial partitions increases the run time of the algorithm. However, using a heuristic for linear ordering doesn't ensure the best solution. Moreover, using this approach, it is possible to obtain a worse solution in comparison with that obtained from a less number of initial partitions. By increasing the number of initial partitions, after a

Table 3.3: The effect of different initial partition number on the net cut and run time targeting 4 tier IC

Cicuit	Net cut			Run time		
	Initial partition numbers			Initial partition numbers		
	4	16	32	4	16	32
ibm01	1101	831	1367	2	95	607
ibm02	1494	1143	1642	4	103	659
ibm03	3278	3186	3731	4	101	633
ibm04	2334	2396	2950	4	115	702
ibm05	5426	4992	6528	9	304	1994
ibm06	2812	2756	3285	4	142	833
ibm07	3424	3118	3584	8	174	1302
ibm08	3977	3910	4557	7	193	1232
ibm09	2983	2638	3571	6	123	853
ibm10	4099	4668	5207	5	267	1932
ibm11	4051	3485	4530	8	241	1359
ibm12	6397	6623	6793	15	460	2743
ibm13	3725	3165	3573	3	271	1821
ibm14	5326	6366	7371	15	365	3274
ibm15	8884	9486	8794	31	636	5139
ibm16	7862	7076	8201	35	699	5812
Average	4198	3738	4730	10	268	1931

certain value, not only the run time of linear ordering increases, but also the quality of solution decreases.

3.4.3 Linear Ordering Results

In this section, the number of net cuts after merging considering different orders, i.e. hMetis order and linear ordering order, are compared. The objective here is to demonstrate how important the order of stacking partition is in reducing the number of net cuts. To achieve this goal, a certain linear ordering heuristic was used (see Section 3.3.2). A seed cell is selected as a random bottom layer, which could be any of the initial partitions obtained from the previous step of algorithm (i.e., initial partitioning phase). Because we have a limited number of partitions, each of them is

Table 3.4: The effect of different initial partition number on the net cut and run time targeting 5 tier IC

Cicuit	Net cut			Run time		
	Initial partition numbers			Initial partition numbers		
	3	15	30	3	15	30
ibm01	1618	1096	2120	2	88	540
ibm02	1962	1479	2439	3	91	637
ibm03	4146	3879	4854	4	92	578
ibm04	3427	2608	3696	3	109	662
ibm05	7894	7298	8372	11	284	1769
ibm06	4493	3256	4023	4	120	700
ibm07	4604	4134	5443	7	168	1108
ibm08	5576	5391	5893	7	170	1148
ibm09	3470	3538	3886	5	105	664
ibm10	6330	5183	7773	8	240	1743
ibm11	5231	4776	6053	8	201	1228
ibm12	8403	9919	9106	20	420	2583
ibm13	5245	4052	4783	5	205	1822
ibm14	7231	6906	8740	12	349	3212
ibm15	11602	11546	10699	28	601	4973
ibm16	9741	8146	12772	30	670	5445
Average	5686	5200	6291	10	245	1801

Table 3.5: The effect of different initial partition number on the net cut on example circuit in Figure 3.5 and Figure 3.4

Number of initial partitions	4	8	12
net cut number	45	37	41

assigned as the bottom layer or seed cell. Using this approach, the bottom layer that produces the smallest possible net cut is used as the seed cell in stacking the partitions (or ordering partitions). The maximum number of net cuts and average number of net cuts are referred to the maximum and average number of net cuts obtained by different orders with different partitions as their bottom layer, respectively. Tables 3.6, 3.7, and 3.8 compare the maximum and average number of net cuts exceeding the number of net cuts obtained by using the default order of initial partitioning phase in hMetis algorithm for 3 tier IC, 4 tier IC, and 5 tier IC, respectively. The last column in these tables is the decrement in the number of net cuts relative to the net cuts obtained by using the default order of initial partitioning phase in hMetis algorithm. Each decrement corresponds to a minimum value of net cut. The order that generates these decrements is selected as the one producing the smallest possible net cut. Thus, the proposed order is the one that generated the smallest number of net cuts. On average, the desired order generates 1038 net cuts less than that being generated using the default order in hMetis for 5 tier IC. Graphical representation of some of the data reported in these three tables is given in 3.6. The net cuts for 3 tier IC, 4 tier IC, and 5 tier IC, which were generated by the proposed order in this section, are reported in Table 3.9.

3.5 Summary

In this chapter, one of the major contributions of this thesis is presented where partitioning and floor planning techniques are combined to obtain better 3D partitions.

Table 3.6: Maximum and average number of net cuts exceeding the number of net cuts obtained by using the default order in hMetis algorithm for 3 tier IC

Circuit	hMetis order	Increment in net cuts		
		Max	Ave	Proposed order
ibm01	1073	702	-43	-682
ibm02	1499	874	-52	-1097
ibm03	2346	822	328	-370
ibm04	2641	215	-216	-1343
ibm05	4315	655	311	-734
ibm06	2815	95	-170	-1275
ibm07	3341	579	7	-1449
ibm08	3273	609	60	-716
ibm09	2118	922	406	-378
ibm10	3913	388	-106	-1509
ibm11	3974	797	-158	-1948
ibm12	6043	328	-1230	-2470
ibm13	2919	1036	143	-1366
ibm14	4854	1183	317	-1307
ibm15	6890	1212	-167	-2268
ibm16	5769	725	483	-1851
Average	3781	696	-5	-1117

Table 3.7: Maximum and average number of net cuts exceeding the number of net cuts obtained by using the default order in hMetis algorithm for 4 tier IC

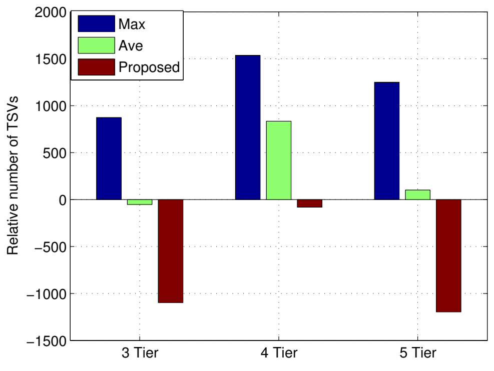
Circuit	hMetis order	Increment in net cuts		
		Max	Ave	Proposed order
ibm01	907	642	405	-76
ibm02	1224	1536	835	- 81
ibm03	3088	1232	592	98
ibm04	2392	969	508	4
ibm05	5463	1437	572	-471
ibm06	2812	1608	656	-56
ibm07	3300	2272	835	-182
ibm08	3893	2575	783	17
ibm09	2404	2000	1127	234
ibm10	3457	3939	1973	1211
ibm11	3824	2423	935	-339
ibm12	6855	3662	748	-232
ibm13	3111	1771	1058	54
ibm14	6704	1922	-146	-338
ibm15	8184	2546	1456	1302
ibm16	5871	5288	3706	1205
Average	3968	2239	984	147

Table 3.8: Maximum and average number of net cuts exceeding the number of net cuts obtained by using the default order in hMetis algorithm for 5 tier IC

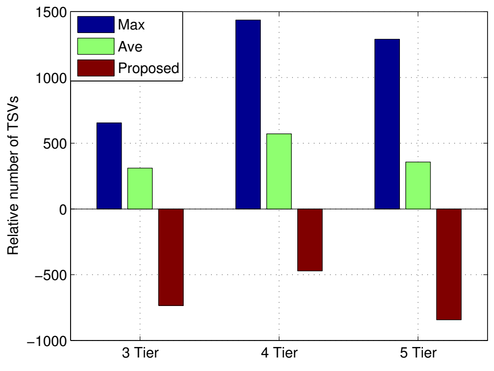
Circuit	hMetis order	Increment in net cuts		
		Max	Ave	Proposed order
ibm01	1847	740	215	-751
ibm02	2675	1250	102	-1196
ibm03	4398	1523	435	-519
ibm04	4465	878	188	-1857
ibm05	8141	1291	357	-843
ibm06	4521	538	268	-1265
ibm07	5536	1967	204	-1402
ibm08	5176	2274	1080	215
ibm09	4039	1243	48	-501
ibm10	6547	2310	584	-1364
ibm11	6530	2032	-305	-1754
ibm12	10964	1634	-416	-1045
ibm13	5121	4001	1269	-1069
ibm14	8232	1492	393	-1326
ibm15	11785	2677	1149	-239
ibm16	9836	3577	2547	-1690
Average	6238	1839	445	-1038

Table 3.9: Best net cut number achieved

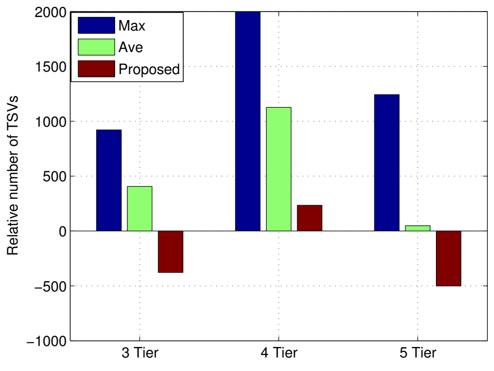
Circuit	Number of TSVs		
	3Tier	4 Tier	5 Tier
ibm01	391	831	1096
ibm02	402	1143	1479
ibm03	1976	3186	3879
ibm04	1298	2396	2608
ibm05	3581	4992	7298
ibm06	1540	2756	3256
ibm07	1892	3118	4134
ibm08	2557	3910	5391
ibm09	1740	2638	3538
ibm10	2404	4668	5183
ibm11	2026	3485	4776
ibm12	3573	6623	9919
ibm13	1553	3165	4052
ibm14	3547	6366	6906
ibm15	4622	9486	11546
ibm16	3918	7076	8146
Average	2314	4114	5200



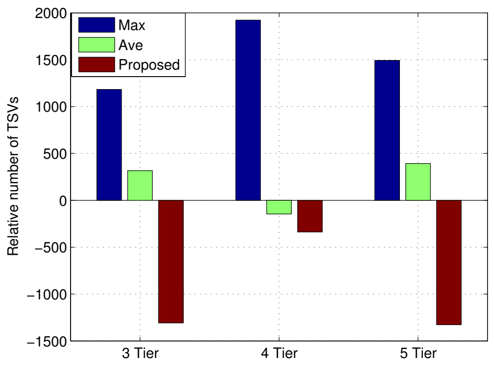
(a) ibm02



(b) ibm05



(c) ibm09



(d) ibm14

Figure 3.6: Graphical representation of maximum and average increments/decrements in number of net cuts (the blue and the green bars respectively) and the increments/decrements in number of net cuts obtained by using the proposed order (the red bars) over the number of net cuts obtained by using the default order given by hMetis algorithm

Experimental results presented in the chapter show that the proposed technique reduces the number of TSVs effectively.

Other contributions of the chapter include studying different partitioning techniques to obtain initial 2D partitions, determining the best trade-off between number of initial partitions and run time, development of a force-directed one dimensional placement problem for finding best locations for partitions and partition merging suited for the final simulated annealing refinement phase which will be described in Chapter 4.

Chapter 4

Force Based Simulated Annealing

4.1 Introduction

In this chapter, a force-based iterative improvement technique is introduced so as to further improve the partitioning solution.

The inspiration of the technique presented in this work comes from the annealing process and the analogy between the molecular forces in a material and the forces between the cells of a circuit.

First major contribution of this chapter is proposing a Force-based Simulated Annealing (FSA) technique, which can be used in different optimization problems. This approach aims to better imitate the annealing process and modify the already existing SA method, by including molecular forces as part of the annealing process.

Another major contribution of this chapter is proposing a new improvement method for 3D IC partitioning. The proposed method, combines the introduced force-based simulated annealing technique with placement techniques to perform 3D IC partitioning.

The general idea of the algorithm is to put the cells in a vertical line, calculate the forces on each cell, and to move the cells according to the forces imposed on them before they are assigned to tiers. In the proposed algorithm, the location of the cells are proposed to be analogous to the state of atoms and the number of net cuts is proposed to be analogous to the internal energy of the system. Forces between the cells that try to expand or contract a net is analogous to the forces between the molecules. The proposed method, introduces a new selection method that is incorporated into the SA. In this new selection method, the probability of selecting a cell to be moved is based on the forces on the cell rather than being random. A method of selection based on force rather than being random is more promising in finding an appropriate move although this method doesn't always result in an appropriate move. Therefore, the algorithm has the advantage of both greedy algorithms, to converge quickly and of metaheuristic methods, to escape from local optima.

The result of the proposed algorithm is to locate the connected cells close to each other and increasing the probability of their assignment to the same tier or adjacent tiers.

The rest of this chapter is organized as follows: In Section 4.2, the essence of FSA is explained in details. The numerical results are then presented in Section 4.3. Finally, in Section 4.4, a brief summary is provided

4.2 Force Based Simulated Annealing

A customized simulated annealing is developed to move the cells. In this algorithm, the probability of a cell to be selected is proportional to the force exerted on that cell. The selected cell is moved in the direction of its force.

The number of TSVs is further reduced by improvements where cells are moved between different layers based on the forces applied onto them. A FSA technique is

Algorithm 5 Force Based Simulated Annealing

Input: Initial Solution s_{init}

Output: Refined Solution

```
1:  Begin
2:  $T \leftarrow T_0$
3:  $i \leftarrow 0$
4:  $s \leftarrow s_{init}$
5:  $f_{current} \leftarrow Force(s)$
6:  $cost_{current} \leftarrow Cost(s)$
7:  while $T > T_{min}$ do
8:      while stopping criteria do
9:          $i \leftarrow i + 1$
10:         $s' \leftarrow$ neighbor($s$)
11:         $cost_{new} \leftarrow Cost(s')$
12:         $f_{new} \leftarrow Force(s')$
13:         $\Delta cost = cost_{new} - cost_{current}$
14:         if $\Delta cost < 0$ then
15:             $s \leftarrow s'$
16:             $cost_{current} \leftarrow cost_{new}$
17:             $f_{current} \leftarrow f_{new}$
18:         else
19:             $r \leftarrow Random(0, 1)$
20:             if $r < e^{-\Delta cost/T}$ then
21:                 $s \leftarrow s'$
22:                 $cost_{current} \leftarrow cost_{new}$
23:                 $f_{current} \leftarrow f_{new}$
24:             end if
25:         end if
26:     end while
27:     $T \leftarrow \alpha.T$
28: end while
29: End.
```

developed and implemented in the initial tiers. In this phase, the algorithm allows the cells to move through layers according to the forces imposed by the connected cells. A greater force on a cell is synonymous with a greater probability of selection for that cell. On one hand, considering the forces in selecting the cells rather than random selection of a cell triggers more promising moves. On the other hand, combining the force-based selection with SA adds randomness to the algorithm and prevents the algorithm from being trapped in local optima.

The FSA algorithm is given in Algorithm 5. The force calculation, neighbor function and cost function are described in the following subsections.

4.2.1 Force Calculation

Adopted from force-based placement approach, the circuit is modeled as a mass-spring system, where cells and nets are analogous to the masses and springs, respectively. An expanded spring imposes force to the connected masses and attracts them. Hence, the amount of force applied by a cell to another is directly proportional to their distance from one another. That is to say, the long connections impose a strong force on the connected cells.

In developing this algorithm, different approaches are considered in calculating the forces. The first approach was to perform a one dimension placement by solving quadratic optimization and to assign an initial location to cells. In order to do this, cells belonging to the top and bottom of the ordered layers from previous phase are fixed. These cells serve as a starting point of quadratic optimization (similar to I/O pads in analytical placement). In a general analytical placement, there are some cells with preassigned locations (usually on the boundary) along with the rest of the cells placed according to their connection with the preassigned cells and other cells. If there are no fixed cells in analytical placement, all the cells would be placed in the middle. Moreover, some constraints are considered in the quadratic optimization. The constraints state that the cells belonging to an initial tier would remain in the same tier. The reason for adding this constraint is that moving cells between the layers affect the number of net cuts. This approach locates most of the cells in tier boundaries, resulting in loosing the information about initial layers. In Figure 4.1, the location of the cells on the z-axes and the amount of force exerted on them after quadratic optimization is given for **ibm03** benchmark circuit.

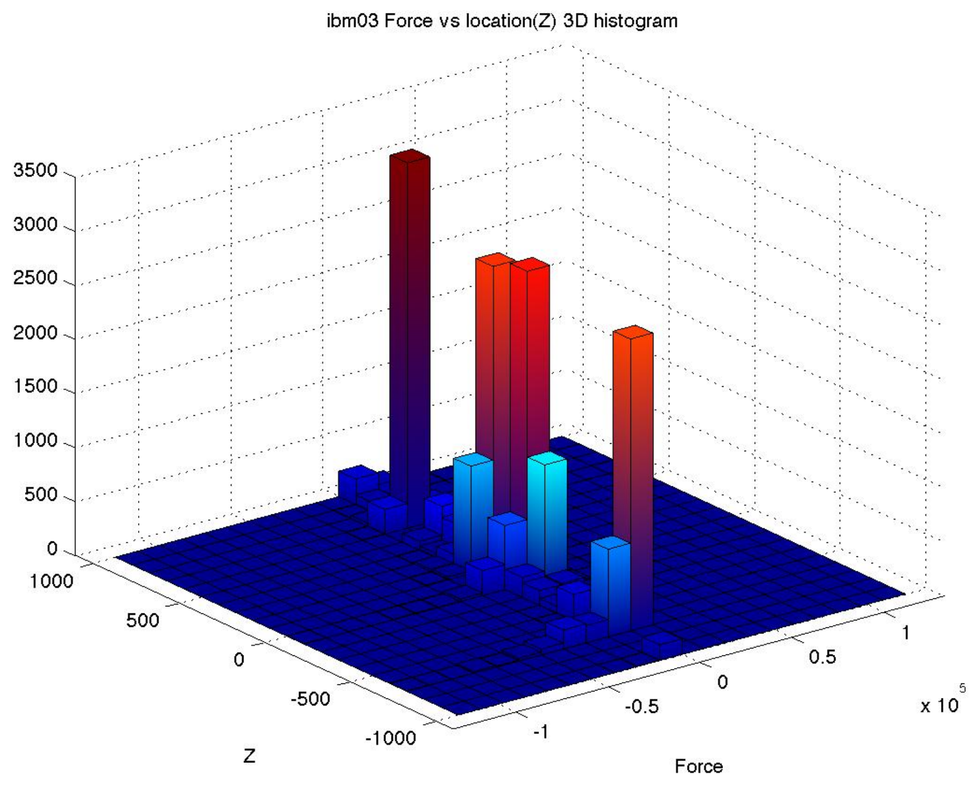


Figure 4.1: Location of the cells on the z-axes and the amount of force exerted on them after quadratic optimization

Therefore, cells belonging to one layer are kept in their original location so that they don't impose force to one another; and it is only the connection between the cells belonging to different partitions that generates forces. By setting the same coordinate on the z-axes to cells from the same layer, the vertical component of their imposed force to one another becomes zero. The forces are calculated using the following equation:

$$F_{ab} = c_{a,b}(z_b - z_a)$$

where $c_{a,b}$ is the connection weight between cell a and cell b and z_b and z_a are the locations of the cell b and a on the z-axes, respectively. The connection weight between cells, $c_{a,b}$, is equal to the number of connections between the cells.

This equation is inspired from the hook's law. The force measure is the Euclidean distance between two cells. The Euclidean distance between two points (x_i, y_i) and (x_j, y_j) is given in following:

$$L = \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2} \tag{4.1}$$

Since in 3D partitioning the partitions are placed in one dimension and the cells move only in that direction (Z direction) the equation can be reduced to

$$L = z_i - z_j,$$

The force imposed on a cell is equal to the sum of the force imposed by all other cells.

$$F_i = \sum_j F_{i.j}$$

4.2.2 Neighbor Function

The neighbor function is a function that generates a new solution by changing the value of one or more variables. In 3D partitioning problem, a new solution is proposed to be generated by moving one cell from one tier to an adjacent tier.

In the original SA, the neighbor function selects a random cell and moves it to a random adjacent tier. However, in the proposed FSA, the probability of the selection of a cell is proportional to the magnitude of the force exerted on that cell and it is moved in the direction imposed by its force.

The FSA neighbour function consists of two steps:

1. Selecting the cell to be moved:

   The cell to be moved is selected randomly with different probabilities for each cell, based on the forces exerted on them. That means If F_i is the force on an individual cell, its probability of being selected is

   $$P_i = F_i \sum_i F_i$$

2. Assigning the new location of selected cell: The selected cell is moved to the adjacent tier in the direction that its force impose.

4.2.3 Cost Function

The cost of a solution is equal to the number of 3D net cuts, i.e., the number of TSV usage. An FM-based gain update method is used to update the gains.

4.2.4 Selecting SA parameters

In order to apply the SA and FSA method to the 3D partitioning problem, the algorithm parameters must be specified. Tuning these parameters can have a significant impact on the effectiveness of the method. Unfortunately, there is not a general way to find the best values for a given problem. In order to tune these parameters different values are assigned to each parameter and the values that result in best solution are chosen. The parameters and the values assigned to them, as well as the stopping criteria of the algorithm, are given in the following.

Stopping criteria

Two stopping criteria is considered in this implementation:

1. The algorithm stops when the number of iterations exceeds this maximum number of iterations.
2. The algorithm runs until the temperature reaches to its minimum value (T_{min}).

Initial temperature (T_0)

The initial temperature T_0 is set in a way that satisfies the following equation:

$$0.9 = e^{-Ave(cost)/T_0}$$

where $Ave(cost)$ is the average cost of moving the cells from their initial position. By doing this we allow the average probability of 0.9 for selecting a given cell in the first iteration.

Maximum number of iterations

The maximum number of iterations is set to be 5000.

Minimum temperature (T_{min})

The minimum temperature is set to be 0.01.

Annealing schedule (α)

The annealing schedule or the rate of decreasing the temperature is set to be 1/2.

$$\alpha = 1/2$$

Number of iterations for each temperature ($Iter_T$)

The number of iterations for each temperature is set in a way that satisfies the following equation:

$$Iter_T = 100/T$$

4.3 Experimental Results

The improvements are applied to the initial solutions obtained in the previous phase of the algorithm (i.e., constructive 3D partitioning) given in Section 3. The initial solutions are the partitioning solutions of the **ISPD04** benchmark suits, which were introduced in Section 3.

SA demonstrates a better performance in terms of the number of TSVs in comparison to FSA. On the other hand, FSA outperforms SA in terms of convergence speed. Therefore, a Modified Force based Simulated Annealing (MFSA) is developed in a way that it possesses both advantages. In summary, MFSA outperforms SA and

FSA in terms of the number of TSVs and it quickly converges to the semi-optimal solution.

Since force calculation is added in each iteration of FSA and the first 500 iterations of MFSA, SA was expected to have a better run time for the same number of iterations; however, the experiments show that this is not always the case and MFSA and FSA demonstrate a better performance in terms of run time in some cases. This lies in that the FSA and MFSA candidate cell selection is based on the cell forces whereas this selection is done randomly in SA. Therefore, the probability of accepting a candidate cell in SA is lower than that in the FSA and MFSA algorithms. In each iteration, the number of unsuccessful selections in SA is thus more than the number of unsuccessful selections in FSA. Consequently, the run time of a given iteration is higher in SA.

The results of SA, FSA and MFSA improvements are given in Table 4.1. The number of iterations to achieve the final solution within 2% accuracy are given .

Table 4.1 and 4.2 tabulates the effect of force based refinement for 4 tier IC. It summarizes the resulting number of TSVs (for different approaches including: (1) simulated annealing- SA, (2) force-based simulated annealing- FSA, and (3) modified force-based simulated annealing- MFSA), and the values of 2% iteration, 2% run time, and the total run time. The value of 2% iteration number represents the iteration number at which the number of TSVs reaches within 2% difference of the ultimate number of TSVs at which there will be no significant change in the number of TSVs and the corresponding curve levels off. The value of 2% run time represents the run time when the number of TSVs reaches within 2% difference of the ultimate number

Table 4.1: Effect of force based refinement 4 tier using SA, FSA, MFSA on net cut

Circuit	Number of TSVs				2%iteration		
	initial	SA	FSA	MFSA	SA	FSA	MFSA
ibm01	831	824	828	824	3120	1412	1553
ibm02	1143	1103	1106	1098	3484	1766	1180
ibm03	3186	3110	3135	3108	2224	992	1112
ibm04	2396	2307	2330	2308	3229	2225	1764
ibm05	4992	4809	4827	4780	3363	2001	2179
ibm06	2756	2715	2716	2704	2773	655	1015
ibm07	3118	3078	3085	3071	3131	501	673
ibm08	3910	3888	3882	3889	2961	573	744
ibm09	2638	2608	2603	2602	2146	516	1040
ibm10	4668	4667	4636	4610	4004	520	522
ibm11	3485	3409	3408	3392	3334	905	1344
ibm12	6623	6430	6370	6359	4185	1495	2058
ibm13	3165	3136	3123	3109	3011	501	506
ibm14	6366	6336	6271	6278	3808	539	930
ibm15	9486	9390	9341	9309	3830	512	873
ibm16	7076	7025	7027	6999	3185	501	616

Table 4.2: Effect of force based refinement 4 tier using SA, FSA, MFSA on run time

Circuit	2%run time			total run time		
	SA	FSA	MFSA	SA	FSA	MFSA
ibm01	132	97	71	211	343	230
ibm02	185	162	65	266	459	277
ibm03	71	36	34	160	180	155
ibm04	147	93	77	228	210	218
ibm05	120	92	74	178	229	169
ibm06	108	23	35	194	179	171
ibm07	191	24	34	305	237	254
ibm08	439	93	78	742	808	526
ibm09	234	34	95	546	331	457
ibm10	297	41	33	371	392	316
ibm11	424	73	139	636	401	517
ibm12	420	141	191	502	470	464
ibm13	580	74	81	963	737	798
ibm14	1411	114	263	1853	1054	1414
ibm15	1940	175	309	2532	1709	1772
ibm16	2022	152	271	3174	1514	2199

Table 4.3: Effect of force based refinement 4 tier

Circuit	Number of TSVs							
	Initial	Final	I/O Pin methods	imp. %	Unlocked Pins	imp. %	Alt. Pins	imp. %
ibm01	831	824	837	2%	838	2%	977	19%
ibm02	1143	1098	1156	5%	1214	11%	1340	22%
ibm03	3186	3108	2610	-16%	2693	-13%	3602	16%
ibm04	2396	2308	2371	3%	2516	9%	2461	7%
ibm05	4992	4780	6489	36%	6653	39%	7037	47%
ibm06	2756	2704	2934	9%	3128	16%	3429	27%
ibm07	3118	3071	3219	5%	3302	8%	3482	13%
ibm08	3910	3889	4018	3%	4184	8%	4183	8%
ibm09	2638	2602	2495	-4%	2763	6%	3757	44%
ibm10	4668	4610	4004	-13%	4675	1%	4358	-5%
ibm11	3485	3392	3685	9%	3958	17%	4923	45%
ibm12	6623	6359	6581	3%	7259	14%	8996	41%
ibm13	3165	3109	3099	0%	3264	5%	4618	49%
ibm14	6366	6278	5342	-15%	6584	5%	7564	20%
ibm15	9486	9309	7022	-25%	9082	-2%	11144	20%
ibm16	7076	6999	5774	-18%	6235	-11%	9525	36%
Avg	3738	4027	3852	-1%	4271	7%	5081	26%

Table 4.4: Effect of force based refinement 3 tier

Circuit	Number of TSVs							
	Initial	Final	I/O Pin methods	imp. %	Unlocked Pins	imp. %	Alt. Pins	imp. %
ibm01	391	379	525	39%	857	126%	881	132%
ibm02	402	392	747	91%	882	125%	829	111%
ibm03	1976	1934	2174	12%	2282	18%	2530	31%
ibm04	1298	1279	1511	18%	1583	24%	1619	27%
ibm05	3581	3566	4311	21%	5372	51%	5428	52%
ibm06	1540	1511	1642	9%	1827	21%	1729	14%
ibm07	1892	1824	2050	12%	3442	89%	3423	88%
ibm08	2557	2460	2697	10%	2814	14%	3431	39%
ibm09	1740	1695	1872	10%	2828	67%	2186	29%
ibm10	2404	2366	2661	12%	3565	51%	4062	72%
ibm11	2026	1988	2240	13%	3477	75%	3629	83%
ibm12	3573	3561	4094	15%	5350	50%	5569	56%
ibm13	1553	1543	1893	23%	3037	97%	2912	89%
ibm14	3547	3512	3886	11%	4561	30%	5090	45%
ibm15	4622	4609	4827	5%	7863	71%	7970	73%
ibm16	3918	3896	4316	11%	5816	49%	6216	60%
Average	2314	2282	2590	20%	3472	60%	3594	63%

Table 4.5: Effect of force based refinement 5 tier

Circuit	Number of TSVs							
	Initial	Final	I/O Pin methods	imp. %	Unlocked Pins	imp. %	Alt. Pins	imp. %
ibm01	1096	1081	1162	7%	1439	33%	1372	27%
ibm02	1479	1463	1533	5%	1600	9%	1691	16%
ibm03	3879	3872	3974	3%	4020	4%	4366	13%
ibm04	2608	2586	2852	10%	3202	24%	4275	65%
ibm05	7298	7289	9193	26%	9651	32%	12400	70%
ibm06	3256	3243	3477	7%	3566	10%	3507	8%
ibm07	4134	4116	4400	7%	4605	12%	6523	58%
ibm08	5391	5341	5346	0%	5698	7%	6327	18%
ibm09	3538	3487	3343	-4%	3518	1%	3556	2%
ibm10	5183	5158	5216	1%	7116	38%	8492	65%
ibm11	4776	4673	4620	-1%	5697	22%	7437	59%
ibm12	9919	9756	8191	-16%	9158	-6%	12515	28%
ibm13	4052	3813	3742	-2%	4557	20%	4874	28%
ibm14	6906	6849	6667	-3%	8085	18%	10113	48%
ibm15	11546	10931	9283	-15%	11707	7%	13857	27%
ibm16	8146	7761	7172	-8%	9300	20%	10903	40%
Average	5200	5089	5011	1 %	5807	16%	7013	36%

Table 4.6: Tier area 3 Tier

Circuit	Tier area		
	3 Tier	4 Tier	5 Tier
ibm02	954376	714790	577788
ibm03	1170105	897060	695968
ibm04	1479500	1110800	885137
ibm05	1512452	1143501	907927
ibm06	1244346	947810	760484
ibm07	2172318	1737200	1311549
ibm08	2225728	1789500	1368832
ibm09	2695445	2053800	1589169
ibm10	4273202	3375500	2580247
ibm11	3459745	2618500	2031029
ibm12	4646314	3550400	2827221
ibm13	4024270	3042500	2465918
ibm14	7305739	5680900	4445652
ibm15	7405975	5550100	4456779
ibm16	9375372	7099400	5662680
Average	3596326	2754117	2171092

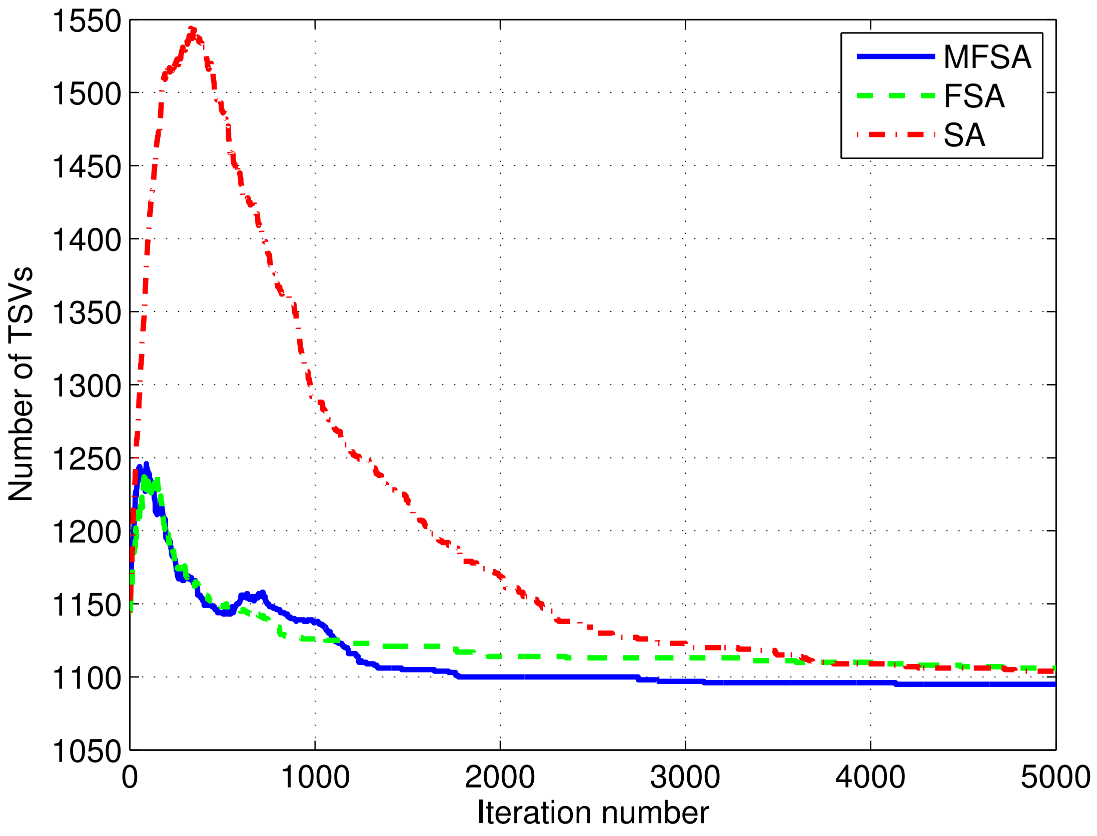


Figure 4.2: Number of TSVs versus iteration number in SA, FSA, and MFSA for IBM02

of TSVs where the corresponding curve tends to levels off. The first column of results in this table were generated using the previously-proposed approach SA (see Chapter 2) whose computational code was adopted and developed in this study. FSA has been proposed in this chapter in order to better imitate the annealing process. In addition to that, FSA is far more advantageous over SA in the sense that it converges more quickly (see the corresponding figures in Appendix). The computational code for this approach was developed in MATLAB. As shown in Table 4.1 and 4.2 , the ultimate answer in SA is smaller while the computation process in FSA converges more quickly. In order to incorporate both of these two features in a single computational approach, MFSA was developed in this study. Thus, MFSA not only produces a more desirable ultimate answer but it also converges far more quickly than both SA and FSA.

Figure 4.2 shows the number of TSVs versus iteration number in SA, FSA, and MFSA. Beyond just 1000 iterations, MFSA generates the best number of TSVs in comparison to SA and FSA, meaning that it has a faster convergence speed. The results also show that SA generates exceedingly high number of TSV during the early iterations (below 500). This is attributed to the random selection of cells that are being moved during the computation. After the early iterations, SA performance starts to revamp. In spite of this self-correction act during SA, the iteration number in SA at which the number of TSVs reaches within 2% of the ultimate answer is still significantly larger than that in FSA and MFSA (see Appendix A).

Table 4.3 shows a comparison of number of TSVs for 4 tier IC, which were obtained from Chapter 3 with the final number of TSVs obtained after MFSA improvements are implemented. Moreover , the number of TSVs, which were obtained from Chapter 3 are compared with the number of TSVs obtained using other approaches including: (1) I/O pins, (2) unlocked pins, and (3) alternate pins. The percentage of improvements made through using MFSA relative to each of the above-mentioned three approaches is also tabulated. Tables 4.4 and 4.5, show a comparison of number of TSVs for 3 and 5 tier IC, respectively, which were obtained from Chapter 3 with the final number of TSVs obtained after MFSA improvements are implemented. The percentage of improvements made through using MFSA relative to the results from I/O pins, unlocked pins, and alternate pins for 3 and 5 tier IC are also given in Tables 4.4 and 4.5, respectively. The results overwhelmingly indicate that the proposed constructive 3D partitioning algorithm followed by MFSA improvements is capable of significantly decreasing the number of TSVs. And in Table 4.6, the tier area of each benchmark is given for 3 tier, 4 tier and 5 tier ICs.

4.4 Summary

In this chapter, a new computational approach (i.e., MFSA) was developed in order to simultaneously capture and incorporate two attractive features of SA and FSA: SA generate the best ultimate answer while FSA converges more quickly to the ultimate desired answer. MFSA, as the newly developed approach in this chapter, generates the desired number of TSVs in a far shorter run time. To sum up, much smaller number of iterations are performed in MFSA in reaching the desired number of TSVs.

Chapter 5

Conclusion and Future Work

This thesis presented a new method for 3D IC partitioning. The proposed method is based on partitioning the circuit into several partitions using traditional partitioning methods and placing and merging the partitions to form tiers. Then a force directed based simulated annealing is introduced and used to improve the partitions and further reduce the net cuts.

5.1 Contributions

The major contributions of this thesis were:

Development of the a force directed partitioning technique:

In this thesis the state of the art technique for 2D circuit partitioning which minimizes the number of connections between all partitions is used to divide the circuit into several initial partitions. However in a 3D IC, objective function is different from that of 2D IC and the goal is to minimize the number of vias between partitions which are stacked on one dimension. Therefore, a one dimensional placement is used to find a suitable ordering for the partitions. Ordering the initial partitions results in locating the partitions that have more connections close to each other and consequently reduce the length of connections and number of long connections.

Development of force-directed simulated annealing technique:

An iterative improvement technique is developed and used to improve the partitioning solution after the initial 3D partitioning solution is obtained. The developed algorithm is a variation of SA where random moves are replaced with moves that are directed by forces applied to cells. This method is combined with other methods in solving the particular 3D IC partitioning problem; However, it can be used in many other optimization problems. Unlike the original SA in which the current solution is replaced by a random neighbor solution, in developed Force-based Simulated Annealing (FSA), the selection of the new solution is probabilistic, i.e., not only the acceptance of a move, but also selecting the new solution is based on a probability. This probability of selection is based on some information that we have about the system. In 3D partitioning problem, this information is the forces that the connected cells impose to each other. Finally, numerical results on benchmarks released by IBM show that the proposed techniques outperform the existing 3D partitioning solutions for most cases.

5.2 Future Work

In this work it was shown that using several techniques for solving complicated problem can work better than just using a single heuristic. The same framework of thought can be used in solving other and more complicated problems.

The future work in 3D IC partitioning includes, but is not limited to, using better linear ordering techniques, and applying clustering techniques such as AMG to obtain better partitioning solutions. In addition, the modified force directed simulated annealing can be applied to placement and floor planning problems.

Appendix A

Supplementary Results

The number of TSVs versus iteration number in SA, FSA, and MFSA for different benchmarks is shown in Figures A.1 to A.14.

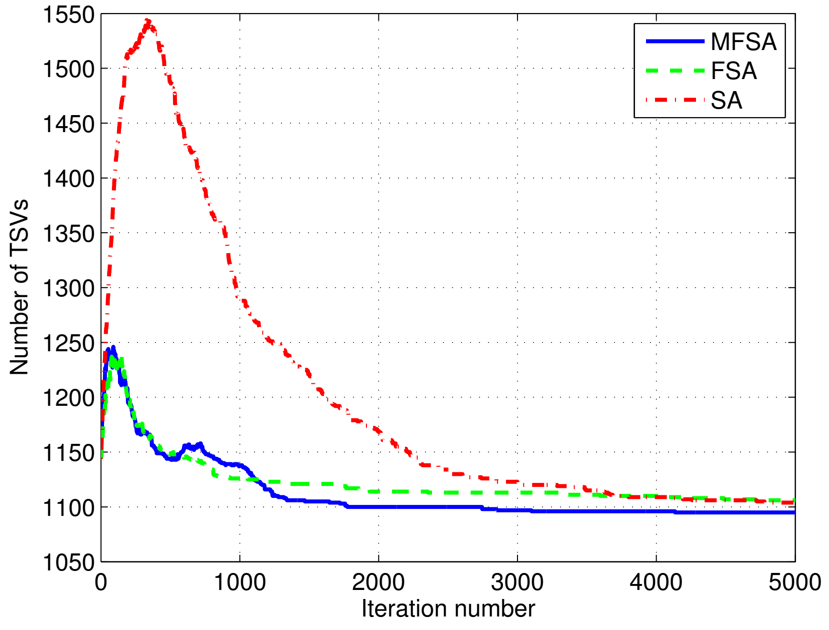


Figure A.1: IBM02 SA, FSA, and MFSA convergence

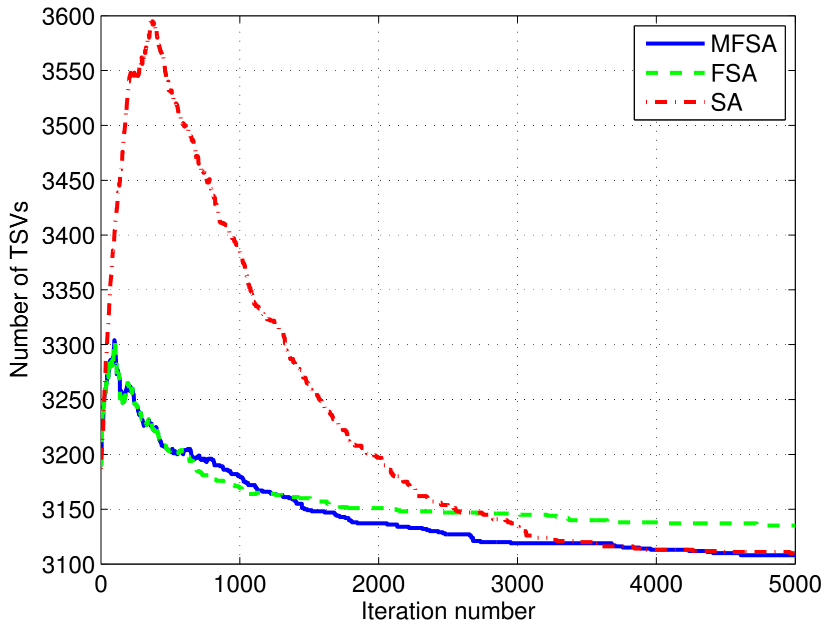


Figure A.2: IBM03 SA, FSA, and MFSA convergence

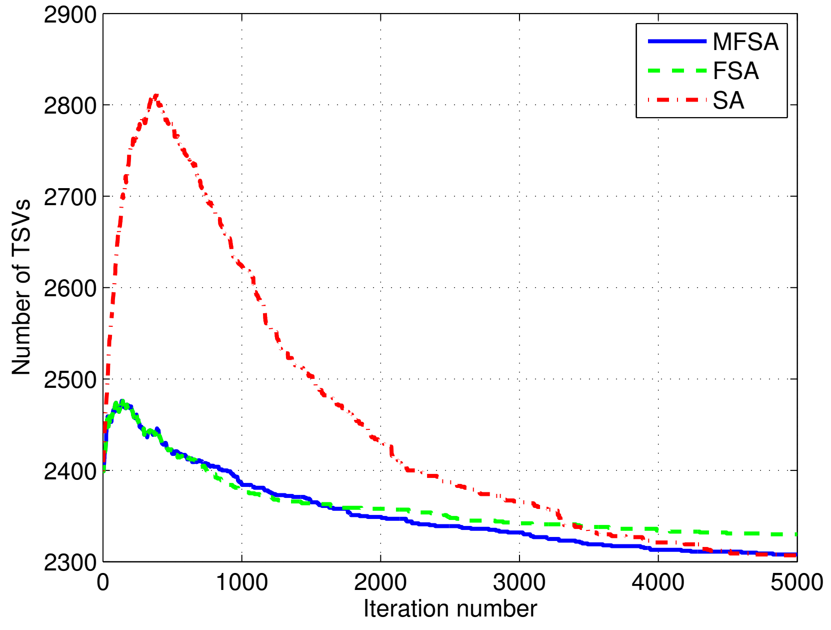


Figure A.3: IBM04 SA, FSA, and MFSA convergence

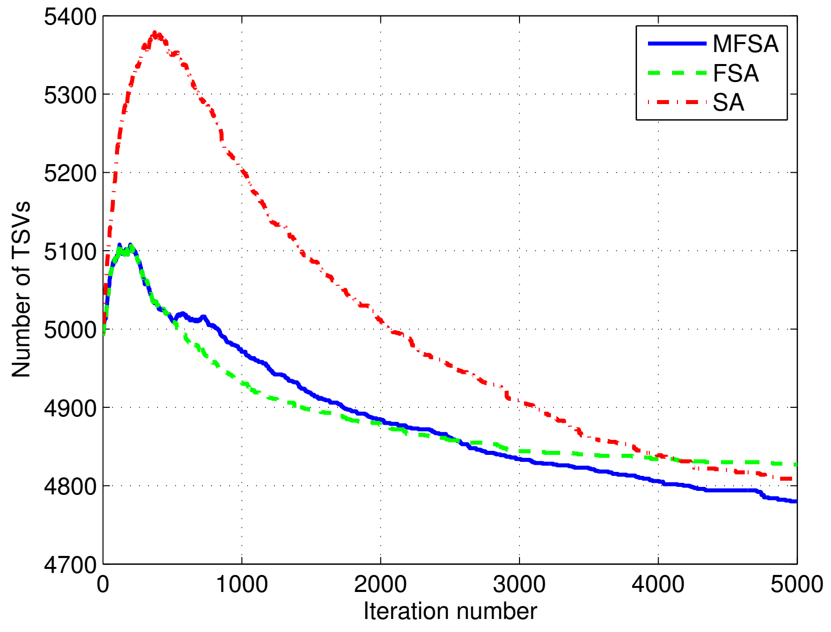


Figure A.4: IBM05 SA, FSA, and MFSA convergence

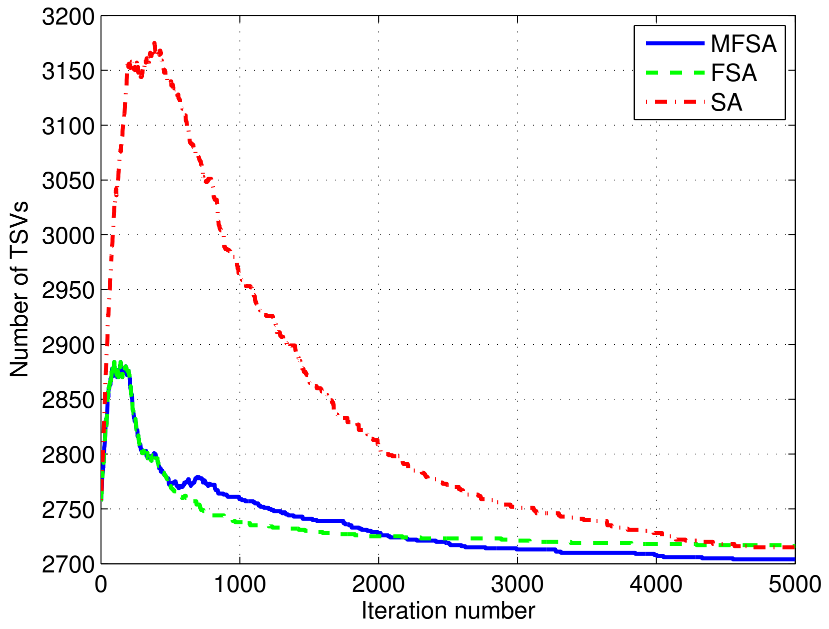


Figure A.5: IBM06 SA, FSA, and MFSA convergence

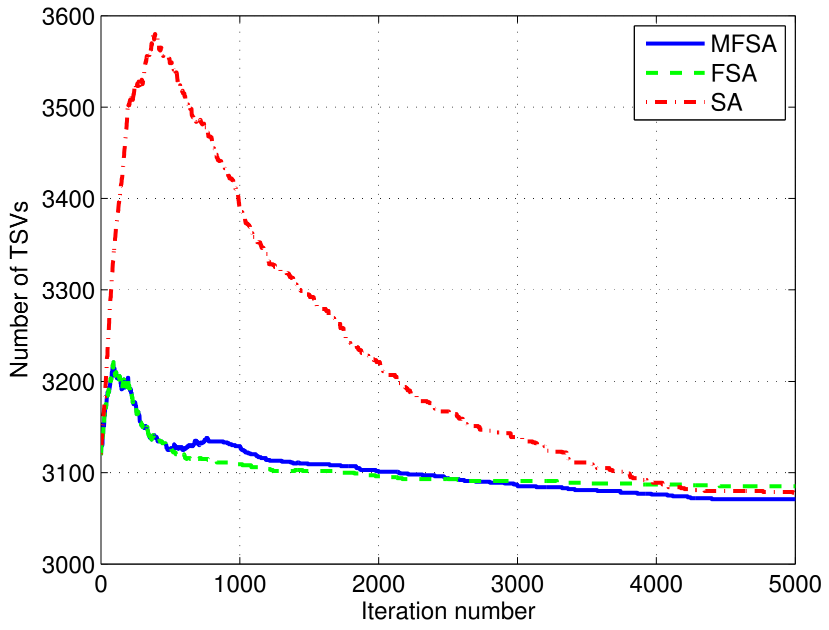


Figure A.6: IBM07 SA, FSA, and MFSA convergence

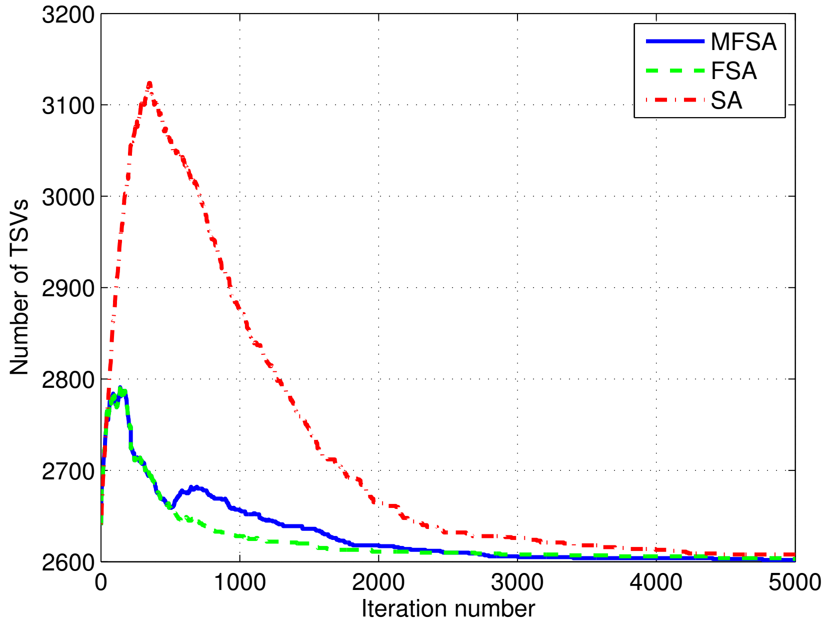


Figure A.7: IBM09 SA, FSA, and MFSA convergence

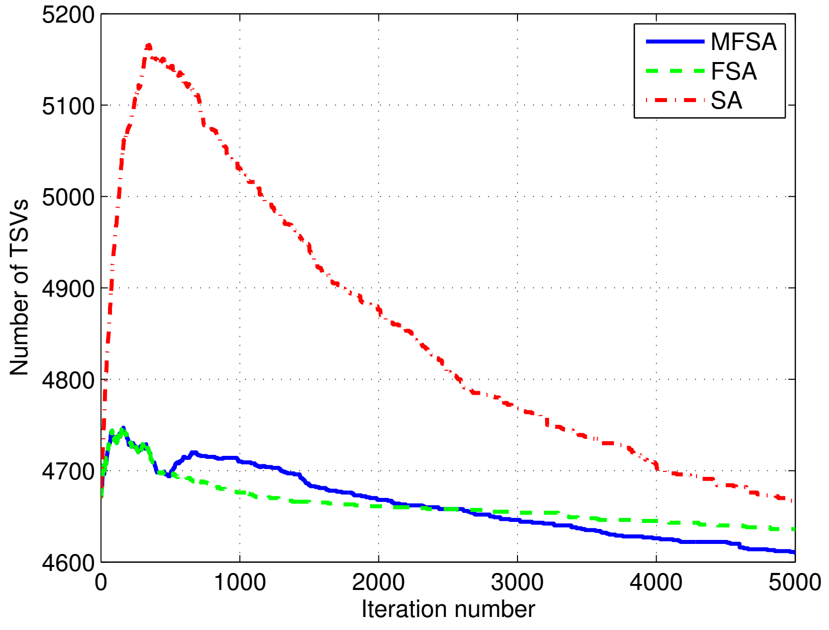


Figure A.8: IBM10 SA, FSA, and MFSA convergence

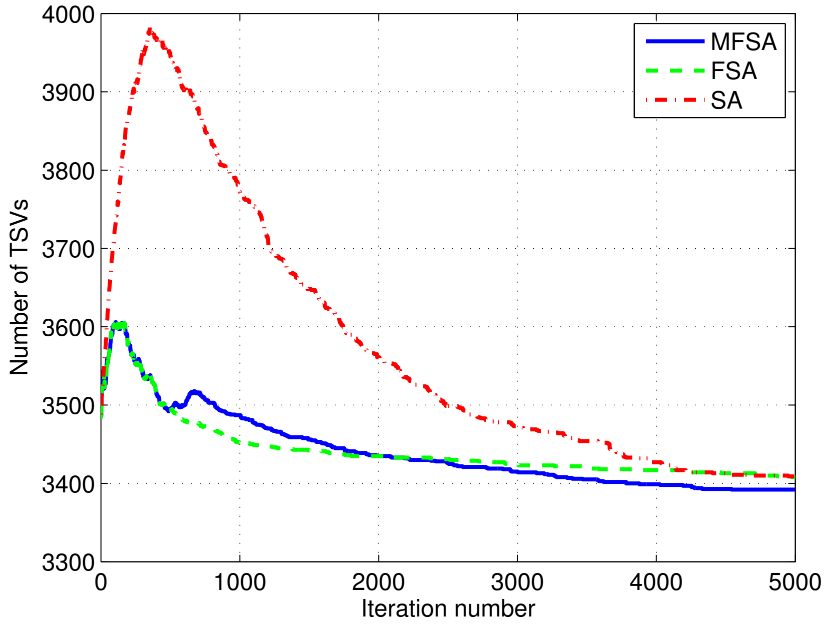


Figure A.9: IBM11 SA, FSA, and MFSA convergence

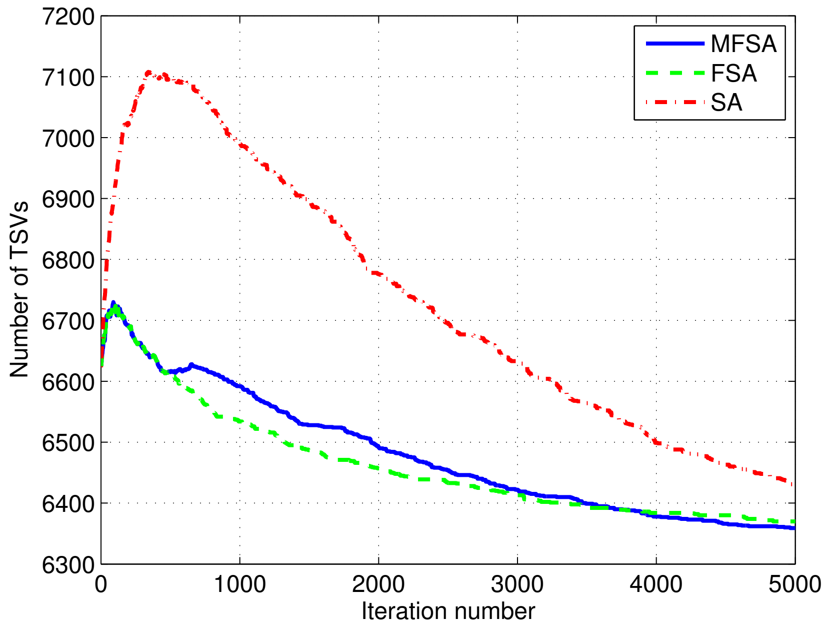


Figure A.10: IBM12 SA, FSA, and MFSA convergence

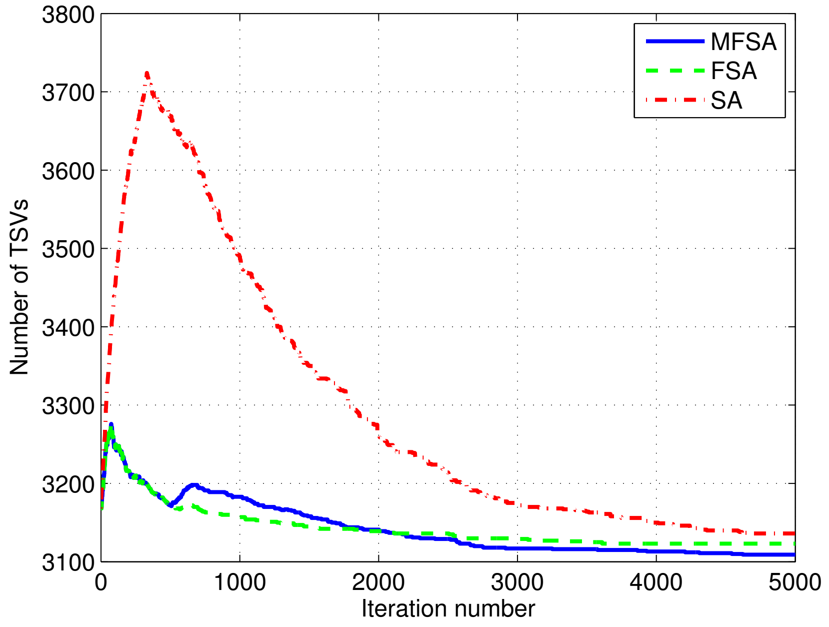


Figure A.11: IBM13 SA, FSA, and MFSA convergence

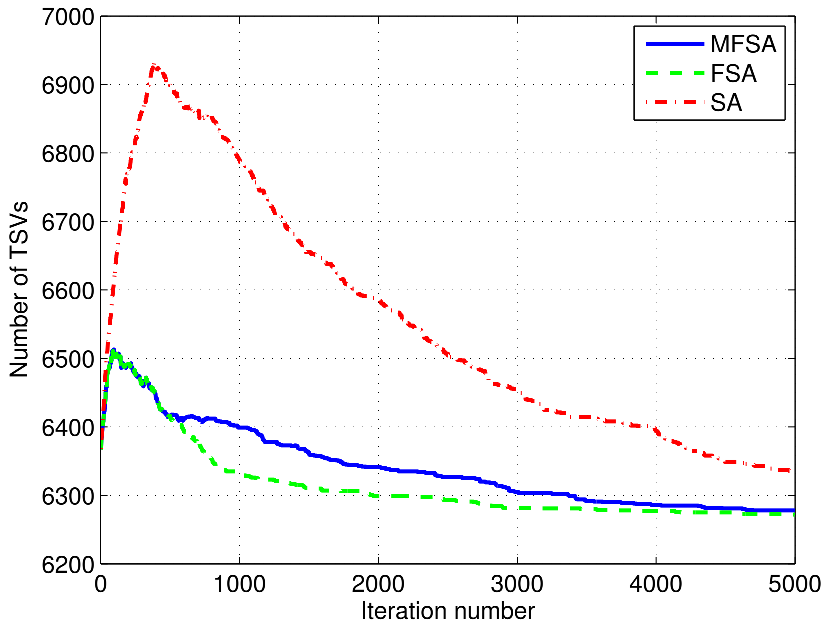


Figure A.12: IBM14 SA, FSA, and MFSA convergence

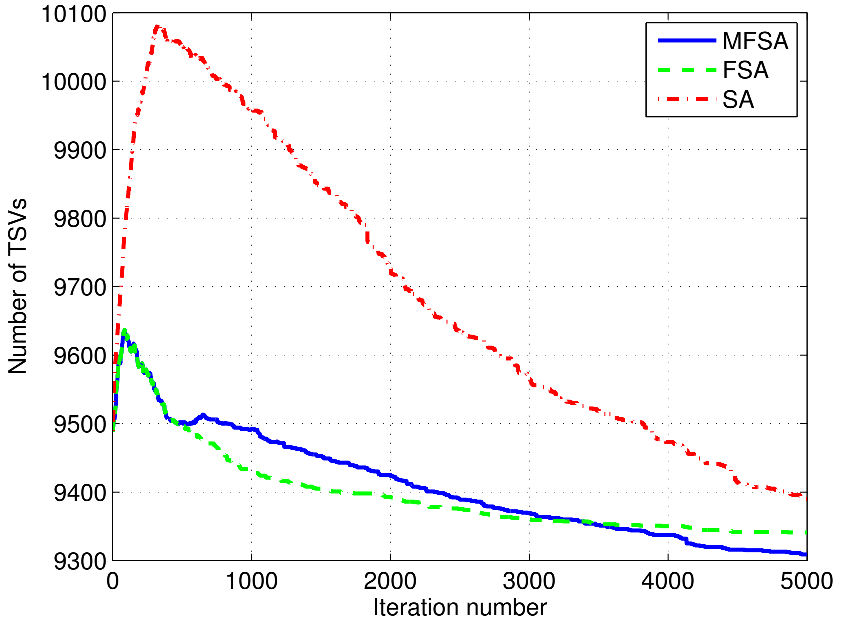


Figure A.13: IBM15 SA, FSA, and MFSA convergence

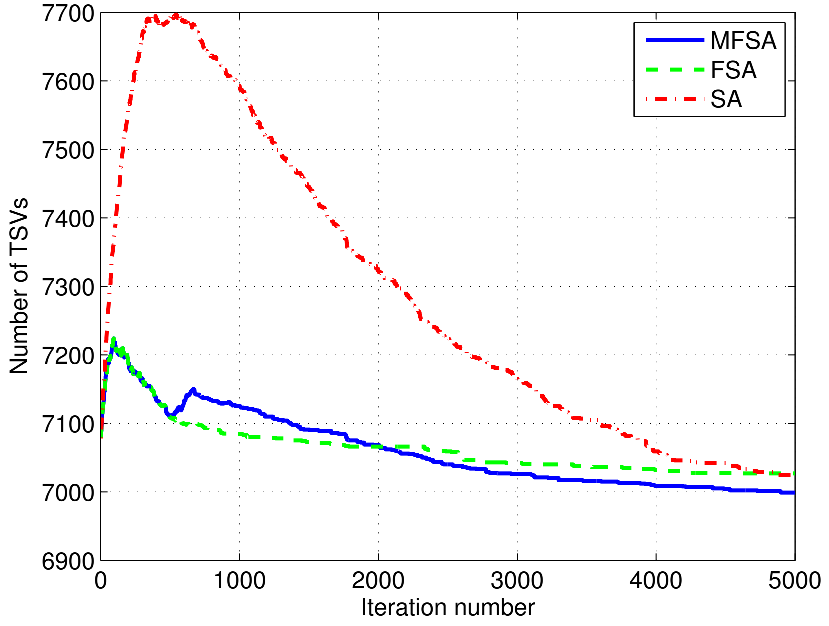


Figure A.14: IBM16 SA, FSA, and MFSA convergence

Bibliography

[1] Intel. Microprocessor quick reference guide, December 2012.

[2] Gordon E. Moore. Cramming more components onto integrated circuits. *Electronics*, 38(8), April 1965.

[3] Kyu-Myung Choi. An industrial perspective of 3d ic integration technology: from the viewpoint of design technology. In *Proceedings of the 2010 Asia and South Pacific Design Automation Conference*, ASPDAC '10, pages 544–545, Piscataway, NJ, USA, 2010. IEEE Press.

[4] Vasilis F. Pavlidis and Eby G. Friedman. *Three-dimensional Integrated Circuit Design*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2009.

[5] Yuan Xie, Jason Cong, and Sachin Sapatnekar. *Three-Dimensional Integrated Circuit Design: EDA, Design and Microarchitectures*. Springer Publishing Company, Incorporated, 1st edition, 2009.

[6] S. Borkar. 3d integration for energy efficient system design. In *Design Automation Conference (DAC), 2011 48th ACM/EDAC/IEEE*, pages 214 –219, june 2011.

[7] A. Kahng, J. Lienig, I. Markov, and J. Hu. *VLSI Physical Design - From Graph Partitioning to Timing Closure*. Springer, 2011.

[8] N. Sherwani. *Algorithms for VLSI Physical Design Automation*. Kluwer Academic Publishers, Dorrecht, The Netherlands, 1999.

[9] C. Alpert, D. Mehta, and S. Sapatnekar. *Handbook of Algorithms for Physical Design Automation.* CRC Press, 2009.

[10] S. Sait and H. Youssef. *VLSI Physical Design Automation: Theory and Practice.* World Scientific, 1998.

[11] Sandro Sawicki, Gustavo Wilke, Marcelo O. Johann, and Ricardo Reis. 3d-via driven partitioning for 3d vlsi integrated circuits. *CLEI Electron. J.*, 13(3), 2010.

[12] S. Sawicki, R. Hentschke, M. Johann, and R. Reis. An algorithm for i/o pins partitioning targeting 3d vlsi integrated circuits. In *Circuits and Systems, 2006. MWSCAS '06. 49th IEEE International Midwest Symposium on*, volume 2, pages 699 –703, aug. 2006.

[13] D.L. Lewis and H.-H.S. Lee. Testing circuit-partitioned 3d ic designs. In *VLSI, 2009. ISVLSI '09. IEEE Computer Society Annual Symposium on*, pages 139 –144, may 2009.

[14] Hua-Sin Ye, M.C. Chi, and Shih-Hsu Huang. A design partitioning algorithm for three dimensional integrated circuits. In *Computer Communication Control and Automation (3CA), 2010 International Symposium on*, volume 1, pages 229 –232, may 2010.

[15] Bernd Hoefflinger. Itrs: The international technology roadmap for semiconductors. In Bernd Hoefflinger, editor, *Chips 2020*, The Frontiers Collection, pages 161–174. Springer Berlin Heidelberg, 2012.

[16] George Karypis, Rajat Aggarwal, Vipin Kumar, and Shashi Shekhar. Multilevel hypergraph partitioning: Application in vlsi domain. In *IEEE TRANS. VERY LARGE SCALE INTEGRATION (VLSI) SYSTEMS*, pages 69–529, 1999.

[17] George Karypis and Vipin Kumar. Multilevel k-way hypergraph partitioning. In *Proceedings of the 36th annual ACM/IEEE Design Automation Conference*, DAC '99, pages 343–348, New York, NY, USA, 1999. ACM.

[18] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi. Optimization by simulated annealing. *Science*, 220:671–680, 1983.

[19] F. Harary. *Graph Theory.* Addison-Wesley Publishing Company, Philippines, 1972.

[20] M. Aigner. *Combinatorial Theory.* Springer-Verlag, New York, 1997.

[21] S. Areibi and A. Vannelli. Tabu Search: A Meta Heuristic for Netlist Partitioning. Technical report, University of Waterloo, Waterloo, ON, N2L-3G1, 1997.

[22] E. R. Barnes. An Algorithm for Partitioning the Nodes of a Graph. *SIAM Journal of Algebraic and Discrete Methods*, 3(4):541–550, 1982.

[23] S. W. Hadley, B. L. Mark, and A. Vannelli. An Efficient Eigenvector Approach for Finding Netlist Partitions. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 11(7):885–892, 1992.

[24] F. Hadlock. Finding a Maximum Cut of a Planar Graph in Polynomial Time. *SIAM Journal of Computing*, 4(3):221–225, 1975.

[25] B. W. Kerninghan and S. Lin. An Efficient Heuristic Procedure for Partitioning Graphs. *Bell System Technical Journal*, pages 291–307, 1970.

[26] C. M. Fiduccia and R. M. Mattheyses. A linear-time heuristic for improving network partitions. In *Proceedings of the 19th Design Automation Conference*, pages 175–181, 1982.

[27] George Karypis, Rajat Aggarwal, Vipin Kumar, and Shashi Shekhar. Multilevel hypergraph partitioning: Application in vlsi domain. In *In Proceedings ACM/IEEE Design Automation Conference*, pages 526–529, 1997.

[28] Charles Alpert, Andrew Kahng, Gi-Joon Nam, Sherief Reda, and Paul Villarrubia. A semi-persistent clustering technique for vlsi circuit placement. In *Proceedings of the 2005 international symposium on Physical design*, ISPD '05, pages 200–207, New York, NY, USA, 2005. ACM.

[29] Jianhua Li, L. Behjat, and A. Kennings. Net cluster: A net-reduction-based clustering preprocessing algorithm for partitioning and placement. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, 26(4):669 – 679, april 2007.

[30] Haixia Yan, Zhuoyuan Li, Xianlong Hong, and Qiang Zhou. Unified quadratic programming approach for 3-d mixed mode placement. In *Circuits and Systems, 2007. ISCAS 2007. IEEE International Symposium on*, pages 3411 –3414, may 2007.

[31] Yu Cheng Hu, Yin Lin Chung, and Mely Chen Chi. A multilevel multilayer

partitioning algorithm for three dimensional integrated circuits. In *ISQED*, pages 483–487, 2010.

[32] I.H.-R. Jiang. Generic integer linear programming formulation for 3d ic partitioning. In *SOC Conference, 2009. SOCC 2009. IEEE International*, pages 321 –324, sept. 2009.

[33] Hsien-Kai Kuo, B.C. Lai, and Jing-Yang Jou. Unleash the parallelism of 3dic partitioning on gpgpu. In *SOC Conference (SOCC), 2010 IEEE International*, pages 127 –132, sept. 2010.

[34] Cristinel Ababei, Yan Feng, Brent Goplen, Hushrav Mogal, Tianpei Zhang, Kia Bazargan, and Sachin S. Sapatnekar. Placement and routing in 3d integrated circuits. *IEEE Design and Test*, 22, 2005.

[35] Renato Hentschke, Sandro Sawicki, Marcelo Johann, and Ricardo Reis. A method for i/o pins partitioning targeting 3d vlsi circuits. In Giovanni Micheli, Salvador Mir, and Ricardo Reis, editors, *VLSI-SoC: Research Trends in VLSI and Systems on Chip*, volume 249 of *IFIP International Federation for Information Processing*, pages 259–279. Springer US, 2008.

[36] Jonathan S. Rose, W. Martin Snelgrove, Zvonko, G. Vranesic, and Senior Member. Parallel standard cell placement algorithms with quality equivalent to simulated annealing. *IEEE Transactions on Computer-Aided Design*, 7:387–396, 1988.

[37] Nicholas Metropolis, Arianna W. Rosenbluth, Marshall N. Rosenbluth, Au-

gusta H. Teller, and Edward Teller. Equation of state calculations by fast computing machines. *Journal of Chemical Physics*, 21:1087–1092, 1953.

[38] A. B. Kahng, S. Reda, and Q. Wang. APlace: A general analytical placement framework. In *Proceedings of ISPD*, pages 233–235, 2005.

[39] Natarajan Viswanathan, Min Pan, and Chris Chu. Fastplace 3.0: A fast multilevel quadratic placement algorithm with placement congestion control. In *Proc. of ASP-DAC*, pages 135–140, 2007.

[40] T. Chan, J. Cong, M. Romesis, J. Shinnerl, K. Sze, and M. Xie. mPL6: Enhanced multilevel mixed-size placement. In *Proc. of ISPD*, pages 212–214, 2006.

[41] Hans Eisenmann and Frank M. Johannes. Generic global placement and floorplanning. In *DAC*, pages 269–274, 1998.

[42] Sungho Kang. Linear ordering and application to placement. In *Proceedings of the 20th Design Automation Conference*, DAC '83, pages 457–464, Piscataway, NJ, USA, 1983. IEEE Press.

[43] ISPD04. ibm standard cell benchmarks with pads, 2006.