THE UNIVERSITY OF CALGARY

Clustering Techniques for Circuit Partitioning and Placement Problems

by

Jianhua Li

A THESIS

SUBMITTED TO THE FACULTY OF GRADUATE STUDIES

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE

DEGREE OF DOCTOR OF PHILOSOPHY

DEPARTMENT OF ELECTRICAL AND COMPUTER

ENGINEERING

CALGARY, ALBERTA

SEPTEMBER, 2007

THE UNIVERSITY OF CALGARY

FACULTY OF GRADUATE STUDIES

The undersigned certify that they have read, and recommend to the Faculty of Graduate Studies for acceptance, a thesis entitled "Clustering Techniques for Circuit Partitioning and Placement Problems" submitted by Jianhua Li in partial fulfillment of the requirements for the degree of Doctor of Philosophy.

Supervisor, Dr. Laleh Behjat
Department of Electrical and Computer Engineering

Co-supervisor, Dr. Graham Jullien
Department of Electrical and Computer Engineering

Dr. Wael Badawy
Department of Electrical and Computer Engineering

Dr. Vassil Dimitrov
Department of Electrical and Computer Engineering

Dr. Michael J. Jacobson
Department of Computer Science

External, Dr. Patrick H. Madden
Department of Computer Science, Binghamton University

23/07/2007
Date

Abstract

Partitioning and placement are two fundamental problems in Very Large Scale Integrated (VLSI) circuit physical design. Clustering techniques are extensively used in today's partitioning and placement algorithms, due to their ability to speed up the partitioning and placement solving process, and improve the final solution quality.

In this thesis, a series of clustering techniques with applications to circuit partitioning and placement is proposed. These clustering techniques are based on identification of a set of high quality clusters, defined as "gain" clusters in this thesis. It is proved that all these algorithms to identify gain clusters have linear time complexity. Compared with existing clustering algorithms, the gain cluster-based clustering techniques proposed in this thesis take more circuit connectivity information into account with a global view of the circuit structure. Therefore, better clustering solutions can be achieved.

In order to deal with the cluster overlap problem, three techniques for finalizing clustering solutions are proposed in this thesis. Among these techniques, a net scoring technique is shown to be able to solve the cluster overlap problem very effectively. This net scoring technique solves the clustering overlap problem by selecting the best set of nets for clustering.

The effectiveness of the proposed clustering techniques has been verified using the state-of-the-art circuit partitioners and placers on different public benchmark circuits. When using the proposed clustering techniques as a preprocessing step, the performance of leading-edge partitioning and placement tools can be further improved.

Acknowledgements

I would like to express my thanks to all of those people who have helped me to make this thesis complete.

First of all, I would like to thank my supervisor, Dr. Laleh Behjat. I am very happy to have her as my supervisor for my PhD study. I want to thank her for every thing that she has ever done for me, thank her for providing me the opportunity to come to Canada and pursue my PhD at the University of Calgary, thank her for the guiding and always encouraging on my study, research and job hunting, thank her for her great efforts to revise my numerous writing of technical reports, papers and this thesis, thank her for lots of valuable discussions with me, thank her for the funny talks, the free meals, and the nice gifts for me and my daughter, which really make life colorful. I gain a lot from her and the most impressing thing in my opinion is how to think and analyze a problem in a more mathematical and scientific way.

Secondly, I want to thank all of the committee members for their valuable comments that make the final thesis better than the original one.

I want to thank my officemates, Lan, Andy, Blair, Jie and Logan. It is very happy to work with you guys in the same cubicle for such a long time. I would like to thank Andy for the help he offered to me when I first came to Calgary in 2003. He drove me around the city and helped me to set up everything. I want to thank Logan and Jie for helping me to proofread the thesis. I also want to thank Jonathan for answering me various questions about the software configurations on the ATIPS lab machines, and thank Janelle for helping me to check the thesis format.

I want to thank my friend, J.C., for his introduction of the University of Calgary to me when I started to search universities in North America to pursue PhD study.

Finally, I want to express my thanks and love to my family. My mum and dad, and

my elder brother, they always support me from the day of my birth, until now. My wife, Meng, my one-year old daughter, Sophie, make me feel the life is so wonderful. I also want to thank my father in law, Shaolin. He stayed with us for more than one year since my daughter was born. He took care of my daughter and cooked for us.

I wish all of the people above the best in future.

Table of Contents

List of Tables

List of Figures

List of Terms

Acronyms:

CCR	:	The Cell Clustering Ratio
FPGA	:	Field-Programmable Gate Arrays
NCR	:	The Net Clustering Ratio
NP	:	Non-Polynomial
RMST	:	Rectilinear Minimum Spanning Tree
RSMT	:	Rectilinear Steiner Minimum Tree
VLSI	:	Very Large Scale Integration

Definitions:

Functions:

$\|\|.\|\|$	:	The number of elements in a set
$A(.)$	:	The area of a cell or a cluster
$conn(v_i, v_j)$	:	The connectivity between cells v_i and v_j
$degree(.)$	:	The degree for a cell or a net
$f(.)$	:	An arbitrary optimization problem objective function
$FS(i)$	:	The number of nets that have cell i as their only cell in cell i's partition
$gain(.)$	:	The cell gain in terms of netcut if the cell is moved from the original partition to the opposite partition
$gain_{in}$	:	The cell gain in terms of netcut if the cell is moved into a partition
$gain_{out}$	:	The cell gain in terms of netcut if the cell is moved out of a partition
$Min.$	:	Minimize
$s_c(C_i)$	:	The score for cluster C_i
$s_n(n_i)$	:	The score for net n_i
$size(C_i)$	:	The size of cluster C_i
$TE(i)$	:	The number of nets that contain cell i and have no cell in i's opposite partition

Scalars:

a_{ji}	:	A constant coefficient for net score calculation
c	:	A cell
c_i	:	Cell i
deg_{max}	:	The maximum cell degree
e	:	An edge or a hyperedge in a graph or a hypergraph
he	:	A hyperedge in a hypergraph
i	:	Index
j	:	Index
k_{ij}	:	A constant coefficient for cell connectivity calculation
L	:	Lower bound for a cluster size
$maxgain$	:	The index for the list of cells that have maximum gain value
n	:	A net in a netlist
n_c	:	The number of cells in a netlist
$n_c(C_i)$	:	The number of cells in cluster C_i
n_e	:	The number of edges or hyperedges in a graph or a hypergraph
n_i	:	Net i
n_n	:	The number of nets in a netlist
$n_n(C_i)$	:	The number of nets in cluster C_i
n_v	:	The number of vertices in a graph
$neighbors(c_i)$	:	The number of neighbors of cell i
$n_{neighbor_j}$	:	The number of first level neighbors of cell j
$n_{inCluster}^{max}$	:	The maximum number of times that a cell is clustered
$n_{neighbor}^{max}$	:	The maximum number of first level neighbors of a cell
n_p	:	The number of cell pins in a netlist
n_{pin_j}	:	The number of pins of cell j
p	:	The number of pins in a netlist
$p_{cluster}$	:	The number of pins in an initial cluster
$p_{cluster_i}$	:	The number of cell pins in an initial cluster made from the seed cell i
U	:	Upper bound for a cluster size
u	:	A cell or a vertex
v	:	A cell or a vertex
$w(e)$	:	The weight for an edge or a hyperedge e
$w(u)$	:	The weight for a vertex or a cell u
x	:	The variable for a given function

Sets:

C	:	A set of clusters
C_{gain}	:	A gain cluster
C_i	:	Cluster i
E	:	Edge or hyperedge set in a graph or a hypergraph
E'	:	Edge set in a sub-graph of a graph
$E_{cut}(C)$	:	The set of nets that are cut by clusters in the cluster set C

$E_{cut}(C_i)$	:	The set of nets that are cut by cluster C_i
G	:	A graph or a hypergraph
G'	:	A sub-graph of a graph
H	:	A graph or a hypergraph
V	:	Vertex set in a graph or a hypergraph
V'	:	Vertex set in a sub-graph of a graph

Chapter 1

Introduction

1.1 VLSI Circuit Physical Design

Very Large Scale Integration (VLSI) refers to the technology through which a circuit with a large number of components can be integrated and implemented in a small silicon chip [63, 69]. For today's VLSI circuit design, it is very common to have millions of transistors integrated on a chip. The process of VLSI circuit design is very complicated. It starts from the system specification, which describes the circuit performance, functionality and the physical dimensions, and ends with a packaged chip, which should have already been tested to meet all design requirements [69].

Physical design is a key step in the overall VLSI circuit design cycle. The input to the physical design is a circuit diagram, or schematic, which can be represented by a "netlist" format. Here the netlist is a list of nets. Each net in the netlist represents an interconnection between circuit components. The circuit components are usually referred to as "cells" in the netlist. The output of physical design is the circuit layout. The process of physical design can be further divided into three stages: partitioning, placement and routing, as illustrated in Figure 1.1. In the following, each stage of the physical design process is discussed briefly.

1.1.1 Partitioning

Today's VLSI circuits may easily contain several million logic gates [57, 58]. For such large circuits, it is necessary to partition the circuits into several smaller relatively independent sub-circuits. The smaller sub-circuits can be designed and tested separately and efficiently [63, 69]. This process of dividing a large circuit into smaller

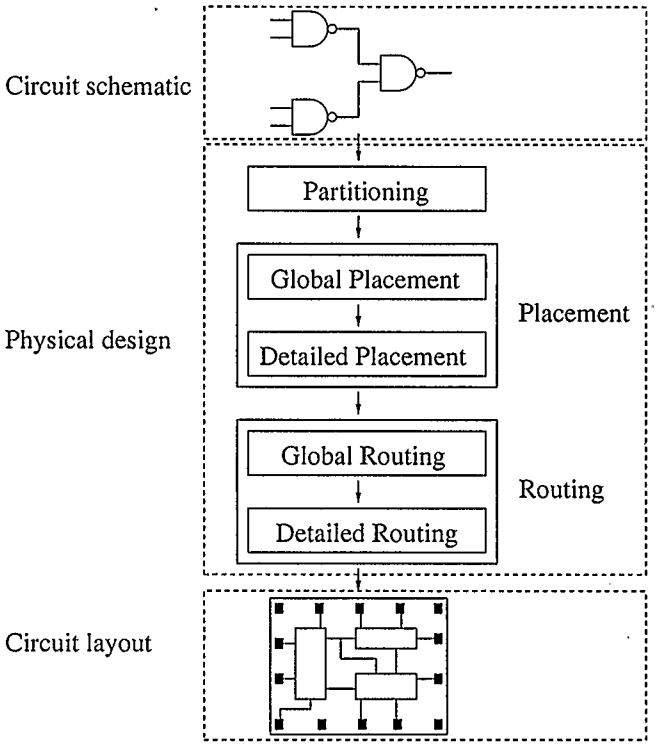


Figure 1.1: Schematic of the VLSI circuit physical design process

sub-circuits is called partitioning. A typical objective of partitioning is to divide a circuit into several relatively independent partitions with roughly equal sizes. The independence of the partitions is measured using the number of nets that are cut, "netcut", between partitions.

Partitioning is a fundamental problem in VLSI physical design. It can also be used to solve other problems in VLSI physical design, such as placement. However, partitioning is an NP-hard problem [16], therefore, practical partitioning algorithms are heuristics.

1.1.2 Placement

In this stage, the circuit components, such as logical gates and terminals, are arranged on a layout surface, such that each circuit component has a unique position and there is no overlap between these components [63, 69]. A general objective of the place-

ment problem is to minimize the total estimated wire length for the interconnections between the circuit components. This objective corresponds to the minimization of timing, signal delay, and power consumption of the circuit [68].

In practice, placement is usually performed in two steps: global placement and detailed placement, as illustrated in Figure 1.1. In the global placement step, an initial placement solution is generated. This solution may have a few overlaps between circuit components. In the detailed placement step, the components' overlaps are removed by performing placement legalization, and the placement solution is improved by performing local refinement.

1.1.3 Routing

In the routing stage, the paths for the interconnections that connect the circuit components on the layout surface are determined. Usually the objective of routing is to minimize the total wire length for the routed interconnections. [17, 63, 73].

Similar to the placement process, routing can be divided into two steps: global routing and detailed routing. In the global routing step, the approximate routes for the interconnections are determined. In the detailed routing step, the output of global routing is further refined and the precise routes for interconnections are determined.

1.2 Research Motivations and Contributions

The proposed contributions in this thesis are focused on the VLSI circuit partitioning and placement problems. Partitioning and placement are important steps in VLSI circuit physical design. The results of partitioning and placement can affect the subsequent design stages and the final performance of the designed circuit. For example, a "poor" placement can result in an infeasible routing [6, 30]. In this case, the placement has to be redone.

In today's VLSI physical design, the sizes of the circuits are growing drastically. In order to deal with this increasing circuit size and design complexity, numerous circuit clustering algorithms have been proposed and successfully used in partitioning and placement algorithms [13, 14, 28, 30, 38, 42, 45, 47, 62]. The main idea of applying clustering algorithms to partitioning and placement is that by using clustering algorithms, a circuit can be reduced to a smaller size, and the design can be completed on the smaller circuit more efficiently. Currently, almost all state-of-the-art partitioning and placement academic tools use various clustering techniques, for example, [20, 23, 26, 47, 59]. It has been demonstrated that clustering plays an important role in the performance of circuit partitioning and placement algorithms. A good clustering algorithm not only speeds up the whole partitioning and placement process, but also helps in producing high quality solutions.

Current clustering algorithms are generally efficient; however, the clustering solutions produced by current clustering techniques can be further improved. The main motivation and goal for this thesis are to develop clustering techniques that use the circuit structure to identify high quality clusters, and improve the efficiency of state-of-the-art of circuit partitioning and placement algorithms. In the case where there are cell overlaps between clusters, the clustering techniques should be able to remove cell overlaps and select the best set of clusters.

The main contributions in this thesis are summarized as follows.

- A clustering concept, called "gain" cluster, is proposed and used to define clusters with high quality.
- Three "single cluster" identification algorithms are proposed to identify gain clusters in a circuit. These algorithms have different characteristics and can be applied in different situations.
- A net scoring technique is proposed to solve the cell overlap problem in a set of

identified gain clusters.

- A net clustering algorithm is proposed which combines the single clustering identification and net scoring technique.

- The above proposed clustering algorithms are compared with other clustering techniques and tested as a preprocessing step for circuit partitioning and placement. The experimental results on public standard benchmark circuits demonstrate the effectiveness of the proposed algorithms by improving the state-of-the-art partitioning and placement algorithms consistently.

1.3 Thesis Structure

The rest of this thesis is organized as follows.

- **Chapter 2**

  In this chapter, the circuit partitioning and placement problems, the two main application areas of the proposed clustering algorithms, are introduced. Different partitioning and placement algorithms are classified and reviewed.

- **Chapter 3**

  In this chapter, the existing clustering techniques for circuit partitioning and placement are reviewed.

- **Chapter 4**

  In this chapter, the first contribution of this thesis, gain cluster definition and three algorithms to identify single gain clusters are proposed. The characteristics of the proposed algorithms are studied using the clustering statistics on standard benchmark circuits used in physical design.

- **Chapter 5**

  In this chapter, the second contribution of this thesis, a net scoring technique and a net clustering algorithm are proposed. The effectiveness of the proposed clustering techniques in Chapters 4 and 5 are verified using state-of-the-art circuit partitioning and placement tools on public standard benchmark circuits.

- **Chapter 6**

  In this chapter, a summary of this thesis and suggestions for future work are given.

Chapter 2

Circuit Partitioning and Placement

2.1 Introduction

In this chapter, circuit partitioning and placement, which are the two main application areas for the proposed research contributions, are introduced. Partitioning and placement are two fundamental steps in the overall circuit physical design flow. In the partitioning process, a given circuit is divided into several smaller sub-circuits, such that each sub-circuit has a manageable size and complexity. In the placement process, the cells in a given circuit are arranged on a layout surface, such that each cell has a unique position in the layout surface and there is no overlap between cells.

The rest of this chapter is organized as follow. In Section 2.2, some of the terminologies used throughout this thesis for partitioning and placement are given. In Section 2.3, the circuit partitioning problem is introduced, and the existing partitioning algorithms are reviewed in two categories: flat and multilevel approaches. In Section 2.4, the circuit placement problem is introduced, and the current placement algorithms are reviewed in three categories: simulated annealing, partitioning based, and analytical approaches. Finally, in Section 2.5, a summary of this chapter is provided.

2.2 Terminology

In this section, the terminologies that are used in this thesis are listed. Most of these terminologies are used in the context of graph theory [63, 69].

- In graph theory, a *graph* G is represented by two sets (V, E), where V is a set

of *vertices* $\{v_1, v_2, ..., v_{n_v}\}$, with n_v as the total number of vertices, and E is a set of *edges* $\{e_1, e_2, ..., e_{n_e}\}$. Here n_e is the number of edges. In a graph each edge connects only **two** distinct vertices.

- In graph theory, a *hypergraph* H is represented by two sets (V, E), where V is a set of vertices, and $E = \{he_1, he_2, ..., he_{n_e}\}$ is a set of *hyperedges.* Each hyperedge can connect **two or more** distinct vertices.

From the above definitions, a graph can be regarded as a special kind of hypergraph; i.e., a hypergraph with each hyperedge connecting exactly two cells. In the following, unless specified, only hypergraphs are used to discuss the rest of the terminology. For example, e can be used for either an edge or a hyperedge representation.

The following terms are used to describe the characteristics of a hypergraph or a graph.

- In a hypergraph, two vertices u and v are *adjacent* if there is one hyperedge e connecting them.
- A hyperedge e is *incident* with the vertices u and v, if u and v are adjacent through e.
- The *degree* of a vertex u is the number of hyperedges which are incident to u.
- The *degree* of an edge or hyperedge, e, is the number of vertices that e is incident to.
- In a hypergraph, depending on the specific applications, the vertices and hyperedges may have associated weights, which can be represented by $w(u)$ or $w(e)$, where u and e are a vertex or a cell and a hyperedge.
- A graph, $G' = (V', E')$, is a *sub-graph* of a graph $G = (V, E)$, if $V' \subseteq V$ and $E' \subseteq E$.

- In a graph, a *clique* is a sub-graph in which every two vertices are connected by an edge [69].

In VLSI physical design, a circuit is described in a "netlist" format, which can be represented either as a graph or a hypergraph, depending on the specific applications. The vertices in the graph or hypergraph represent the circuit modules, or cells, such as the logic gates and external terminals. The edges or hyperedges represent the interconnections, or nets, between the cells. In Figure 2.1, a simple circuit, and its corresponding hypergraph and graph models are shown respectively. In Figure 2.1 (a), $g0$, $g1$ and $g2$ represent the logic gates of a circuit, and $P1$ to $P4$ are the Input/Output (Abbreviated as I/O in subsequent text) pads. The nets, $n1$ to $n6$, connect the gates and I/O pads. In Figure 2.1 (b), the hypergraph model for the circuit is shown. The gates, $g0$ to $g2$, and I/O pads, $P1$ to $P4$, are modeled as cells ① to ⑦. The nets $n1$ to $n6$ are modeled by hyperedges $he1$ to $he6$, respectively. In this hypergraph, the degree for cells ① and ② is 3. In Figure 2.1 (c), the graph model for the circuit is shown. In this figure, the gates and I/O pads are also modeled as cells. However, the net $n2$, which connects more than two cells, is transformed to a *clique* in the corresponding graph model. All edge degrees in this graph model are equal to 2. The degree for cell ① is changed to 4 as shown in Figure 2.1 (c).

From Figure 2.1, it can be seen that the graph and the hypergraph models for a circuit can be different. A hypergraph model has exactly the same topology as the original circuit; but a graph model changes the circuit topology. In practice, the graph and the hypergraph models are both widely used, depending on the specific application circumstances.

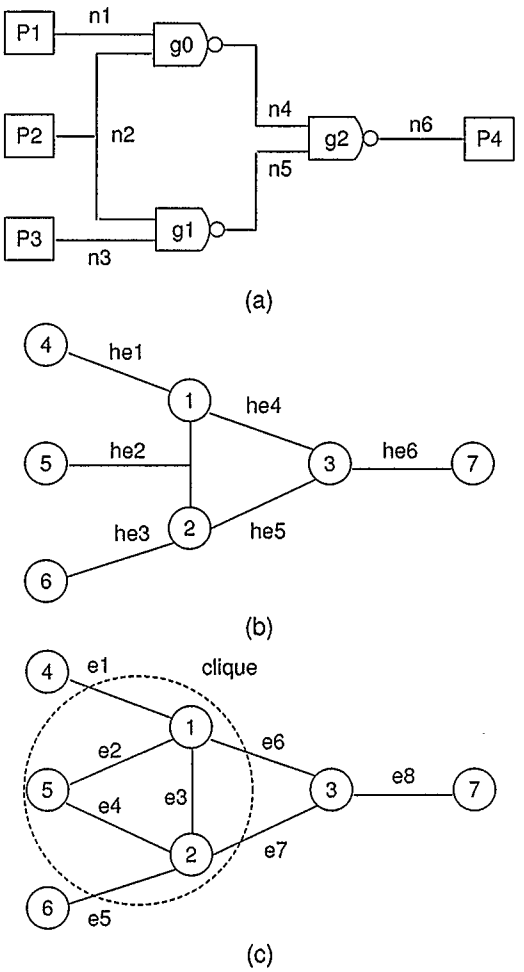


Figure 2.1: (a) A simple circuit (b) The hypergraph model (c) The graph model with a clique transformation

2.3 Circuit Partitioning

2.3.1 Problem Definition

VLSI circuit partitioning is an NP-complete optimization problem [45, 62] and has been extensively studied over the past three decades, for example [13, 14, 30, 33, 38, 45, 47, 62]. This problem can be described as follows: given a circuit, the partitioning objective is to divide the circuit into several smaller sub-circuits with roughly equal

sizes, such that the interconnections between these sub-circuits are minimized.[1] Each sub-circuit is referred to as a partition.

In practice, the bipartitioning problem, i.e., partitioning the circuit into two sub-circuits, is the most widely used application. Bipartitioning can be used recursively to partition a circuit into an arbitrary number of sub-circuits. It is also widely used to solve other physical design problems, such as placement [10, 61, 72]. Figure 2.2 shows the bipartitioning of a simple circuit. The number of interconnections between the two partitions is equal to 2.

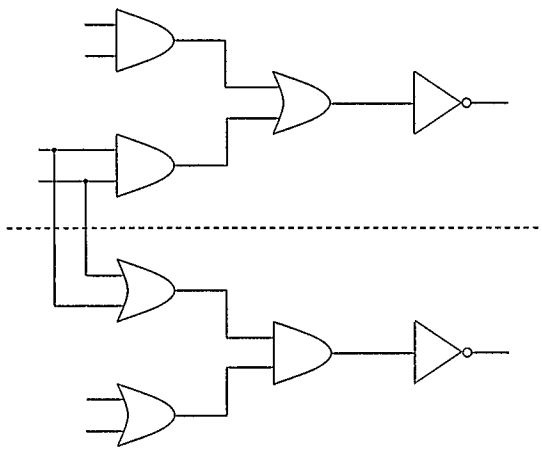


Figure 2.2: Bipartitioning of a simple circuit

Based on the operation mode of the circuit, partitioning algorithms can be classified into two categories: **flat** approaches and **multilevel** approaches [14]. Each category is briefly reviewed as follows.

2.3.2 Flat Partitioning Approaches

Flat partitioning approaches refer to those algorithms that directly work on the original input circuit, i.e., on the "flat" netlist [32, 33, 34, 36, 49, 64, 65], without per-

[1]It should be noted that depending on the specific applications, the partitioning objective may vary, such as signal delay optimization. However, the minimization of interconnections between different partitions, i.e., "mincut", is the most typical objective function in partitioning problems. It has been demonstrated [63] that the mincut objective can enhance the whole system performance and reduce the cost of manufacturing.

forming any circuit simplification, such as clustering. There are two types of flat algorithms: iterative improvement based [24, 34, 49] and optimization based [17, 35, 37]. These algorithms were developed when the circuit sizes were not as large as today's circuits. Because of the complexity of optimization based algorithms, they proved not to be scalable to today's circuits. Typical representatives of iterative improvement based algorithms are Kernighan and Lin (KL) algorithm [49], Fiduccia-Mattheyses (FM) algorithm [32, 33, 34], and the simulated annealing based partitioning algorithms [24]. Generally these algorithms start with an arbitrary initial solution, and iteratively improve the solution by swapping or moving cells between different partitions, until a local or global minimum is obtained. In the following, the three flat partitioning algorithms mentioned above are described respectively.

Kernighan and Lin (KL) Algorithm

The KL algorithm is a bisecting partitioning heuristic that swaps pairs of cells between two partitions. This algorithm was introduced in 1970, and has a high runtime complexity of $O(n^3)$ and serious limitations; but because of historical importance, it is briefly reviewed in this section.

The KL algorithm starts with converting a netlist to its corresponding graph model, then an initial bisecting partitioning solution, the assignment of each cell to one of the two partitions, is generated. This initial solution can be generated randomly, but the size of the partitions needs to be equal. Then the number of interconnections, or nets, between the two partitions, referred to as "netcut" in partitioning problems, is calculated. The "*gains*" for swapping every possible pair of cells, which consist of one cell from one partition and an other cell from its opposite partition, are calculated. Here, the gain for swapping a pair of cells is equal to the number of netcut decreases, if the two cells are exchanged. After calculating the gains for all possible pair of cells exchanges, the pair of cells that has maximum gain is selected and "tentatively"

exchanged, and this exchange is stored in a table. After the swapping of a pair of cells, these cells are "locked", and can not be swapped or used in the following gain calculation. Next, the same procedure is repeated for the unlocked cells and another pair of cells is selected and tentatively exchanged. This process is continued until all the cells are locked, indicating there is no more cell swapping possible. At this point, the table that has stored all the tentative cell exchanges is inspected, and the series of exchanges that can result in the lowest netcut are selected and made permanent. This completes an iteration of the KL algorithm, which is referred to as a "pass" in the algorithm. If there is a netcut decrease at the end of a pass, another iteration will start; otherwise, the algorithm has reached a minimum and stops.

It has been proved that the KL algorithm has a $O(n_c^3)$ time complexity, where n_c is the number of cells in the netlist. In practice, this complexity is considered to be very expensive for large scale circuits [69]. In addition, the sizes of the two partitions have to be equal and only a graph model can be used.

Fiduccia-Mattheyses (FM) Algorithm

The FM algorithm is another gain based iterative improvement algorithm. However, in the FM, several improvements have been made over the KL in the gain calculation. The main modification in the FM algorithm is that only one single cell, instead of a pair of cells, is moved from its original partition to the opposite partition at each movement. A "bucket" data structure is used in FM to efficiently implement the cell gain calculations, storage and updating. As a result, FM has a linear time complexity of $O(n_p)$, where n_p is the total number of cell pins in the netlist, and therefore runs faster than the KL algorithm. Furthermore, FM is not limited to using the graph model of a circuit and can use a hypergraph model. The sizes of the two partitions in FM can also be different.

The basic procedure of the FM algorithm is similar to that of the KL algorithm:

starting with an initial partition for the netlist, the "gain" for each cell movement is calculated. In the FM algorithm, the gain of a cell is defined to be the number of nets by which the netcut will decrease, if this cell is moved from its current partition to the opposite partition. Here, the netcut has the same meaning as in the KL algorithm; i.e., the number of interconnections between two partitions. More formally, the gain for cell i, $gain(i)$, can be calculated as follows:

$$gain(i) = FS(i) - TE(i),$$

where $FS(i)$ is the number of nets that have cell i as their only cell in cell i's original partition, and $TE(i)$ is the number of nets that contain cell i and have no cell in cell i's opposite partition.

After calculating all the cell gains, a suitable cell for movement, referred to as "base" cell, is found. This base cell must have the following three characteristics:

- The base cell can be moved freely.
- The base cell has the maximum gain value among all cells; thus it can result in the maximum reduction in netcut after its movement.
- The movement of base cell should not violate the partition size constraint. In other words, after the base cell movement, the size of each partition should be still within the range of the specified lower and upper limits.

The base cell is moved and "locked"; any locked cell can not be moved again during the current "pass". Here, a pass refers to a sequence of cell movements that result in netcut to reach a local or global minimum. After each cell movement, the gains of the neighbors of the base cell are updated. This process is repeated until there are no available base cells for further movement. Then, the algorithm checks the partitioning solutions generated during the cell moving sequence, and chooses the one that results

in the minimum netcut as the output of the current pass. At this point, the current pass terminates, all locked cells are released, and the output of this pass is compared with the initial partitioning solution at the beginning of this pass. If the output of this pass is the same as the initial partitioning solution, i.e., the initial partitioning solution is not improved in this pass, the FM algorithm stops; otherwise, a new pass will start. The new pass uses the output of the previous pass; i.e., the best solution obtained at that pass, as its initial partitioning solution. A similar process will be repeated in this pass.

An important property of the FM algorithm is its use of a "bucket" data structure to efficiently choose the base cells for movement. Figure 2.3 shows the schematic for the bucket structure [34, 69]. For each partition, an integer cell gain array is created,

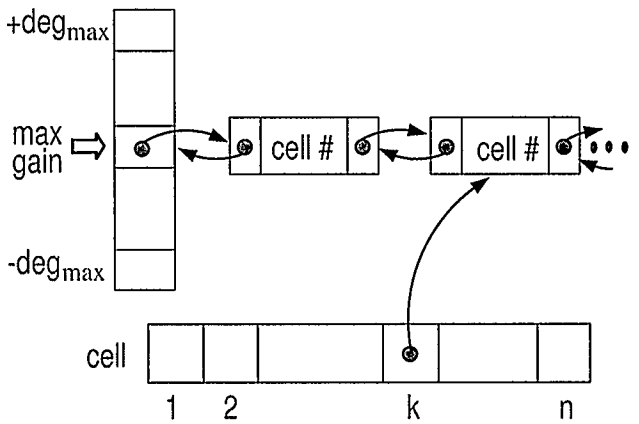


Figure 2.3: The data structure for cell gain manipulation [34, 69]

and each entry of the array is a double linked list, which is used to accommodate the cell numbers with the same gain values. It should be noted that for any cell in the netlist, its gain value is an integer with range from $-deg_{max}$ to $+deg_{min}$, where deg_{max} is the maximum cell degree in the netlist. Therefore, the length of cell gain array in Figure 2.3 is $2 \times deg_{max} + 1$. Once all cell gains have been calculated, the cell numbers will be added into the array based on their gain values. The cells that have the same

gain values will have the same index and are put into the same linked list in the array. Furthermore, in the cell gain array, a "maxgain" index is maintained to keep track of the index for the list of cells that have the maximum gain value.[2] A base cell can be decided on by directly visiting the list of cells with current "maxgain" index in the array. Once a base cell is moved and locked, its number is removed from the gain value list in the cell gain array. Then, the maxgain index and the array are updated appropriately. In practice, this bucket data structure has been demonstrated to be very efficient.

The most remarkable characteristic of the FM algorithm is its runtime efficiency. FM has a linear behavior: the complexity of FM algorithm is $O(n_p)$, where n_p is the number of pins (the cells and I/O pad terminals) in the circuit [34]. However, although the FM algorithm has been shown to produce very good partitioning results, even optimal solutions, for small size circuits [20, 21], for large scale circuits, it can easily get stuck in a local minimum [12, 46].

Simulated Annealing (SA) Algorithm

The inspiration of simulated annealing algorithms comes from the real physical process of metal annealing. In the real annealing process, a metal is heated to a very high temperature, such that its atoms can have enough energy to move freely. Then, the metal is cooled down slowly, until its temperature reaches the recrystallization point, or is low enough. During this process, the atoms will move such that the metal crystal structure is changed towards the lower global energy equilibrium. Finally, when the temperature is low enough, the atoms are fixed and do not move again. At this point, the metal is stabilized with the global minimum energy [69]. Simulated annealing algorithms mimic the real annealing process and optimize a problem in a similar way as annealing metals [63, 69]. In VLSI physical design, simulated annealing has been

[2]Note that this index is not necessarily at the top of the array.

successfully used as an iterative technique for many different problems, for example, partitioning [63], placement [54], and routing [66].

In the context of VLSI partitioning, simulated annealing partitioning algorithms try to reach a global optimal solution by mimicing the real metal annealing process. Generally, simulated annealing partitioning techniques start with a random partitioning solution, then, a series of cell movements is performed. Cell movements can be either the exchange of two cells, like the KL algorithm, or the movement of one single cell, like the FM algorithm. The cost of a movement, for example, change in netcut, is calculated. If this cost lowers the partitioning cost, the movement is accepted. If the cost increases the cost of the partition, it can be accepted based on a probability function. This process is repeated until a stopping criterion is reached. Then, another iteration starts with a decreased probability of accepting "bad" movements.

An outstanding advantage of the simulated annealing over other iterative algorithms is that simulated annealing can "escape" from a local minimum state and find better solutions; e.g., a better local minimum or hopefully a global minimum [63, 69]. An example of this effect is shown in Figure 2.4 for an arbitrary problem. In this figure, the x-axis corresponds to possible partitioning solutions, and the y-axis provides the corresponding partitioning netcut results. It can be seen that there are many local optimal points in the partitioning results. Depending on the starting points, different local optimum points can be obtained. Once a local minimum is found, unless the optimization algorithm is able to take steps that will result in worsening the solution, i.e., uphill moves, no better solution can be explored. The behavior of simulated annealing algorithms is different from other local optimization techniques, such as Newton's method [18], Lagrangian techniques [18], and Interior-Point Methods [16, 18]. In simulated annealing algorithms, given a current point, the next iteration point can be a point that either has a better result, or has a worse result than the

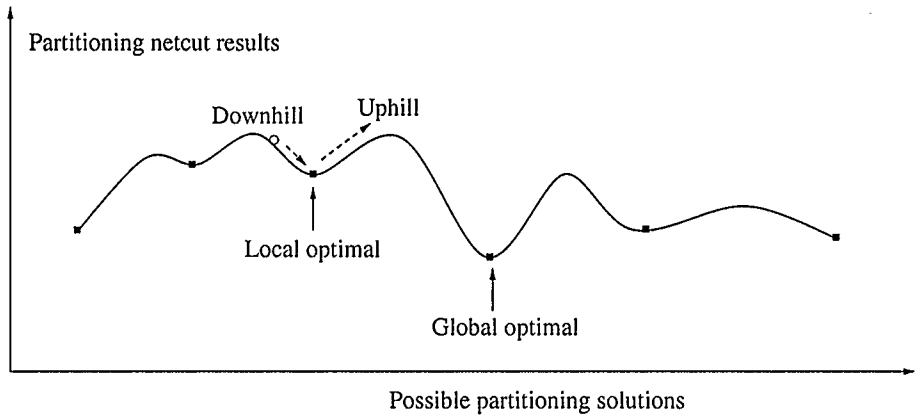


Figure 2.4: Illustration of the downhill and uphill movement in a simulated annealing algorithm for partitioning problem optimization

current point with a controlled probability. An "uphill" movement is allowed with probability, as illustrated in Figure 2.4.

The performance of the simulated annealing algorithm depends on fine tuning many parameters. Furthermore, for large scale circuits, simulated annealing algorithms can be too slow.

Overall, the flat partitioning approaches can provide reasonable solutions for small to medium size circuits, but as circuit sizes become larger, the solution quality deteriorates [33, 45] and the runtime cost is also very expensive. Therefore, for today's big design problems, flat partitioning algorithms are seldom solely used [20, 47].

2.3.3 Multilevel Partitioning Approaches

Multilevel partitioning approaches refer to the algorithms that perform partitioning on a small circuit derived from the original circuit and then refine the solution in multiple levels [13, 14, 28, 31, 42, 45, 47, 62]. Generally the multilevel partitioning approaches consist of three phases: multilevel clustering, initial partitioning and multilevel refinement. In Figure 2.5, the three stages of the multilevel partitioning algorithms are shown [47]. In this figure, a hierarchy of 3 levels of clustering/refinement

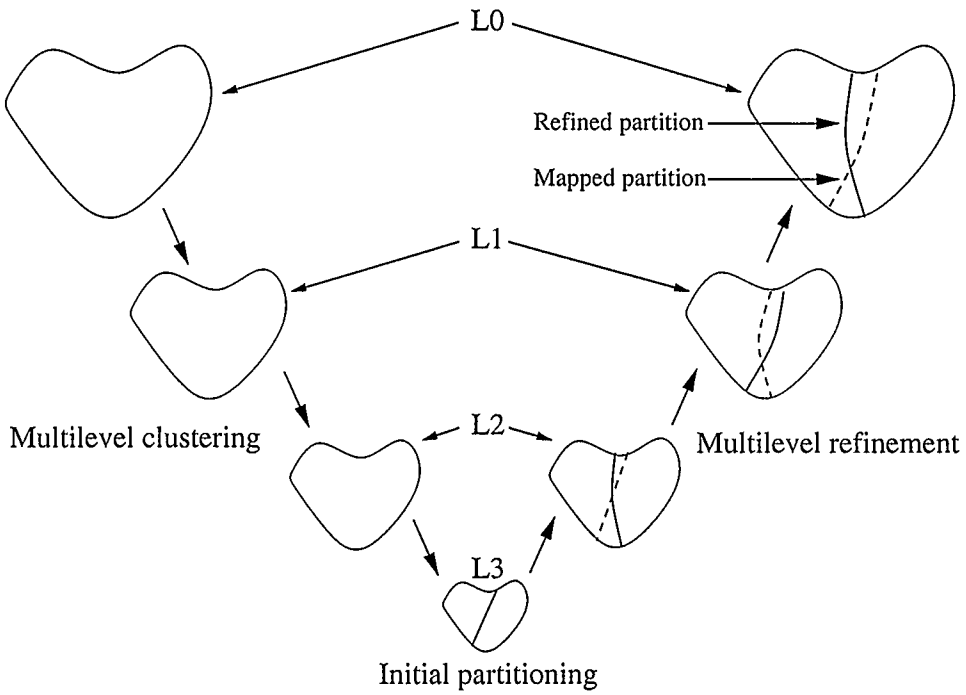


Figure 2.5: Multilevel partitioning algorithm procedure [47]

is constructed for a given circuit. L0, L1, L2, and L3 refer to circuits at each clustering/refinement level. L0 is the original circuit and L3 is the smallest clustered circuit. In the first phase, multilevel clustering, the original circuit is clustered successively into a sequence of smaller circuits in several levels. In each level, a new smaller circuit is generated by clustering its upper level circuit. Specifically, at each clustering level, the cells that are highly connected are grouped together to form a cluster. This cluster will be considered as a normal cell in the subsequent circuit level. The clustering process is repeated until the clustered circuit is reduced to a desirable size. Then, in the second phase, initial partitioning, one or several initial partitioning solutions on the lowest level circuit; i.e., the smallest clustered circuit, are obtained. In practice, these initial solutions are usually generated by a random assignment or a flat partitioning algorithm, such as the FM algorithm, since flat partitioning algorithms can produce high quality solutions efficiently on a circuit with small size. In the last

phase, multilevel refinement, each of the initial partitioning solutions on the lowest level circuit is mapped to its upper level circuit, and this solution is further refined by a flat partitioning algorithm. Then the improved solution at the current level is mapped to its upper level circuit and further refinement is performed. This process of solution mapping and refinement is repeated in the reverse order of clustering, until the original circuit is reached.

Two state-of-the-art circuit multilevel partitioners, hMetis [45, 46, 47, 67] and MLPart [13, 20] follow the scheme shown in Figure 2.5. However, they use different clustering, initial partitioning, and refinement techniques. In hMetis, five clustering techniques have been implemented, including edge coarsening, hyperedge coarsening, modified hyperedge coarsening, FirstChoice, and hybrid FirstChoice. These clustering techniques will be discussed in the next Chapter. The initial partitioning solution in hMetis can be either generated randomly, or by an FM algorithm. The refinement techniques used in hMetis are standard FM and two variations of FM, one-way FM and early-exit FM. In MLPart, the heavy-edge matching clustering technique is implemented [13]. Furthermore, the netlist is dynamically updated during the clustering process. The initial partitioning solution in MLPart is generated by a variation of the FM algorithm, called CLIP-FM which can produce better partitioning results but needs more runtime than the standard FM algorithm. The refinement technique in MLPart uses the standard FM algorithm.

2.3.4 Comparison between Flat and Multilevel Approaches

Although multilevel partitioning algorithms are sophisticated procedures and require the implementation of efficient data structures, they offer several advantages over the traditional flat approaches.

Low Runtime

The overall runtime for the partitioning process is significantly reduced. The runtime speedup is mainly due to the application of clustering. Clustering reduces the size of circuits, making the partitioning problem easier to solve. As a result, less time is required to obtain or improve a partitioning solution. Furthermore, the refinement is performed in an iterative way with a refined solution from its previous level as a starting point. This good starting solution will reduce the number of iterations in the iterative refinement procedure and provide a more rapid convergence to a local or global minimum, further reducing the runtime.

High Solution Quality

The solution quality of the multilevel partitioning approaches is greatly enhanced, compared to that of the flat approaches [30, 46]. This is due to the combination of multilevel clustering and refinement.

In the multilevel framework, multilevel clustering and refinement benefit each other to improve the solution quality. Clustering reduces the circuit size and for a smaller circuit, a high quality solution can be obtained by flat partitioning algorithms [20, 50]. This solution is the starting point for the subsequent multilevel refinement phase. It should be noted that due to the nonlinear nature of the partitioning problem, finding a good starting point is extremely important to the quality of the final result. The difference between starting points for a nonlinear programming problem is illustrated in Figure 2.6, where an arbitrary function $F(x)$ is plotted. In this figure, the function, $F(x)$, has many local minima and one global minimum. Different starting points can lead to different minimum values. For example, in Figure 2.6, starting point 1 can lead to a local minimum, while starting point 2 leads to the global minimum.

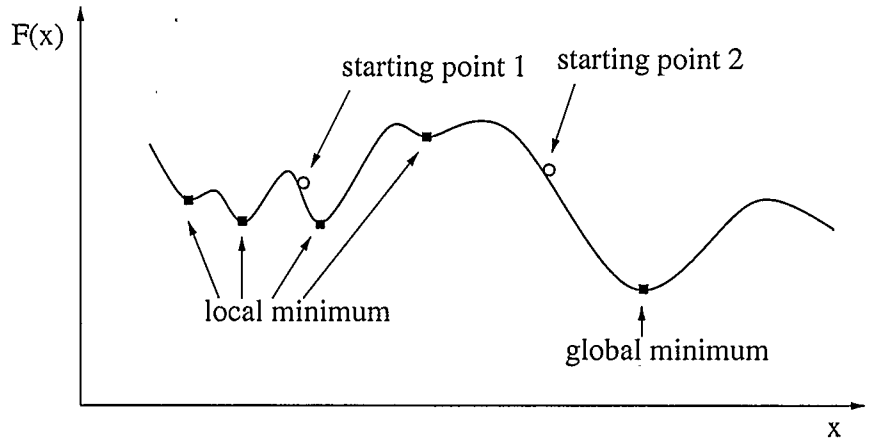


Figure 2.6: Starting point effect for a polynomial optimization problem

High Performance Stability

The third advantage of multilevel partitioning approaches is the higher performance stability, in terms of both the solution quality and runtime, compared to the flat approaches. Here, "stability" refers to variations between different results from several times of running a partitioning algorithm implementation; i.e., a partitioner.

Due to the intractability of circuit partitioning problems, most practical algorithms use "heuristic" approaches. In other words, currently there is no known polynomial algorithm that can find the optimal results for partitioning problems. Most practical partitioning heuristics, including both flat and multilevel approaches, have a random nature. As a result, partitioning results obtained by the same algorithm in different runs can be different. Here, a run refers to a one time execution of the algorithm implementation. In order to get a high quality partitioning solution, usually several independent runs of a partitioner are performed and the best partitioning result is chosen.

In terms of the difference between partitioning results from different runs, it can be concluded that the performance of multilevel partitioning algorithms is much more stable than that of flat partitioning algorithms, since the variations in partitioning

results from multilevel algorithms tend to be quite small, compared to those of flat algorithms. In Figure 2.7, the bipartitioning results of 20 runs on benchmark circuit ibm01 by a flat partitioner, "FMPart" [34], and a multilevel partitioner, "hMetis" [47], are plotted respectively. In this figure, the x-axis is the run number for both partitioners, and the y-axis is the corresponding partitioning result in terms of netcut by each partitioner per run. It can be seen that the multilevel partitioner produced

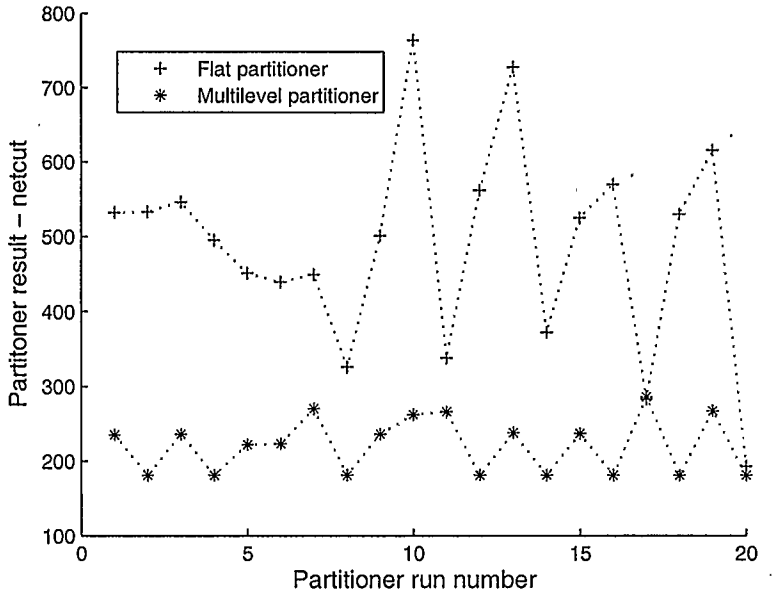


Figure 2.7: The bipartitioning result comparison between flat and multilevel partitioners on benchmark circuit ibm01

more stable results than the flat partitioner. The partitioning results of 20 runs by the flat partitioner fluctuate drastically. Furthermore, the solution quality of the multilevel partitioner is better than that by the flat partitioner. More stability experiments for different types of partitioning algorithms can be found in [12]. In [12], it is shown that for the whole ISPD98 benchmark suite, the multilevel partitioning algorithms produce more stable and better partitioning results than flat partitioning approaches.

In summary, compared with the flat partitioning approaches, the multilevel ap-

proaches have better optimality, scalability, and stability.

2.4 Circuit Placement

2.4.1 Problem Definition

Placement, an NP-hard problem, is another important optimization problem in the physical design process. It is also an NP-complete problem [68, 69]. Generally, circuit placement problems can be defined as follows: given a circuit, the interconnections between the circuit components (including cells and I/O pads), called a "netlist", and the dimensions for these components (height, width), are known; the process of placement is to arrange or find a valid physical position for each circuit component on the layout surface, such that the total **estimated** wire length for interconnections is minimized. The main constraint on the placement problem is that there should be no overlap between components. It should be noted that depending on the circumstances, the objective of placement may vary. For example, the objective can be minimization of the layout area, reduction of circuit congestion, minimization of power consumption, timing, or some combination of these objectives [55]. Due to the intractability of the placement problem, placement is generally further divided into two stages: global placement and detailed placement. In the global placement stage, the cell positions are roughly determined and some cells may overlap. In the detailed placement stage, the placement result from the global placement stage is first legalized by removing any cell overlap, and then the legalized placement result is further refined. It should be mentioned that, in practice, the cells are usually placed in rows on a rectangular layout surface. Furthermore, there may be areas on a layout surface that can not be used for cell placement.

Placement has a deep impact on the final performance criteria, such as signal integrity, timing delay, and power consumption [43, 44, 55, 70]. After the placement,

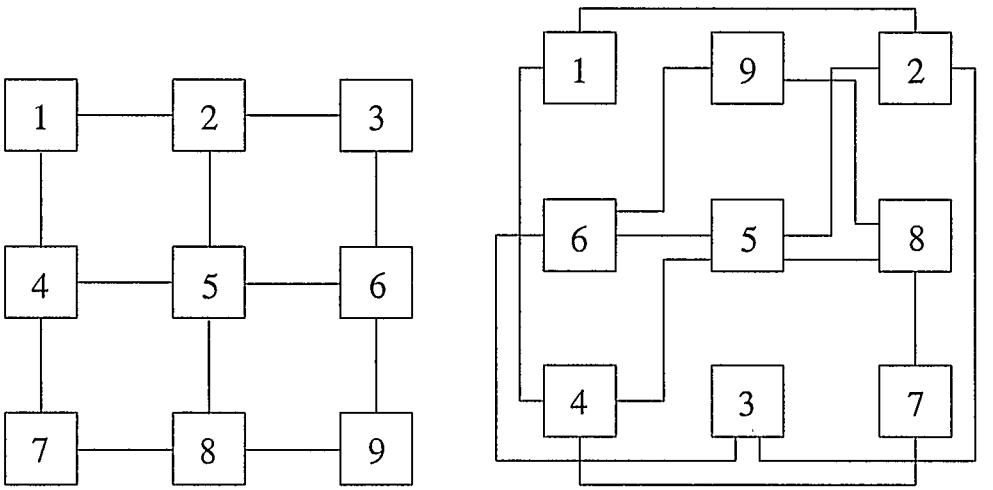


Figure 2.8: An example of good placement VS bad placement [63] (a) Optimal placement with estimated wire length = 12 (b) Alternative placement with estimated wire length = 22

all cells will be fixed on the layout surface, and all subsequent procedures will be performed based on this placement result. A good placement will benefit the optimization of subsequent design objectives; however, a bad placement will lead to poor subsequent design objectives, and, in the worst case, the subsequent design may not be feasible and the placement will have to be redone. The effect of a "poorly" placed circuit is shown in the example of Figure 2.8, where two placement solutions for a mesh are given. It can be seen that the total wire length, from the placement solution given in Figure 2.8(a) is much shorter than that from the solution given in Figure 2.8(b). The placement solution in Figure 2.8(b) can lead to a circuit design with more timing delay, more interconnect congestion, and more power consumption.

2.4.2 Wire Length Estimation

The goal of the placement algorithms used in this thesis is to determine the cell positions such that the total length of **estimated** interconnections between the cells is minimized. The actual completion of interconnections between cells is performed

at the global routing stage. As a result, in placement, the model used to estimate the wire length can become important. On one hand, the model of wire length estimation should be accurate enough to represent the real interconnection wire length. On the other hand, the calculation should be computationally efficient so as to deal with the large sizes of circuits encountered in modern designs. Currently, there are several widely used models for wire length estimation [63, 69]. All these models use Manhattan geometry; i.e., the interconnect tracks are either horizontal or vertical [63]. In the following, three typical wire length estimation models are described.

- Rectilinear Minimum Spanning Tree (RMST)

  In this model, a rectilinear minimum spanning tree [16, 27] is constructed for each interconnection, or net. A rectilinear minimum spanning tree (RMST) is a minimum length tree that connects all the given terminals, or cells, of a net. In an RMST, edge lengths are measured in L1 (Manhattan) metric [4]. The estimated wire length is equal to the tree length.

  Figure 2.9 shows a rectilinear spanning tree for a net that connects 4 cells. If

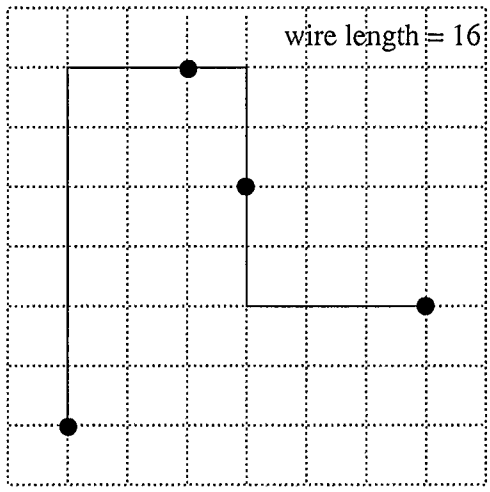


Figure 2.9: Rectilinear minimum spanning tree model [69]

the grid width and length in Figure 2.9 are 1 unit each, then the estimated

wire length for this interconnection is equal to 16 units. An RMST can be constructed using Prim's algorithm [4, 69] in $O(n^2)$ time, where n is the number of terminals. Other algorithms to find RMST are Kruskal's greedy algorithm [69], and an efficient $O(n\log n)$ algorithm that combines divide-and-conquer and Prim's algorithms together [4]. The advantage of the RMST model is that a spanning tree can be constructed rapidly for small nets. However, the model doesn't represent the real interconnection accurately, since this model can cause redundant interconnections and consequently longer estimated wire length [16, 27, 69].

- Rectilinear Steiner Minimum Tree (RSMT)

  In this model, a rectilinear Steiner minimum tree [16, 27, 69] is constructed to represent an interconnection and the wire length calculation is performed on the constructed Steiner tree. A Steiner tree is a tree that contains not only all of the original vertices of a graph, but also some additional vertices, referred to as "Steiner" points. The purpose of introducing Steiner points is to reduce the cost of the tree. A minimum Steiner tree is a tree with minimum cost. The cost is defined to be equal to the sum of weight of edges connecting the vertices and Steiner points in the Steiner tree. A rectilinear minimum Steiner tree is a minimum Steiner tree where all the edges are either horizontal or vertical.

  Figure 2.10 shows a constructed Steiner tree for the same net as in Figure 2.9. It can be seen that the wire length for the Steiner tree is 13 units. The rectilinear Steiner tree model is the most accurate representation of the real interconnection and the optimal wire length can be obtained from the Steiner tree model. However, the problem of finding a Steiner tree for an interconnection is NP-hard [16] and therefore, in practice, this model is not widely used.

- Half-Perimeter Wire length

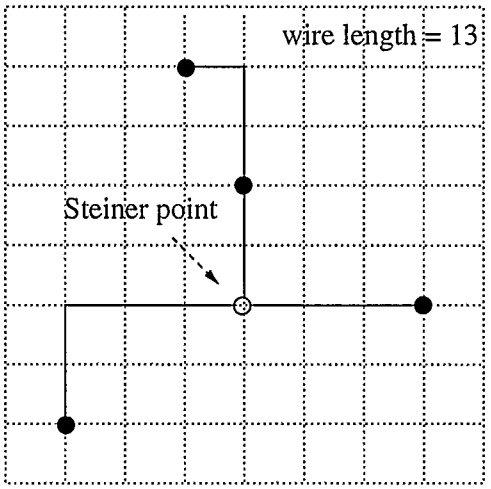


Figure 2.10: Rectilinear Steiner tree model [69]

In this model, the smallest bounding rectangle that encloses all the interconnection points is found, and the estimated wire length is equal to half of the perimeter of the bounding rectangle.

In Figure 2.11, the wire length for the half-perimeter model is 12 units. Compared

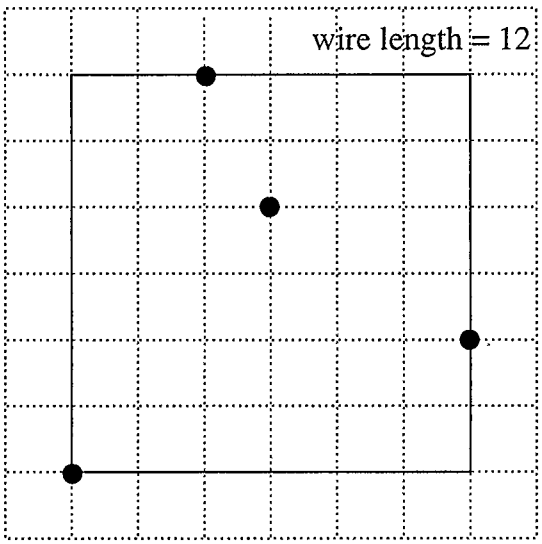


Figure 2.11: Half-perimeter model [63]

with the spanning tree and Steiner tree models, the half-perimeter model is the most widely used model in practical placement algorithms. This model is very

fast; i.e., the construction of the bounding rectangle and the computation of the perimeter can be done in linear time. This model is accurate to represent interconnections connecting 2 or 3 cells, where the wire length is equal to that obtained from a Steiner tree model, but this model underestimates the wire length of nets with higher degrees. In real integrated circuits, the majority nets connect only 2 or 3 cells [12, 57, 58]. Thus, this model represents a good trade-off between time complexity and accuracy.

2.4.3 Existing Placement Algorithms

There has been numerous research on placement, such as [10, 23, 25, 26, 30, 43, 61, 68, 74, 76]. Since placement problems are also known to be NP-complete [68], most practical placement algorithms are heuristics. With the size and complexity of integrated circuits increasing rapidly[3], numerous placement algorithms have been proposed in recent years to tackle with the new challenges and difficulties arising in modern VLSI circuit design. Currently, the state-of-the-art placement heuristics can be classified into 3 major categories: simulated annealing [54, 66, 77], partitioning based [10, 25, 61], and analytical methods [26, 43, 59, 71]. In the subsequent sections, each category is reviewed.

Simulated Annealing Placement Approaches

In Section 2.3.2, the basic concept behind the simulated annealing algorithms was introduced. Simulated annealing placement algorithms have the same characteristics as the simulated annealing partitioning algorithms discussed earlier. In solving the placement problem using simulated annealing, wire length minimization is usually the objective. The algorithm starts from an initial placement solution, and generates a

[3]According to the famous Moore's Law [56], the number of transistors on an integrated circuit is doubled every 18 months. It is predicted that Moore's Law will continue for several chip generations [8].

series of new placement solutions by operations such as cell movements or swappings. For each solution in the series, the cost is estimated, and the new solution is accepted if the cost is lower, or if the cost is higher, and where the annealing function allows the annealing operation (random jump to a new solution).

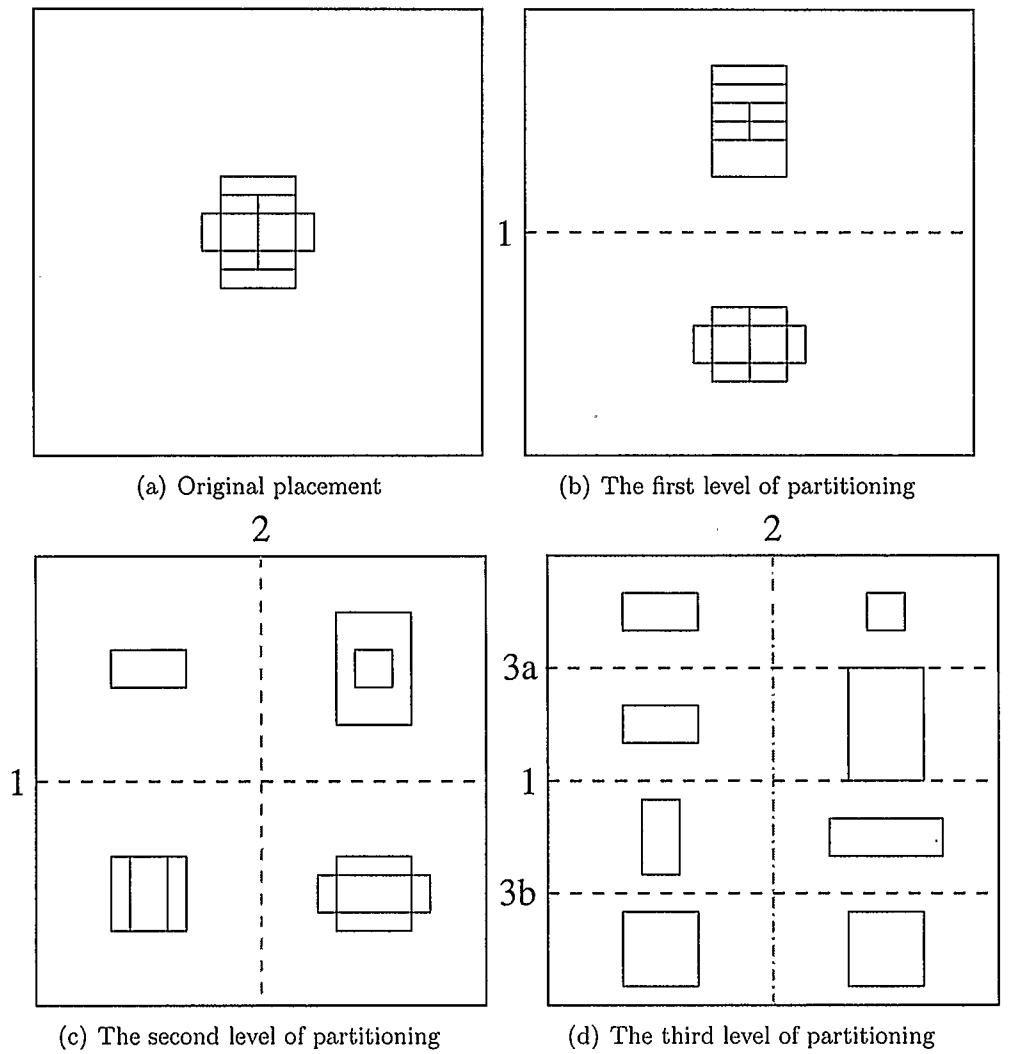
Typical representatives of simulated annealing placement tools are TimberWolf [66] and Dragon2000 [77]. The main drawback of simulated annealing based placement algorithms is that the process is very time consuming. For modern large scale circuit designs, the runtime cost of simulated annealing can be prohibitive [7, 74].

Partitioning Based Placement Approaches

Partitioning based placement approaches are an important class of placement algorithms. Essentially, these algorithms use a divide-and-conquer strategy to solve the placement problem. In these algorithms, first a given circuit is partitioned into smaller sub-circuits, where each sub-circuit is placed into one subsection of the layout area. At this point, this level of partitioning of both netlist and layout area is complete and a new level starts, where each sub-circuit is considered independently. The circuit and layout area partitioning process are repeated until the sub-circuits are small enough, and each sub-circuit contains just a few cells.

In Figure 2.12, an example of a simple circuit placement using partitioning is illustrated.[4] In Figure 2.12(a), originally all of the cells are placed at the centre of the layout area and they overlap with one another. In Figure 2.12(b), the original circuit is bipartitioned and the layout area is also partitioned horizontally into two subsections. Each subsection accommodates one sub-circuit. In Figure 2.12(c), another level of partitioning is performed. Note that this time, the layout area is partitioned vertically. In practice, the layout area partitioning is usually performed in horizontal and vertical directions alternatively. Finally, in Figure 2.12(d), the third level parti-

[4]For the purpose of simplicity, the interconnections between cells in Figure 2.12 are not plotted.



(a) Original placement (b) The first level of partitioning

(c) The second level of partitioning (d) The third level of partitioning

Figure 2.12: An example of the partitioning based placement [63]. Here, each partition is shown using the dotted line. (a) Original placement (b) After the first bipartitioning (horizontal) (c) After the second bipartitioning (vertical) (d) After the third bipartitionings (two horizontal)

tioning is performed and each component occupies an unique layout subsection. From Figure 2.12, it can be seen that after the partitioning of the netlist and layout area is complete, the placement for each small sub-circuit on the corresponding small layout area section can be performed efficiently. For example, the branch-and-bound, or even enumeration methods can be used to solve such a placement problem optimally since the problem instance is small [21].

It should be noted that depending on the final placement objectives, the objective functions used in partitioning may vary [69]. A general objective function during partitioning is to minimize the interconnections between different partitions. It has been verified that interconnect minimization during partitioning is roughly equivalent to minimizing the half perimeter wire length for final placement [68, 69, 77]. During the circuit partitioning sequences, depending on the problem scale, i.e., the circuit sizes, different partitioning approaches can be used to achieve the best trade off between solution quality and runtime. For example, when the circuit is "large", e.g., more than 10000 cells, the multilevel partitioning algorithms are used; when the circuit is "small", the flat partitioning algorithms are typically used.

Typical representatives of partitioning based placers include Capo [61], FengShui [10] and NTUplace2 [25]. Generally these kind of placers are scalable. Furthermore, the global placement results from these placers are generally cell overlap free, or have only a small percentage of cell overlap. As a direct result, less effort is required in the subsequent detailed placement stage to legalize the result from global placement. Despite the above advantages, the placement results by partitioning based placers can be unstable; i.e., different runs of a placer will give different placement results. This is due to the instability of the partitioning algorithms used in the placement procedure. In addition, due to the recursive nature of partitioning based procedure, i.e., the original netlist needs to be partitioned into thousands of smaller sub-circuits by invoking

the partitioner many times, the runtime for these placers is also prohibitively large for large scale circuits.

Analytical Placement Approaches

Analytical placement approaches refer to the algorithms in which mathematical analysis is used to solve the placement objectives. Generally, in these approaches, the original placement problem is formulated by a constrained nonlinear programming model [43]. This optimization problem is solved using nonlinear optimization techniques.

Analytical placement algorithms have numerous variations. Among these variations, force-directed quadratic placement techniques are widely used, due to the ease of solving this type of formulation [74]. In the remainder of this section, the force-directed quadratic placement technique is described in detail. Other analytical placement approaches are similar to this technique, except that the objective functions, or nonlinear optimization techniques used, can be different.

In force-directed placement algorithms, an interconnection, or a net, connecting different cells in a netlist can be regarded as a physical spring with forces imposed on the cells incident to this net. The forces applied on each cell in a net try to pull cells together, so as to obtain a minimum energy solution in order to maintain the system equilibrium. According to Hooke's law in mechanics, the amount of the force exerting on the cells via the spring is proportional to the distance between these cells [63]. In a balanced physical system with objects and springs, each object will be at the exact location where the overall resultant forces from different directions on this object is zero. As a result, the overall energy required to maintain the system equilibrium is minimized; that is, the springs are stretched with minimum tension. Similarly, in force-directed placement algorithms, originally all cells, except for those fixed I/O pads and prefixed blocks, are supposed to move freely. Since these cells have nets

connecting them, they will move toward the direction of the resultant forces applied by the nets. The whole system is stable when all cells have zero resultant force on them. At this point, the minimum wire length objective can be achieved.

Typical quadratic placers are FastPlace [74], hATP [59], Kraftwerk [71], mFAR [40], and BonnPlace [19]. Besides the quadratic wire length objective function, other wire length functions, such as the log-sum-exponential function, have been proposed in recent years and used in some analytical placers, such as APlace [43], NTUplace3 [26], and mPL6 [23]. These non-quadratic analytical placers show comparable or even better performance compared to quadratic placers.

2.5 Summary

In this chapter, the circuit partitioning and placement problems have been introduced. These two problems are the main application areas for clustering algorithms in the context of VLSI circuit physical design.

Current partitioning algorithms can be classified into two categories: flat and multilevel methods. Multilevel partitioning algorithms use clustering and are shown to be able to handle the sizes of circuits encountered today.

Current placement algorithms can be classified into three categories: simulated annealing, partitioning based, and analytical methods. Among these methods, analytical placement algorithms can achieve the best wire length results with the lowest runtime. In ISPD2005 and ISPD2006 placement contests, the analytical placers dominated other types of placers in terms of wire length reduction and runtime cost [57, 58]. However, it should be mentioned that, in practice, wire length minimization is only one of the performance metrics for placement algorithms. In commercial placement tools, there are other key concerns, such as the routability of the placement results. A placement solution that has a minimum of estimated wire length does not guarantee

that the routing wire length for this placement solution is also minimum [6].

Chapter 3

Existing Clustering Algorithms for Partitioning and Placement

3.1 Introduction

In this chapter, background information on the current clustering algorithms for circuit partitioning and placement is provided. These algorithms are classified into two categories: scoreless and score-based.

The rest of this chapter is organized as follows. In Section 3.2, clustering in the context of circuit partitioning and placement is introduced. In Section 3.3, scoreless clustering algorithms are discussed. In Section 3.4, score-based clustering algorithms are discussed. In Section 3.5, a comparison between scoreless and score-based clustering algorithms is given. In Section 3.6, the motivations for the proposed research in this thesis are presented. Finally in Section 3.7, a summary of this chapter is given.

3.2 Problem Definition

In the context of VLSI physical design, the clustering problem can be described as finding a group of subsets of cells in a netlist, such that certain constraints, such as the cluster lower and upper size bounds are satisfied. A formal definition of the clustering problem is as follows.

Given a hypergraph $H(V, E)$ which represents a netlist, where V is the set of cells

and E is the set of nets in the netlist, find a set of clusters $C_1, C_2, ..., C_k$, such that

$$\begin{aligned} k &\leq n_c \\ C_i &\neq \phi,\ 1 \leq i \leq k \\ C_i &\subset V,\ 1 \leq i \leq k \\ \cup_{i=1}^{k} C_i &= V \\ C_i \cap C_j &= \phi,\ 1 \leq i, j \leq k,\ i \neq j \\ size(C_i) &\leq U,\ 1 \leq i \leq k \\ size(C_i) &\geq L,\ 1 \leq i \leq k, \end{aligned}$$

where n_c is the number of cells in the netlist; i.e., $\|V\| = n_c$, $size(C_i)$ is the size of cluster C_i, and U and L are the upper and lower bounds for the cluster size. In the context of VLSI physical design, the size of cluster can be either the number of cells inside the cluster, or sum of areas of cells inside the cluster.

Depending on the specific circumstances, the optimization objective for clustering problems can be different. A typical objective can be the minimization of the number of nets that are cut by different clusters, or maximization of nets that are absorbed by clusters. For a cluster C_i, the set of nets that are cut by C_i is expressed as

$$E_{cut}(C_i) = \{e | e \in E,\ \exists \text{ cells } v_i \in e \text{ and } v_j \in e, v_i \in C_i \text{ and } v_j \notin C_i\}.$$

Then, the set of nets that are cut by all clusters can be expressed as

$$E_{cut}(C) = \cup_{i=1}^{k} E_{cut}(C_i).$$

Eventually, the objective of minimization of nets cut by clusters can be expressed as

$$Min.\ \ ||E_{cut}(C)||, \tag{3.1}$$

where $||E_{cut}(C)||$ denotes the number of nets in set $E_{cut}(C)$.

In [15], other objective functions for clustering problems are presented, such as scaled cost and absorption.

In the context of VLSI partitioning and placement, the clustering algorithms are mainly used to reduce the original large scale netlist into a smaller netlist. As illustrated in (3.1), the general objective of a clustering algorithm is to identify the groups of strongly interconnected cells and then make each group into a cluster, subject to the cluster size constraints. Clustering can be performed in multiple levels; i.e., the clustering can be recursively applied to not only the original netlist, but also an already clustered netlist. As a result of multilevel clustering, a hierarchy of netlists are constructed. Then the partitioning and placement are performed in this multilevel hierarchy, as discussed in Section 2.3.3 in Chapter 2.

It should be noted that in the multilevel clustering framework, the clustering process between continuous levels needs to be controlled appropriately. In practice, there is a predefined "clustering ratio" to determine how much the current netlist will be reduced in one single level of clustering. This clustering ratio is usually defined in terms of the number of cells in the netlist, and is expressed as

$$\text{Cell-Clustering-Ratio (CCR)} = \frac{\#\text{ cells in clustered netlist}}{\#\text{ cells in original netlist}}.$$

At each level of clustering, clusters are generated sequentially. Once a desirable clustering ratio has been reached, indicating that the netlist has been reduced small enough at the current level, the clustering stops and the clustered netlist is finalized. The main purpose of having a fixed clustering ratio is to control the clustering process. The value of the clustering ratio is dependent on the specific application circumstances. In some cases, the clustering ratio is set to be high to adequately smooth the clustering process. For example, in the multilevel partitioning process, the clustering ratio is usually set to be greater than 0.5; i.e., the size of next level clustered netlist is greater than half the size of the current level netlist [3, 20, 45, 47].

Experiments by other researchers have demonstrated that when using a higher clustering ratio, better partitioning results can be obtained [20]. However, in the context of multilevel placement, a smaller clustering ratio is generally desired. For example, in state-of-the-art placers, e.g., mPL6 [23] and APlace [43], the default clustering ratios are 0.25 and 0.1, respectively. The reason for choosing a small clustering ratio value for the placement problem is that compared to partitioning, placement is a more difficult problem and the process of solving placement requires more runtime and memory resources.

With the size and complexity of today's VLSI circuits, the task of circuit physical design is becoming more difficult than ever. One main challenge arising with the large circuits is that the performance of many existing algorithms deteriorates as the circuit size increases. As a result, the solution quality can also deteriorate significantly. At the same time, the runtime increases drastically. For example, the FM algorithm [34] is a standard partitioning method for circuits with small to medium size. For small circuits, FM can produce optimal partitioning results. However, when much larger circuits are encountered, e.g., circuits with more than 100,000 cells, the partitioning results by FM are generally 2 times worse than known upper bounds [12].

In order to tackle today's large scale designs, numerous clustering algorithms have been proposed in recent years. Due to their capability to reduce circuit size and complexity, clustering algorithms are receiving more attentions. Previous empirical results have shown that by using clustering, both runtime reduction and better solutions are obtained [11, 13, 22, 28, 39, 45]. Currently, clustering algorithms have been successfully applied in almost all aspects of the circuit physical design, for example, partitioning [20, 47, 62], placement [10, 22, 25, 59, 61, 72, 76], and routing [60].

Based on how clusters are formed, as discussed earlier, the clustering algorithms for circuit partitioning and placement can be classified into two categories: scoreless

and score-based methods. These two categories are discussed in detail below.

3.3 Scoreless Approaches

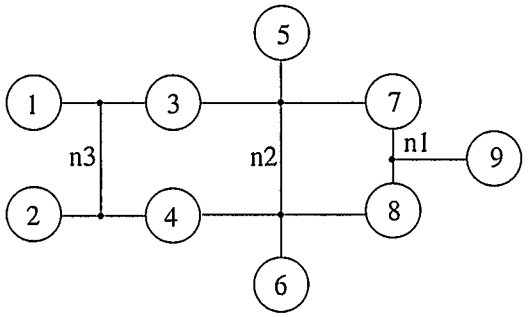
A clustering algorithm is defined as scoreless, if once a potential cluster is identified, it is finalized and no comparison between different potential clusters is made. The main scoreless clustering algorithms used for VLSI physical design are briefly discussed below.

3.3.1 Edge Coarsening (EC)

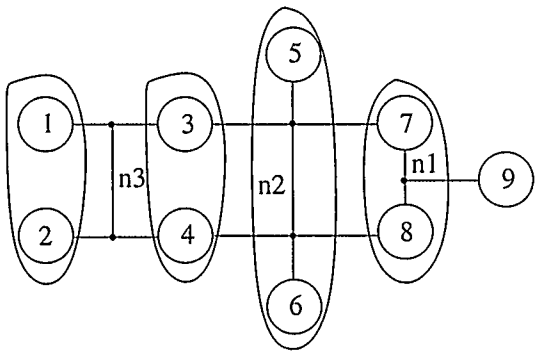
In [45], an edge coarsening algorithm is proposed. In edge coarsening, cells in a circuit, or netlist, are visited in a random order in order to form clusters. For each randomly selected cell, first the clustering status of this cell is examined. If the visited cell has already been clustered, it will be skipped and the next cell is examined. Otherwise, the connectivity between this "seed" cell and all of its unclustered neighbors is computed. The visited seed cell is clustered with the cell with the highest connectivity. Therefore, each time, edge coarsening clusters two cells and generates one new cluster.

Figure 3.1 shows an example where edge coarsening is used to cluster a simple circuit. In Figure 3.1 (a), the original netlist is shown, where all cells are unclustered cells. If the cells are visited according their numbers, cell ① will be the first visited cell. It has the same connections with cells ②, ③, ④ and can be grouped with any of the them to form a cluster. If cell ② is chosen and grouped with cell ①, the first cluster is produced. Then, cell ② is visited, since it is already a clustered cell, it is skipped. Cell ③ is visited and it will cluster with cell ④, since cell ③ has the maximum connectivity with cell ④. The similar cell visiting and clustering process is repeated until cells ⑦ and ⑧ are clustered, as shown in Figure 3.1 (b), or the predefined clustering ratio is reached. The clustered netlist is shown in Figure 3.1

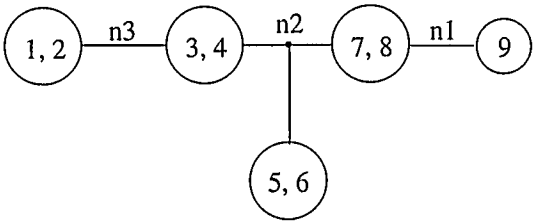
(c).



(a) Original netlist



(b) 4 clusters are formed in turn



(c) Clustered netlist

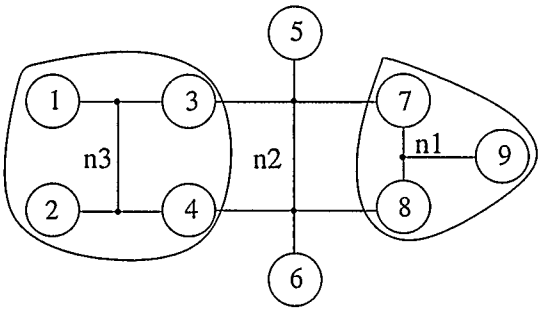
Figure 3.1: An example of the edge coarsening algorithm for a small circuit. Here, the cells are ordered as (①, ②, ③, ④, ⑤, ⑥, ⑦, ⑧ and ⑨).

It should be noted that the edge coarsening algorithm is the basic clustering technique used for VLSI physical design. Many other clustering algorithms are derived from edge coarsening algorithm, and have similar characteristics.

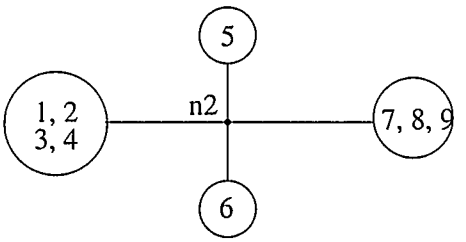
3.3.2 Hyperedge Coarsening (HEC)

In [45], a hyperedge coarsening algorithm is proposed. In hyperedge coarsening, clusters are formed by visiting the hyperedges or nets in the netlist. The nets in a circuit are first ordered ascendingly based on the net degree; i.e., the number of cells connected to the net. Then, each net is visited in turn. Similarly to the edge coarsening algorithm, first the clustering status of the visited net is examined: if the visited net contains any clustered cell, it will be skipped and the next net is visited. Otherwise, all of the cells in this net are grouped and a new cluster is produced. This process is repeated until all the nets have been visited and handled appropriately, or a predefined clustering ratio has been reached. From the above description, it can be seen that, unlike edge coarsening that has exactly two cells in one cluster, in hyperedge coarsening, each time one hyperedge is clustered, the number of cells in a cluster is arbitrary and only dependent on the net degree.

In Figure 3.2, an example where hyperedge coarsening is used to cluster the circuit in Figure 3.1 (a) is given. Originally all the cells and nets are unclustered. The nets in the circuit are visited based on their degrees: $n1$, $n3$ and $n2$. Suppose the sum of areas of the cells in net $n1$ does not exceed the cluster size upper bound, then, a new cluster is generated which consists of cells ⑦, ⑧ and ⑨, as shown in Figure 3.2 (a). Next, net $n3$ connecting cells ①, ②, ③ and ④ is examined. No cell in this net belongs to a cluster. If the sum of areas of the cells in this net does not exceed the cluster size upper bound, then another new cluster is generated which consists of cells ①, ②, ③ and ④, as shown in Figure 3.2 (a). Finally, net $n2$ is visited. Since the cells ③, ④, ⑦ and ⑧ in net $n2$ are already clustered, net $n2$ is skipped. The final clustered netlist is shown in Figure 3.2 (b).



(a) All identified clusters



(b) Clustered netlist

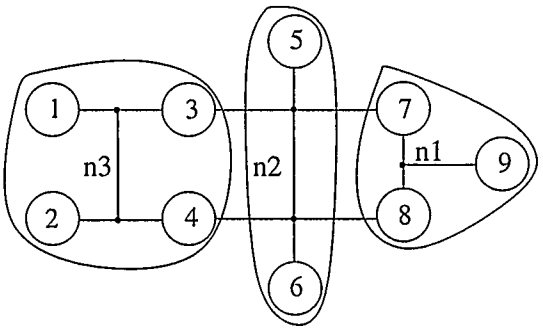
Figure 3.2: An example of the hyperedge coarsening algorithm for a small circuit. Here, the nets are ordered as ($n1$, $n3$ and $n2$).

3.3.3 Modified Hyperedge Coarsening (MHEC)

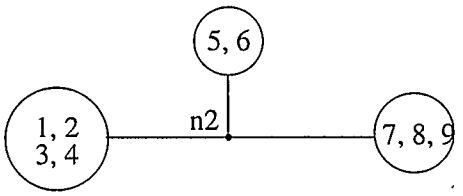
In [45], a modified hyperedge coarsening algorithm is proposed. Modified hyperedge coarsening is based on the same procedure as hyperedge coarsening, with the difference that hyperedges containing clustered cells can still be considered for further clustering. In other words, modified hyperedge coarsening first uses hyperedge coarsening procedure to cluster the netlist. Then, the nets are visited again, and those unclustered nets that were left in the hyperedge coarsening process due to containing clustered cells are, themselves, clustered. During the process, if clustering a net or partial cells of a net does not satisfy the cluster size constraint, no cluster is produced. Furthermore, if the specified clustering ratio is reached, the clustering stops immediately.

In Figure 3.3, an example is shown where modified hyperedge coarsening is used

to cluster the same circuit as in Figures 3.1 and 3.2. For the purpose of simplifica-



(a) All identified clusters



(b) Clustered netlist

Figure 3.3: An example of the modified hyperedge coarsening algorithm for a small circuit. Here, the nets are ordered as ($n1$, $n3$ and $n2$).

tion, the hyperedge coarsening procedure is omitted and the resulting netlist after hyperedge coarsening is directly used in this example. Then, the modified hyperedge coarsening algorithm examines each unclustered net. In Figure 3.3 (a), cells ⑤ and ⑥ are unclustered cells in net $n2$. Therefore, a new cluster is generated by grouping cells ⑤ and ⑥. In Figure 3.3 (b), the final clustered netlist is shown. In modified hyperedge coarsening, the number of cells in a cluster is also arbitrary.

3.3.4 FirstChoice Clustering (FC)

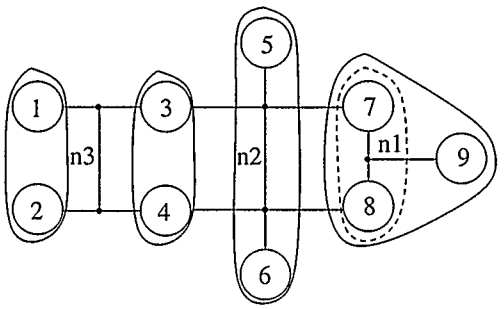
In [47], the FirstChoice clustering algorithm is proposed. In FirstChoice, similar to edge coarsening, cells are visited randomly, and each visited seed cell, v_i, is clustered with the neighbor, v_j, that has the maximum connectivity with v_i. Here the

connectivity, *conn*, between two cells v_i and v_j is calculated as follows:

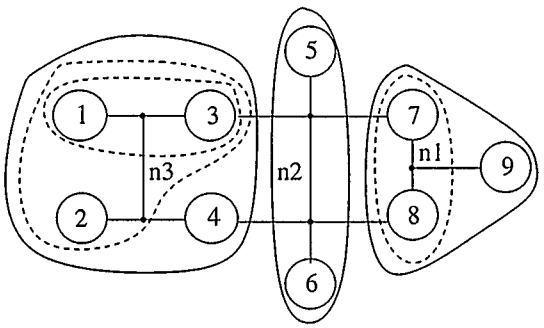
$$conn(v_i, v_j) = \sum_{e \in \{e | v_i \in e, v_j \in e\}} \frac{1}{|e| - 1},$$

where $e \in E$, and is the net connecting cells v_i and v_j, $|e|$ is the degree of net e. In FirstChoice, when a cell is visited, the connectivities between the visited seed cell and all of its neighbor cells are calculated and the visited cell is clustered with the neighbor that has the maximum connectivity regardless of the neighbor's clustering status. However, a seed cell can not be a clustered cell. This clustering is different from edge coarsening, where only the unclustered cells can be used to generate a cluster. As a result, when FirstChoice groups a visited cell and its selected neighbor, if the neighbor is a clustered cell already, then no new cluster is produced and the visited cell is added into an existing cluster.

In Figure 3.4, an example where FirstChoice is used to cluster the circuit in Figure 3.1 (a) is given. Originally, all cells are unclustered and are visited randomly. Suppose, in this example, the cells are visited in the order of their cell numbers. Cell ① has the same connectivity with cells ②, ③ and ④ and is randomly clustered with cell ②. The next seed cell is ③ which has the highest connectivity with cell ④. Seed cell ⑤ is randomly matched with ⑥ and seed cell ⑦ has the highest connectivity with ⑧. Finally, cell ⑨ is visited. Although both neighbors of cell ⑨, i.e., cells ⑦ and ⑧, are clustered already, FirstChoice still calculates the connectivity between cells ⑨ and ⑦, and cells ⑨ and ⑧. As a result, cell ⑨ is added into the existing cluster consisting of cells ⑦ and ⑧, as illustrated in Figure 3.4 (a). In Figure 3.4 (b), another clustering result using the same cell ordering is illustrated. As it can be seen in this example, the cluster sizes are varied.



(a) FC clustering solution 1



(b) FC clustering solution 2

Figure 3.4: An example of different clustering results by FirstChoice algorithm using the same cell ordering on a small circuit. Here, the cells are ordered as (①, ②, ③, ④, ⑤, ⑥, ⑦, ⑧ and ⑨).

3.3.5 Heavy-Edge Matching (HEM)

In [13], a variation of edge coarsening algorithm, called heavy-edge matching, is proposed. The principle procedures of this algorithm are similar to edge coarsening, but the connectivity definition and computation are different.

In heavy-edge matching [13], a connectivity measure for a pair of cells is proposed which considers both the areas of the cells and the degrees of the edges that connect them. Specifically, the connectivity, *conn*, between two cells v_i and v_j, is calculated as follows:

$$conn(v_i, v_j) = \frac{1}{A(v_i) \cdot A(v_j)} \sum_{e \in \{e | v_i \in e, v_j \in e\}} \frac{1}{|e|}, \tag{3.2}$$

where $A(v_i)$ represents the area of cell v_i, e represents the net that connects v_i and v_j, and $|e|$ represents the number of cells in net e, i.e., the degree of the net. This

connectivity measure emphasizes nets with lower degrees; i.e., nets with a low number of incident cells. It also gives priority to the cells with small area to prevent forming big clusters. Similar to edge coarsening, all the cells in a netlist are visited randomly. When an unclustered cell is visited, it will be clustered with one of its unclustered neighbors such that the connectivity between them is highest among all unclustered candidate neighbors.

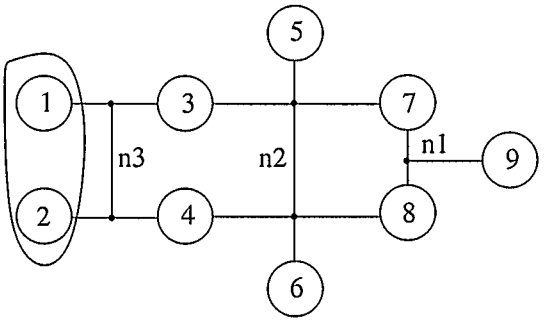
3.3.6 PinEC Clustering (PinEC)

In [20], a variation of the heavy-edge matching algorithm, called PinEC, is proposed.

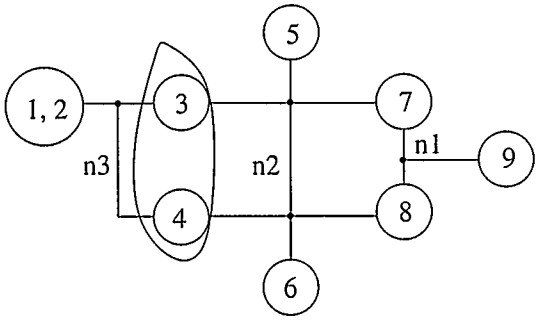
In PinEC, the first modification to the heavy-edge matching connectivity is that the weight of hyperedges with degree equal to two is doubled to give higher priority to these nets. Second, in PinEC, the netlist is dynamically updated after the formation of each cluster. Subsequent connectivity calculations and clustering operations are aware of previously produced clusters.

In Figure 3.5, an example where PinEC is used to cluster the circuit in Figure 3.1 (a) is given. In this example, the cells are visited in the same order as edge coarsening; i.e., ①, ②, ③, ④, ⑤, ⑥, ⑦, ⑧ and ⑨. Cell ① is visited first and cells ① and ② are clustered, as illustrated in Figure 3.5 (a). Once the cluster is generated, cells ① and ② are merged into a new cell, whose area is equal to the sum of cells ① and ②, and the nets incident to this cell is the set of nets originally incident to cells ① and ②, as illustrated in Figure 3.5 (b). In the remainder of the clustering process, the new cell merged from ① and ② is treated as a normal cell with the rest of the cells in the netlist. Figure 3.5 (c) shows the clustered netlist. For this small example, the netlist updating and adding cell areas to the connectivity computation prevent formations of clusters such as in the example shown in Figure 3.4 (b).

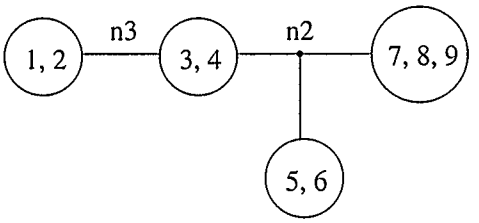
Although in heavy-edge matching and PinEC, the connectivity, which can be regarded as a clustering score, is calculated, these algorithms are put into the scoreless



(a) First cluster



(b) Second cluster after netlist update



(c) Clustered netlist

Figure 3.5: An example of the PinEC algorithm for a small circuit. Here, the cells are ordered as (①, ②, ③, ④, ⑤, ⑥, ⑦, ⑧ and ⑨).

category since the score is not used in ordering the cells.

3.3.7 Application of Scoreless Clustering Algorithms

The scoreless clustering algorithms discussed above have been widely used in circuit partitioning and placement.

In state-of-the-art circuit partitioner, hMetis [1, 45, 47], edge coarsening, hyperedge coarsening, modified hyperedge coarsening, and FirstChoice have been implemented. In another leading-edge circuit partitioner, MLPart [3, 20], heavy-edge matching and PinEC have been implemented.

In the context of placement, most of these scoreless clustering algorithms are indirectly used in partitioning based placement tools, such as Capo [3] and FengShui [10]. Both of these two placers invoke either hMetis or MLPart to perform bisecting for placement. Furthermore, FirstChoice is directly used to construct the multilevel placement hierarchy in some other placement tools, such as mPL5 [22], NTUplace2 [25], NTUplace3 [26], and FDP [48, 76]. In these placers, like the multilevel partitioning scheme, the clustering algorithms are used to cluster the original netlist into a series of smaller netlists. An initial placement is performed on the smallest netlist in the hierarchy of netlists. This initial placement is further refined when the series of clustered netlists is mapped back to its upper level netlist, until the original netlist is recovered.

In Table 3.1, the application of scoreless clustering algorithms on partitioning and placement is summarized. It should be noted that among all these scoreless clustering algorithms, FirstChoice has been shown to be the most used scoreless clustering technique in practice.

Table 3.1: A summary of scoreless clustering algorithm applications in partitioning and placement tools

Clustering Methods	Tools Using Clustering Methods	
	Partitioners	Placers
Edge coarsening	hMetis	
Hyperedge coarsening	hMetis	
Modified hyperedge coarsening	hMetis	
FirstChoice	hMetis	Capo, FengShui, mPL5, FDP, NTUplace2/3
Heavy-edge matching	MLPart	
PinEC	MLPart	

3.4 Score-Based Approaches

A clustering algorithm can be said to be a score-based approach, if a set of potential clusters are first identified, with a score assigned to each potential cluster, which indicates the priority or quality of that cluster among all potential clusters. Then, all potential clusters are ordered based on their associated clustering scores, and clustered in turn until a desired clustering ratio is reached. The following clustering algorithms are all score-based approaches.

3.4.1 Edge Separability-Based Clustering (ESC)

In [28, 29], clustering is performed based on *edge separability*, which is essentially another measurement of connectivity between a pair of cells in a netlist. In this technique, first the netlist is converted to its corresponding graph. Then, all the cells in the graph are visited in turn and the maximum flow between each visited cell and its adjacent cells is computed. Subsequently, different pairs of cells are ordered descendingly based on their maximum flow values; the pair of cells with the highest flow will be clustered first, and the pair of cells with the second highest flow will be clustered next. This process is repeated until a predefined clustering ratio is achieved. The edge separability clustering algorithm provides global connectivity information to guide the clustering process, since it computes the flows for all pairs of cells and

orders them globally.

3.4.2 Fine Granularity Clustering (FGC)

In [39], a Fine Granularity Clustering (FGC) algorithm is presented. Similar to edge separability, this algorithm first converts a given netlist into its corresponding graph model. This graph is initially clustered using a greedy seed cell growing algorithm: The cells in the netlist are visited randomly and for each visited cell, if it already belongs to a cluster, it is skipped; otherwise, the seed cell attracts its neighbor with the highest connectivity to the cluster. This process is repeated until the cluster size constraint is reached. After all the cells are visited as seed cells, the initially formed clusters are further refined by an adapted FM algorithm [34], to improve the quality of clusters iteratively by moving cells between different clusters. This cell movement process for refining clusters is like an iterative refinement for multi-way partitioning [47]. The refined clusters by FGC algorithm typically contain a few, 2 to 6, cells in each cluster.

The edge separability and FGC algorithms both make clusters based on a global comparison of the possible clusters. However, they both transform a given hypergraph into a graph by replacing each hyperedge with degree greater than 2 to a clique model. In some cases this transformation can result in incorrect edge weights, as illustrated in [11].

3.4.3 Best Choice Clustering (BC)

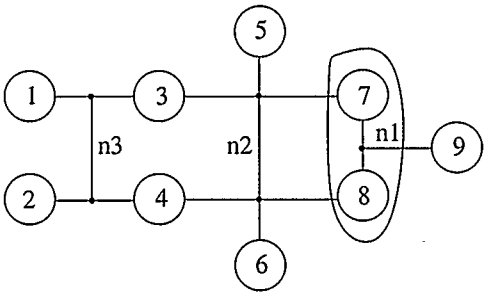
In [11, 59], a clustering algorithm called "best choice" is presented. This technique consists of two phases. In the first phase, the cells are visited randomly. For each visited cell, the connectivity between this cell and each of its neighbors is calculated.

$$conn(v_i, v_j) = \frac{1}{A(v_i) + A(v_j)} \sum_{e \in \{e | v_i \in e, v_j \in e\}} \frac{1}{|e|}.$$

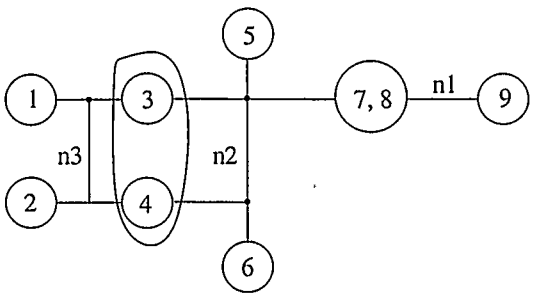
This connectivity computation is similar to the equation (3.2). Once the connectivities for a visited cell and its neighbors are available, the neighbor that has the maximum connectivity with the visited cell is chosen, and a pair which contains the numbers for the selected neighbor and the visited cell is created and inserted into a priority queue, based on the value of the connectivity for this pair of cells. In best choice, the connectivity value for the pair of cells is referred to as the "clustering score". The above process is continued for each cell in the netlist. At the end of this phase, the priority queue will have exactly n_c items inside, where n_c is the number of cells in the netlist. The priority of these items are determined by their associated clustering scores. In the second phase, the pair of cells at the top of priority queue is made into a new cluster, and the netlist is dynamically updated. As a result of netlist updates, the connectivities between the new cell, made of the cluster, and its neighbors can be changed. Therefore, the clustering score of the new cell and the cells connected to it are re-calculated and their positions are updated in the priority queue. The above process is continued until a predefined clustering ratio has been reached.

In Figure 3.6, an example where best choice is used to cluster the circuit in Figure 3.1 (a) is given. In this example, first the clustering score for each cell and its best match neighbor is calculated and inserted into a priority queue. It turns out that the pair of cells consisting of ⑦ and ⑧ has the highest score, 0.25. Cells ⑦ and ⑧ are grouped as the first cluster, as illustrated in Figure 3.6 (a). Then, both the netlist and the clustering scores are updated. As a result, the pair of cells made of ③ and ④ has the highest score of 0.225. Therefore, a new cluster is made by grouping cells ③ and ④, as illustrated in Figure 3.6 (b). This process is continued and in Figure 3.6 (c), the clustered netlist is shown.

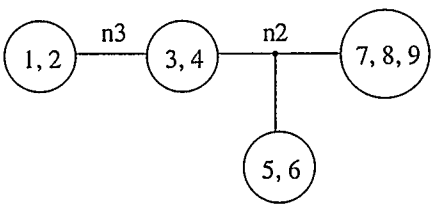
Best choice combines the advantages of the previous clustering techniques by utilizing the connectivity computation in the heavy-edge matching algorithm. Also,



(a) First cluster, clustering score is 0.25



(b) Second cluster, clustering score is 0.225



(c) Clustered netlist

Figure 3.6: An example of the best choice algorithm for a small circuit. (a) The first cluster is made of cells ⑦ and ⑧. The clustering score is 0.25. (b) After netlist and clustering score update, the second cluster is made of cells ③ and ④. The clustering score is 0.225. (c) Final clustered netlist.

a clustering score is assigned to each potential cluster when it is identified. These clustering scores are used to guide the clustering sequence. As a result, at each time, the best pair of cells are guaranteed to be clustered [11]. Finally, in order to speed up the whole algorithm process, a lazy-update technique is proposed. The idea of this technique is to skip the clustering score updating for the neighbors of a newly created cluster cell. The empirical results show that by using this lazy-update technique, the overall runtime for clustering process is reduced by around 50%, with almost no loss of cluster quality [11, 59].

3.4.4 Application of Score-Based Clustering Algorithms

The score-based clustering algorithms have been extensively used in circuit partitioning and placement.

In [28, 29], edge separability has been used to construct a multilevel circuit partitioner LR/ESC-PM. This edge separability based partitioner can provide comparable results to hMetis. However, the computation cost and runtime for this partitioner is higher compared to hMetis. The main reason is that the edge separability technique has a longer runtime than the FirstChoice algorithm used in hMetis.

In [39], the FGC algorithm has been used as a preprocessing step for partitioning based placer Capo. The experimental results show that the overall placement runtime is decreased by about 3 to 5 times, with comparable or even better placement solution quality. Furthermore, in [40, 41], FGC is used in a quadratic analytical placer mFAR to construct the design hierarchy.[1]

The best choice algorithm is the most used score-based clustering technique for circuit placement. It has been implemented in many state-of-the-art placement tools, such as hATP [59], mPL6 [23] and APlace2/APlace3 [43, 44]. The empirical results have verified the effectiveness of best choice algorithm. For example, compared to

[1] mFAR won the second place in ACM ISPD2005 placement contest.

the flat ATP placer, best choice based hierarchy placer hATP is 2.1 times faster and has a 1.4% improvement on the placement half perimeter wire length. Among almost all of the well-known academic placement tools, APlace2 and mPL6 ranked first and second place in ISPD2005 and ISPD2006 placement contests, respectively [57, 58].

In Table 3.2, the application of score-based clustering algorithms on partitioning and placement is summarized. It clearly shows that best choice is the favorite technique for many placers.

Table 3.2: A summary of score-based clustering algorithm applications in partitioning and placement tools

Clustering Methods	Tools Using Clustering Methods	
	Partitioners	Placers
Edge separability Fine granularity clustering Best choice	LR/ESC-PM	Capo, mFAR APlace 2.0/3.0, mPL6, hATP, FastPlace3.0

3.5 Comparison between Scoreless and Score-Based Approaches

Currently, both scoreless and score-based clustering algorithms are extensively used for circuit partitioning and placement. Compared to score-based algorithms, scoreless clustering algorithms are relatively easier to implement, and have lower runtime. However, a common drawback of these algorithms is their random behavior, which can cause different clustering results to be produced in different runs. Furthermore, compared to the score-based algorithms, the clustering results by scoreless algorithms are usually suboptimal.

In [11, 15], it is demonstrated that the clustering solution quality can be greatly improved if the cells are "properly" ordered. Score-based clustering algorithms can be regarded as an improved version of scoreless approaches with the score manipulation.

Due to the introduction of clustering scores, clustering decisions can be made based on a comparison of cell connectivity and cluster areas, and consequently the cluster quality can be further improved [11]. However, the clustering score computation and manipulation require increased runtime for score-based clustering techniques than for scoreless algorithms.

3.6 Research Motivations

In Table 3.3, the various clustering algorithms and their applications on circuit partitioning and placement are summarized.[2] Currently, almost all state-of-the-art par-

Table 3.3: A summary of clustering algorithm applications in partitioning and placement tools

Academic	Clustering techniques								
	Scoreless						Score-based		
tools	EC	HEC	MHEC	FC	HEM	PinEC	ESC	FGC	BC
hMetis	×	×	×	×					
MLPart					×	×			
LR/ESC-PM							×		
Capo	×	×	×	×				×	
FengShui	×	×	×	×					
mPL5				×					
FDP				×					
NTUplace2/3				×					
mFAR								×	
APlace2.0/3.0									×
mPL6									×
hATP									×
FastPlace3.0									×
Kraftwerk									

titioning and placement tools use clustering techniques. Although these successful clustering applications on circuit layout design have been demonstrated, there is still considerable improvement room for current clustering algorithms. Generally, current clustering algorithms for circuit partitioning and placement problems suffer the

[2]To the author's knowledge, most of the current partitioners and placers are summarized in Table 3.3.

disadvantages discussed below.

First, the solutions produced by current clustering algorithms are not always optimal. Among the different clustering techniques, edge clustering algorithms, including edge coarsening and its numerous variations, such as FirstChoice and best choice heuristics, are the most popular algorithms in practice, due to their high runtime efficiency. This can also be observed in Table 3.3. However, the trade-off for the lower runtime is the suboptimality of solution quality. Essentially, edge clustering algorithms compute connectivity and form clusters in a "pair wise" manner. On one hand, this pair wise scheme is very fast and easy to implement. On the other hand, the view of the cell connectivity is limited, and the clustering decision is made locally and in a greedy manner. As a result, the optimal clustering results may not be obtained.

An analysis of current edge clustering algorithms from two points of view: cell connectivity and force-directed model, reveals the suboptimality of these algorithms.

- Cell connectivity example

Current edge clustering algorithms cluster pairs of cells that are highly interconnected, but if "natural" clusters made of more than 2 cells in a netlist exist, the current edge clustering algorithms produce suboptimal results. Here, a "natural" cluster refers to a group of cells that are strongly connected and should be identified and clustered as a whole. In Figure 3.7, a netlist containing two natural clusters, made of cells (①, ②, ③, ④), and (⑤, ⑥, ⑦, ⑧), is shown. However, if cell ② is visited first, assuming that all cells in Figure 3.7 have equal area, an edge cluster based algorithm groups cells ② and ⑤ together, eliminating the two natural clusters. For score-based edge clustering algorithms, such as best choice, the pair of cells ② and ⑤ end up having the highest clustering score and therefore are clustered first, thus, also not identifying the natural clusters. In summary, in either case, the optimal

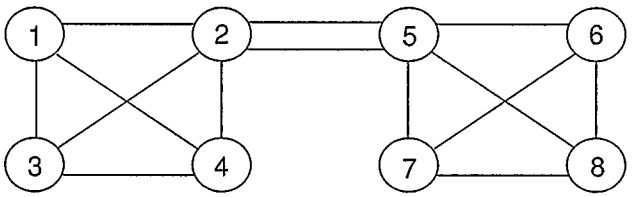


Figure 3.7: Example of natural clusters in a circuit

clustering is not obtained.

- Force-directed model example

The suboptimality of current edge clustering algorithms can also be illustrated in the context of force-directed placement. As it has been discussed in Section 2.4.3, force-directed models have been widely used in many state-of-the-art placement tools, such as Kraftwerk, FastPlace, mFAR and hATP. An essential reason for the great success of force-directed placers is that all nets that exert forces on a cell are considered simultaneously; therefore, the final solution is based on a global view of the netlist connectivity information. This mechanism is totally different from the pair wise method that is used by edge clustering algorithms. It can be claimed that, in essence, current edge clustering algorithms are not consistent with the force-directed model.

In Figure 3.8, a simple circuit is shown. In this circuit, cell ③ has 5 incident nets. Among these nets, the 2 nets connecting cells ③ and ⑤ will try to pull cell ③ towards cell ⑤, and the other 3 nets will apply forces in the opposite direction. If each net in the circuit is assumed to have the same force, then cell ③ will be pulled leftward a little more, even though locally it looks like cell ③ and cell ⑤ have more connections and should be pulled closer to each other.

For the circuit in Figure 3.8, current edge clustering algorithms, such as FirstChoice or best choice, which are the two most commonly used edge clustering techniques, cluster cells ③ and ⑤ together. This simple example clearly shows that due to the decision made based on a local and greedy pair wise manner, edge clustering approaches

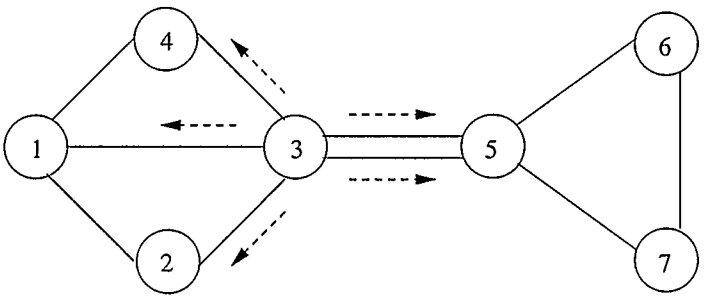


Figure 3.8: The force directed model for clustering

may produce suboptimal results given certain circuit structures. But in reality, the cells in a netlist interact with each other through nets and the connectivity should be considered as a whole, instead of with pairs of cells. This inconsistency with the force-directed model accounts for the suboptimality of current edge clustering algorithms [52].

Another disadvantage of current clustering algorithms is that most of them are capable of reducing the circuit sizes, but tend to focus on clustering "cells" rather than "nets" in a netlist [53]. As a result, compared to the original netlist, the clustered netlist has a much lower cell clustering ratio versus net clustering ratio; i.e., in a clustered netlist originating from a large design, the number of nets is much higher than the number of cells. However, in practice, as an indication of the extent of a netlist structure's changes, a smaller discrepancy between cell and net clustering ratios, or a smaller net clustering ratio[3], is more desirable. For example, it has been demonstrated that in hierarchical FPGA design, a clustering algorithm which produces a lower net clustering ratio can also improve the routability, area, and power consumption of the designed device [70].

Overall, the main motivations of this research work are to improve the performance of current clustering algorithms, and consequently enhance the performance

[3]Since all clustering algorithms produce larger net clustering ratio than cell clustering ratio, in order to reduce the gap between cell and net clustering ratio, the net clustering ratio should be minimized as much as possible.

of present leading-edge partitioning and placement tools in VLSI physical design, since clustering is extensively used in these tools. Based on the analysis of previous clustering algorithms, a series of new clustering algorithms is proposed in this thesis. Basically, this series of clustering algorithms have the following characteristics:

- The proposed clustering algorithms use a novel cell connectivity formulation, which is consistent with the force-directed model, to identify a set of potential high quality clusters. Specifically, during the potential cluster identification process, instead of focusing on only the connections between pairs of cells, all involved connections are considered as a whole, and furthermore, instead of focusing on only the immediate, or first level, connections, second and higher level connections are also considered. Once all potential clusters have been identified, these clusters are globally compared with one another to decide their priority and are clustered in turn. The experimental results demonstrate that better clustering results can be obtained by the proposed clustering algorithms.

- One of the proposed clustering algorithms has an explicit objective of minimizing the net clustering ratio, so as to reduce the discrepancy gap between the cell clustering ratio and net clustering ratio. This objective can be achieved by carefully selecting a set of nets, and clustering these nets directly instead of cells after all potential clusters have been identified, thus the proposed clustering algorithm is a "net" clustering method. This net clustering mechanism differentiates the proposed method from most existing clustering techniques. As it will be illustrated in subsequent sections, this net clustering scheme is also a good method to resolve the cell overlapping problem between different potential clusters.

- The main ideas of the proposed clustering algorithms, i.e., global cluster identification and selection, can be used in other existing clustering algorithms to

improve their performance. For example, the proposed connectivity formulation can be adapted in edge clustering techniques to obtain a global connectivity computation, which may eventually lead to improved clustering results.

3.7 Summary

Clustering algorithms play an important role in current large scale circuit partitioning and placement tools. This chapter reviews the existing clustering algorithms for circuit partitioning and placement in two categories: scoreless and score-based methods. Scoreless clustering algorithms are usually fast, but the clustering results are random. On the contrary, score-based clustering algorithms require a longer runtime than scoreless methods, but the clustering results are deterministic and have improved quality. Since, in practice, especially for large scale circuit designs, clustering occupies only a small percentage of the whole process[4], it is more desirable to spend more runtime and implement score-based clustering algorithms in partitioning and placement tools. The research motivation in this thesis is to further improve current clustering algorithms, and consequently enhance the state-of-the-art status of circuit partitioning and placement tools.

[4]The runtime difference of scoreless and score-based clustering in terms of the percentage of the whole partitioning and placement process can be negligible.

Chapter 4

Algorithms for Single Cluster Identification

4.1 Introduction

Recently there has been a considerable amount of interest in clustering algorithms and they have been successfully used in many VLSI physical design problems. However, there is still improvement room for current algorithms. One of the main drawbacks of the current clustering algorithms is that they are greedy in nature. In these algorithms, the clustering decisions are made based on connectivity of pairs of cells versus groups of cells. For example, most edge clustering based algorithms, such as FirstChoice, consider the cell connectivity of a seed cell and its immediate neighbors and cluster the pair with highest connectivity. This pairwise clustering manner can result in suboptimal results [52]. There are algorithms that trade runtime for solution quality. For example, edge separability clustering – ESC [28, 29], can produce good clustering solutions by computing the maximum flows between pairs of cells and clustering the pairs with highest flows. But, the computation of maximum flows for all pair of cells is very time consuming. However, with the increase in sizes of circuits, these trade offs can affect the performance of the circuit.

In this chapter, one of the contributions of this thesis, clustering algorithms for single cluster identification, are proposed. The development of these algorithms are targeted to identify high quality clusters efficiently. The algorithms try to find clusters by taking a considerable amount of connectivity information into account when making the clustering decisions, therefore, obtaining better clustering results. Furthermore, the clustering algorithms are implemented based on a modified FM algorithm, which has a linear time complexity, and, therefore, are efficient. In order to be

able to measure the quality, a new concept called "gain" cluster has been proposed. The formal definition of a gain cluster is given in Section 4.2. Three techniques that can identify gain clusters efficiently, a scoreless algorithm, a seed cell based and a seed net based algorithms, are also proposed.

The remainder of this chapter is organized as follows: In Section 4.2, gain cluster definition is given. In Section 4.3, a scoreless gain clustering algorithm is proposed in details, along with a theory proving its complexity. In Section 4.4, two score-based gain clustering algorithms along with their complexity analysis are given. In Section 4.5, the numerical results on the ISPD98 benchmark suite [5, 12] are reported. Finally in Section 4.6, a summary of this chapter is made.

4.2 Gain Cluster Definition and Terms

In the context of VLSI physical design, a high quality cluster can be described as a group of cells that are tightly connected with each other and loosely connected with the other neighboring cells. Since this definition does not result in identification of clusters, in this thesis, a certain type of high quality clusters, referred to as gain clusters, are defined. The formal definition of gain cluster is as follows:

Definition 1 (Gain Cluster): Given a netlist $H(V, E)$, which consists of a set of cells, $V = \{c_1, c_2, ..., c_{n_c}\}$, where n_c is the total number of cells in the netlist, and a set of nets, $E = \{e_1, e_2, ..., e_{n_n}\}$, where n_n is the total number of nets in the netlist, a *gain cluster*, $C_{gain} = \{c_i, ..., c_j\}$, is a subset of V with the following characteristics:

1. C_{gain} is **not** an empty set.

2. For each cell in the cluster, the gain in terms of netcut decrease to move the cell out of the cluster is less than or equal to zero:

$$\forall c_i \in C_{gain}, \quad gain_{out}(c_i) \leq 0. \tag{4.1}$$

Here, $gain_{out}$ is the gain in terms of netcut decrease to move a cell **out** of a cluster.

3. For each cell outside the cluster, the gain in terms of netcut decrease to move the cell into the cluster is less than zero:

$$\forall c_j \notin C_{gain}, \quad gain_{in}(c_j) \; < \; 0. \tag{4.2}$$

Here, $gain_{in}$ is the gain in terms of netcut decrease to move a cell **into** a cluster.

The first characteristic in the definition is to ensure that a cluster is not an empty set. The second characteristic described in (4.1) is to ensure that the "attraction" from the cells outside the cluster is no more than the attraction from the cells inside the cluster. The third characteristic, (4.2), is to ensure that there are no cells outside the cluster that have greater attraction with those inside the cluster and should be moved. It should be mentioned that a gain cluster does not guarantee that the number of nets inside a cluster is more than the number of nets cut by the cluster. It simply states that the attraction forces on the cells inside the cluster must be greater than or equal to the forces outside the cluster.

The following definitions are also used for the remainder of this thesis to describe the algorithms.

***Definition 2** (Seed Cell):* A seed cell is any unclustered cell that can be used to form an initial cluster.

***Definition 3** (Seed Net):* A seed net is a net that can be used to form an initial cluster.

***Definition 4** (Accessory Cell):* An accessory cell is any non-seed cell that already belongs to a cluster.

***Definition 5** (Origin):* A partition used to accommodate the netlist at the beginning of the cluster identification procedure.

Definition 6 (InitCluster): A partition that can accommodate an initial cluster.

Definition 7 (NeighborCluster): A partition that can accommodate the neighbor cells of an initial cluster.

Definition 8 (The First Level Neighbors of A Seed Cell): For a seed cell, the first level neighbors are the cells that are connected directly to the seed cell.

Definition 9 (The Second Level Neighbors of A Seed Cell): For a seed cell, the second level neighbors are the cells that are connected to the first level neighbors of the seed cell, excluding the seed cell itself.

Definition 10 (The n^{th} Level, $n > 2$, Neighbors of A Seed Cell): For a seed cell, the n^{th} level neighbors are the cells that are connected to the $(n-1)^{th}$ level neighbors of the seed cell, but do not belong to the $(n-2)^{th}$ level neighbors.

In Figure 4.1, a graph schematic is shown, where the first, second, and third level neighbors of a seed cell are illustrated respectively. The level number for neighbors of

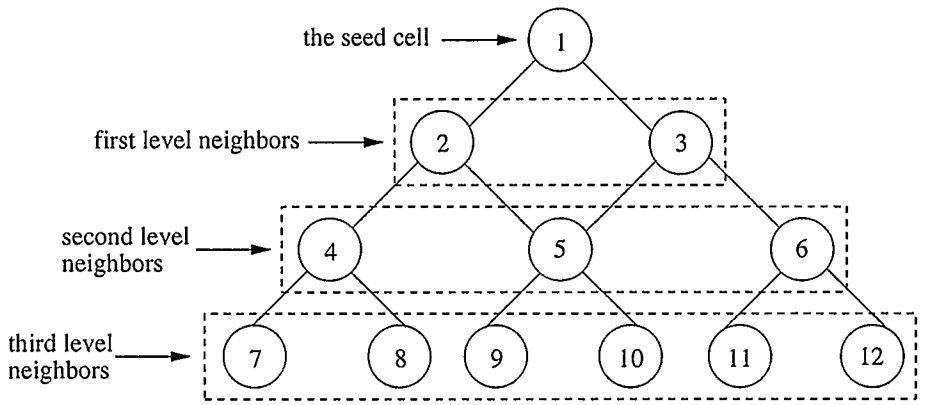


Figure 4.1: Illustration of different level neighbors for a seed cell

a seed cell indicates the distance from the seed cell to the neighbors; if the circuit is regarded as a tree, with the seed cell as the tree root, as illustrated in Figure 4.1, then the n^{th} level neighbors are the cells that are located at the n^{th} depth level of the tree; i.e., all cells at that level have a depth value of n. For a given tree, a breadth-first search algorithm can be used to efficiently search different depth levels of nodes; i.e.,

different levels of neighbors of the seed cell.

4.3 Scoreless Gain Cluster Identification

In this section, a scoreless gain clustering algorithm is proposed. In this algorithm, unlike the other scoreless algorithms, cells are ordered based on their characteristics. This ordering is described in Section 5.2.1. Once the cells are ordered, each cell is visited in turn. Depending on the clustering status of the visited cell, it can be either a *Seed Cell* or an *Accessory Cell.* If the visited cell is an accessory cell, it is skipped; otherwise, it is considered as a seed cell and a single gain cluster can potentially be identified.

4.3.1 Scoreless Single Gain Cluster Identification

Initial Cluster Creation

For a seed cell, first an initial cluster is created. The created initial cluster, referred to as partition "InitCluster", is made by grouping the seed cell and its different level unclustered neighbors. In this scoreless gain clustering algorithm, by default, the created initial cluster is made of a seed cell and its first level neighbors. In our implementation, this strategy can produce good clustering results efficiently. However, for some circuits, this default mode cannot cluster original netlists to low clustering ratios. In this case, including the second level of neighbors, or even higher level of neighbors, can reduce the cell clustering ratio to the desired value. In addition, for some circuits, including higher level neighbors as well as the first level neighbors benefits the final clustering solution quality. This is motivated by the fact that different circuits can have different structures. For a circuit with unknown inner structure, different initial cluster creation strategies might be tried in order to get a better clustering solution. Therefore, if multiple runs of clustering are allowed, it is suggested

to try different initial cluster creations so as to obtain better clustering results. In Figure 4.2, a simple example is shown where including higher level neighbor cells improves the cluster quality. In this figure, If PAD 1 is selected to be the seed cell,

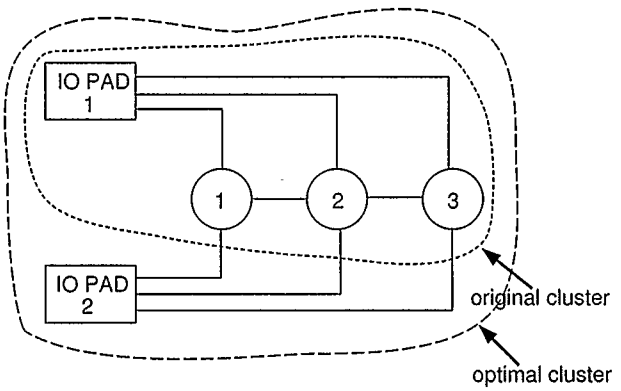


Figure 4.2: Example of including the second level neighbor cells into initial cluster

cells ①, ②, and ③ are the first level neighbors of PAD 1, and PAD 2 is the second level neighbor. Including PAD 2 in the cluster reduces the number of connections between the cluster and the rest of the circuit by 3 nets. The drawback of including higher level neighbors is the runtime increase.

It should be noted that during the initial cluster creation process, a cluster area constraint is in effect. If adding a cell into the initial cluster results in a cluster area violation, this cell cannot be put into the initial cluster. The main purpose of the cluster area constraint is to control the cluster size so as to produce clusters with balanced sizes. It has been demonstrated that when applied in multilevel partitioning and placement, balanced sized clusters can improve the final partitioning and placement solution qualities [13, 31, 38, 39].

Initial Cluster Refinement

As described above, in initial cluster creation, the seed cell and its neighbors are grouped together to form InitCluster. Since this grouping does not consider the cell

connectivities, the cells in InitCluster are not necessarily tightly connected. In most cases, there exist some cells in InitCluster that have more connections with cells outside. Such cells should be identified and removed. The task of the initial cluster refinement step is to rectify the assumptions made in the initial cluster creation, and consequently improve the initial cluster quality.

The specific procedure for this step is as follows: first, another group of cells, referred to as "NeighborCluster", is formed by the putting the cells that are directly connected to cells in InitCluster. A modified FM algorithm that restricts movements from InitCluster to NeighborCluster is performed. Like the standard FM algorithm procedure, the gains of cells in InitCluster need to be computed first. Next, the cell with highest positive gain in InitCluster is permanently moved to NeighborCluster, and the gains for the remaining cells in InitCluster are updated. This process of cell movement and gain updating is repeated until either of the following conditions is met:

- Condition 1: The seed cell is moved out of InitCluster, indicating that the trial of finding a refined cluster that has the seed cell as its "central" cell has failed.
- Condition 2: All cells are moved out of InitCluster, indicating that all cells in InitCluster have more connections with outside of InitCluster than inside of it.
- Condition 3: All remaining cells in InitCluster have non-positive gains, indicating that all remaining cells have more connections inside than outside of it.

In any case, this step terminates.

Final Cluster Formation

After the initial cluster refinement, a post processing step, final cluster formation, is performed. In this step, depending on the refinement results, different procedures are

invoked.

- Case 1: InitCluster is not empty, but the seed cell is moved out of InitCluster.

  This case is the first condition for stopping the cluster refinement process. In this case, even though there are still cells left in InitCluster, no cluster is formed. All the cells in both InitCluster and NeighborCluster are moved out and released for further cluster formation.

- Case 2: There are no cells left in InitCluster.

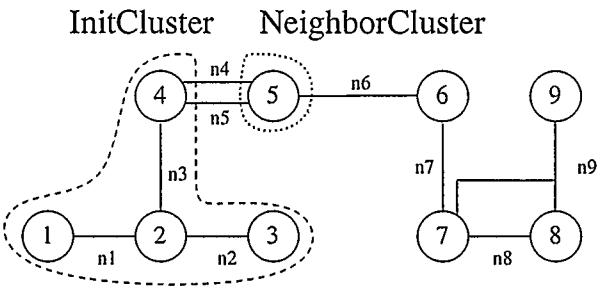
  This case corresponds to the second condition for stopping the cluster refinement process. In this case, all the cells have been moved out of InitCluster, which means that the initial created cluster was not tightly connected, and it is reasonable to discard this cluster. All cells in NeighborCluster are released for further cluster formation.

- Case 3: There are at least two cells left in InitCluster, including the seed cell.

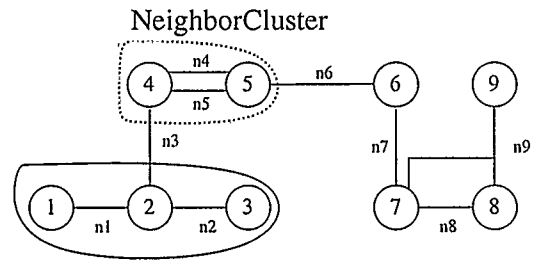
  This case corresponds to the third condition for stopping the cluster refinement process. This is the only case where a cluster can be formed. The remaining cells in InitCluster are shown to have more interconnections with each other; they are labeled with the seed cell number to form a cluster, and become accessory cells. All cells are released for further cluster identification.

4.3.2 Illustrative Example for Scoreless Gain Clustering Algorithm

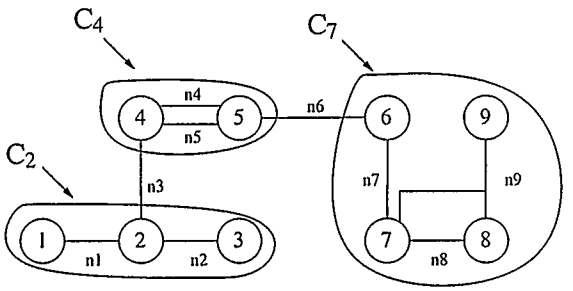
An example of the scoreless gain clustering algorithm is shown in Figure 4.3. Suppose in this example, the cells are visited based on their numbers, then, the first cell to be visited as a seed cell is cell ①. Cell ① and its unclustered neighbor, i.e., cell ② are put into partition "InitCluster". The neighbors of cells in InitCluster, i.e., cells ③ and ④ are put into partition "NeighborCluster". Then, the gains of the cells in InitCluster



(a) InitCluster and NeighborCluster partitions for seed cell ②



(b) Gain cluster identified for seed cell ②



(c) All identified gain clusters

Figure 4.3: Example of gain cluster identification using the proposed scoreless algorithm for a small circuit

are computed. In this example, the gain for cell ① is: $gain_{out}(①)$ = -1, and the gain for cell ② is: $gain_{out}(②)$ = 1. Cell ② is moved from InitCluster to NeighborCluster, and the gain for remaining cells in InitCluster, i.e., cell ① is updated. The gain for cell ① has changed from -1 to 1, and therefore, will also be moved into Neighbor. Since at this point, there is no cell left in InitCluster, the cluster identification based on seed cell ① is considered to be a failure, and the cells in NeighborCluster are released. The next cell, cell ②, is visited as a seed cell. Cell ② and its neighbors, cells ①, ③ and ④ are put into InitCluster, and their neighbor, cell ⑤ is put into NeighborCluster. In Figure 4.3(a), InitCluster and NeighborCluster partitions by seed cell ② are shown by dashed and dotted lines, respectively. Similar to the previous procedure, the cell gains in InitCluster are computed. Since cell ④ has the highest positive gain at this point, it is moved from InitCluster to NeighborCluster. After this move, the gains of the remaining cells in InitCluster are updated. At this point the gain of all cells in InitCluster is negative, therefore, no more cells are moved. The cells in InitCluster are considered a gain cluster, as illustrated in Figure 4.3(b). The algorithm continues until all the cells have been visited. All of the gain clusters that can be obtained using this algorithm are shown in Figure 4.3(c). In this example, seed cell ② identifies the gain cluster C_2, made of cells ①, ② and ③. Seed cell ④ identifies the gain cluster, C_4, made of cells ④ and ⑤. Seed cell ⑦ identifies the gain cluster, C_7, made of cells ⑥, ⑦, ⑧, and ⑨.

4.3.3 Complexity Analysis of Scoreless Gain Clustering Algorithm

Theorem 1: The time complexity of the scoreless gain clustering algorithm to find gain clusters in a circuit, is $O(n_p)$, where n_p is the number of pins, or cell terminals, in a circuit.

Proof. In the proposed clustering method, the FM algorithm is used to identify a

single cluster. Since FM has a complexity of $O(p)$ [34], the complexity of a single cluster identification will be $O(p_{cluster})$, where $p_{cluster}$ is equal to the number of pins in the cluster.

The worst case complexity for the proposed algorithm happens when all the cells are visited as seed cells. If a circuit has n_c cells, then the worst case complexity of the proposed algorithm is:

$$\sum_{i=1}^{n_c} O(p_{cluster_i}) = O(\sum_{i=1}^{n_c} p_{cluster_i}), \tag{4.3}$$

where $p_{cluster_i}$ is the total number of pins of all the cells in a cluster made from the seed cell i.

If the initial cluster is formed by only choosing the first level neighbor cells, this cluster contains the pins of seed cell i and the cells that are directly connected to it, therefore, $p_{cluster_i}$ is equal to:

$$p_{cluster_i} = \sum_{j=1}^{n_c} k_{ij} \times n_{pin_j}, \tag{4.4}$$

where n_{pin_j} is the number of pins of cell j (also the number of nets that cell j is connected to), and k_{ij} is the connectivity coefficient: $k_{ij} = 1$ if cell i and cell j are connected or $i = j$, otherwise $k_{ij} = 0$.

From equation (4.4), the total number of pins that will be visited can be calculated as:

$$\sum_{i=1}^{n_c} p_{cluster_i} = \sum_{i=1}^{n_c} \sum_{j=1}^{n_c} k_{ij} \times n_{pin_j} = \sum_{j=1}^{n_c} n_{pin_j} \sum_{i=1}^{n_c} k_{ij}.$$

Let $n_{neighbor_j}$ denote the number of neighbors of cell j, i.e., the number of cells that connect to cell j, $n_{neighbor}^{max}$ denote the maximum $n_{neighbor_j}$, i.e., $n_{neighbor_j} \leq n_{neighbor}^{max}$,

then:

$$\begin{aligned}\sum_{j=1}^{n_c} n_{pin_j} \sum_{i=1}^{n_c} k_{ij} &= \sum_{j=1}^{n_c} n_{pin_j}(n_{neighbor_j}+1) \\ &\leq \sum_{j=1}^{n_c} n_{pin_j}(n_{neighbor}^{max}+1) \\ &\leq (n_{neighbor}^{max}+1)\sum_{j=1}^{n_c} n_{pin_j} \\ &\leq (n_{neighbor}^{max}+1)n_p. \end{aligned} \tag{4.5}$$

Similarly, if the initial cluster is formed by choosing the first and second level neighbor cells, it can be deduced that

$$\sum_{i=1}^{n_c} p_{cluster_i} < (n_{neighbor}^{max}+1)^2 n_p. \tag{4.6}$$

In practice, $n_{neighbor}^{max}$ can be regarded as a constant, since its value does not increase with the circuit size increasing, therefore, from equations (4.3), (4.5) and (4.6), the overall time complexity of proposed algorithm is $O(n_p)$. □

4.3.4 Scoreless Clustering Algorithm Analysis

The clustering algorithm described in this section has the following advantages.

- The algorithm finds refined clusters made of highly connected cells.
- The algorithm is linear in time.
- The algorithm produces clusters with no cell overlap.

Note that once a cell has been assigned to a cluster, it is not considered for further clustering. Therefore, higher quality clusters may be ignored. This is due to the scoreless nature of the algorithm. In the rest of this chapter, two score-based algorithms that are capable of comparing clusters are introduced.

4.4 Score-Based Gain Cluster Algorithms

The number of gain clusters in a circuit is very large and these clusters can overlap. The sizes, in terms of the number of cells, of these clusters can vary from two cells to the whole netlist. Because of the complexity and sizes of the circuits encountered in today's VLSI designs, having a smaller number of cells in a cluster can usually result in a higher quality solution. Therefore, in this thesis it is proposed to use a bottom up approach to find gain clusters. In the bottom up approach, small initial clusters are formed and refined.

Two methods to form gain clusters and score them are proposed in this section. In the approach explained in Section 4.4.1, single cells are used as attractors to form initial clusters. These clusters are then refined and a score is assigned to them to measure the quality of the clusters. In the second proposed technique, instead of a seed cell, a seed net is used to form an initial cluster. This approach is described in detail in Section 4.4.4.

4.4.1 Gain Cluster Identification Using Seed Cell

Initial Cluster Creation and Refinement

A single cluster can be formed by grouping a cell and its different level neighbors. To be able to identify as many gain clusters as possible, all the cells in a circuit are visited and considered as potential seed cells. First, all the cells in a circuit are put in the partition *Origin*. Another partition, referred to as *InitCluster*, is also created. This partition can accommodate a cluster, but initially is empty. Once a cell is visited as a seed cell, it is moved from Origin to InitCluster. Once a seed cell is moved to InitCluster, the FM algorithm is used to move the cell with the highest non-negative gain from Origin to InitCluster and the cell gains are updated. The cell movement process is repeated until all the cells in Origin have negative gains. This indicates

that all the cells in InitCluster have more connections with cells inside of InitCluster compared to the cells in Origin and cells in InitCluster form a natural cluster. At this stage, the cell numbers in InitCluster are saved for each seed cell as a linked list, and all the cells are returned to Origin and their original gains are recovered.

There are two situations when a seed cell can fail to produce a cluster:

- A seed cell fails to attract any other cells into InitCluster; i.e., all cells in Origin have negative gains.
- The area of a cluster becomes more than a specified percentage of the total area of the circuit. This condition is imposed to avoid creating extremely large clusters.

To reduce the runtime for moving the cells from Origin to InitCluster, the following algorithm is proposed. At the start of the algorithm, the gains of moving all the cells from Origin to InitCluster, $gain_{in}$, are calculated and stored in a bucket array. This gain for each cell is equal to the negative of the degree of the cell, making the calculations linear in time. When a cell is moved from Origin to InitCluster, the gains of cells in Origin should be updated. But, only the gains of the cells directly connected to the moved cell need to be updated, which makes this algorithm very fast. In addition, once a cluster is either found or discarded, all the cells associated with the cluster are moved back to Origin and the original gain array is recovered. This makes the cluster identification algorithm very fast and efficient.

Clustering Score Calculation

After a gain cluster is identified, its quality is evaluated using a clustering score. The score of each cluster C_i is calculated as:

$$s_c(C_i) = \begin{cases} \frac{n_n(C_i)}{n_c(C_i)} \times \frac{1}{A(C_i)} & n_c(C_i) \neq 0 \\ 0 & n_c(C_i) = 0, \end{cases} \tag{4.7}$$

where, $s_c(C_i)$ is the score of the refined cluster C_i, $n_n(C_i)$ denotes the number of nets that lie entirely inside cluster C_i, and $n_c(C_i)$ represents the number of cells inside the refined cluster. $A(.)$ is a function representing the area of the clusters, and is equal to the summation of cell areas in a cluster. This score function is designed to give higher priority to cluster formations that reduce the number of nets in a circuit. In addition, it penalizes clusters that have a high area, resulting in more balanced clusters.

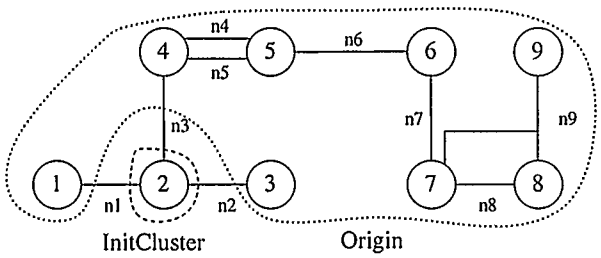
The outline of the proposed seed cell based gain cluster identification algorithm is shown in Figure 4.4.

input: a flat netlist output: a set of clusters
For all cells a. Single gain cluster identification: 1. Initial cluster creation using a seed cell 2. Initial cluster refinement b. Cluster score calculation for the identified gain cluster

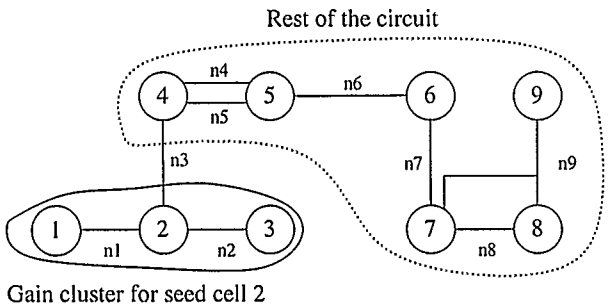
Figure 4.4: Cell based gain cluster identification algorithm

4.4.2 Illustrative Example for Gain Clustering Algorithm Using Seed Cell

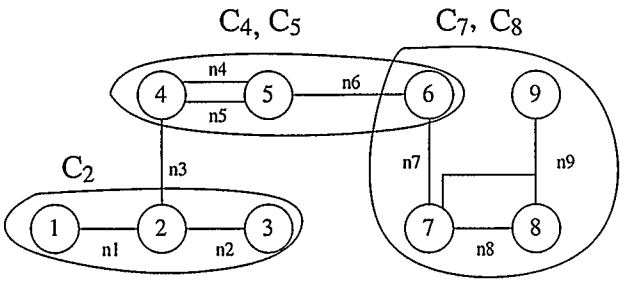
An example of this process is shown in Figure 4.5. At the beginning, the gain of all the cells to be moved to InitCluster, the negative of the cell degree, is calculated. For example, the gain for cell ① is: $gain_{in}(①)$ = -1, and the gain for cell ② is: $gain_{in}(②)$ = -3. All cells are visited as potential seed cells according to their cell numbers. Therefore, the first cell to be visited is cell ①. Cell ① fails to attract any cell and no cluster is identified. Then, the next cell, cell ②, is visited. In Figure 4.5(a), Origin and InitCluster partitions for seed cell ② are shown by dotted and dashed lines, respectively. Once cell ② is moved from Origin to InitCluster, the gains for cells ①, ③ and ④ have to be updated. The gain of cells ① and ③ changes from -1 to 1 and the gain of cell ④ changes from -3 to -1. Since, cells ① and ③ have the highest positive gain at this point, they are also moved from Origin to InitCluster. At this



(a) Origin and InitCluster partitions for seed cell ②



(b) Gain cluster identified for seed cell ②



(c) All identified gain clusters

Figure 4.5: Example of gain cluster identification using the proposed seed cell based algorithm for a small circuit

point the gain of all cells in Origin is negative, therefore, no more cells can move to InitCluster and this cluster is considered a gain cluster. For the example in Figure 4.5, the gain cluster made for seed cell ② consists of cells ①, ② and ③, as shown in Figure 4.5(b). All gain clusters that can be obtained using this algorithm are shown in Figure 4.5(c). In this example, seed cells ④ and ⑤ form the same initial clusters, C_4 and C_5, made of cells ④, ⑤, and ⑥. Seed cells ⑦ and ⑧ form the same initial clusters, C_7 and C_8, made of cells ⑥, ⑦, ⑧, and ⑨. Other cells fail to construct any gain clusters. The initial clusters (C_4, C_5) and (C_7, C_8) overlap on cell ⑥. In this example no area constraints have been imposed.

For the example in Figure 4.5, if all cells have unit area, the cluster scores are:

$$\begin{aligned} s_c(C_2) &= \frac{2}{3} \times \frac{1}{3} = 0.22, \\ s_c(C_4) = s_c(C_5) &= \frac{3}{3} \times \frac{1}{3} = 0.33, \\ s_c(C_7) = s_c(C_8) &= \frac{3}{4} \times \frac{1}{4} = 0.19. \end{aligned}$$

4.4.3 Complexity Analysis of Seed Cell-Based Gain Cluster Identification

Theorem 2: The time complexity of the cell based clustering algorithm to find gain clusters in a circuit, is $O(n_p)$, where n_p is the number of pins, or cell terminals, in a circuit.

The proof of this theorem is similar to the proof of theorem 1, but for the purpose of completeness, it is repeated here.

Proof. Figure 4.4 shows that the proposed cell based clustering algorithm is performed in two steps: initial gain cluster identification and cluster score calculation. In the following, the complexity of each of the steps is developed.

Complexity Analysis of Initial Gain Cluster Identification

The first step of initial gain cluster identification is to calculate the gain of moving each cell from Origin to InitCluster. In this step, the cell gains are computed and the bucket pointer for each cell is inserted into the bucket list. Since the cell gains are computed by visiting each net and its incident pins, if p_i represents the number of pins in net i, and n_n represents the total number of nets, then, the total time required to calculate all gains is:

$$O(\sum_{i=0}^{n_n} p_i) = O(n_p),$$

where n_p is the total number of pins in the circuit.

The gain insertion into the bucket list will take $O(n_c)$, where n_c is the number of cells in the circuit. Since each cell has more than one pin, $n_c < n_p$, the total required time for initialization is:

$$O(n_p) + O(n_c) = O(n_p).$$

This means that the gain insertion implementation has linear time complexity.

The initial cluster identification uses FM which has a linear complexity of $O(n_p)$ [34]. Therefore, the complexity for each cluster identification is: $O(p_{cluster_i})$, where $p_{cluster_i}$ is the number of pins in cluster i. To identify all initial clusters, all cells are visited as seed cells, and initial clusters are formed for each seed cell. Therefore, the complexity of finding all initial clusters is:

$$\sum_{i=1}^{n_c} O(p_{cluster_i}) = O(\sum_{i=1}^{n_c} p_{cluster_i}). \tag{4.8}$$

In the rest of this section, an upper bound for $\sum_{i=1}^{n_c} p_{cluster_i} = p_{total}$ or the total number of pins that will be visited, is developed.

The total number of pins for $cluster_i$, $p_{cluster_i}$, made from seed cell i is:

$$p_{cluster_i} = \sum_{j=1}^{n_c} k_{ij} \times p_{c_j},$$

where p_{c_j} is the number of pins of cell j (also the number of nets that cell j is connected to), and k_{ij} is a coefficient where $k_{ij} = 1$ if cell j is inside the initial cluster formed by cell i, otherwise $k_{ij} = 0$. The number of times cell j has been clustered is equal to $\sum_{i=1}^{n_c} k_{ij}$. Therefore, the total number of pins that will be visited, p_{total} is:

$$p_{total} = \sum_{i=1}^{n_c} p_{cluster_i} = \sum_{j=1}^{n_c} p_{c_j} \sum_{i=1}^{n_c} k_{ij}.$$

If $n_{inCluster}^{max}$ denote the maximum number of times that a cell is clustered, i.e. $\max_j \sum_{i=1}^{n_c} k_{ij}$, then,

$$\begin{aligned} p_{total} = \sum_{j=1}^{n_c} p_{c_j} \sum_{i=1}^{n_c} k_{ij} &\leq \sum_{j=1}^{n_c} p_{c_j} \times n_{inCluster}^{max}, \\ p_{total} &\leq n_{inCluster}^{max} \sum_{j=1}^{n_c} p_{c_j}, \\ p_{total} &\leq n_{inCluster}^{max} n_p. \end{aligned} \tag{4.9}$$

From (4.8) and (4.9), it can be deduced that

$$\sum_{i=1}^{n_c} O(p_{cluster_i}) = O(\sum_{i=1}^{n_c} p_{cluster_i}) = O(n_{inCluster}^{max} n_p). \tag{4.10}$$

In real VLSI circuits, $n_{inCluster}^{max}$ does not increase with the circuit size increasing, therefore, $n_{inCluster}^{max}$ can be regarded as a constant, and (4.10) can be written as:

$$\sum_{i=1}^{n_c} O(p_{cluster_i}) = n_{inCluster}^{max} O(n_p) = O(n_p).$$

So, the time complexity for the cluster identification step is $O(n_p)$.

Complexity Analysis of Cluster Score Calculation

The score calculations entail only a constant number of addition and multiplications for each, at most n_c, object, which are all linear in time.

Overall, the time complexity for the whole seed cell based clustering algorithm is $O(n_p)$. □

4.4.4 Gain Cluster Identification Using Seed Net

Forming clusters using a seed cell can be effective, but in some cases the percentage of the clusters that can be formed using this technique is very low. Therefore, a large circuit cannot be efficiently clustered with such a low clustering percentage. In this thesis, a new technique that uses a net as a seed to form a cluster is proposed. Such nets are called seed nets, as described in Definition 3.

Initial Cluster Creation and Refinement

Originally, all cells in the netlist are in partition, Origin. Another partition, InitCluster, that is supposed to accommodate any cluster is empty. The gains for moving all cells from Origin to InitCluster is calculated. These gains, as discussed in Section 4.4.1, are equal to the negative of the cell degree.

To form clusters using seed nets, first the nets in a circuit are visited. When a net is visited as a seed net, all movable cells in this net and their neighboring cells, are moved from Origin to InitCluster. A movable cell is a cell that is not a terminal or a fixed object. After the movement, the gains for the cells that have been moved, and their neighbors, are recalculated. This initial cluster formation is different from other clustering techniques, since it visits nets, as opposed to cells, to produce initial clusters, and the resulting clusters can be larger.

At this stage, the initial cluster needs to be refined. This phase is equivalent to calculating all the forces applied on a net through its connections to other nets. Because of the efficiency of the FM algorithm, it is proposed to use this algorithm to refine an initial cluster. As in the standard FM process, the cell with the maximum gain is moved from its original partition to the opposite partition. After the cell movement, the gains of the neighboring cells are updated. To give higher priority to cell movements into InitCluster, if the maximum cell gains in both partitions are equal, the cell in partition Origin is moved first. This process is repeated until the

gain to move any cell in InitCluster to Origin is less than or equal to zero, and the gain to move any cell in Origin to InitCluster is also less than zero. This indicates that the cells inside partition InitCluster have more inner-connections between each other than the outer-connections. At this point, a gain cluster is formed by saving the cell numbers in InitCluster a linked list with the seed net number used as the pointer. All the cells are returned to the Origin partition and their original gains are recovered.

There are two situations when a seed net fails to produce a cluster:

- All cells are moved out from the partition InitCluster.
- A seed net has a high degree. In this case the net is not considered as a seed net. In a real electronic circuit, the majority of nets are those with low degree; i.e., nets connecting a small number of pins. The percentage of nets having a large number of pins is very small. These nets are usually clock or power nets. As a result, in practice, if a group of cells are indeed highly interconnected, then, it is more likely that these cells are mainly connected by low degree nets. However, the benefits of not including nets with high degrees are as follows: The proposed algorithm either uses a net as a seed net for starting an initial cluster, or uses a net to add the neighboring cells of a seed net into the initial cluster. In either case, if the net has a higher degree, considerable cell gain calculations and cell movements need to be performed, which can be very time consuming. At the same time, the cells added into the initial cluster via the net with high degree probably do not have a high connectivity with the other cells in the initial cluster. Thus, skipping a long net does not necessarily result in losing any connectivity information. In addition, if a net with a high degree is clustered, the resulting cluster area can be very large. The experiments on ISPD05 benchmark circuits [7] show that compared to the implementation in

[52], skipping the nets with high degree in the initial cluster formation can result in a clustering runtime reduction of up to two times, and also improved final placement results. Overall, skipping the nets with high degree during initial cluster formation can greatly improve the clustering runtime, with almost no loss of, or even better, cluster quality.

It should be mentioned that in this algorithm no area constraints are used. The reason for this is that the clusters made in this step are scored based on their area. Therefore, it is not necessary to add cluster area constraints.

Compared with the cluster refinement technique in Section 4.4.1, which only allow one directional cell movement, this initial cluster refinement step follows the standard FM algorithm to naturally select the best cell to move in either direction based on the cell gain. This refinement step captures the connectivity information from both groups. As a result, the refined cluster can be more accurate and have a higher quality. This step is analogous to identifying cells that have more internal forces than external forces applied to them.

Clustering Score Calculation

The cluster score calculations for this algorithm are the same as discussed in Section 4.4.1, and are not repeated here.

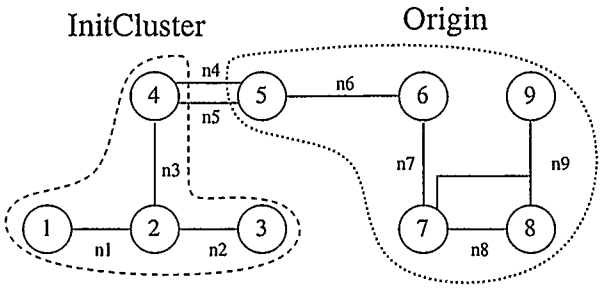
The outline of the proposed seed net based gain cluster identification algorithm is shown in Figure 4.6.

input: a flat netlist output: a set of clusters
For all nets with low degree a. Single net based gain cluster identification: 1. Initial cluster creation using a seed net 2. Initial cluster refinement b. Cluster score calculation for the identified gain cluster

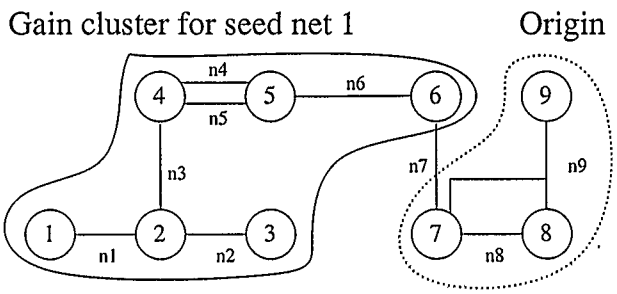
Figure 4.6: Net based gain cluster identification algorithm

4.4.5 Illustrative Example for Gain Clustering Algorithm Using Seed Net

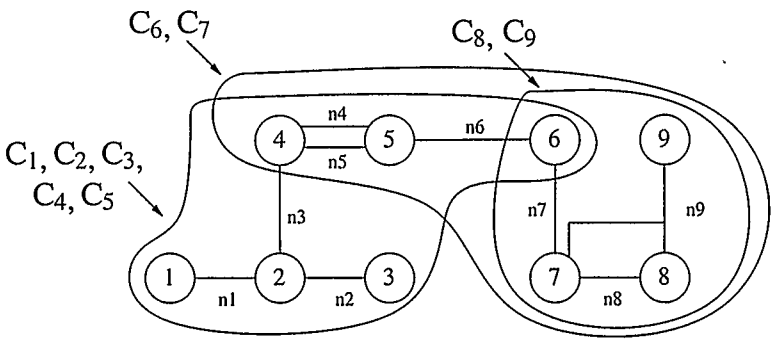
An example of gain cluster identification using seed nets is shown in Figure 4.7. At



(a) Origin and InitCluster partitions for seed net $n1$



(b) Gain cluster identified for seed net $n1$



(c) All identified gain clusters

Figure 4.7: Example of gain cluster identification using the proposed seed net based algorithm for a small circuit

the beginning the gain of all the cells to be moved to InitCluster, negative of the cell degree, is calculated. For example, the gain for cell ① is: $gain_{in}(①)$ = -1, and the gain for cell ② is: $gain_{in}(②)$ = -3. All nets are visited to become seed nets according

to their net numbers. Therefore, the first net to be visited as a seed net is net $n1$. All the cells in net $n1$ and their neighbors; i.e., cells ①, ②, ③ and ④, are moved from Origin to InitCluster. In Figure 4.7(a), Origin and InitCluster partitions are shown by dotted and dashed lines, respectively. Since cell ⑤ has the highest positive gain at this point, it is moved from Origin to InitCluster. After this move, the gain for cell ⑥ is updated from -2 to 0. Since the gain for ⑥ is nonnegative, it is also moved to InitCluster. At this point the gain of all cells in Origin is negative, and the gains of all cells in InitCluster are nonpositive, therefore, no more cells can be moved. The cells in InitCluster are considered to form a gain cluster. For the example in Figure 4.7, the gain cluster made for seed net $n1$ consists of cells ①, ②, ③, ④, ⑤ and ⑥ as shown in Figure 4.7(b). All the gain clusters that can be obtained using this algorithm are shown in Figure 4.5(c). In this example, seed nets $n1$, $n2$, $n3$, $n4$ and $n5$ identify the same gain clusters, C_1, C_2, C_3, C_4 and C_5, made of cells ①, ②, ③, ④, ⑤ and ⑥. Seed nets $n6$ and $n7$ identify the same clusters, C_6 and C_7, made of cells ④, ⑤, ⑥, ⑦, ⑧ and ⑨. Seed nets $n8$ and $n9$ identify the same initial clusters, C_8 and C_9, made of cells ⑥, ⑦, ⑧, and ⑨.

These three clusters all overlap on cell ⑥. In addition, cells ④, ⑤, ⑦, ⑧, and ⑨ belong to two clusters.

For the example in Figure 4.7, if all cells have unit area, the cluster scores are:

$$\begin{aligned} s_c(C_1) = s_c(C_2) = s_c(C_3) = s_c(C_4) = s_c(C_5) &= \frac{6}{6} \times \frac{1}{6} = 0.17, \\ s_c(C_6) = s_c(C_7) &= \frac{6}{6} \times \frac{1}{6} = 0.17, \\ s_c(C_8) = s_c(C_9) &= \frac{3}{4} \times \frac{1}{4} = 0.19. \end{aligned}$$

4.4.6 Complexity Analysis of Seed Net-Based Gain Cluster Identification

Theorem 3: The time complexity of the net based clustering algorithm to find gain clusters in a circuit, is $O(n_p)$, where n_p is the number of pins, or cell terminals, in a

circuit.

Proof. The proof of this theorem is the same as theorem 2. Minor changes are discussed as follows.

First, in the seed net based clustering technique, an initial cluster is formed by a seed net, instead of a seed cell in cell based technique. However, the seed net based technique also used a modified FM algorithm to refine a cluster, and, therefore, the time complexity of the gain cluster identification for the seed net based technique can also be expressed as

$$n_{inCluster}^{max} O(n_p) = O(n_p),$$

where $n_{inCluster}^{max}$ denotes the maximum number of times that a cell is clustered, as defined in Section 4.4.3.

Second, in the seed net based technique, since each net is visited as the seed net, the total number of clusters is n_n, where n_n is the number of nets in the netlist. Therefore, for the cluster score calculation, it requires $O(n_n)$ runtime, instead of $O(n_c)$ runtime in the seed cell based technique. However, both n_n and n_c are in linear time and smaller than n_p.

Overall, the time complexity for the whole net based clustering algorithm is $O(n_p)$.

□

4.5 Numerical Experiments

All the proposed clustering algorithms have been implemented in C++ and tested on ISPD98 benchmark circuits. The ISPD98 benchmark suite was released by IBM Austin research laboratory in 1998. It consists of 18 real industry circuits[1] with sizes ranging from 12,752 to 210,613 modules, including bus arbitrators, bus bridge

[1]In order to facilitate the application on partitioning, these circuits have been slightly changed, for example, all nets with more than 200 pins are removed from the netlist [12].

Table 4.1: ISPD98 benchmark circuit statistics

Circuit	# Cells	# Pads	# Modules	# Nets	# Pins	Max%
ibm01	12506	246	12752	14111	50566	6.37
ibm02	19342	259	19601	19584	81199	11.36
ibm03	22853	283	23136	27401	93573	10.76
ibm04	27220	287	27507	31970	105859	9.16
ibm05	28146	1201	29347	28446	126308	0.00
ibm06	32332	166	32498	34826	128182	13.56
ibm07	45639	287	45926	48117	175639	4.76
ibm08	51023	286	51309	50513	204890	12.10
ibm09	53110	285	53395	60902	222088	5.42
ibm10	68685	744	69429	75196	297567	4.80
ibm11	70152	406	70558	81454	280786	4.48
ibm12	70439	637	71076	77240	317760	6.43
ibm13	83709	490	84199	99666	357075	4.22
ibm14	147088	517	147605	152772	546816	1.99
ibm15	161187	383	161570	186608	715823	11.00
ibm16	182980	504	183484	190048	778823	1.89
ibm17	184752	743	185495	189581	860036	0.94
ibm18	210341	272	210613	201920	819697	0.96

chips, memory and PCI bus interfaces, communication adaptors, memory controllers, processors, and graphics adaptors [12]. The characteristics of ISPD98 benchmark circuits are given in Table 4.1. For each circuit, a cell is an internal movable object, a pad is an external object, and a module is either a cell or a pad. Max% gives the percentage of the total cell areas occupied by the largest module in the circuit [12].

The purpose of experiments presented in this section is to investigate the effect of clustering by studying various clustering statistics of different algorithms. The clustering statistical study can also help reveal the characteristics of each clustering algorithm, and furthermore, provide a guideline for making the best use of each clustering algorithm on a specific application field.

In the following, different clustering statistics by the three proposed gain clustering algorithms are reported from Table 4.2 to Table 4.7, and explained respectively. In each table, columns 2 and 3 represent the clustering results of the scoreless gain cluster identification algorithm, columns 4 and 5 are clustering results of the gain

cluster identification algorithm using score-based seed cell technique, and finally in columns 6 and 7, the clustering results of the gain cluster identification algorithm using score-based seed net technique are reported.

Table 4.2: Clustering statistics in terms of number of seed cells or seed nets that result in the formation of gain clusters

	Scoreless		Score using seed cell		Score using seed net	
Circuit	# cells	percentage	# cells	percentage	# nets	percentage
ibm01	828	6.49	2863	22.45	10775	76.36
ibm02	1293	6.60	7629	38.92	13373	68.29
ibm03	1255	5.42	5028	21.73	16310	59.52
ibm04	1972	7.17	5246	19.07	18899	59.11
ibm05	3171	10.81	8195	27.92	23429	82.36
ibm06	887	2.73	4762	14.65	13437	38.58
ibm07	2097	4.57	11006	23.96	26070	54.18
ibm08	2849	5.55	17221	33.56	35659	70.59
ibm09	2627	4.92	10450	19.57	37901	62.23
ibm10	4581	6.60	12734	18.34	49452	65.76
ibm11	4119	5.84	12499	17.71	50465	61.96
ibm12	3499	4.92	13748	19.34	41877	54.22
ibm13	3397	4.03	10457	12.42	49866	50.03
ibm14	6582	4.46	27706	18.77	75619	49.50
ibm15	5351	3.31	15324	9.48	103644	55.54
ibm16	6968	3.80	21962	11.97	111295	58.56
ibm17	7445	4.01	21701	11.70	104169	54.95
ibm18	13420	6.37	53993	25.64	132062	65.40
Average	-	5.42	-	20.40	-	60.40

In Table 4.2, the clustering results in terms of number of seed cells or seed nets that result in the formation of gain clusters, and their percentage to the number of total cells or nets, are reported.

Table 4.2 indicates that among all three techniques, the score-based algorithm using seed net technique can identify a larger number of gain clusters when compared to either the score-based algorithm using seed cell technique, or the scoreless algorithm. Therefore, if a low clustering ratio is required, the score-based algorithm using seed nets can be more suitable.

In Table 4.3, the clustering results in terms of number of cells per cluster and

Table 4.3: Clustering statistics in terms of number of cells per cluster

Circuit	Scoreless		Score using seed cell		Score using seed net	
	avg # cells	STD	avg # cells	STD	avg # cells	STD
ibm01	4.24	2.58	2.42	1.29	30.41	36.07
ibm02	3.42	2.99	5.21	2.93	27.15	58.65
ibm03	6.96	6.63	2.55	2.34	28.67	34.68
ibm04	4.44	4.26	2.61	2.55	41.40	99.74
ibm05	3.78	1.54	2.29	0.79	39.08	85.87
ibm06	4.67	3.23	2.27	1.18	28.28	36.92
ibm07	4.91	4.42	2.34	1.45	26.08	32.82
ibm08	3.79	5.04	5.13	3.14	68.56	159.01
ibm09	7.19	6.01	2.28	1.40	32.93	39.91
ibm10	4.31	4.89	2.69	4.97	38.30	54.16
ibm11	5.48	4.79	2.42	1.60	24.02	30.17
ibm12	4.95	9.19	2.65	9.63	68.74	139.33
ibm13	6.00	6.13	2.43	1.36	34.39	50.43
ibm14	4.53	5.40	2.30	1.86	35.83	78.71
ibm15	6.94	8.66	2.43	1.38	37.26	60.52
ibm16	5.79	4.64	2.28	2.02	38.71	53.55
ibm17	5.02	3.48	2.29	1.26	47.03	61.94
ibm18	3.61	2.47	4.88	3.14	33.51	63.79

the standard deviation are reported. For each algorithm, the column "avg # cells" represents the average number of cells in a cluster, and the column "STD" represents the standard deviation for numbers of cells in a cluster.

The results obtained in this experiment illustrate that on average for all benchmark circuits, the score-based algorithm using seed cell has a minimum average number of cells in one cluster and the score-based algorithm using seed net has a maximum average number of cells in one cluster. In other words, the clusters are relatively small when using the score-based technique via seed cells, and large when using the score based technique via seed nets.

In the following, three figures are shown, where the detailed compositions of the clusters containing different number of cells are plotted. Figures 4.8 shows the percentages for clusters containing 2 cells, 3 cells, 4-10 cells and more than 10 cells, respectively, when using the scoreless gain clustering identification algorithm. Fig-

ures 4.9 shows the similar result, when using the seed cell based scoring algorithm. Figures 4.10 shows a similar result, when using the seed net based scoring algorithm. These figures demonstrate consistent results with the data reported in Table 4.3. Furthermore, from these figures, it can be seen that different clustering algorithms favor clustering different types of clusters. For example, a large percentage of the clusters identified by the score-based algorithm using the seed cell technique are clusters that contain 2 cells. However, for the score-based algorithm using seed nets, most of the identified clusters contain large numbers of cells.

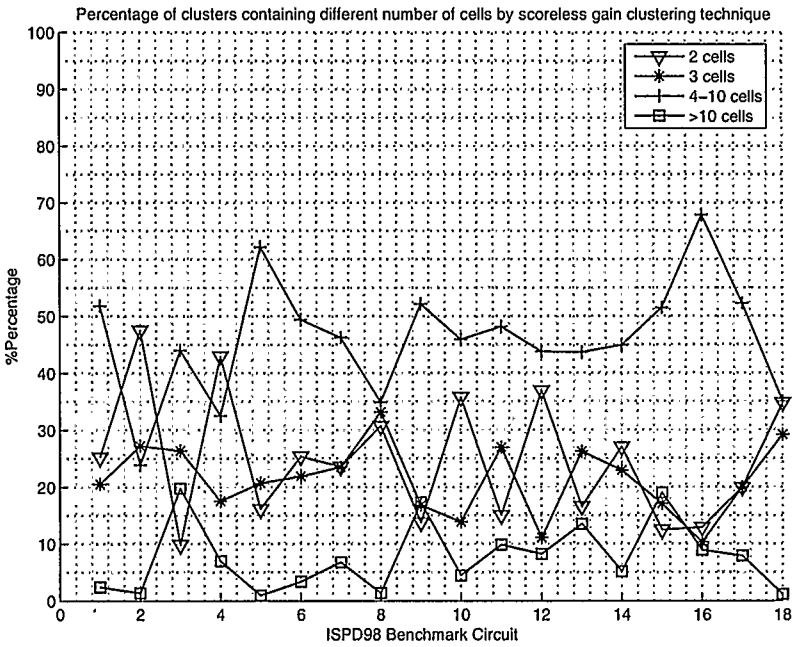


Figure 4.8: Clustering statistics in terms of number of cells per cluster for the scoreless clustering algorithm

In Table 4.4, the clustering results in terms of area per cluster and the standard deviation are reported. For each algorithm, the column "avg. area" represents the average area for an identified cluster, and the column "STD" represents the standard deviation for the areas of the clusters.

The results obtained in this experiment illustrate that on average, for all bench-

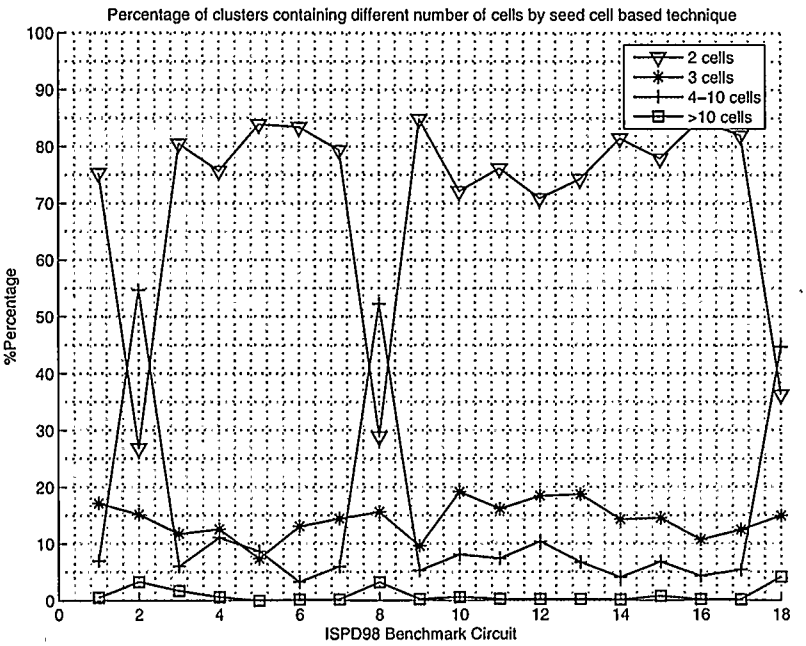


Figure 4.9: Clustering statistics in terms of number of cells per cluster for the seed cell based clustering algorithm

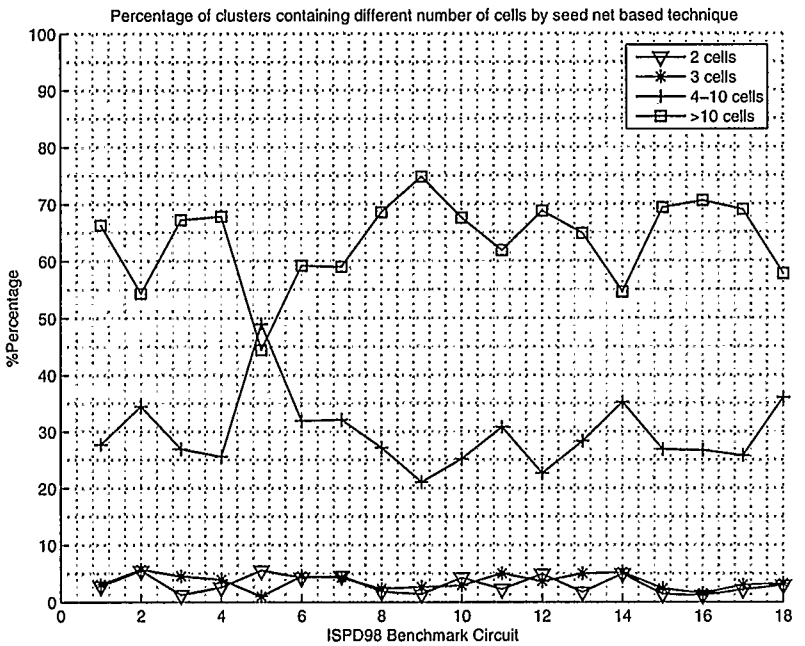


Figure 4.10: Clustering statistics in terms of number of cells per cluster for the seed net based clustering algorithm

Table 4.4: Clustering statistics in terms of area per cluster

Circuit	Scoreless		Score using seed cell		Score using seed cell	
	avg. area	STD	avg. area	STD	avg. area	STD
ibm01	816.08	1251.32	1105.75	5686.78	12785.40	26477.90
ibm02	912.98	12035.30	1314.00	16403.10	22527.30	173474.00
ibm03	2254.48	18540.60	1533.41	23265.60	17160.00	65390.50
ibm04	823.95	1314.13	1245.18	15629.40	23641.20	123495.00
ibm05	575.14	213.71	318.38	119.43	6324.18	13816.40
ibm06	562.51	1075.46	1057.81	23907.20	13744.00	141096.00
ibm07	1555.41	19556.50	753.75	10850.00	15682.90	72224.10
ibm08	1477.72	46242.70	1013.43	15189.90	96635.40	478679.00
ibm09	2497.35	30195.20	1100.87	22171.30	25070.10	129149.00
ibm10	3362.77	96201.70	2938.00	56543.30	88140.90	560654.00
ibm11	2404.27	32586.10	1096.52	20573.10	19024.10	108401.00
ibm12	3925.34	73252.90	2133.11	51651.90	169737.00	589594.00
ibm13	1792.93	25982.20	1367.21	20481.60	20288.50	90895.70
ibm14	782.06	5922.07	556.02	5068.29	11900.30	48450.20
ibm15	2158.75	55957.20	1064.92	41293.30	19970.00	219958.00
ibm16	2821.71	66352.20	1397.29	25011.60	52458.10	380801.00
ibm17	1403.44	9528.37	733.62	6341.45	12485.00	28858.90
ibm18	533.97	2478.09	727.16	2866.27	6272.64	36434.10

mark circuits, the score-based algorithm using seed nets has the maximum average area per cluster. This is consistent with previous experiments, since the score-based algorithm using seed net has been shown to have more cells in a cluster on average.

In Table 4.5, the clustering results, in terms of numbers of clustered cells and the percentage of those cells over the total number of cells, are reported. For each algorithm, the column "# cells" represents the number of clustered cells, and the column "percentage" represents the ratio of these cells over the total cells.

The results obtained in this experiment show consistency with previous results: the score-based algorithm using seed net has the maximum number of cells that are being clustered.

In Table 4.6, the clustering results in terms of number of overlapped clustered cells, i.e., those cells belonging to different clusters, and the percentage of those cells over the total number of clustered cells, are reported. For each algorithm, the column

Table 4.5: Clustering statistics in terms of number of clustered cells

Circuit	Scoreless		Score using seed cell		Score using seed net	
	# cells	percentage	# cells	percentage	# cells	percentage
ibm01	3514	27.56	5124	40.18	12582	98.67
ibm02	4425	22.58	9142	46.64	16610	84.74
ibm03	8735	37.76	9687	41.87	21068	91.06
ibm04	8757	31.84	9338	33.95	25118	91.31
ibm05	12000	40.89	13725	46.77	27801	94.73
ibm06	4140	12.74	8485	26.11	26270	80.84
ibm07	10291	22.41	18691	40.70	41238	89.79
ibm08	10799	21.05	21083	41.09	48827	95.16
ibm09	18876	35.35	18871	35.34	50370	94.33
ibm10	19736	28.43	20847	30.03	65698	94.63
ibm11	22576	32.00	22306	31.61	64569	91.51
ibm12	17326	24.38	23713	33.36	65399	92.01
ibm13	20371	24.19	20289	24.10	77012	91.46
ibm14	29837	20.21	45765	31.01	127259	86.22
ibm15	37154	23.00	28406	17.58	146995	90.98
ibm16	40327	21.98	39600	21.58	171326	93.37
ibm17	37362	20.14	39412	21.25	166965	90.01
ibm18	48416	22.99	71926	34.15	194742	92.46
Average	-	26.08	-	33.18	-	91.29

"# cells" represents the number of overlapped cells, and the column "percentage" represents the ratio of these cells over the total clustered cells.

It should be noted that since the scoreless algorithm finalizes a cluster once it is identified, so there are no overlapped cells, therefore all the data for the scoreless algorithm are 0 in Table 4.6. Comparing the two score-based techniques, it can be seen that the technique using seed net has a much higher number of overlapped cells.

In Table 4.7, the clustering results in terms of number of clusters that a cell belongs to, i.e., the number of times that a cell has been clustered, and the standard deviation are reported. For each algorithm, the column "# clusters" represents the number of clusters that one cell can belong to, and the column "STD" represents the corresponding standard deviation.

Comparing the two score-based techniques, it can be seen that the technique using seed nets demonstrates a much higher number of times that a cell can be clustered

Table 4.6: Clustering statistics in terms of number of overlapped cells

Circuit	Scoreless		Score using seed cell		Score using seed net	
	# cells	percentage	# cells	percentage	# cells	percentage
ibm01	0.00	0.00	1019	19.89	12403	98.58
ibm02	0.00	0.00	2333	25.52	14940	89.95
ibm03	0.00	0.00	2775	28.65	20067	95.25
ibm04	0.00	0.00	2396	25.66	23657	94.18
ibm05	0.00	0.00	2693	19.62	26150	94.06
ibm06	0.00	0.00	1451	17.10	23937	91.12
ibm07	0.00	0.00	4684	25.06	39190	95.03
ibm08	0.00	0.00	4917	23.32	47275	96.82
ibm09	0.00	0.00	3387	17.95	48870	97.02
ibm10	0.00	0.00	6502	31.19	63768	97.06
ibm11	0.00	0.00	5419	24.29	61609	95.42
ibm12	0.00	0.00	6103	25.74	63007	96.34
ibm13	0.00	0.00	3541	17.45	73236	95.10
ibm14	0.00	0.00	11372	24.85	117776	92.55
ibm15	0.00	0.00	5559	19.57	140242	95.41
ibm16	0.00	0.00	7811	19.72	165645	96.68
ibm17	0.00	0.00	6930	17.58	160665	96.23
ibm18	0.00	0.00	19798	27.53	187493	96.28
Average	-	0	-	22.82	-	95.17

to different clusters. This suggests that, for the score-based algorithm using seed net, the clusters are highly overlapped with each other.

4.6 Summary

In this chapter, three algorithms are proposed to identify a single gain cluster. The first algorithm is a scoreless technique, and the other two algorithms are score-based techniqués: one of them uses a seed cell to identify a gain cluster and the other one uses a seed net to find a gain cluster. All of the three algorithms have a linear time complexity. The characteristic of each algorithm can summarized as follows.

- The scoreless algorithm finalizes a cluster when it is identified; however, since no comparison between clusters is made, the clustering solution quality can be further improved.

Table 4.7: Clustering statistics in terms of number of clusters that one cell belongs to, i.e., number of overlaps for one cell

	Scoreless		Score using seed cell		Score using seed net	
Circuit	# clusters	STD	# clusters	STD	# clusters	STD
ibm01	1.00	0.00	1.35	0.97	26.04	18.15
ibm02	1.00	0.00	4.35	10.01	21.86	38.00
ibm03	1.00	0.00	1.32	0.63	22.20	22.22
ibm04	1.00	0.00	1.47	1.40	31.15	67.33
ibm05	1.00	0.00	1.37	0.88	32.94	40.53
ibm06	1.00	0.00	1.27	0.89	14.47	14.92
ibm07	1.00	0.00	1.38	0.88	16.49	18.21
ibm08	1.00	0.00	4.19	6.89	50.07	149.74
ibm09	1.00	0.00	1.27	0.74	24.78	23.04
ibm10	1.00	0.00	1.64	1.22	28.83	30.30
ibm11	1.00	0.00	1.36	0.84	18.77	20.44
ibm12	1.00	0.00	1.54	1.15	44.01	95.11
ibm13	1.00	0.00	1.25	0.70	22.27	23.22
ibm14	1.00	0.00	1.39	0.99	21.29	40.89
ibm15	1.00	0.00	1.31	1.01	26.27	33.95
ibm16	1.00	0.00	1.27	0.67	25.15	40.25
ibm17	1.00	0.00	1.26	0.71	29.34	26.26
ibm18	1.00	0.00	3.66	5.90	22.73	30.71

- The score-based algorithm using seed cell is the fastest algorithm compared to the other two techniques; however, this technique cannot reduce the netlist to a low clustering ratio.

- The score-based algorithm using seed net can achieve the lowest clustering ratio among all three techniques; however, the number of cells in a cluster, and the area of a cluster can be large. In addition, a large number of clusters produced by this technique overlap.

Chapter 5

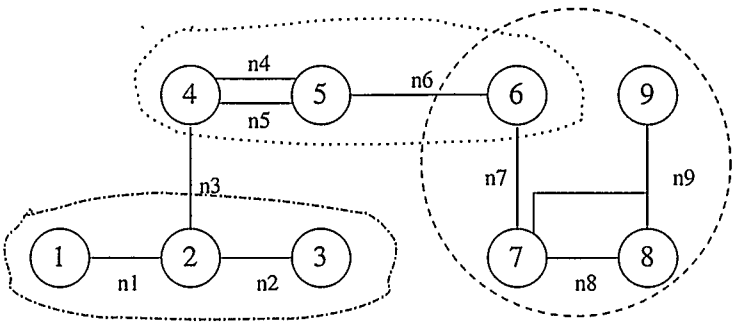
Algorithms for Obtaining Clustering Solutions

5.1 Introduction

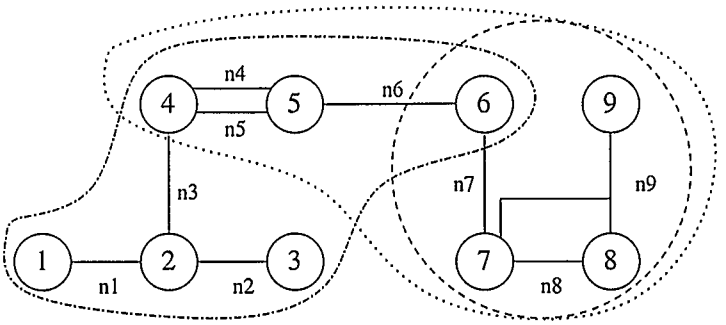
In Chapter 4, three algorithms for identifying single gain clusters in a circuit were proposed. In this chapter, another significant contribution of this thesis, clustering algorithms to form final clusters are introduced. The development of these algorithms is targeted to extend the application of clustering techniques proposed in Chapter 4 to cluster a whole circuit. The efficiency of these algorithms are illustrated in the context of circuit partitioning and placement.

The clusters produced using the score-based algorithms proposed in Chapter 4 can have overlap, as shown in Table 4.7, making the decision of the best group of clusters to be finalized complicated. An example of the cluster overlap is illustrated in Figure 5.1. Figure 5.1 (a) illustrates the original circuit and the gain clusters identified using the seed cell based approach. In this example, the two clusters shown by dashed and dotted lines overlap in cell ⑥. In Figure 5.1 (b), all the potential clusters using the seed net based technique are illustrated. In this example, three clusters overlap on cell ⑥. In addition, cells ④, ⑤, ⑦, ⑧, and ⑨ belong to two clusters. In most clustering techniques, these cell overlaps are removed by either choosing the cluster that is deemed to have the higher quality, or merging the clusters. In this thesis, three different techniques for forming the final clusters are proposed and implemented.

- The first technique is a scoreless technique, where cells are ordered and clusters are made based on the order of the cells. In this technique, cell overlap is not allowed and once a cluster is formed, all of the cells in that cluster are taken out of the possible cells that can be clustered.



(a) All gain clusters identified using seed cell based technique



(b) All gain clusters identified using seed net based technique

Figure 5.1: Example of initial clusters identified using the proposed seed cell based and seed net based algorithms in a small circuit

- In the second technique, a score is assigned to each cluster and clusters with the highest scores are finalized. In this technique, the cell overlap problem is solved by scoring the clusters.

- In the third technique, cluster overlaps are dealt with in a novel method. In this technique, the identified clusters are used to score nets. Then, nets that have a higher score are collapsed. Therefore the cluster overlap problem is solved without discarding cluster information.

The rest of this chapter is organized as follows. In Section 5.2, the proposed scoreless clustering technique is presented. In Section 5.3, the cluster score-based technique is proposed. In Section 5.4, the new net score-based clustering algorithm

is proposed. Numerical experiments and analysis for the proposed techniques are reported in Section 5.5. These experiments are conducted on ISPD98 benchmark circuits [5, 12], ICCAD04 benchmark circuits [2, 9] and ISPD2005 placement contest circuits [7, 58] in the context of partitioning and placement. Finally a summary of the chapter is given in Section 5.6.

5.2 Scoreless Algorithm for Obtaining A Clustering Solution

Most clustering algorithms for VLSI circuit partitioning and placement start by ordering all the cells in a netlist [13, 38, 39, 46]. In scoreless clustering algorithms, such as FirstChoice and heavy-edge matching, the cell ordering criterion is usually random. Therefore, the clustering result is also random, resulting in an "unstable" clustering performance. When clustering algorithms with random cell ordering are used in multilevel partitioning, different runs of a partitioner can produce different and greatly varying partitioning results. Here, a run refers to one execution of a partitioner. For example, in Figure 5.2, the partitioning results of 20 runs of hMetis on benchmark circuit ibm03 are plotted. In hMetis, FirstChoice is used to cluster the netlist. The x-axis is the run number for hMetis, and the y-axis is the partitioning result in terms of netcut for each run of hMetis. From Figure 5.2, it can be seen that the partitioning results from different runs have a wide range, from a minimum netcut of 700 nets to a maximum netcut of 800 nets.

In this thesis, two cell ordering techniques that can improve the results without noticeably increasing the runtime are given. The algorithm proposed in this section is a scoreless clustering technique, and it starts by visiting cells in a netlist. To be able to deal with the cell overlap problem, if the visited cell already belongs to a cluster, it is skipped and the next cell in the queue is examined. There are two important factors that can change the results obtained by this scoreless technique. The first one

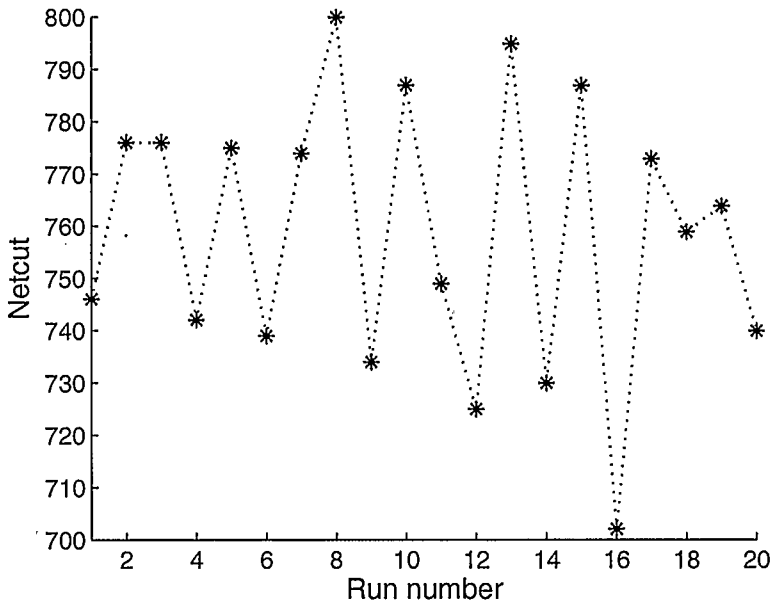


Figure 5.2: Illustration of fluctuation in the partitioning results in terms of netcut for 20 runs of hMetis on benchmark circuit ibm03

is the cluster formation that was discussed in Section 4.3, and the second one is the cell ordering. In order to improve the clustering solution quality, two nonrandom cell ordering criteria, cell degree based and number of neighbors based, are proposed and discussed in the rest of this section. Furthermore, in order to speed up the clustering process, sophisticated ordering computation is avoided in the proposed cell ordering criteria. Therefore, a good trade-off between clustering solution quality and runtime is achieved.

5.2.1 Cell Ordering Criteria

The two proposed ordering criteria in the scoreless clustering algorithm are as follows:

- *Criterion 1 (Cell degree based ordering):* Cells are ordered in ascending order according to cell degree, the number of nets incident to the cell.

- *Criterion 2 (Cell neighbor number based ordering):* Cells are ordered in ascend-

ing order according to the number of their connected neighbors.

Both criteria are based on local cell connectivity and favor cells with fewer incident nets or neighbors. To find gain clusters, each cell potentially can be used to form an initial cluster by grouping it with its neighbors. If a cell with a high degree or a large number of neighbor cells is visited at the beginning of the algorithm, a cluster with a large area is likely to be formed which can lead to unbalanced cluster sizes. Therefore, the nonrandom cell ordering methods are designed to give high priority to the cells that can lead to clusters with smaller area. The experimental results about the cell orderings are reported in Section 5.5.1.

An example of how the proposed cell ordering techniques work is shown in Figure 5.3, using a small sample circuit. In this circuit, the degree of each cell c_i,

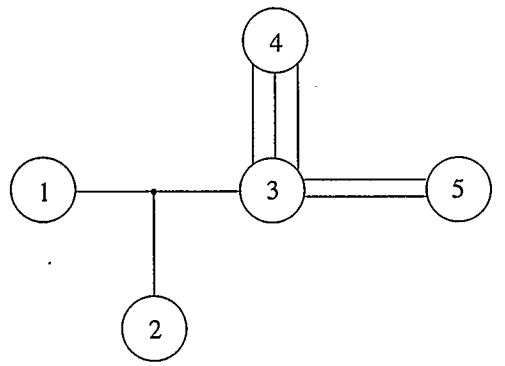


Figure 5.3: A simple circuit

$degree(c_i), 1 \leq i \leq 5$, is: $degree(c_1) = 1$, $degree(c_2) = 1$, $degree(c_3) = 6$, $degree(c_4) = 3$, and $degree(c_5) = 2$. If the cell degree based ordering criterion is used, then the cell ordering result is (①, ②, ⑤, ④, ③). If criterion 2 is used, then the number of neighbors for each cell, $neighbors(c_i), 1 \leq i \leq 5$, is calculated: $neighbors(c_1) = 2$, $neighbors(c_2) = 2$, $neighbors(c_3) = 4$, $neighbors(c_4) = 1$, and $neighbors(c_5) = 1$. The cell ordering result is (④, ⑤, ①, ②, ③).

Once all the cells are ordered, each cell is visited, and the cluster formation step discussed in Section 4.3 is performed, and either a refined cluster is finalized or the

cluster is discarded. Then, the next cell in the ordered cell sequence is examined. If the cell already belongs to another cluster, it is skipped; otherwise the same single cluster formation process, which involves initial cluster creation, initial cluster refinement, and final cluster formation, is performed. The whole algorithm terminates when all cells have been visited.

5.2.2 Clustered Netlist Generation

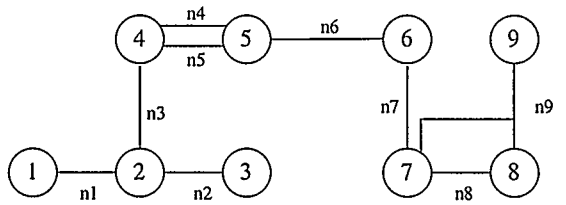
Once all the cells have been visited and processed appropriately, the complete clustering results are available: if a cell is a clustered cell, it is labeled with its seed cell number, otherwise, no label number is attached to this cell. With the complete clustering results, the process of generating a clustered netlist starts. The cells that have the same label number are merged into a newly created cell. The area of this new cell is equal to the sum of the area of the cells that are merged, and each net incident to the new cell is either a net that connects a merged cell to either an unmerged cell, or a merged cell with different label number. At the end of this step, a new clustered netlist is created. Figure 5.4 shows the outline of the whole scoreless clustering algorithm.

Input: A flat netlist Output: A clustered netlist
a. Order all cells based on the ordering criterion b. For each unclustered cell 1. Single cluster identification using the algorithm proposed in Section 4.3 2. Finalize the identified cluster c. Clustered netlist generation

Figure 5.4: Scoreless clustering algorithm procedure

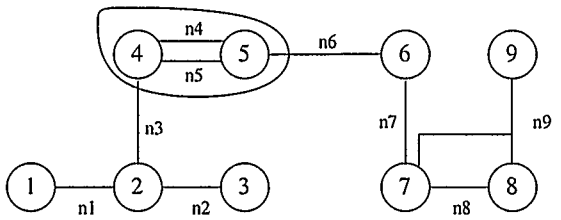
5.2.3 Illustrative Example for Scoreless Clustering Algorithm

For illustrative purposes, a simple circuit shown in Figure 5.5 is used to demonstrate the procedure for the gain clustering algorithm. In Figure 5.5(a), the original circuit

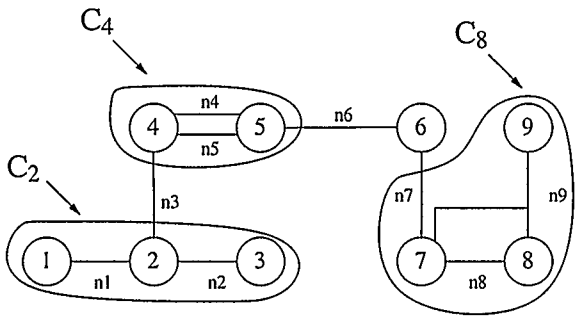


(a) The original flat netlist

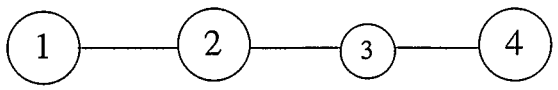
Identified cluster for seed cell 4



(b) The first identified cluster by seed cell ④



(c) All identified clusters



(d) The final clustered netlist

Figure 5.5: Illustration of the scoreless clustering algorithm procedure on a simple circuit

is shown. The algorithm starts by ordering all the cells in the circuit based on either cell degree or cell neighbor number criterion. Suppose in this example, the cells are ordered based on cell neighbor number; then the ordered cell sequence is as follows:

(①, ③, ④, ⑤, ⑥, ⑧, ⑨, ② and ⑦). The first cell in the list, cell ①, is visited as a seed cell. The cluster identification step discussed in Section 4.3 is performed, but no cluster is identified for this seed cell. Then, the next cell in the ordered cell sequence, ③, is examined as a seed cell. This seed cell also fails to form a cluster. Cell ④ is visited as the third seed cell, and it can result in a cluster, C_4, made of cells ④ and ⑤, as shown in Figure 5.5(b). This process is continued. Cells ⑤, ⑥, ⑨ and ⑦ all fail to result in a cluster. Cell ⑧ is a successful seed cell resulting in cluster, C_8, made of cells ⑦, ⑧ and ⑨. Cell ② is the last cell that results in a cluster, C_2, made of cells ①, ② and ③. In Figure 5.5(c), all identified clusters are shown.

By using the above clustering results, a clustered netlist is created, as shown in Figure 5.5(d). In this new netlist, the new cell ① is created by merging cells ①, ② and ③ in the original netlist, the new cell ② is created by merging cells ④ and ⑤ in the original netlist, and similarly, the new cell ④ is created by merging cells ⑦, ⑧ and ⑨ in the original netlist. The new cell ③ is copied from the cell ⑥ in the original netlist.

5.3 Cluster Score-Based Algorithm for Obtaining A Clustering Solution

In this section, the cluster score-based technique for obtaining the final clustering solution is proposed. In this technique, once a set of gain clusters are identified and the corresponding scores are calculated for those clusters, the final clustering solution can be obtained as follows. First, the clusters are ordered descendingly based on their scores. Then each cluster is visited in turn. The cluster with the highest score is visited first and clustered. The next cluster is examined. If this cluster contains cells that belong to another cluster, then this cluster is skipped and the next cluster is visited; otherwise, this cluster is finalized. This process is repeated until a desired

clustering ratio has been reached.

A high level outline of the cluster score-based algorithm is shown in Figure 5.6. The algorithm consists of two phases: *potential gain cluster identification* and *final cluster formation*. In the first phase, potential gain cluster identification, a score-based single gain cluster identification algorithm, as proposed in Section 4.4, is used to find a single gain cluster. Then the score for the identified cluster is calculated. At the end of the first phase, a set of gain clusters is identified where each cluster has a clustering score. In the second phase, the final cluster formation, the clusters are ordered based on their associated scores, which indicate the quality of the clusters. The clusters with higher scores will be clustered first, if the cluster area constraint is satisfied. This process is continued until a desirable clustering ratio is reached.

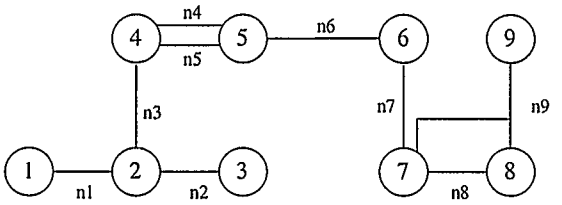
Input: A flat netlist Output: A clustered netlist
Phase 1: Potential gain cluster identification a. Identification of a set of single gain clusters using the algorithms proposed in Section 4.4 **Phase 2: Final cluster formation** b. Ordering of clusters c. Finalization of clusters in turn

Figure 5.6: Cluster score-based algorithm procedure

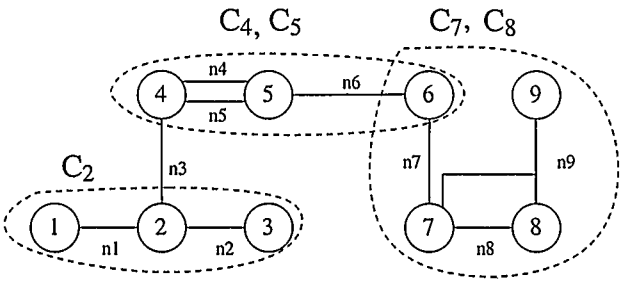
5.3.1 Illustrative Example for Cluster Score-Based Algorithm

For illustrative purposes, a simple circuit shown in Figure 5.7 is used to demonstrate the procedure of the proposed cluster score-based clustering algorithm.

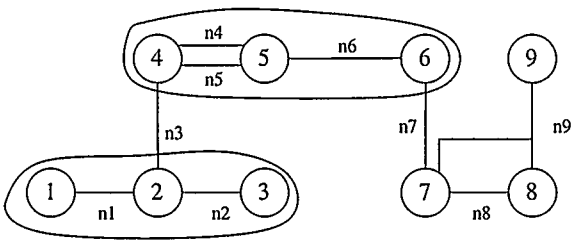
In Figure 5.7(a), the original netlist is shown. In Figure 5.7(b), all identified gain clusters by the seed cell based technique, as proposed in Section 4.4, are shown. The



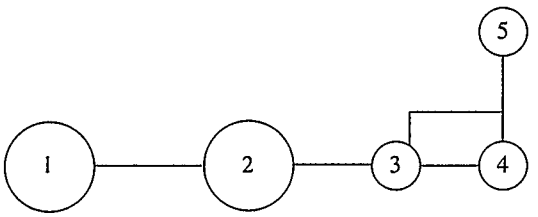
(a) The original flat netlist



(b) All identified clusters



(c) The finalized clusters



(d) The final clustered netlist

Figure 5.7: Illustration of the cluster score-based clustering algorithm procedure on a simple circuit

scores for clusters are:

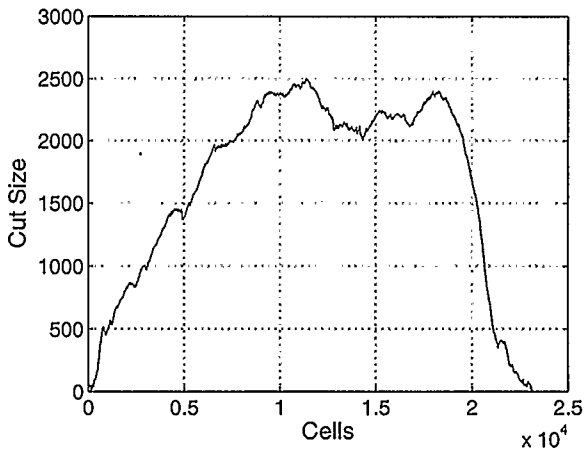
$$\begin{aligned} s_c(C_2) &= \frac{2}{3} \times \frac{1}{3} = 0.22, \\ s_c(C_4) = s_c(C_5) &= \frac{3}{3} \times \frac{1}{3} = 0.33, \\ s_c(C_7) = s_c(C_8) &= \frac{3}{4} \times \frac{1}{4} = 0.19. \end{aligned}$$

Therefore, the first finalized cluster is either C_4 or C_5. The second finalized cluster is C_2, as illustrated in Figure 5.7(c). Then the remaining clusters, C_7 and C_8 are examined. Since they contain the clustered cell ⑥, they cannot be clustered.

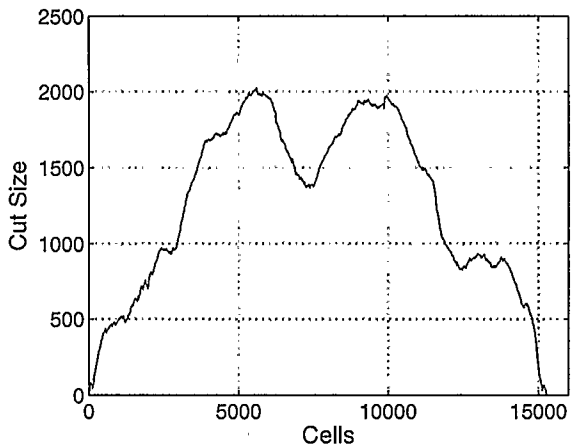
By using the above clustering results, a clustered netlist is created, as shown in Figure 5.7(d). In this new netlist, the cell ① is created by merging cells ①, ② and ③ in the original netlist, and cell ② is created by merging cells ④, ⑤ and ⑥ in the original netlist. The rest of the cells are just copied from the original netlist.

5.3.2 Spectral Analysis of the Cluster Score-Based Technique

In this section, the clustering effects of the proposed cluster score-based algorithms on the netlist structure are studied by the spectral technique proposed in [17]. In Figure 5.8, an example of how the structure of a circuit is maintained during the clustering process, as highly connected cells are grouped together, is shown. The figures in 5.8 are obtained by linearly ordering the cells in the benchmark circuit, ibm03, using the eigenvalue based technique in [17]. This technique is a spectral analysis method based on finding the second largest eigenvalue of the connectivity matrix of a circuit. In Figures 5.8 (a) and (b), the x-axis indicates the cell position number after the linear ordering. It should be noted that the x-axis in the two figures do not match since in Figure 5.8 (b) cells have formed clusters and therefore there is a smaller number of cells in the circuit. The y-axis is the netcut if the circuit is partitioned at the specified cell. The y-axis for both circuits match to show how the netcut has been reduced. In Figure 5.8(a), cells in the original test circuit ibm03 are linearly ordered. In Figure



(a) No clustering



(b) Two levels of clustering

Figure 5.8: Illustration of the linear ordering of the cells and the respective netcut for circuit ibm03

5.8(b), the linear order of the same circuit, after applying two levels of clustering, is shown. In Figure 5.8(a), the netcut peak has occurred around cell 12000 and the number of netcuts is at 2500. This peak corresponds to the peak around cell 5500 in Figure 5.8(b), with a maximum value of 2000. The second peak in Figure 5.8(a) has occurred around cell 18000 and the number of netcuts is over 2400. This peak corresponds to the peak around 10000 in Figure 5.8(b), with a maximum netcut lower than 2000 nets. By comparing these two figures, it can be seen that the number of netcuts in the linear order is reduced when clustering is applied. In addition, in Figure 5.8(b), a valley has appeared around cell 7000, which may mean that the clustering algorithm is trying to naturally separate the circuit into different partitions.

5.4 Score-Based Net Cluster Algorithm for Obtaining A Clustering Solution

The two score-based algorithms proposed in Chapter 4 can both produce a set of gain clusters. The clustering statistics from these two techniques, as shown in Section 4.5, indicate that the number of clusters produced by the seed net based technique is much larger than that by the seed cell based technique. Therefore, the seed net based technique potentially can reduce a netlist to a much lower clustering ratio, compared to the seed cell based technique. For today's large scale designs, a clustering technique producing a lower clustering ratio can be better suited to the design requirements. However, the statistics for the seed net based technique in Section 4.5 also show that there is a large number of cell overlaps between the clusters produced by the seed net technique.

In most existing score-based clustering algorithms [11, 51, 59], the cell overlap problem is solved by ordering the clusters descendingly based on their scores, and then finalizing the clusters with highest scores. As a result, some clusters are discarded.

In this thesis, a new net scoring technique is proposed and used to solve the problem of cell overlap between clusters. The inspiration of the net scoring technique comes from the force-directed model.

For a single net, its geometry relationship to a cluster can be one of the following three situations:

1. The net is fully embraced by the cluster; i.e., all the cells in the net are inside the cluster. In this case, the relationship of the net to the cluster is referred to as "*clustered*". This is equivalent to a contraction force on the net, from the perspective of the force-directed model.

2. The net is partially embraced by the cluster; i.e., some of the cells in the net are inside the cluster. In this case, the relationship of the net to the cluster is referred to as "*cut*". This is equivalent to an expanding force on the net.

3. The net has no connection with the cluster; i.e., none of the cells in this net is inside the cluster. In this case, the relationship of the net to the cluster is referred to as "*nonconnected*" This is equivalent to no force on the net.

In the set of clusters produced by a score-based technique, for a single net, it can be fully embraced by some clusters, be partially embraced by some clusters, and have no connections with other clusters, simultaneously. The final status of the net should be determined by considering all these clusters that have different relationships with the net, since different relationships indicate different intentions, or forces, on the net. The clusters that fully embrace the net have contraction forces on the net, the clusters that partially embrace the net have expanding forces on the net, and the clusters that have no connection with the net have no force on the net.

Since once a cluster is identified, a score is calculated to evaluate the quality of that cluster, this score can be regarded as the force that the cluster applies to nets.

The force applied to each net is calculated based on the value of the cluster score, amount of force, and the relationship, direction of the force. This process is discussed in detail in the rest of this section.

5.4.1 Net Score Calculation

Originally, each net has a score with an initial value set to be 0. After the score-based algorithm identifies a gain cluster, as discussed in Section 4.4, the cluster score is calculated. The cluster score calculation is the same as in equation 4.7:

$$s_c(C_i) = \begin{cases} \frac{n_n(C_i)}{n_c(C_i)} \times \frac{1}{A(C_i)} & n_c(C_i) \neq 0 \\ 0 & n_c(C_i) = 0, \end{cases}$$

where, $s_c(C_i)$ is the score of the identified gain cluster C_i, $n_n(C_i)$ denotes the number of nets that lie entirely inside the cluster C_i, and $n_c(C_i)$ represents the number of cells inside the refined cluster. $A(.)$ is the function representing the area of the clusters, and is equal to the summation of cell areas in a cluster.

After calculating the cluster score, the nets that have connections with this cluster are examined. Based on the geometry relation of the net and the cluster, the score of the cluster, $s_c(C_i)$, is added or subtracted from the net score.

$$s_n(n_j) = s_n(n_j) \ + a_{ji} s_c(C_i), \tag{5.1}$$

where $s_n(n_j)$ is the clustering score for net n_j, and $s_c(C_i)$ is the clustering score for cluster C_i calculated above. a_{ji} is a coefficient, where

$$a_{ji} = \begin{cases} 1 & \text{if } n_j \text{ is clustered by } C_i \\ -1 & \text{if } n_j \text{ is cut by } C_i \\ 0 & \text{otherwise.} \end{cases}$$

From (5.1), the overall clustering score for a net n_j can be calculated as:

$$s_n(n_j) = \sum_{i}^{m} a_{ji} s_c(C_i), \tag{5.2}$$

where m is total number of the identified gain clusters in a netlist. An overall positive net score in (5.2) means that there are more attraction forces on the net and the net should be contracted. An overall negative score in (5.2) for a net means that the net is being stretched. A zero score can be thought of as a case where the total forces on a net are zero.

5.4.2 Net Cluster Formation

In this step, cluster overlaps are removed and final clusters are formed based on the calculated net scores. First, all nets with a clustering score greater than 0 are ordered in descending order. Then, each such net is processed. If clustering a net satisfies the cluster area constraint, it will be considered to be clustered. Three different situations can occur when a net is examined:

1. *None of the cells in this net belong to a cluster:* In this case, all the cells in this net are clustered and a new cluster is created.

2. *At least one cell in this net has already been clustered and all of the clustered cells belong to one cluster:* In this case, the cluster made by this net has overlap with only one existing cluster. If the cluster area constraint is satisfied, all unclustered cells in this net join the overlapping cluster. If the cluster area constraint is not satisfied, the net is not clustered.

3. *At least two cells in this net have been clustered and these clustered cells belong to at least 2 different existing clusters:* In this case, the cluster formed by this net has overlap with different existing clusters. The cells of the net under consideration and all of its overlapping clusters will be tried to see if they can be merged into a cluster that encompasses them all. If the area of this cluster is no more than the cluster area constraint, then, the cells of the current net

and the overlapping clusters are merged into a new bigger cluster. Otherwise, the net is not clustered.

A high-level outline of the proposed score-based net cluster algorithm is shown in Figure 5.9. The algorithm consists of two phases: *potential gain cluster identification*

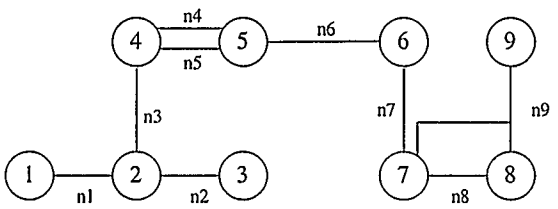
Input: A flat netlist Output: A clustered netlist
Phase 1: Potential gain cluster identification For each net with low degree a. Single cluster identification using the algorithm proposed in Section 4.4 b. Cluster score calculation c. Net score calculation **Phase 2: Final cluster formation** d. Net cluster formation

Figure 5.9: Score-based net cluster algorithm procedure

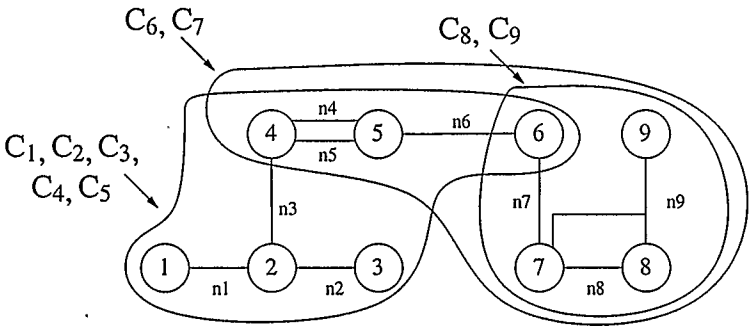
and *final cluster formation.* In the first phase, potential gain cluster identification, the score-based single gain cluster identification algorithm proposed in Section 4.4 is used for each net. Then the score for the identified cluster is calculated and is used to update the scores for the nets in an incremental manner. At the end of the first phase, each net has a clustering score. In the second phase, final cluster formation, nets in the netlist are ordered based on their associated scores, which measures the quality of nets. The nets with high scores are clustered first. This net clustering process is continued until there are no remaining nets with score greater than 0, indicating that all "good" nets for clustering have been processed.

5.4.3 Illustrative Example for the Score-Based Net Cluster Algorithm

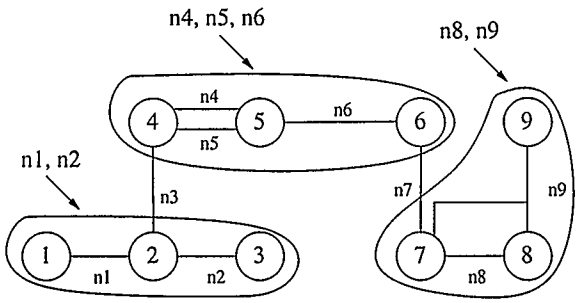
For illustrative purposes, a simple circuit shown in Figure 5.10 is used to demonstrate the procedure of the proposed score-based net cluster algorithm. In Figure 5.10(a), a netlist is shown. In Figure 5.10(b), all identified gain clusters using the seed net



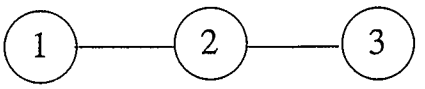
(a) The original flat netlist



(b) All identified clusters



(c) The clusters by net scores



(d) The final clustered netlist

Figure 5.10: Illustration of the score-based net cluster algorithm procedure on a simple circuit

based technique proposed in Section 4.4 are shown. The scores for the clusters are:

$$\begin{aligned} s_c(C_1) = s_c(C_2) = s_c(C_3) = s_c(C_4) = s_c(C_5) &= \frac{6}{6} \times \frac{1}{6} = 0.17, \\ s_c(C_6) = s_c(C_7) &= \frac{6}{6} \times \frac{1}{6} = 0.17, \\ s_c(C_8) = s_c(C_9) &= \frac{3}{4} \times \frac{1}{4} = 0.19. \end{aligned}$$

The net score calculation is as follows. Nets $n1$ and $n2$, are clustered by C_1, C_2, C_3, C_4 and C_5; therefore, their scores can be calculated as

$$s_n(n_1) = s_n(n_2) = 0 + \sum_{i=1}^{5} s_c(C_i) = 0 + 0.17 \times 5 = 0.85.$$

Net $n3$ is clustered by C_1, C_2, C_3, C_4 and C_5, and cut by C_6 and C_7; therefore, its score is:

$$s_n(n_3) = 0 + \sum_{i=1}^{5} s_c(C_i) - \sum_{j=6}^{7} s_c(C_j) = 0 + 0.17 \times 5 - 0.17 \times 2 = 0.51.$$

Nets $n4$ and $n5$ are both clustered by C_1, C_2, C_3, C_4, C_5, C_6 and C_7, therefore, their scores are:

$$s_n(n_4) = s_n(n_5) = 0 + \sum_{i=1}^{7} s_c(C_i) = 0 + 0.17 \times 7 = 1.19.$$

Net $n6$ is clustered by C_1, C_2, C_3, C_4, C_5, C_6 and C_7, and cut by C_8 and C_9, therefore, its score can be calculated as

$$s_n(n_6) = 0 + \sum_{i=1}^{7} s_c(C_i) - \sum_{j=8}^{9} s_c(C_j) = 0 + 0.17 \times 7 - 0.19 \times 2 = 0.81.$$

Net $n7$ is clustered by C_6, C_7, C_8 and C_9, and cut by C_1, C_2, C_3, C_4 and C_5, therefore, its score can be calculated as

$$\begin{aligned} s_n(n_7) &= 0 + \sum_{i=6}^{9} s_c(C_i) - \sum_{j=1}^{5} s_c(C_j) \\ &= 0 + 0.17 \times 2 + 0.19 \times 2 - 0.17 \times 5 = -0.13. \end{aligned}$$

Finally nets $n8$ and $n9$ are clustered by C_6, C_7, C_8 and C_9, therefore, their scores are:

$$s_n(n_8) = s_n(n_9) = 0 + \sum_{i=6}^{9} s_c(C_i) = 0 + 0.17 \times 2 + 0.19 \times 2 = 0.72.$$

After calculating the net scores, the nets are ordered based on their scores descendingly, ($n4$, $n5$, $n1$, $n2$, $n6$, $n8$, $n9$, $n3$ and $n7$). Net $n4$ is clustered first. Since all cells in net $n5$ have already been clustered, clustering of net $n5$ is skipped. Net $n1$ is the third clustered net. Net $n2$ has one cell belonging to $n1$, and is merged with $n1$. The final clusters are shown in Figure 5.10(c).

By using the above clustering results, a clustered netlist is created, as shown in Figure 5.10(d). In this new netlist, cell ① is created by merging cells ①, ② and ③ in the original netlist, cell ② is created by merging cells ④, ⑤ and ⑥ in the original netlist, and similarly, cell ③ is created by merging cells ⑦, ⑧ and ⑨ in the original netlist.

5.5 Numerical Experiments

5.5.1 Experimental Results for the Scoreless Clustering Technique

Cell Ordering Experiments

The first experiments in this section are used to study the effects of different cell ordering techniques on cluster formation. ISPD98 benchmark circuits are used as the test circuits.

To illustrate that the proposed technique results in balanced clusters, a set of experiments is performed, whereby the clustering statistics by different cell ordering techniques are obtained. In the following, these statistics are reported in Tables 5.1, 5.2, 5.3 and 5.4. The results in all of these four tables are reported in the same manner. In each table: columns 2 and 3, "Ngr. ordering", are the results for cell neighbor number based ordering; columns 4 and 5, "Deg. ordering" are the results for cell degree based ordering; columns 6 and 7, "Rnd ordering 1" are the results for the first run of cell random ordering; and finally columns 8 and 9, "Rnd ordering 2", are the results for the second run of cell random ordering. For each ordering technique,

the column "Ori." lists the actual values of the clustering results, and the column "Comp." lists the comparison results of the actual values over the actual values of "Ngr. ordering" result.

Table 5.1: Clustering statistics in terms of average number of cells in one single cluster from different cell ordering techniques

	Ngr. ordering		Deg. ordering		Rnd ordering 1		Rnd ordering 2	
Circuit	Ori.	Comp.	Ori.	Comp.	Ori.	Comp.	Ori.	Comp.
ibm01	4.24	1.00	4.29	1.01	4.71	1.11	4.64	1.09
ibm02	3.37	1.00	3.44	1.02	4.08	1.21	4.02	1.19
ibm03	6.88	1.00	6.94	1.01	7.32	1.06	7.32	1.06
ibm04	4.50	1.00	5.35	1.19	5.88	1.31	5.75	1.28
ibm05	3.79	1.00	3.81	1.00	3.92	1.03	3.91	1.03
ibm06	4.68	1.00	5.02	1.07	5.30	1.13	5.26	1.12
ibm07	4.92	1.00	5.00	1.02	5.12	1.04	5.21	1.06
ibm08	3.82	1.00	3.96	1.04	5.17	1.35	5.13	1.34
ibm09	7.06	1.00	7.16	1.01	7.56	1.07	7.47	1.06
ibm10	4.37	1.00	4.45	1.02	4.83	1.11	4.84	1.11
ibm11	5.46	1.00	5.47	1.00	5.74	1.05	5.74	1.05
ibm12	4.69	1.00	4.75	1.01	4.94	1.05	4.98	1.06
ibm13	5.85	1.00	5.91	1.01	6.29	1.08	6.31	1.08
ibm14	4.59	1.00	4.62	1.01	4.90	1.07	4.90	1.07
ibm15	6.79	1.00	6.95	1.02	7.36	1.08	7.38	1.09
ibm16	5.70	1.00	5.84	1.02	6.06	1.06	6.05	1.06
ibm17	5.07	1.00	5.37	1.06	5.68	1.12	5.69	1.12
ibm18	3.63	1.00	3.75	1.03	4.59	1.26	4.60	1.27
Average	-	1.00	-	1.03	-	1.12	-	1.12

In Table 5.1, the average number of cells in one single cluster for different cell ordering techniques are given. The experimental data in this table suggests that among all ordering techniques, the cell neighbor number based ordering produces the minimum average number of cells in one cluster. The cell degree based ordering produces a slightly higher value than cell neighbor number based ordering, but a lower value than both of the random orderings.

In Figure 5.11, the average number of cells in a cluster for all ISPD98 circuits are shown. An interesting point in this figure is that for circuits with similar sizes, for example, ibm08 and ibm09, the cluster sizes can be very different. This is due to

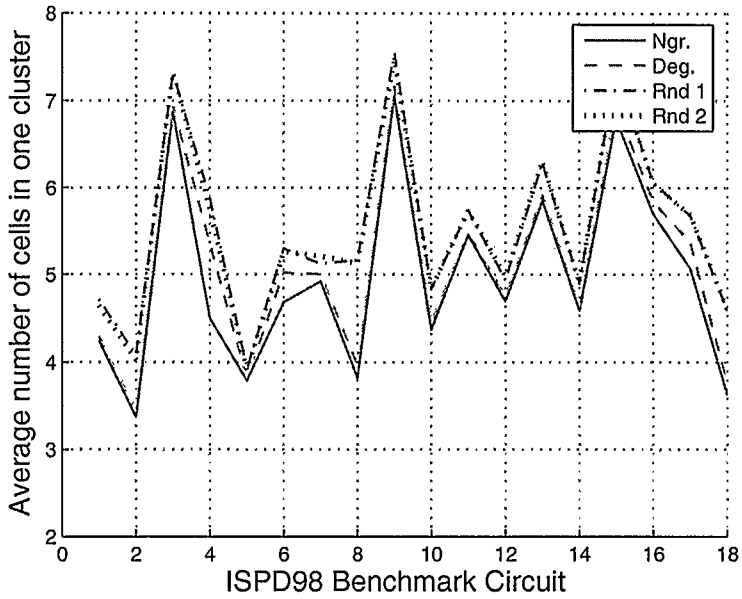


Figure 5.11: Average number of cells per cluster for different cell ordering techniques on ISPD98 benchmark circuit

different circuit structures.

In Table 5.2, the standard deviation for the numbers of cells in one cluster for different cell ordering techniques are given. The experimental data in this table show the difference or fluctuation between the cluster sizes in terms of number of cells for different ordering techniques.

It can be observed that the cell neighbor number based and cell degree based orderings have lower standard deviation values, indicating that the clusters by these two algorithm have a "balanced" size in terms of number of cells in cluster, compared to the random ordering techniques. On average, if the standard deviation for cell neighbor based ordering is chosen as the base, then the cell degree based ordering have 5% more fluctuation in the sizes of clusters, and the two random ordering have 29% and 27% more fluctuations.

In Table 5.3, the maximum numbers of cells in a cluster for different cell ordering techniques are given. The experimental data in this table show the size in terms of

Table 5.2: Clustering statistics in terms of standard deviation for numbers of cells in one single cluster from different cell ordering techniques

	Ngr. ordering		Deg. ordering		Rnd ordering 1		Rnd ordering 2	
Circuit	Ori.	Comp.	Ori.	Comp.	Ori.	Comp.	Ori.	Comp.
ibm01	2.59	1.00	2.78	1.07	3.59	1.39	3.34	1.29
ibm02	2.35	1.00	2.52	1.07	3.16	1.34	3.16	1.34
ibm03	6.41	1.00	6.44	1.01	6.88	1.07	6.85	1.07
ibm04	4.43	1.00	4.93	1.11	10.44	2.36	9.14	2.06
ibm05	1.53	1.00	1.59	1.04	1.80	1.17	1.74	1.13
ibm06	3.16	1.00	3.46	1.09	4.03	1.27	4.08	1.29
ibm07	4.39	1.00	4.42	1.01	4.59	1.04	4.72	1.08
ibm08	4.09	1.00	4.82	1.18	8.62	2.11	8.50	2.08
ibm09	5.83	1.00	5.85	1.00	6.17	1.06	6.06	1.04
ibm10	6.32	1.00	6.44	1.02	7.36	1.17	7.40	1.17
ibm11	4.69	1.00	4.66	0.99	4.96	1.06	4.96	1.06
ibm12	3.94	1.00	4.04	1.03	4.56	1.16	4.51	1.15
ibm13	5.52	1.00	5.54	1.00	6.14	1.11	6.29	1.14
ibm14	5.36	1.00	5.39	1.01	5.95	1.11	6.11	1.14
ibm15	7.89	1.00	7.95	1.01	8.49	1.08	8.70	1.10
ibm16	4.15	1.00	4.26	1.03	4.78	1.15	4.72	1.14
ibm17	3.48	1.00	3.67	1.06	4.07	1.17	4.00	1.15
ibm18	2.47	1.00	3.02	1.22	3.53	1.43	3.55	1.43
Average	-	1.00	-	1.05	-	1.29	-	1.27

number of cells for the biggest cluster for each ordering technique.

It can be observed that the experimental data reported in Table 5.3 are consistent with previous tables: the cell neighbor number based ordering produces the minimum value for the largest cluster size; the cell degree based ordering produces a little higher, on average 6% higher, value for the largest cluster size; and both of the random orderings produce much higher values, on average 55% and 46% respectively, than the cell neighbor number based ordering. It should be noted that for this experiment, each ordering has the same predefined cluster size upper bound.

In Table 5.4, the average cluster areas for different cell ordering techniques are given. This table shows consistent experimental data as previous tables. Although the same maximum cluster area constraint is applied to each ordering technique, the cell neighbor number based and cell degree based orderings produce clusters will smaller

Table 5.3: Clustering statistics in terms of maximum number of cells in one single cluster from different cell ordering techniques

	Ngr. ordering		Deg. ordering		Rnd ordering 1		Rnd ordering 2	
Circuit	Ori.	Comp.	Ori.	Comp.	Ori.	Comp.	Ori.	Comp.
ibm01	32	1.00	32	1.00	35	1.09	28	0.88
ibm02	32	1.00	32	1.00	57	1.78	51	1.59
ibm03	77	1.00	85	1.10	79	1.03	78	1.01
ibm04	61	1.00	53	0.87	364	5.97	312	5.11
ibm05	15	1.00	24	1.60	36	2.40	23	1.53
ibm06	27	1.00	35	1.30	38	1.41	55	2.04
ibm07	110	1.00	110	1.00	110	1.00	116	1.05
ibm08	160	1.00	154	0.96	359	2.24	358	2.24
ibm09	80	1.00	78	0.98	78	0.98	78	0.98
ibm10	305	1.00	305	1.00	305	1.00	305	1.00
ibm11	117	1.00	117	1.00	106	0.91	117	1.00
ibm12	61	1.00	62	1.02	96	1.57	83	1.36
ibm13	107	1.00	107	1.00	107	1.00	107	1.00
ibm14	188	1.00	188	1.00	188	1.00	188	1.00
ibm15	332	1.00	333	1.00	345	1.04	338	1.02
ibm16	90	1.00	90	1.00	109	1.21	109	1.21
ibm17	67	1.00	67	1.00	63	0.94	61	0.91
ibm18	51	1.00	66	1.29	66	1.29	68	1.33
Average	-	1.00	-	1.06	-	1.55	-	1.46

average area, compared to the random ordering techniques. Finally, the cell neighbor number based ordering produces the smallest values for average cluster areas.

In general Tables 5.1 to 5.4 show that cell neighbor number based ordering results in the creation of the most balanced clusters.

Partitioning Experiments

To be able to illustrate the effectiveness of the proposed algorithm, a hybrid multilevel circuit partitioner is developed, where the proposed clustering technique is implemented as a preprocessing step. First, a flat netlist is clustered using the cell ordering algorithm. As a result, a clustered smaller circuit is obtained. Then, the multilevel partitioning process is performed on the clustered smaller circuit. In the implementation, the library interface of state-of-the-art circuit partitioner, hMetis [1, 46], is invoked to perform the initial partitioning on the clustered circuit. Finally,

Table 5.4: Clustering statistics in terms of average area of one single cluster from different cell ordering techniques

	Ngr. ordering		Deg. ordering		Rnd ordering 1		Rnd ordering 2	
Circuit	Ori.	Comp.	Ori.	Comp.	Ori.	Comp.	Ori.	Comp.
ibm01	814.54	1.00	822.14	1.01	963.04	1.18	958.26	1.18
ibm02	579.70	1.00	584.65	1.01	696.92	1.20	683.88	1.18
ibm03	2227.80	1.00	2210.35	0.99	2341.22	1.05	2333.47	1.05
ibm04	828.02	1.00	994.34	1.20	1220.19	1.47	1161.46	1.40
ibm05	575.17	1.00	576.81	1.00	595.67	1.04	594.89	1.03
ibm06	559.20	1.00	561.59	1.00	586.72	1.05	582.69	1.04
ibm07	1547.70	1.00	1716.69	1.11	1761.70	1.14	1785.68	1.15
ibm08	631.88	1.00	675.02	1.07	942.45	1.49	924.82	1.46
ibm09	2051.56	1.00	2076.36	1.01	2297.62	1.12	2223.87	1.08
ibm10	1336.93	1.00	1444.86	1.08	2504.79	1.87	2079.10	1.56
ibm11	2362.11	1.00	2444.49	1.03	2440.75	1.03	2472.80	1.05
ibm12	1818.64	1.00	1826.38	1.00	1910.67	1.05	1833.61	1.01
ibm13	1725.15	1.00	1727.05	1.00	1847.98	1.07	1853.88	1.07
ibm14	791.08	1.00	741.45	0.94	837.38	1.06	780.74	0.99
ibm15	1359.28	1.00	1381.51	1.02	1473.66	1.08	1477.95	1.09
ibm16	1713.96	1.00	1662.18	0.97	1904.79	1.11	1896.57	1.11
ibm17	1382.76	1.00	1427.80	1.03	1408.53	1.02	1414.04	1.02
ibm18	536.92	1.00	571.32	1.06	698.33	1.30	698.41	1.30
Average	-	1.00	-	1.03	-	1.19	-	1.15

the best partitioning result among multiple runs of hMetis on the clustered circuit is chosen, mapped back to original circuit, and further refined by the standard FM algorithm. All the implementations are in C++ and tested on the ISPD98 benchmark suite [5, 12]. Since the actual area bipartitioning, which uses each cell's actual area as its weight to calculate the balance condition during the bipartitioning process, is more typical and practical in the context of VLSI physical design, all experimental data reported in the following are actual area bipartitioning results. All these partitioning experiments are performed on a 900 MHz Sun Ultra Sparc 3 with 4G memory.

The experimental setup for the hybrid partitioner is as follows: for each benchmark circuit, given a partitioning balance condition of either 10% or 2% deviation from an exact bisection,[1] 10 runs are performed. In each run, different sets of parameters

[1]The 10% and 2% deviations are the most commonly used balance conditions for partitioning in current literature.

are used: 5 runs use cell degree based ordering and 5 runs use cell neighbor number based ordering. Each of the two 5 runs are further broken into 3 runs using first level neighbor and 2 runs using second level neighbor as well to form the initial clusters. Then, the library interface of hMetis is called to perform partitioning on the clustered circuit. The number of runs of hMetis is set to be 10. Finally, the partitioning result obtained from hMetis (The best result over 10 runs) is further refined by a flat FM algorithm and the refined partitioning solution is given as the final output of the hybrid partitioner.

After 10 runs of the hybrid partitioner for each circuit with either 10% or 2% deviation, the run that produces the best partitioning result is chosen and reported. In Table 5.5 the partitioning results with 10% deviation are given. In Table 5.6 the partitioning results with 2% deviation are given. In each table, the column "Initial netcut" reports the partitioning results of library interface of hMetis on the clustered circuit; the column "Refined netcut" is the final refined partitioning results of hybrid partitioner on the original circuit. The average runtime in seconds for 10 runs of the hybrid partitioner is also reported in column "Runtime (second)".

In order to evaluate the effectiveness of proposed scoreless clustering algorithm and the hybrid partitioner, the following two experiments are performed:

- Comparison between the proposed scoreless clustering and heavy-edge matching

  The purpose of this experiment is to compare the final results of the proposed scoreless clustering with other commonly used clustering algorithms. In this experiment, the heavy-edge matching algorithm [13] is also implemented as a preprocessing step for multilevel partitioning. First, a circuit is clustered using the heavy-edge matching algorithm, then the resulting clustered circuit is partitioned by library interface of hMetis, and finally the partitioning solution from hMetis is further refined by the FM algorithm. This partitioning process is

Table 5.5: Actual area bipartitioning results on ISPD98 benchmark suite. Solutions are constrained to be within 10% of bisection.

Circuit	Hybrid Partitioner			Heavy-edge + hMetis		hMetis	
	Initial netcut	Refined netcut	Runtime (second)	Refined netcut	Runtime (second)	Final netcut	Runtime (second)
ibm01	217	**215**	5	223	5	**215**	5
ibm02	249	**249**	9	264	9	269	8
ibm03	643	**643**	10	745	11	681	11
ibm04	440	**440**	11	443	12	**440**	12
ibm05	1745	1715	17	**1705**	19	1722	17
ibm06	363	**363**	15	372	15	367	14
ibm07	754	**718**	23	760	23	737	24
ibm08	1161	**1156**	30	1161	28	1157	28
ibm09	530	**520**	22	524	23	524	23
ibm10	740	**735**	38	758	38	756	37
ibm11	709	**694**	31	**694**	33	695	32
ibm12	2033	**1972**	49	1984	51	1976	53
ibm13	913	**833**	46	844	44	**833**	44
ibm14	1614	**1521**	107	1534	105	1527	120
ibm15	1783	**1783**	126	1905	119	1801	128
ibm16	1710	**1667**	143	1674	149	1668	167
ibm17	2389	**2215**	215	2281	206	2257	246
ibm18	1686	**1522**	182	1566	178	**1522**	213

referred to as "heavy-edge + hMetis". The experimental setup for "heavy-edge + hMetis" is the same as that for hybrid partitioner, and its partitioning results are reported under the columns "Heavy-edge + hMetis" in Tables 5.5 and 5.6. Similar to the results of hybrid partitioner, the reported "Refined netcut" data are the best partitioning results of 10 runs of "Heavy-edge + hMetis" for each circuit, and the reported runtime is also the average of 10 runs of "Heavy-edge + hMetis".

- Comparison between the hybrid partitioner and hMetis

  The purpose of this experiment is to compare the hybrid partitioner with an existing state-of-the-art circuit partitioner. In this experiment, the hybrid partitioner is directly compared with hMetis. 10 runs of hMetis are performed directly to each benchmark circuit with either 10% or 2% deviation. The best

Table 5.6: Actual area bipartitioning results on ISPD98 benchmark suite. Solutions are constrained to be within 2% of bisection.

Circuit	Hybrid Partitioner			Heavy-edge + hMetis		hMetis	
	Initial netcut	Refined netcut	Runtime (second)	Refined netcut	Runtime (second)	Final netcut	Runtime (second)
ibm01	219	**217**	5	238	5	230	5
ibm02	267	267	10	279	9	**266**	9
ibm03	742	**742**	11	851	12	748	11
ibm04	522	**494**	11	505	13	506	12
ibm05	1743	**1724**	17	1751	20	1727	18
ibm06	496	**496**	17	583	15	550	16
ibm07	768	**726**	25	804	23	739	23
ibm08	1182	**1180**	32	1206	27	1188	29
ibm09	530	**522**	22	523	23	523	23
ibm10	1085	1085	38	**1066**	41	1133	40
ibm11	818	**770**	31	793	33	781	33
ibm12	2155	**1986**	51	1998	49	1998	51
ibm13	931	**890**	49	894	45	902	45
ibm14	1978	**1756**	106	1794	114	1772	131
ibm15	2092	**2092**	126	2204	121	2099	134
ibm16	1702	**1658**	152	1665	149	1692	180
ibm17	2496	**2321**	223	2361	212	2396	254
ibm18	1962	1693	211	**1664**	196	**1664**	261

partitioning result and the average runtime for these runs are reported under the column "hMetis" in Tables 5.5 and 5.6.

From Table 5.5 and Table 5.6, the following can be observed:

1. For most partitioning results of the hybrid partitioner, the initial netcuts, i.e., the partitioning results of hMetis on the clustered circuit, are close or equal to the final refined netcuts. This suggests that the proposed scoreless clustering algorithm has the ability to identify the gain clusters in a circuit and the clustered circuit is a good representative of the original circuit. In most cases, the hybrid partitioner can maintain the partitioning solution quality when clustering a circuit; although in a few cases, there is a loss of partitioning solution quality during clustering; however, this loss can be recovered by the final FM refinement step.

2. Among the three comparison partitioners: Hybrid, heavy-edge + hMetis, and hMetis, Hybrid achieves the best partitioning solution quality, in terms of net-cuts. The overall partitioning results by Hybrid is better than those by heavy-edge + hMetis or hMetis for most of the test benchmarks. In Tables 5.5 and 5.6, the best partitioning results for each circuit are indicated with bold font. It can be seen that for partitioning with 10% deviation, out of 18 test circuits, Hybrid produces 17 best partitioning results; for partitioning with 2% deviation, Hybrid produces 15 best partitioning results.

3. The runtime of the hybrid partitioner is comparable to other partitioners. For the relatively small test benchmarks that have less than 10,000 cells, ibm01 to ibm13, the three partitioners use roughly equal runtime for each circuit. However, for the large test benchmarks that have more than 10,000 cells, ibm14 to ibm18, Hybrid has less runtime than hMetis, and slightly more runtime than heavy-edge + hMetis. The reason for Hybrid being slower than heavy-edge + hMetis is that the heavy-edge matching algorithm is used in the same manner as the proposed scoreless clustering algorithm; but, in itself, heavy-edge matching is faster than the proposed scoreless algorithm. However, the final refined netcut comparison between heavy-edge + hMetis and Hybrid suggests that the cluster quality by heavy-edge matching is worse than that by the proposed algorithm. The runtime efficiency of Hybrid is mainly due to the following two reasons.

   First, the deterministic behavior of the proposed scoreless algorithm improves the efficiency of Hybrid. The proposed scoreless algorithm is a non-random clustering technique, and, given the same running parameters, it can generate the same clustering results in multiple runs. Therefore, in Hybrid, the clustering results from one run of the proposed clustering algorithm can be used directly in subsequent runs. In contrast with this deterministic characteristic,

in other multilevel partitioning processes, e.g., hMetis, clustering is generally done randomly, therefore, the clustering results of a single run cannot be used in other runs; i.e., in each run, the clustering must start from the original circuit. As a result, more clustering operations, which require more runtime, need to be performed in hMetis than Hybrid.

Second, the high quality clustering results produced by our proposed clustering algorithm improve the runtime for Hybrid. The proposed clustering algorithm has been shown to be able to identify the highly interconnected clusters in a circuit, and maintain the partitioning solution quality during the clustering process. As a result of this, after obtaining the partitioning result of hMetis on the clustered circuit and mapping it to the original circuit, the original circuit already has a very good initial partitioning solution. For this initial partitioning solution, the following FM based refinement step tends to use a low number of iterations to improve it, and consequently less time is required for the FM algorithm to converge. The small difference between the initial netcuts and final refined netcuts for Hybrid in Tables 5.5 and 5.6 also confirms that FM based refinement only has marginal netcut improvement.

In conclusion, because of its deterministic nature and the ability to produce high quality clustering results, the hybrid partitioner can achieve good partitioning solutions in a reasonable runtime.

5.5.2 Experimental Results for the Cluster Score-Based Technique

In these experiments, the seed cell based clustering technique is used to identify gain clusters. Then, at each level, the final clustering solution is obtained using either the cluster scores (CS) or the net scores (NS).

Partitioning Experiments

In the first experiment, the performances of different clustering scoring techniques are studied. A comparison between cluster score based and net score based techniques is performed to empirically determine the effectiveness of each scoring technique. Different combinations of cluster scoring (CS) and net scoring (NS), during two levels of our proposed clustering algorithm, are investigated. Specifically, 4 possible combinations of scoring are considered, namely CS+CS, NS+NS, CS+NS and NS+CS (the notation "x+y" implies that the scoring methods "x" was used in the first level and method "y" was used in the second level of clustering). After clustering each of the ISPD98 benchmark circuits using each scoring combination, each netlist is partitioned with 20 runs of hMetis allowing a 10% deviation from an exact bisection.

The results of this experimentation are provided in Table 5.7. The netcut and

Table 5.7: Comparison between cluster-score based and net-score based techniques in terms of netcut and runtime

Circuit	Average net cut and average runtime comparison							
	CS + CS		NS + NS		CS + NS		NS + CS	
	Netcut	Runtime	Netcut	Runtime	Netcut	Runtime	Netcut	Runtime
ibm01	1.017	1.024	0.988	1.024	0.996	1.056	**0.97**	1.073
ibm02	0.897	1.112	0.99	1.150	**0.894**	1.141	0.897	1.175
ibm03	0.989	0.996	0.973	1.103	0.977	1.048	**0.969**	1.029
ibm04	**0.952**	1.138	0.961	1.112	0.954	1.122	**0.952**	1.118
ibm05	**0.997**	1.035	1.001	1.048	0.999	1.014	**0.997**	1.028
ibm06	**1.021**	1.133	1.032	1.159	1.026	1.156	1.024	1.136
ibm07	1.022	1.041	1.04	1.063	1.041	1.073	**1.01**	1.009
ibm08	1.005	1.089	1.003	1.178	1.003	1.123	**1.002**	1.175
ibm09	0.992	1.048	0.991	1.183	0.991	1.067	**0.990**	1.036
ibm10	1.004	1.015	**0.982**	1.021	1.005	1.052	1.001	1.003
ibm11	1.013	1.080	1.01	1.123	1.004	1.112	**0.993**	1.072
ibm12	1.022	1.094	1.009	1.125	1.027	1.150	**1.006**	1.127
ibm13	0.959	1.064	0.997	1.106	**0.952**	1.090	0.971	1.044
ibm14	**0.973**	0.900	0.992	0.960	0.988	0.888	0.974	0.882
ibm15	1.006	1.041	1.006	1.035	1.009	1.040	**0.992**	1.034
ibm16	0.999	1.105	1.002	1.132	0.998	1.128	**0.996**	1.105
ibm17	1.013	0.942	1.006	1.007	**1.005**	0.960	1.006	0.957
ibm18	**0.995**	0.856	1.011	0.928	0.996	0.859	0.999	0.834
Avg.	0.993	1.040	1.000	1.081	0.992	1.060	**0.986**	1.046

runtime in this table are the normalized netcut and average runtime per run from the different combinations of scoring techniques. Here, both netcut and the runtime are normalized to the average netcut and runtime from 20 runs of hMetis on the flat netlist as reported in Table 5.8. Empirically, it appears that the NS+CS combination is slightly more effective than the other scoring combinations. It produces the best trade-off between solution quality and runtime among all scoring combinations. Hence, this combination is used in the remainder of the experiments in this section.

As the second partitioning experiment, the performance of our proposed clustering techniques, as a preprocessing step, is compared to hMetis and MLPart. In addition, the performance of best choice clustering (with lazy update) used as a preprocessing step is compared to the proposed techniques. This comparison is in terms of the final netcut value and runtime. For the proposed clustering techniques, two levels of clustering: net scoring and cluster scoring, the NS+CS combination, are used in the first and second levels of clustering, respectively. The library interface to hMetis is then called to perform partitioning on the clustered circuit. Finally, the results from hMetis are further refined using a flat FM partitioner. For each netlist, 20 runs with a 10% deviation from an exact bisection are performed. When calling hMetis, default parameters are used; 10 random starts are performed with V-cycling applied to the best result [45].

The experimental setups for hMetis, MLPart and best choice clustering used as preprocessing are similar. For hMetis, the parameters are identical to the experimental setup for proposed techniques (except that hMetis is provided with the original netlist). For MLPart, each run consists of 4 starts (this setup is from [20]). Finally, when using best choice clustering as an alternative preprocessing technique to NS+CS combination, the experimental setup is exactly the same as NS+CS combination. All reported results are for actual area partitioning.

In Table 5.8, partitioning results are provided for hMetis, MLPart, best choice (used as a preprocessor), and the NS+CS combination are given. In this table, the

Table 5.8: Actual area partitioning results on ISPD98 benchmark suite when solutions are constrained to be within 10% of bisection. All experiments were performed on 600 MHz Sun Ultra Sparc 3.

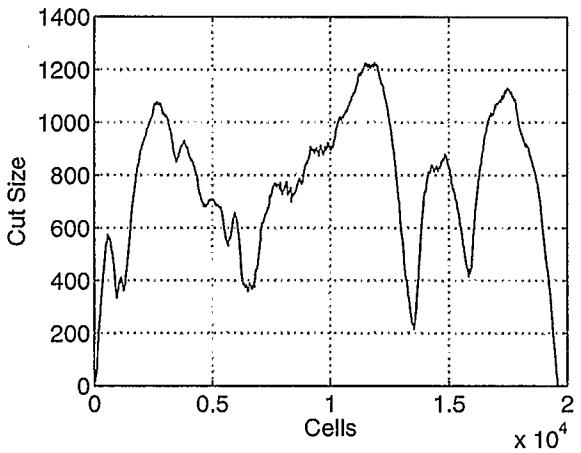
Circuit	hMetis		MLPart (comparison)		best choice + hMetis (comparison)		NS+CS + hMetis (comparison)	
	Avg.	Time	Avg.	Time	Avg.	Time	Avg.	Time
ibm01	236	6.2	0.962	0.903	1.030	0.952	0.970	1.008
ibm02	283	10.3	0.933	1.146	0.936	1.019	0.897	1.092
ibm03	770	13.6	0.949	1.188	1.019	0.893	0.969	0.996
ibm04	482	15.2	0.983	1.204	0.979	0.980	0.952	1.066
ibm05	1725	21.7	1.023	1.074	0.994	0.972	0.997	1.005
ibm06	370	17.3	1.084	1.185	1.008	1.061	1.024	1.136
ibm07	766	29.3	1.013	1.087	0.993	0.937	1.010	1.012
ibm08	1157	40.4	1.016	0.941	1.002	0.902	1.002	1.050
ibm09	524	30.4	1.048	1.232	1.002	0.933	0.990	1.003
ibm10	774	51.6	1.094	1.125	1.003	0.903	1.001	0.955
ibm11	715	44.6	1.036	1.197	1.008	0.955	0.993	1.049
ibm12	2039	67.2	1.103	0.963	0.998	1.061	1.006	1.099
ibm13	928	61.4	0.974	1.150	0.978	0.867	0.971	1.002
ibm14	1623	187.1	1.020	0.712	0.978	0.811	0.974	0.898
ibm15	1845	199.3	1.160	0.845	1.012	0.901	0.992	0.963
ibm16	1717	242	1.119	0.722	1.003	0.923	0.996	1.115
ibm17	2304	368.6	0.995	0.572	1.007	0.857	1.006	0.959
ibm18	1535	326.6	1.023	0.554	1.004	0.849	0.999	0.827
Avg.	1.000	1.000	1.030	0.989	0.998	0.932	0.986	1.013

average netcuts, *Avg.*, and the average runtime, *Time*, (the total runtime/number of runs) in seconds are given for hMetis under columns 2 and 3. For MLPart, best choice and NS+CS, only comparison results are given. In all comparisons, hMetis results are considered to be the base results, and the results from other partitioners are compared with hMetis.

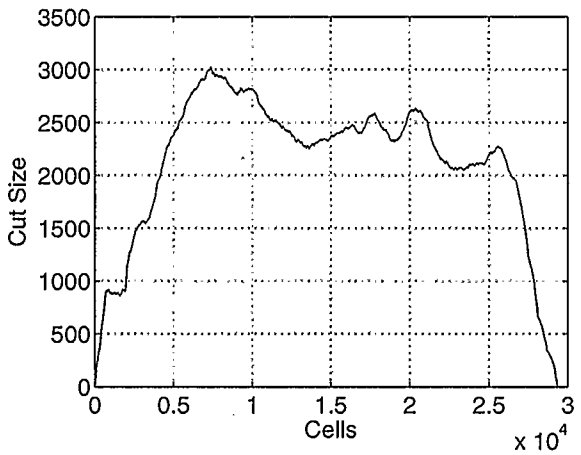
From Table 5.8, it can be seen that the proposed NS+CS clustering combination is effective to improve the netcut. When compared to hMetis, there are a total of 12 circuits with improved results. In comparison with MLPart, there are 15 improved results, where in 13 of them the improvement is about 1%. In comparison with

best choice there are 13 improved results. Overall in terms of netcut, the NS+CS combination is the only algorithm that shows improved results by about 1%. MLPart produces results that are 3% worse and best choice has negligible gain over hMetis.

During experiments with the NS+CS combination, it was discovered that the algorithm consistently does not perform well on certain circuits such as ibm05, where other circuits such as ibm02 consistently provide improved results. This discrepancy is probably a result of different circuit structures of the two benchmark circuits. For example, benchmark circuit ibm02 has 6 macro cells, each with an area of more than 1% of the total area of the chip (one macro block occupies more than 11% of the total area). On the other hand, ibm05 does not have any macro blocks. Another main difference between the two circuits is the structure of the nets and cells. In ibm05, 63% of the nets connect only two cells, where in ibm02, there are 55% of this type of nets (8% reduction). Also, the longest net (the net with the highest number of cells) in ibm05 connects 17 cells, where in ibm02 the longest net connects 134 cells. In ibm05, 74% of the cells have degrees lower than 5 (1 to 4), but in ibm02 this number is reduced to 51%. These differences between net and cell structures can indicate that the ibm05 has a more homogeneous structure, where cells are related together with the same connectivity. The spectral analysis technique, as mentioned in Section 5.3.2, was performed on circuits ibm02, and ibm05 to obtain a visual measure of the circuit structure. These linear orderings are illustrated in Figure 5.12(a) and Figure 5.12(b), respectively. This linear ordering for ibm02 shows many valleys and peaks, where the valleys correspond to a group of cells that are loosely connected. The linear ordering for ibm05 results in a very flat figure that means that there are no loosely connected groups of cells in this circuit.



(a) ibm02



(b) ibm05

Figure 5.12: Comparison between the circuit structure obtained by the linear ordering of circuits ibm02 and ibm05

Placement Experiments

For placement, the ICCAD04 mixed-size placement benchmarks were tested using four well-known academic placement tools: Capo10.1 [61], FengShui5.1 [10], FDP [76], and mPL6 [23]. Capo and FengShui are partitioning based placers. FDP and mPL6 are force directed placers. The main objective in this experiment was to confirm the efficacy of the proposed clustering algorithms within the context of an additional physical design problem other than partitioning. As in partitioning, the NS+CS clustering combination is used as a preprocessing step to each placement tool. Therefore, several revisions to the overall flow were made. Specifically,

1. The NS+CS clustering combination was first used to cluster each of the ICCAD04 placement benchmarks. In performing the clustering, not only the area of the clustered cells, but also the proper sizes of the clustered cells were preserved. Only standard cells were clustered together and the dimensions of a cluster were set such that the height corresponds to the height of a standard cell while the width of the cluster is enough to accommodate all of the clustered cells if linearly arranged. It should be noted that this shaping might result in a set of oddly shaped standard cells and proper shaping remains an avenue for further investigation.

2. The clustered mixed-size placement problems were run through each of the placement tools. Hence, for each clustered circuit, a legalized placement was obtained from each tool.

3. The placement produced by each tool was unclustered. In this phase, the cells were positioned in the flat netlist at the centre of their assigned cluster. The results of the unclustered placement are the re-introduction of cell overlap which must subsequently be removed. To accomplish this task, Capo10.1 in several

different modes[2] was invoked.

The total half perimeter wire length results for the four placers and the comparison when NS+CS clustering combination is used are given in Table 5.9. In Table 5.10 the runtime comparison for all the placers involved in this experiment are given. In each table, the column "Ori." gives the result of a placer, and the column "Comp." gives the comparison value of the placement result when using proposed clustering techniques over the result in column "Ori.".

Table 5.9: Placement wire length results on ICCAD04 mixed-size benchmark suite. The placement solution is measured from pin to pin.

Circuit	Capo10.1		FengShui5.1		FDP		mPL6		Overall
	Ori.	Comp.	Ori.	Comp.	Ori.	Comp.	Ori.	Comp.	Comp.
ibm01	2.45	0.981	2.36	0.990	2.50	1.097	2.14	0.961	1.007
ibm02	4.86	1.014	5.32	0.954	5.25	0.926	4.74	0.999	0.973
ibm03	7.28	0.994	7.97	0.968	7.80	1.071	6.94	0.941	0.994
ibm04	9.02	0.963	8.59	0.978	8.98	0.974	7.30	1.006	0.980
ibm05	10.02	0.997	9.80	1.008	10.85	0.939	9.37	0.996	0.985
ibm06	6.54	1.003	6.67	1.003	7.71	0.867	5.80	0.994	0.967
ibm07	12.32	0.899	11.20	0.978	12.04	0.971	9.81	0.962	0.952
ibm08	13.27	1.035	12.74	1.002	13.05	1.084	11.94	0.951	1.018
ibm09	14.34	0.972	14.19	0.985	15.98	0.971	12.35	0.998	0.981
ibm10	33.89	0.942	32.91	1.013	39.32	1.105	27.83	0.986	1.011
ibm11	21.13	0.969	20.26	0.945	22.71	0.928	17.72	0.977	0.955
ibm12	36.68	1.037	37.38	0.973	40.53	1.239	31.94	1.013	1.066
ibm13	26.28	0.937	25.56	0.951	26.53	0.987	23.08	0.946	0.955
ibm14	39.79	0.945	37.94	1.002	43.76	0.923	35.73	0.968	0.959
ibm15	53.66	0.971	51.07	0.961	59.78	0.967	47.21	0.968	0.967
ibm16	60.12	1.004	58.95	0.997	68.72	0.985	55.19	0.970	0.989
ibm17	75.24	0.928	68.87	0.985	80.39	0.867	64.94	0.981	0.940
ibm18	45.40	0.977	44.33	0.986	50.85	0.934	42.43	0.991	0.972
Avg.	26.24	0.976	25.34	0.982	28.71	0.991	23.14	0.978	0.982

- **Total Wire Length Comparison**

The efficiency of the NS+CS clustering combination, used as a preprocessing step, can be seen from the results presented in Table 5.9. From this table it can be seen that

[2]Capo 10.1 is equipped with several features that enable it to successfully legalize initially illegal placements produced by different placers.

Table 5.10: Placement runtime results on ICCAD04 mixed-size benchmark suite. The original runtime (Org.) of all placers are given in seconds. All experiments have been performed on dual core intel d820s.

Circuit	Capo10.1		FengShui5.1		FDP		mPL6	
	Ori.	Comp.	Ori.	Comp.	Ori.	Comp.	Ori.	Comp.
ibm01	255	0.94	130	1.17	118	1.73	163	1.11
ibm02	452	1.13	227	1.26	268	2.46	253	1.29
ibm03	555	1.09	265	1.17	274	2.05	258	1.20
ibm04	655	1.09	301	1.21	357	1.34	309	1.20
ibm05	558	1.05	315	1.19	538	1.07	320	1.19
ibm06	782	1.04	408	1.22	383	1.45	380	1.15
ibm07	1240	0.93	554	1.19	488	1.46	540	1.18
ibm08	1327	1.18	629	1.27	788	1.30	767	1.12
ibm09	1297	1.13	644	1.19	754	1.10	851	1.05
ibm10	3387	0.84	965	1.27	1054	2.35	1016	1.29
ibm11	1949	1.06	873	1.22	842	1.10	1042	1.16
ibm12	2817	1.62	986	1.26	991	3.79	1170	1.17
ibm13	2478	1.07	1128	1.21	1079	1.20	1190	1.28
ibm14	4448	1.24	2063	1.17	1438	1.67	2061	1.21
ibm15	6097	0.99	2478	1.28	2513	1.48	2487	1.27
ibm16	6578	0.97	2789	1.29	2478	3.04	3231	1.08
ibm17	7792	1.11	2986	1.26	2718	1.80	3076	1.23
ibm18	6077	1.02	3118	1.18	3351	1.00	3082	1.16
Avg.	2708	1.08	1159	1.22	1135	1.74	1233	1.19

for all placement algorithms, on average, the total wire length was reduced by almost 2%. For Capo10.1, FengShui5.1 and FDP, the placement results for 13 benchmarks were improved. For mPL6, 16 of the placement results were improved. The last column in Table 5.9 shows the overall improvement per circuit for all placers. It can be seen that for 14 circuits, improvements in total wire length are achieved. It should be mentioned that the flow and implementation of our preprocessing is far from optimal. Potentially, larger improvements in placement results could be obtained if the proposed clustering techniques are integrated into the placement algorithm.

- **Timing Comparison for Placement**

Table 5.10 shows the runtime comparison for all placers involved in this experiment. The original runtime for all placers is shown in the column labeled "Ori". The

comparison between the original runtime of each placer and total runtime with the preprocessing is shown under column "Comp.". For these placers, on average 8%, 22%, 74% and 19% increase in total runtime was obtained.

- **Comparison with Best Choice Used as Preprocessing**

Placer mPL6 uses best choice clustering scheme, but to show the effectiveness of proposed clustering techniques, best choice clustering with lazy update was implemented as a preprocessing scheme with the same setup as NS+CS clustering combination. The placement results when best choice is applied as a preprocessing step are

Table 5.11: Placement results on ICCAD04 mixed-size benchmark suite for best choice clustering algorithm. The placement solution by best choice is normalized to the each placer's flat solution.

Circuit	Capo10.1		FengShui5.1		FDP		mPL6		Overall comp.	
	WL	Time	WL	Time	WL	Time	WL	Time	WL	Time
ibm01	1.051	1.22	0.990	1.08	0.932	1.15	0.960	0.90	0.983	1.09
ibm02	1.362	1.56	0.954	1.19	0.945	2.60	0.978	1.18	1.060	1.63
ibm03	0.984	1.16	1.026	1.12	1.001	1.19	0.900	1.19	0.978	1.16
ibm04	0.958	1.08	0.936	1.14	0.987	2.74	0.987	1.16	0.967	1.53
ibm05	0.997	1.16	1.003	1.11	0.947	2.17	0.991	1.04	0.985	1.37
ibm06	1.034	1.16	0.979	1.16	1.092	2.15	0.983	1.20	1.022	1.42
ibm07	0.993	1.13	0.995	1.12	1.021	2.72	0.960	1.17	0.992	1.54
ibm08	1.012	1.05	1.010	1.17	1.049	2.24	1.009	1.17.	1.020	1.41
ibm09	0.985	1.23	0.922	1.15	1.002	1.24	0.976	1.11	0.971	1.18
ibm10	0.932	0.81	0.995	1.15	0.967	4.96	1.009	1.31	0.976	2.06
ibm11	0.968	1.19	0.958	1.17	1.139	2.48	0.990	1.11	1.014	1.49
ibm12	1.136	1.13	1.044	1.21	0.973	2.82	1.027	1.17	1.045	1.58
ibm13	0.986	0.85	0.921	1.17	1.026	1.20	0.962	1.29	0.974	1.13
ibm14	0.961	1.03	0.997	1.18	0.929	2.30	0.982	1.22	0.967	1.43
ibm15	0.990	1.31	0.980	1.27	1.068	2.56	0.945	1.27	0.996	1.60
ibm16	0.969	1.05	0.997	1.27	0.999	2.60	0.953	1.10	0.979	1.50
ibm17	0.960	1.00	1.029	1.26	0.980	2.73	0.986	1.28	0.989	1.57
ibm18	0.988	1.02	1.006	1.17	0.927	1.80	0.965	1.15	0.972	1.28
Avg.	1.015	1.12	0.986	1.17	0.999	2.31	0.976	1.17	0.994	1.44

reported in Table 5.11. For each benchmark circuit, best choice is used to preprocess the netlist to the same clustering ratio as NS+CS clustering combination. The subsequent experiment setup, in terms of cluster dimensions and legalization, is exactly

the same as the NS+CS combination. In Table 5.11, the wire length and runtime by using best choice as a preprocessing step are first normalized to the results from each flat placer and then reported in Columns "WL" and "Time", respectively. From Table 5.11, it can be seen that the application of best choice as a preprocessing step also improves the placement wire length for most test placers. However, compared to the wire length results by the NS+CS combination in Table 5.9, the results by best choice are about 1% worse on average. Furthermore, the overall runtime by best choice is higher than that by NS+CS combination even though best choice in itself is faster on clustering the circuit.

5.5.3 Experimental Results for the Score-Based Net Cluster Technique

In this set of experiments, the score-based net cluster algorithm is investigated and applied in the context of circuit placement. In the following, the proposed score-based net cluster algorithm is referred to as SNC. The quality of clusters depends on many factors. Usually the only metric used to determine this quality is the final placement results. Final results are good indications, but placement is a complicated process and many factors can influence the final placement results. In this section, two aspects pertaining to the quality of clusters are considered. These are: the connectivity of cells in a cluster, and the final results.

Experiments on Connectivity

As a general rule, clustering algorithms need to be able to recognize clusters with high connectivity and group them together. In [15], several metrics are proposed to measure the connectivity. In this thesis, two measures for connectivity are tested. The first measure uses the relative net clustering ratio versus cell clustering ratio. A lower net clustering ratio means that most of the connections between the cells are within clusters, indicating that the algorithm has been successful. For example, it has been

demonstrated that a clustering algorithm that can produce a low net clustering ratio can benefit the routability, area, and power consumption in hierarchical FPGA designs [70]. The second set of experiments explore the change in net scores after performing clustering using the proposed SNC clustering technique, and using FirstChoice, and best choice.

- **Net Clustering Ratio**

In these experiments, the clustering ratio achieved by the proposed algorithm is compared to those obtained by best choice and FirstChoice clustering algorithms. In Table 5.12, the clustering results obtained by using the three algorithms on ICCAD04 benchmark circuits [2] are reported. The cell and net clustering ratios indicate the percentage of the clustered netlist size to original netlist size:

$$\text{Cell Clustering Ratio (CCR)} = \frac{\#\text{ cells in clustered netlist}}{\#\text{ cells in original netlist}},$$

and similarly,

$$\text{Net Clustering Ratio (NCR)} = \frac{\#\text{ nets in clustered netlist}}{\#\text{ nets in original netlist}}.$$

It should be noted that the results in Table 5.12 are from one level of clustering, and all cell clustering ratios were matched to the clustering ratio achieved by the proposed SNC clustering technique. From Table 5.12, it can be seen that the net clustering ratios achieved by the proposed technique are the lowest compared to best choice and FirstChoice techniques. It can also be seen that the clustering ratios for different benchmark circuits vary. This is due to different inner structures of the benchmark circuits.

- **Net Scores**

In this section, a comparison between the net scores of the circuit after one level of clustering using the proposed clustering technique, best choice and FirstChoice is

Table 5.12: Clustering ratio comparison for ICCAD04 benchmark circuits

Circuit	CCR	NCR		
		SNC	Best choice	FirstChoice
ibm01	0.511	**0.623**	0.670	0.674
ibm02	0.650	0.703	**0.697**	0.751
ibm03	0.604	**0.700**	0.727	0.732
ibm04	0.627	**0.727**	0.750	0.764
ibm05	0.723	**0.734**	0.749	0.765
ibm06	0.599	**0.684**	0.713	0.726
ibm07	0.604	**0.665**	0.713	0.736
ibm08	0.598	**0.636**	0.671	0.710
ibm09	0.631	**0.719**	0.758	0.765
ibm10	0.572	**0.653**	0.679	0.708
ibm11	0.637	**0.743**	0.779	0.782
ibm12	0.640	**0.718**	0.750	0.766
ibm13	0.591	**0.707**	0.755	0.752
ibm14	0.633	**0.690**	0.716	0.753
ibm15	0.632	**0.728**	0.776	0.781
ibm16	0.587	**0.656**	0.686	0.727
ibm17	0.684	**0.740**	0.760	0.792
ibm18	0.655	**0.691**	0.748	0.780
Average	0.621	**0.695**	0.728	0.748

given. A net's score shows how desirable it is to cluster a net. First, average positive net scores, Avg, and standard deviations, σ, for ICCAD04 benchmark circuits before clustering are calculated and shown in columns 2 and 3 in Table 5.13. These scores are determined before performing any clustering. Then, the average positive net scores and standard deviations for the clustered circuits after one level of clustering using the proposed technique, best choice, and FirstChoice are also reported as a comparison with the original scores. From Table 5.13, it can be seen that the proposed algorithm results in the lowest average and standard deviation of the scores. This means that by performing one level of clustering, most nets that should have been clustered have been identified and clustered. At this stage the score distribution becomes narrower. The nets' score distribution for circuit ibm12 before and after clustering using the mentioned techniques is shown in Figure 5.13. In this figure, the score distribution for the proposed algorithm has the sharpest rise. In addition, only 2 benchmarks have

Table 5.13: Net score comparison for ICCAD04 benchmark circuits. The average positive net scores and standard deviations for the proposed algorithm, best choice and FirstChoice are shown as comparison.

Circuit	Nets' scores before clustering		SNC (comp.)		best choice (comp.)		FirstChoice (comp.)	
	Avg $\times 10^3$	$\sigma \times 10^3$	Avg	σ	Avg	σ	Avg	σ
ibm01	1.460	6.16	**0.503**	**0.417**	0.74	0.535	0.859	0.693
ibm02	0.759	4.368	**0.177**	**0.316**	0.502	0.667	0.661	0.851
ibm03	1.930	9.218	**0.632**	**0.584**	0.74	0.632	0.781	0.767
ibm04	1.480	6.574	**0.563**	**0.622**	0.6	0.713	0.730	0.842
ibm05	1.010	4.96	**0.095**	0.381	0.203	**0.319**	0.644	0.893
ibm06	1.340	7.212	**0.475**	**0.488**	0.753	0.708	0.956	0.918
ibm07	1.420	7.588	**0.402**	**0.499**	0.556	0.5	0.739	0.831
ibm08	0.907	5.698	**0.383**	**0.375**	0.751	0.624	0.973	0.879
ibm09	1.700	7.255	**0.625**	**0.788**	0.751	0.843	0.794	0.842
ibm10	1.030	5.323	**0.248**	**0.303**	0.409	0.41	0.709	0.918
ibm11	1.710	8.125	**0.633**	0.787	0.667	**0.749**	0.771	0.803
ibm12	0.765	4.144	**0.273**	**0.378**	0.494	0.592	0.829	1.899
ibm13	1.490	7.548	**0.663**	**0.638**	0.727	0.707	0.773	0.752
ibm14	1.200	6.558	**0.355**	**0.364**	0.485	0.454	0.772	0.845
ibm15	1.530	7.839	**0.671**	**0.641**	0.851	0.756	0.904	0.846
ibm16	1.080	6.226	**0.345**	**0.417**	0.598	0.617	0.902	1.074
ibm17	0.759	4.473	**0.346**	**0.390**	0.556	0.578	0.901	1.027
ibm18	0.802	4.688	**0.345**	**0.412**	0.595	0.566	0.917	0.883
Avg.	1.243	6.331	**0.430**	**0.514**	0.610	0.622	0.812	0.892

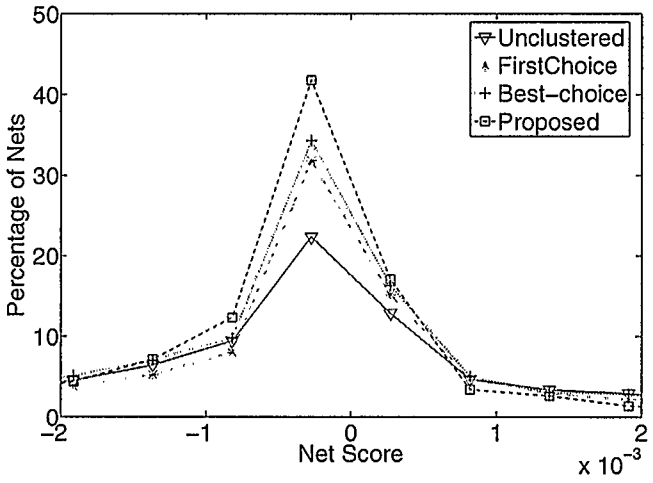


Figure 5.13: Net score distribution for ibm12

lower standard deviations and the average net scores are higher for all the circuits when best choice is used. Compared to FirstChoice all circuits have improved average and standard deviation when using the proposed algorithm.

Experiments on Placement

The final test was performed to compare final placement results of the proposed SNC algorithm with other techniques. The SNC clustering algorithm is implemented and tested as a preprocessor for large scale netlists. Then, five publicly available academic placers are used to produce initial placements on the clustered netlist. Finally, the clustered netlist and the initial placement results are mapped back to the original netlist, and another round of legalization and detailed placement is performed to obtain legal placements for the original netlist. Details of this framework are as follows:

1. During the clustering process, the maximum cluster area is set to be 5 times larger than the average standard cell area. The dimension of a cluster is set to accommodate all cells if they are linearly aligned. Three different shapes for clusters were implemented: all cells were aligned in one row, put in two rows, and a square shape with an area equal to the sum of the areas of all the cells in the cluster. Three placers Capo10.2, mPL6 and FengShui5.1 were used to obtain placement results for each cluster shape. The results show that when cluster shapes are square or the cells are put into two rows, the quality of the final results deteriorated. Based on the above experiments, it was decided to construct clusters with the same height as standard cells. It should be noted that this shaping of clusters might produce some long cells in the clustered netlist. As a result, only one level of clustering is performed, since more levels of clustering will produce very long cells that can make the placement problem more difficult to solve. The experiments also show that the placement results

Table 5.14: ISPD05 benchmark circuit statistics

Circuit	#Total objs	#Mov. objs	#Fixed objs	#Nets	#Total pins	Design density	Design utility	Peri. I/Os
adaptec1	211447	210904	543	221142	944053	75.71%	57.34%	480
adaptec2	255023	254457	566	266009	1069482	78.56%	44.32%	407
adaptec3	451650	450927	723	466758	1875039	74.53%	33.66%	0
adaptec4	496045	494716	1329	515951	1912420	62.67%	27.23%	0
bigblue1	278164	277604	560	284479	1144691	54.19%	44.67%	528
bigblue2	557866	534782	23084	577235	2122282	61.80%	37.94%	0
bigblue3	1096812	1095519	1293	1123170	3833218	85.65%	56.68%	0
bigblue4	2177353	2169183	8170	2229886	8900078	65.30%	44.35%	0

deteriorate when more levels of clustering are performed; this is because of the odd-shaped cells.

2. When a placer is invoked to run on the clustered netlist, the generated placement result is a legal placement.

3. When a cluster is unclustered and its component cells are mapped back into the original netlist, the positions of its cells are set to be at the bottom-left corner of the cluster. Since this process introduces cell overlaps, further legalization and detailed placement is required which is performed by Capo10.1 with the option "noCapo" for ICCAD04 benchmarks. For ISPD05 benchmarks, the FastPlace legalizer was used due to its high efficiency [75].

The placement results generated by the above framework are compared with the results from each placer directly. The total runtime for this framework is the summation of the time for each step. In the following experiments, ICCAD04 mixed-size placement benchmarks [2] and ISPD05 placement contest benchmarks [7] are chosen as the test circuits. The statistics of ISPD05 benchmark circuits are shown in Table 5.14. In this table, "# Total objs" is the total number of cells in the circuit. "# Mov. objs" is the number of movable cells. "# Fixed objs" is the number of fixed cells. Design density is the ratio of area sum of total objects over the area of

placement region. Design utility is the ratio of area sum of only movable objects over the available free space. Four well-known academic placers, Capo10.1 [61], FengShui5.1 [10], mPL6 [23], and NTUplace3-LE [25], are tested by using the proposed clustering algorithm as a preprocessing step for ICCAD04 benchmark circuits. For ISPD05 benchmark circuits, FengShui5.1 was replaced by FastPlace3.0. Capo10.1 and FengShui5.1 are partitioning-based placers. NTUplace3-LE, mPL6, and FastPlace3.0 are force-directed, or analytical placers. All placers use clustering, directly or indirectly, to achieve manageable circuit sizes. All experiments are performed on a dual-processor 2.8 GHz Xeon with 4 GB RAM running RedHat Linux.

- **Effects of Skipping Long nets on Clustering**

In Table 5.15, the statistics of clustering results without and with skipping long nets on ISPD05 benchmark circuits are given. In Table 5.15, the column "Max net deg." lists the maximum net degree for each benchmark circuit. It can be seen that in ISPD05 circuits, there are some very long nets inside, for example, for circuit bigblue4, the longest net contains over 20000 cells. If such long nets are treated as the short nets in SNC algorithm, the runtime will definitely become much more expensive. In column "CCR", the cell clustering ratio is given, and similarly, column "NCR" gives the net clustering ratio. It can be seen that the two schemes, without skipping long nets and skipping long nets, almost produce the same cell and net clustering ratios, i.e., the clustering effects in terms of clustering ratios are equal. However, the runtime for the two schemes are totally different. The last column "Runtime comp."in Table 5.15 reports the runtime comparison between two schemes. It can be seen that when skipping the long nets, the runtime is decreased by around 50%.

- **Placement Wire Length and Runtime Results**

In this set of experiments, the efficiency of the SNC algorithm in terms of reducing the total half perimeter wire length and runtime is tested.

Table 5.15: Statistics of clustering results with and without skipping long nets on ISPD05 benchmark circuits

Circuit	Max net deg.	Without skipping long nets			Skipping long nets			Runtime comp.
		CCR	NCR	Runtime(s)	CCR	NCR	Runtime(s)	
adaptec1	2271	63.80	67.15	161	63.81	69.06	111	0.69
adaptec2	1935	57.53	60.87	225	57.64	62.49	168	0.75
adaptec3	3713	56.76	61.39	543	56.77	62.08	328	0.60
adaptec4	3974	53.81	57.27	627	53.82	57.87	305	0.49
bigblue1	2621	61.25	65.04	288	61.28	65.68	197	0.68
bigblue2	11869	59.65	63.86	1949	59.77	64.18	790	0.41
bigblue3	7623	50.72	52.66	2534	50.74	52.88	860	0.34
bigblue4	20766	56.12	57.98	6212	56.17	58.35	3066	0.49
Average	–	57.45	60.78	–	57.50	61.57	–	0.56

In Figures 5.14 and 5.15, the placement wire length and runtime comparisons on ICCAD04 benchmark circuits are plotted, respectively. From Table 5.16 to Table 5.23, the placement results of different placers, Capo10.1, FengShui5.1, mPL6, and NTUplace3-LE, on ICCAD04 benchmark circuits are reported respectively. In Table 5.24, the overall placement wire length and runtime comparisons on ICCAD04 benchmark circuits are given.

In Figures 5.16 and 5.17, the placement wire length and runtime comparisons on ISPD05 benchmark circuits are plotted, respectively. From Table 5.25 to Table 5.32, the placement results of different placers, Capo10.1, FastPlace3.0, mPL6, and NTUplace3-LE, on ISPD05 benchmark circuits are reported respectively. In Table 5.33, the overall placement wire length and runtime comparisons on ISPD05 benchmark circuits are given.

For each of the test placers, the experimental data are given in two tables. The first table reports the final placement results with and without the SNC clustering application, and the comparison between these two modes. The second table reports the runtime percentage of each stage when the SNC algorithm is used as a preprocessing step. For example, Tables 5.16 and 5.20 gives the placement results of placer Capo10.1 on ICCAD04 benchmark circuits. In Table 5.16, under "Origi-

nal Capo10.1", the original half perimeter wire length results and runtime of placer Capo10.1 are given in column "HPWL" and "Runtime" respectively. Under "SNC + Capo10.1", the placement results when SNC is applied as a preprocessing step are given in a similar way. The comparison results between "SNC + Capo10.1" and "Original Capo10.1" are given under "Comparison".

- ICCAD04 Benchmark Circuits

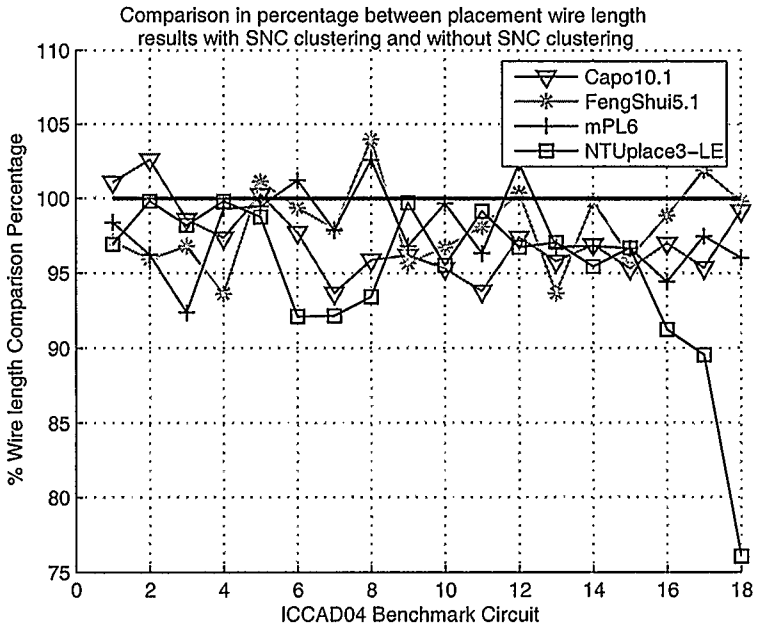


Figure 5.14: Placement wire length comparison in percentage on ICCAD04 benchmark circuits

From Tables 5.16 to 5.24, it can be seen that for ICCAD04 benchmark circuits, the proposed clustering algorithm consistently improves the placement HPWL results for all tested placers on most benchmark circuits, with a comparable or lower runtime. The overall average HPWL and runtime improvements for the 4 test placers are 3% and 5%, respectively. For Capo10.1, out of 18 test circuits, the HPWL results for 15 circuits are improved. The average improvement is about 3%, and the maximum improvement is up to 6% for benchmark ibm07. Furthermore, the runtime is also

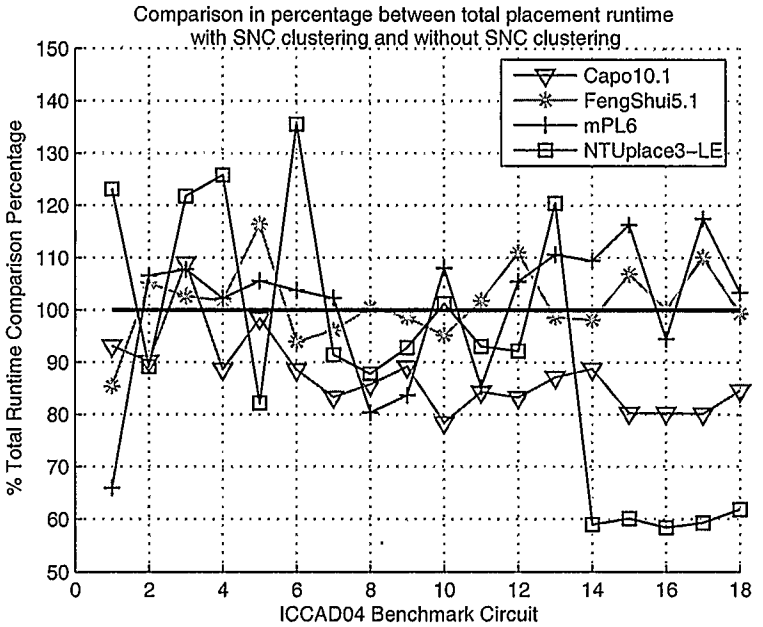


Figure 5.15: Placement runtime comparison in percentage on ICCAD04 benchmark circuits

Table 5.16: Placement results of placer Capo10.1 on ICCAD04 benchmark circuits

ICCAD04	Original Capo10.1		SNC + Capo10.1		Comparison	
circuit	HPWL	Runtime	HPWL	Runtime	HPWL	Runtime
ibm01	2.45E+06	255	2.47E+06	237	1.011	0.931
ibm02	4.86E+06	452	4.99E+06	407	1.026	0.902
ibm03	7.28E+06	555	7.18E+06	605	0.986	1.090
ibm04	9.02E+06	655	8.78E+06	581	0.974	0.887
ibm05	1.00E+07	558	1.00E+07	549	1.002	0.984
ibm06	6.54E+06	782	6.39E+06	692	0.977	0.886
ibm07	1.23E+07	1240	1.15E+07	1033	0.937	0.834
ibm08	1.33E+07	1327	1.27E+07	1138	0.959	0.857
ibm09	1.43E+07	1297	1.38E+07	1157	0.962	0.892
ibm10	3.39E+07	3387	3.23E+07	2657	0.953	0.784
ibm11	2.11E+07	1949	1.98E+07	1644	0.938	0.843
ibm12	3.67E+07	2817	3.57E+07	2347	0.974	0.833
ibm13	2.63E+07	2478	2.52E+07	2159	0.958	0.871
ibm14	3.98E+07	4448	3.86E+07	3947	0.969	0.887
ibm15	5.37E+07	6097	5.11E+07	4897	0.953	0.803
ibm16	6.01E+07	6578	5.83E+07	5282	0.970	0.803
ibm17	7.52E+07	7792	7.17E+07	6248	0.954	0.802
ibm18	4.54E+07	6077	4.50E+07	5142	0.992	0.846
Average	–	–	–	–	0.972	0.874

Table 5.17: Placement results of placer FengShui5.1 on ICCAD04 benchmark circuits

ICCAD04	Original FengShui5.1		SNC + FengShui5.1		Comparison	
circuit	HPWL	Runtime	HPWL	Runtime	HPWL	Runtime
ibm01	2.36E+06	130	2.29E+06	111	0.971	0.854
ibm02	5.32E+06	227	5.11E+06	238	0.959	1.051
ibm03	7.97E+06	265	7.71E+06	272	0.968	1.026
ibm04	8.59E+06	301	8.04E+06	306	0.936	1.018
ibm05	9.80E+06	315	9.91E+06	367	1.011	1.163
ibm06	6.67E+06	408	6.63E+06	384	0.993	0.939
ibm07	1.12E+07	554	1.10E+07	532	0.980	0.961
ibm08	1.27E+07	629	1.32E+07	632	1.040	1.004
ibm09	1.42E+07	644	1.36E+07	634	0.956	0.986
ibm10	3.29E+07	965	3.18E+07	917	0.967	0.951
ibm11	2.03E+07	873	1.99E+07	889	0.981	1.018
ibm12	3.74E+07	986	3.75E+07	1093	1.004	1.109
ibm13	2.56E+07	1128	2.39E+07	1112	0.936	0.986
ibm14	3.79E+07	2063	3.79E+07	2023	0.998	0.981
ibm15	5.11E+07	2478	4.90E+07	2647	0.960	1.068
ibm16	5.89E+07	2789	5.83E+07	2794	0.988	1.002
ibm17	6.89E+07	2986	7.02E+07	3286	1.019	1.100
ibm18	4.43E+07	3118	4.42E+07	3096	0.998	0.993
Average	–	–	–	–	0.981	1.012

Table 5.18: Placement results of placer mPL6 on ICCAD04 benchmark circuits

ICCAD04	Original mPL6		SNC + mPL6		Comparison	
circuit	HPWL	Runtime	HPWL	Runtime	HPWL	Runtime
ibm01	2.14E+06	163	2.10E+07	107	0.984	0.660
ibm02	4.74E+06	253	4.56E+07	270	0.962	1.066
ibm03	6.94E+06	258	6.41E+07	278	0.924	1.078
ibm04	7.30E+06	309	7.25E+07	316	0.993	1.022
ibm05	9.37E+06	320	9.31E+07	338	0.994	1.055
ibm06	5.80E+06	380	5.87E+07	395	1.012	1.038
ibm07	9.81E+06	540	9.60E+07	552	0.978	1.023
ibm08	1.19E+07	767	1.22E+08	617	1.026	0.804
ibm09	1.24E+07	851	1.20E+08	712	0.968	0.837
ibm10	2.78E+07	1016	2.77E+08	1097	0.997	1.080
ibm11	1.77E+07	1042	1.71E+08	889	0.963	0.853
ibm12	3.19E+07	1170	3.27E+08	1234	1.024	1.054
ibm13	2.31E+07	1190	2.23E+08	1316	0.968	1.106
ibm14	3.57E+07	2061	3.46E+08	2255	0.968	1.094
ibm15	4.72E+07	2487	4.56E+08	2892	0.967	1.163
ibm16	5.52E+07	3231	5.21E+08	3053	0.944	0.945
ibm17	6.49E+07	3076	6.33E+08	3612	0.975	1.174
ibm18	4.24E+07	3082	4.07E+08	3185	0.960	1.033
Average	–	–	–	–	0.978	1.005

Table 5.19: Placement results of placer NTUplace3-LE on ICCAD04 benchmark circuits

ICCAD04	Original NTUplace3-LE		SNC + NTUplace3-LE		Comparison	
circuit	HPWL	Runtime	HPWL	Runtime	HPWL	Runtime
ibm01	2.23E+06	47	2.16E+06	58	0.969	1.231
ibm02	4.64E+06	135	4.63E+06	120	0.998	0.892
ibm03	6.64E+06	132	6.52E+06	161	0.982	1.218
ibm04	7.47E+06	164	7.46E+06	206	0.998	1.258
ibm05	9.69E+06	403	9.57E+06	331	0.987	0.822
ibm06	6.12E+06	165	5.64E+06	224	0.921	1.356
ibm07	1.03E+07	395	9.47E+06	361	0.922	0.915
ibm08	1.26E+07	476	1.17E+07	418	0.934	0.878
ibm09	1.21E+07	522	1.21E+07	485	0.997	0.928
ibm10	2.95E+07	546	2.82E+07	553	0.955	1.012
ibm11	1.78E+07	658	1.76E+07	613	0.991	0.931
ibm12	3.25E+07	820	3.14E+07	756	0.967	0.922
ibm13	2.27E+07	738	2.20E+07	889	0.970	1.204
ibm14	3.56E+07	3048	3.40E+07	1798	0.954	0.590
ibm15	4.69E+07	4152	4.54E+07	2497	0.967	0.601
ibm16	5.79E+07	3995	5.28E+07	2334	0.912	0.584
ibm17	7.67E+07	8288	6.87E+07	4918	0.895	0.593
ibm18	5.55E+07	9338	4.22E+07	5776	0.761	0.618
Average	–	–	–	–	0.949	0.920

Table 5.20: Runtime statistics for different stages of **SNC + Capo10.1** framework on ICCAD04 benchmark circuits

ICCAD04	SNC Clustering		Initial Placement		Mapping Placement		Legalization	
circuit	runtime	ratio%	runtime	ratio%	runtime	ratio%	runtime	ratio%
ibm01	3	1.27	202.44	85.41	0.1	0.04	31.47	13.28
ibm02	13	3.19	325.65	79.97	0.18	0.04	68.4	16.80
ibm03	13	2.15	513.68	84.95	0.2	0.03	77.8	12.87
ibm04	12	2.07	478.33	82.35	0.25	0.04	90.27	15.54
ibm05	20	3.64	424.15	77.21	0.28	0.05	104.93	19.10
ibm06	16	2.31	570.92	82.47	0.28	0.04	105.05	15.18
ibm07	16	1.55	858.98	83.12	0.4	0.04	157.99	15.29
ibm08	43	3.78	923.31	81.14	0.44	0.04	171.14	15.04
ibm09	22	1.90	944.41	81.65	0.47	0.04	189.81	16.41
ibm10	39	1.47	2317.51	87.22	0.57	0.02	300.12	11.29
ibm11	26	1.58	1355.72	82.46	0.6	0.04	261.74	15.92
ibm12	49	2.09	1954.84	83.30	0.61	0.03	342.33	14.59
ibm13	53	2.45	1780.9	82.49	0.73	0.03	324.38	15.02
ibm14	58	1.47	3268.35	82.82	1.26	0.03	618.92	15.68
ibm15	124	2.53	3980.25	81.28	1.39	0.03	791.06	16.15
ibm16	96	1.82	4256.69	80.59	1.57	0.03	927.81	17.57
ibm17	148	2.37	5112.03	81.82	1.68	0.03	985.87	15.78
ibm18	105	2.04	4191.1	81.51	1.84	0.04	843.74	16.41
Average	–	2.20	–	82.32	–	0.04	–	15.44

Table 5.21: Runtime statistics for different stages of **SNC + FengShui5.1** framework on ICCAD04 benchmark circuits

ICCAD04	SNC Clustering		Initial Placement		Mapping Placement		Legalization	
circuit	runtime	ratio%	runtime	ratio%	runtime	ratio%	runtime	ratio%
ibm01	3	2.70	75.81	68.12	0.11	0.10	32.37	29.09
ibm02	13	5.45	161.41	67.70	0.18	0.08	63.82	26.77
ibm03	13	4.78	178.99	65.87	0.2	0.07	79.55	29.27
ibm04	12	3.92	203.52	66.48	0.24	0.08	90.37	29.52
ibm05	20	5.45	235.23	64.09	0.28	0.08	111.51	30.38
ibm06	16	4.17	263.37	68.68	0.28	0.07	103.85	27.08
ibm07	16	3.01	356.92	67.05	0.4	0.08	159.03	29.87
ibm08	43	6.81	407.45	64.50	0.44	0.07	180.77	28.62
ibm09	22	3.47	433.75	68.39	0.46	0.07	178.06	28.07
ibm10	39	4.25	591.85	64.52	0.6	0.07	285.83	31.16
ibm11	26	2.92	603.7	67.91	0.62	0.07	258.61	29.09
ibm12	49	4.48	690.41	63.14	0.62	0.06	353.46	32.32
ibm13	53	4.77	729.71	65.65	0.74	0.07	328.09	29.52
ibm14	58	2.87	1354.36	66.94	1.28	0.06	609.59	30.13
ibm15	124	4.68	1676.9	63.35	1.4	0.05	844.6	31.91
ibm16	96	3.44	1783.7	63.85	1.56	0.06	912.38	32.66
ibm17	148	4.50	2148.69	65.38	1.7	0.05	987.97	30.06
ibm18	105	3.39	2134.87	68.96	1.87	0.06	854.15	27.59
Average	–	4.17	–	66.14	–	0.07	–	29.62

Table 5.22: Runtime statistics for different stages of **SNC + mPL6** framework on ICCAD04 benchmark circuits

ICCAD04	SNC Clustering		Initial Placement		Mapping Placement		Legalization	
circuit	runtime	ratio%	runtime	ratio%	runtime	ratio%	runtime	ratio%
ibm01	3	2.80	70.83	66.06	0.1	0.09	33.39	31.14
ibm02	13	4.82	196.43	72.78	0.16	0.06	60.47	22.40
ibm03	13	4.68	193.85	69.79	0.19	0.07	70.9	25.53
ibm04	12	3.80	215.43	68.24	0.24	0.08	88.25	27.96
ibm05	20	5.93	212.09	62.84	0.27	0.08	105.44	31.24
ibm06	16	4.06	268.78	68.14	0.27	0.07	109.69	27.81
ibm07	16	2.90	366.82	66.44	0.38	0.07	169.28	30.66
ibm08	43	6.98	402.97	65.40	0.43	0.07	170.21	27.62
ibm09	22	3.09	501.00	70.37	0.45	0.06	188.92	26.54
ibm10	39	3.56	789.00	71.94	0.56	0.05	268.69	24.50
ibm11	26	2.93	612.93	68.98	0.59	0.07	249.69	28.10
ibm12	49	3.97	878.88	71.28	0.6	0.05	305.2	24.75
ibm13	53	4.03	915.35	69.59	0.68	0.05	347.05	26.38
ibm14	58	2.57	1550.42	68.80	1.24	0.06	645.21	28.63
ibm15	124	4.29	1938.53	67.07	1.34	0.05	827.67	28.64
ibm16	96	3.15	1977.31	64.81	1.54	0.05	977.63	32.04
ibm17	148	4.10	2417.24	66.95	1.6	0.04	1045.24	28.95
ibm18	105	3.30	2249.91	70.68	1.83	0.06	828.11	26.02
Average	–	3.94	–	68.34	–	0.06	–	27.72

Table 5.23: Runtime statistics for different stages of **SNC + NTUplace3-LE** framework on ICCAD04 benchmark circuits

ICCAD04	SNC Clustering		Initial Placement		Mapping Placement		Legalization	
circuit	runtime	ratio%	runtime	ratio%	runtime	ratio%	runtime	ratio%
ibm01	3	5.18	24	41.47	0.11	0.19	30.76	53.15
ibm02	13	10.80	50	41.54	0.18	0.15	57.2	47.52
ibm03	13	8.09	69	42.92	0.21	0.13	78.55	48.86
ibm04	12	5.82	86	41.68	0.24	0.12	108.09	52.39
ibm05	20	6.04	204	61.56	0.27	0.08	107.09	32.32
ibm06	16	7.15	104	46.50	0.28	0.13	103.38	46.22
ibm07	16	4.43	172	47.61	0.4	0.11	172.84	47.85
ibm08	43	10.29	201	48.09	0.44	0.11	173.51	41.51
ibm09	22	4.54	268	55.31	0.47	0.10	194.08	40.05
ibm10	39	7.06	219	39.62	0.6	0.11	294.13	53.21
ibm11	26	4.24	323	52.73	0.61	0.10	263	42.93
ibm12	49	6.48	390	51.59	0.65	0.09	316.28	41.84
ibm13	53	5.96	483	54.34	0.72	0.08	352.06	39.61
ibm14	58	3.23	1125	62.56	1.34	0.07	613.92	34.14
ibm15	124	4.97	1529	61.24	1.46	0.06	842.16	33.73
ibm16	96	4.11	1254	53.72	1.63	0.07	982.55	42.09
ibm17	148	3.01	3562	72.43	1.72	0.03	1206.36	24.53
ibm18	105	1.82	4800	83.11	1.93	0.03	868.57	15.04
Average	–	5.73	–	53.22	–	0.10	–	40.94

Table 5.24: Overall wire length and runtime comparison results of 4 test placers on ICCAD04 benchmark circuits

ICCAD04 circuit	Overall Comparison	
	HPWL	Runtime
ibm01	0.984	0.919
ibm02	0.986	0.978
ibm03	0.965	1.103
ibm04	0.975	1.046
ibm05	0.999	1.006
ibm06	0.976	1.054
ibm07	0.954	0.933
ibm08	0.990	0.886
ibm09	0.971	0.911
ibm10	0.968	0.957
ibm11	0.968	0.911
ibm12	0.992	0.980
ibm13	0.958	1.042
ibm14	0.972	0.888
ibm15	0.962	0.909
ibm16	0.954	0.833
ibm17	0.961	0.917
ibm18	0.928	0.873
Average	0.970	0.953

reduced by about 13% on average. For FengShui5.1, the HPWL results for 14 circuits are improved. The average improvement is about 2%. The maximum improvement is about 6% for circuits ibm04 and ibm13. However, the runtime is increased by around 1% on average. For mPL6, there are overall 15 improved HPWL results. The average improvement is about 2% and maximum improvement is 7.6% for benchmark ibm03. The runtime increase is about 0.5% on average. For NTUplace3-LE, the HPWLs for all test circuits are improved, with an average 5% improvement. The runtime is also reduced by 8% on average. Especially, for the relatively big ICCAD04 benchmark circuits, ibm14 to ibm18, both the HPWL and runtime are significantly improved. For example, for circuit ibm18, the HPWL is improved by about 24% and the runtime is decreased by 32%.[3]

[3]It should be noted that there are two different versions of NTUplace3, called NTUplace3 and NTUplace3-LE respectively. NTUplace3 is based on the log-sum-exp wire length model, which is

- ISPD05 Benchmark Circuits

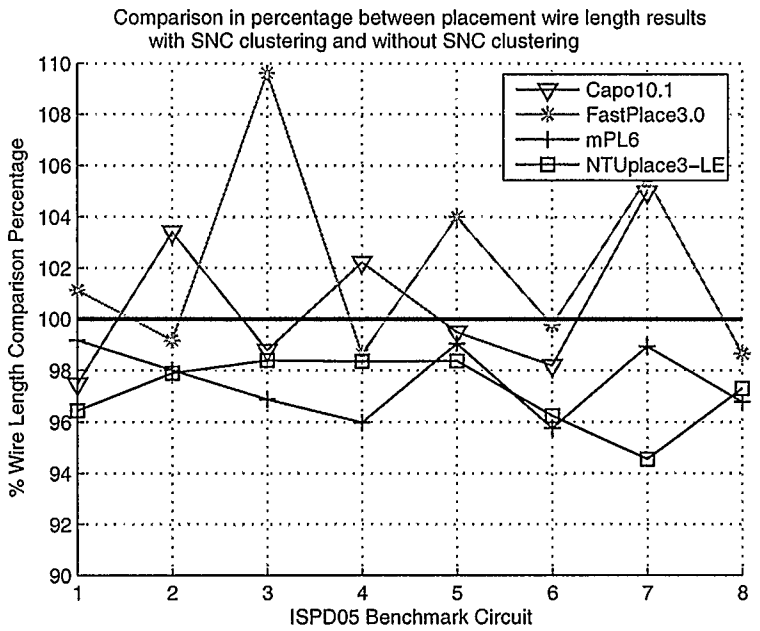


Figure 5.16: Placement wire length comparison in percentage on ISPD05 benchmark circuits

Table 5.25: Placement results of placer Capo10.1 on ISPD05 benchmark circuits

ISPD05 circuit	Original Capo10.1		SNC + Capo10.1		Comparison	
	HPWL	Runtime	HPWL	Runtime	HPWL	Runtime
adaptec1	9.00E+07	6255.82	8.78E+07	5109.87	0.975	0.817
adaptec2	9.65E+07	7957.9	9.97E+07	5993.14	1.034	0.753
adaptec3	2.35E+08	17962.2	2.32E+08	12497.55	0.988	0.696
adaptec4	2.06E+08	17851.1	2.10E+08	11981.43	1.022	0.671
bigblue1	1.08E+08	9972.82	1.07E+08	7545.89	0.995	0.757
bigblue2	1.64E+08	19871.8	1.61E+08	15689.84	0.982	0.790
bigblue3	3.85E+08	48447.2	4.04E+08	32923.68	1.050	0.680
Average	–	–	–	–	1.006	0.738

For ISPD05 benchmark circuits, the reported results are also promising. The overall average HPWL and runtime improvements for the test 4 placers are 1% and

a patented technology from Synopsys. NTUplace3-LE is based on the Lp-norm wire length model. Generally the placement results produced by NTUplace3 are better than those by NTUplace3-LE, in terms of both wire length and runtime. However, at this time the author is not able to get a license to use NTUplace3. Therefore, NTUplace3-LE was used.

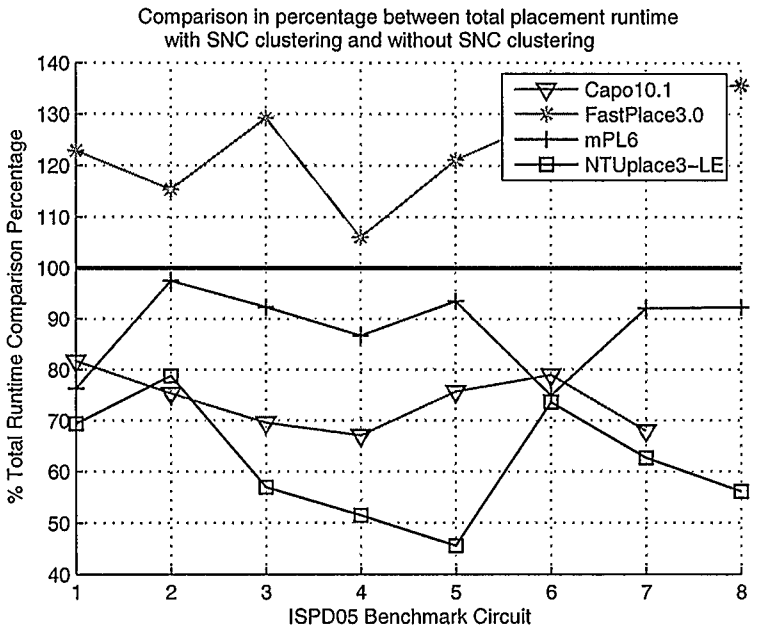


Figure 5.17: Placement runtime comparison in percentage on ISPD05 benchmark circuits

Table 5.26: Placement results of placer FastPlace3.0 on ISPD05 benchmark circuits

ISPD05 circuit	Original FastPlace3.0		SNC + FastPlace3.0		Comparison	
	HPWL	Runtime	HPWL	Runtime	HPWL	Runtime
adaptec1	7.88E+07	568.21	7.97E+07	697.99	1.011	1.228
adaptec2	9.34E+07	949.65	9.26E+07	1095.15	0.992	1.153
adaptec3	2.12E+08	2093.3	2.33E+08	2703.99	1.096	1.292
adaptec4	1.99E+08	1810.18	1.96E+08	1918.56	0.986	1.060
bigblue1	9.70E+07	935.93	1.01E+08	1132.64	1.040	1.210
bigblue2	1.53E+08	2388.75	1.53E+08	3086.03	0.998	1.292
bigblue3	3.66E+08	4804.68	3.86E+08	6485.53	1.055	1.350
bigblue4	8.37E+08	10207.3	8.26E+08	13833.22	0.987	1.355
Average	–	–	–	–	1.021	1.243

Table 5.27: Placement results of placer mPL6 on ISPD05 benchmark circuits

ISPD05 circuit	Original mPL6		SNC + mPL6		Comparison	
	HPWL	Runtime	HPWL	Runtime	HPWL	Runtime
adaptec1	7.79E+07	3049.67	7.73E+07	2326.5	0.992	0.763
adaptec2	9.20E+07	3177.07	9.02E+07	3097.1	0.980	0.975
adaptec3	2.14E+08	9619.88	2.07E+08	8879.39	0.969	0.923
adaptec4	1.94E+08	8904.11	1.86E+08	7717.06	0.960	0.867
bigblue1	9.68E+07	3812.8	9.59E+07	3563.76	0.990	0.935
bigblue2	1.52E+08	10355.1	1.46E+08	7757.63	0.958	0.749
bigblue3	3.44E+08	13994.9	3.40E+08	12885.08	0.989	0.921
bigblue4	8.29E+08	31721.1	8.03E+08	29264.87	0.968	0.923
Average	–	–	–	–	0.976	0.882

Table 5.28: Placement results of placer NTUplace3-LE on ISPD05 benchmark circuits

ISPD05 circuit	Original NTUplace3-LE		SNC + NTUplace3-LE		Comparison	
	HPWL	Runtime	HPWL	Runtime	HPWL	Runtime
adaptec1	8.06E+07	1635	7.77E+07	1134.29	0.964	0.694
adaptec2	9.05E+07	1655	8.86E+07	1303.21	0.979	0.787
adaptec3	2.15E+08	5102	2.12E+08	2906.45	0.984	0.570
adaptec4	1.96E+08	5267	1.93E+08	2711.24	0.983	0.515
bigblue1	9.70E+07	3420	9.55E+07	1557.89	0.984	0.456
bigblue2	1.53E+08	5378	1.48E+08	3957.6	0.962	0.736
bigblue3	3.53E+08	13163	3.34E+08	8248.7	0.946	0.627
bigblue4	8.36E+08	30369	8.13E+08	17043.03	0.973	0.561
Average	–	–	–	–	0.972	0.618

Table 5.29: Runtime statistics for different stages of **SNC + Capo10.1** framework on ISPD05 benchmark circuits

ISPD05 circuit	SNC Clustering		Initial Placement		Mapping Placement		Legalization	
	runtime	ratio%	runtime	ratio%	runtime	ratio%	runtime	ratio%
adaptec1	111	2.17	4786.47	93.67	2	0.04	210.4	4.12
adaptec2	168	2.80	5475.16	91.36	3	0.05	346.98	5.79
adaptec3	328	2.62	11477.5	91.84	6	0.05	686.05	5.49
adaptec4	305	2.55	11164.5	93.18	5	0.04	506.93	4.23
bigblue1	197	2.61	7006.9	92.86	3	0.04	338.99	4.49
bigblue2	790	5.04	13739.2	87.57	6	0.04	1154.64	7.36
bigblue3	860	2.61	30669.7	93.15	11	0.03	1382.98	4.20
Average	–	2.91	–	91.95	–	0.04	–	5.10

Table 5.30: Runtime statistics for different stages of **SNC + FastPlace3.0** framework on ISPD05 benchmark circuits

ISPD05 circuit	SNC Clustering		Initial Placement		Mapping Placement		Legalization	
	runtime	ratio%	runtime	ratio%	runtime	ratio%	runtime	ratio%
adaptec1	111	15.90	249.53	35.75	2	0.29	335.46	48.06
adaptec2	168	15.34	371.07	33.88	4	0.37	552.08	50.41
adaptec3	328	12.13	693.11	25.63	5	0.18	1677.88	62.05
adaptec4	305	15.90	554.2	28.89	9	0.47	1050.36	54.75
bigblue1	197	17.39	389.11	34.35	4	0.35	542.53	47.90
bigblue2	790	25.60	633.59	20.53	6	0.19	1656.44	53.68
bigblue3	860	13.26	1180.01	18.19	11	0.17	4434.52	68.38
bigblue4	3066	22.16	4060.21	29.35	24	0.17	6683.01	48.31
Average	–	17.21	–	28.32	–	0.27	–	54.19

Table 5.31: Runtime statistics for different stages of **SNC + mPL6** framework on ISPD05 benchmark circuits

ISPD05 circuit	SNC Clustering		Initial Placement		Mapping Placement		Legalization	
	runtime	ratio%	runtime	ratio%	runtime	ratio%	runtime	ratio%
adaptec1	111	4.77	2009.71	86.38	2	0.09	203.79	8.76
adaptec2	168	5.42	2544.61	82.16	3	0.10	381.49	12.32
adaptec3	328	3.69	7564.98	85.20	5	0.06	981.41	11.05
adaptec4	305	3.95	6712.86	86.99	6	0.08	693.2	8.98
bigblue1	197	5.53	3108.08	87.21	3	0.08	255.68	7.17
bigblue2	790	10.18	5631.15	72.59	6	0.08	1330.48	17.15
bigblue3	860	6.67	10290.5	79.86	11	0.09	1723.58	13.38
bigblue4	3066	10.48	22090.2	75.48	23	0.08	4085.67	13.96
Average	–	6.34	–	81.98	–	0.08	–	11.60

Table 5.32: Runtime statistics for different stages of **SNC + NTUplace3-LE** framework on ISPD05 benchmark circuits

ISPD05 circuit	SNC Clustering		Initial Placement		Mapping Placement		Legalization	
	runtime	ratio%	runtime	ratio%	runtime	ratio%	runtime	ratio%
adaptec1	111	9.79	811	71.50	3	0.26	209.29	18.45
adaptec2	168	12.89	878	67.37	3	0.23	254.21	19.51
adaptec3	328	11.29	1888	64.96	5	0.17	685.45	23.58
adaptec4	305	11.25	1881	69.38	6	0.22	519.24	19.15
bigblue1	197	12.65	1105	70.93	4	0.26	251.89	16.17
bigblue2	790	19.96	2137	54.00	7	0.18	1023.6	25.86
bigblue3	860	10.43	5380	65.22	13	0.16	1995.7	24.19
bigblue4	3066	17.99	10316	60.53	26	0.15	3635.03	21.33
Average	–	13.28	–	65.49	–	0.20	–	21.03

Table 5.33: Overall wire length and runtime comparison results of 4 test placers on ISPD05 benchmark circuits

ISPD05 circuit	Overall Comparison	
	HPWL	Runtime
adaptec1	0.986	0.875
adaptec2	0.996	0.917
adaptec3	1.009	0.870
adaptec4	0.988	0.778
bigblue1	1.002	0.839
bigblue2	0.975	0.892
bigblue3	1.010	0.894
bigblue4	0.976	0.946
Average	0.993	0.877

12%, respectively. For Capo10.1, out of 7 test circuits,[4] the HPWL results for 4 circuits are improved. On average, the HPWL results are increased by only 0.6%. Furthermore, the runtime is reduced by about 26% on average. For FastPlace3.0, the HPWL results for 4 circuits are improved. On average, the HPWL results have a 2% increase. Besides, the runtime is increased by around 24% on average. For mPL6, all HPWL results are improved. The average improvement is about 2% and maximum improvement is 4.2% for benchmark circuit bigblue2. In addition, the runtime is reduced by about 12% on average. For NTUplace3-LE, the HPWLs for all test circuits are improved too. The average improvement is about 3% and maximum improvement is 5.4% for benchmark circuit bigblue3. At the same time, the runtime is significantly improved by 38% on average.

5.6 Summary

In this chapter, three algorithms are proposed to obtain the final clustering solutions. The first algorithm is a scoreless technique, and the other two algorithms are score-based clustering techniques. All of these algorithms are able to remove cell overlap between clusters. The characteristic of each algorithm can be summarized as follows.

[4]Capo10.1 run out of memory for benchmark circuit bigblue4 on our workstation.

- The scoreless algorithm finalizes a cluster once the cluster is identified. The cells in this cluster are not considered for further clustering. Two nonrandom cell ordering techniques are proposed in this algorithm to obtain the clustering solutions with balanced sizes.

- The cluster score-based algorithm uses the scores to determine the cluster priorities for clustering. The clusters with higher scores are clustered first. The cell overlap problem is solved by discarding some clusters.

- The score-based net cluster algorithm represents a major contribution of this thesis. It introduces a net scoring technique to solve the cell overlap problem by mimicing the widely used force-directed model. As a result, this technique can produce high quality clustering solutions and at the same time, reduce the netlist to a low clustering ratio.

The scoreless and score-based algorithms have been tested in the context of circuit partitioning and placement. The experimental results verify the effectiveness of these algorithms. State-of-the-art circuit partitioning and placement tools have been further improved by using these clustering techniques.

Chapter 6

Conclusions and Future Work

6.1 Summary and Contributions

The research work presented in this thesis is focused on clustering algorithms for VLSI circuit partitioning and placement. Circuit partitioning and placement are two fundamental optimizations problems in VLSI physical design, and the algorithms for circuit partitioning and placement have a deep impact on the overall circuit performance. With the increasing size and complexity for today's circuits, clustering algorithms have become popular in current partitioning and placement tools to effectively deal with the large scale designs. Most of current clustering algorithms are greedy in nature, therefore, the performance of current clustering algorithms can be further improved. This improvement on clustering algorithms can result in improvement of the partitioning and placement tools.

The main contributions of this thesis can be summarized as follows.

- The gain clustering concept, a new clustering concept, is proposed. The cell connectivities in a gain cluster are considered as a whole, instead of in the standard greedy pair wise manner. As a result, gain clusters have improved quality.

- Three single gain cluster identification algorithms are proposed. One of these algorithms is a scoreless clustering technique, and the other two are score-based clustering techniques. The clustering statistical study shows that the score-based gain cluster algorithm, using the seed net technique, can produce a solution with a low clustering ratio, which is more suitable for large scale circuit

design.

- A net scoring technique is proposed. This net scoring technique is motivated by the force-directed model, and can be used to effectively solve cell overlaps between different gain clusters identified by our new algorithms.

- A score-based net cluster algorithm is proposed. This algorithm combines the score-based single gain clustering algorithm, using seed nets, and the net scoring technique together. It can produce high quality clustering solutions with lower clustering ratios.

- All of the proposed clustering algorithms have been compared with other popular clustering techniques, such as FirstChoice and best choice. The proposed algorithms are also tested as a preprocessing step for circuit partitioning and placement. The experimental results on ISPD98, ICCAD04 and ISPD05 benchmark circuits show that by using the proposed clustering algorithm, the performance of state-of-the-art partitioning and placement algorithms are consistently improved.

- A software package in C++ has been developed. The code will be released as open source software. The package includes the implementations of the proposed clustering algorithms, and other useful utilities for circuit clustering, partitioning and placement study.

6.2 Future Work

As for the future work, the following directions deserve to be investigated.

- The implementation of a library interface of the proposed score-based net cluster algorithm. This algorithm overall shows great potential to improve both the

placement solution quality and runtime. However, currently it is not seamlessly integrated into a placer. The library interface of proposed clustering algorithm will be available on line soon.

- A more efficient implementation of proposed clustering algorithm. The runtime of the proposed clustering algorithm can be further improved.

- Investigation of the new cluster score functions. Different cluster score functions can result in different clustering results. The design of new cluster score functions is still on open problem.

- Extension of the connectivity model in gain clusters to other clustering algorithms. The connectivity model proposed in gain clusters can be applied to other clustering techniques, and improve the solution quality by those techniques.

- Application of the proposed clustering techniques on industrial placers. The proposed clustering techniques have shown to be effective for academic placers, and can further be used or adapted for industrial placers.

Bibliography

[1] http://glaros.dtc.umn.edu/gkhome/metis/hmetis/overview. July 2007.

[2] http://vlsicad.eecs.umich.edu/BK/ICCAD04bench/. July 2007.

[3] http://vlsicad.eecs.umich.edu/BK/PDtools/. July 2007.

[4] http://vlsicad.ucsd.edu/GSRC/bookshelf/Slots/RSMT/RMST/. July 2007.

[5] http://vlsicad.ucsd.edu/UCLAWeb/cheese/ispd98.html. July 2007.

[6] http://www.ispd.cc/ispd07_contest.html. July 2007.

[7] http://www.sigda.org/ispd2005/contest.htm. July 2007.

[8] International technology raodmap for semiconductors. http://www.itrs.net/. July 2007.

[9] S. Adya, S. Chaturvedi, J. Roy, D. Papa, and I. Markov. Unification of partitioning, floorplanning and placement. In *Proc. ICCAD*, pages 550–557, 2004.

[10] A. Agnihotri, S. Ono, and P. Madden. Recursive bisection placement: Feng Shui 5.0 implementation details. In *Proc. ISPD*, pages 230–232, 2005.

[11] C. Alpert, A. Kahng, G.-J. Nam, S. Reda, and P. Villarrubia. A semi-persistent clustering technique for VLSI circuit placement. In *Proc. ISPD*, pages 200–207, 2005.

[12] C. J. Alpert. The ISPD98 circuit benchmark suite. In *Proc. ISPD*, pages 80–85, 1998.

[13] C. J. Alpert, D. Huang, and A. B. Kahng. Multilevel circuit partitioning. *IEEE Trans. on CAD*, 17(8):655–667, 1998.

[14] C. J. Alpert and A. B. Kahng. Recent directions in netlist partitioning: A survey. *The VLSI Journal on Integration*, 19:1–81, 1995.

[15] C. J. Alpert and A. B. Kahng. A general framework for vertex orderings with applications to circuit clustering. *IEEE Trans. on VLSI Systems*, 4(2):240–246, 1996.

[16] L. Behjat. *New Modeling and Optimization Techniques for the Global Routing Problem.* PhD thesis, University of Waterloo, Waterloo, Ontario, 2002.

[17] L. Behjat, D. Kucar, and A. Vannelli. A novel eigenvector technique for large scale combinatorial problems in VLSI layout. *Journal of Combinatorial Optimization*, 6(3):271–286, 2002.

[18] D. P. Bertsekas. *Nonlinear Programming.* Athena Scientific, Belmont, Mass, 1995.

[19] U. Brenner and M. Struzyna. Faster and better global placement by a new transportation algorithm. In *Proc. DAC*, pages 591–596, 2005.

[20] A. E. Caldwell, A. B. Kahng, and I. L. Markov. Improved algorithms for hypergraph bi-partitioning. In *Proc. Asia and South Pacific DAC*, pages 661–666, 2000.

[21] A. E. Caldwell, A. B. Kahng, and I. L. Markov. Optimal partitioners and end-case placers for standard-cell layout. *IEEE Trans. on CAD*, 19(11):1304–1313, 2000.

[22] T. Chan, J. Cong, and K. Sze. Multilevel generalized force-directed method for circuit placement. In *Proc. ISPD*, pages 185–192, 2005.

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

[24] A. Chatterjee and R. Hartley. A new simultaneous circuit partitioning and chip placement approach based on simulated annealing. In *Proc. DAC*, pages 36–39, 1990.

[25] T. Chen, T. Hsu, Z. Jian, and Y. Chang. NTUplace: A ratio partitioning based placement algorithm for large-scale mixed-sized designs. In *Proc. ISPD*, pages 236–238, 2005.

[26] T.-C. Chen, Z.-W. Jiang, T.-C. Hsu, H.-C. Chen, and Y.-W. Chang. A high-quality mixed-size analytical placer considering preplaced blocks and density constraints. In *Proc. ICCAD*, pages 187–192, 2006.

[27] A. Chiang. *A Fast, Congestion Based Concurrent Global Routing Technique.* Master thesis, University of Calgary, Calgary, Alberta, 2005.

[28] J. Cong and S. Lim. Edge separability-based circuit clustering with application to multilevel circuit partitioning. *IEEE Trans. on CAD*, 23(3):346–357, 2004.

[29] J. Cong and S. K. Lim. Edge separability based circuit clustering with application to circuit partitioning. In *Proc. Asia and South Pacific DAC*, pages 429–434, 2000.

[30] J. Cong, M. Romesis, and M. Xie. Optimality, scalability, and stability study of partitioning and placement algorithms. In *Proc. ISPD*, pages 88–94, 2003.

[31] J. Cong and M. Smith. A parallel bottom-up clustering algorithm with applications to circuit partitioning in VLSI design. In *Proc. DAC*, pages 755–760, 1993.

[32] S. Dutt and W. Deng. VLSI circuit partitioning by cluster-removal using iterative improvement techniques. In *Proc. ICCAD*, pages 194–200, 1996.

[33] S. Dutt and W. Deng. Probability-based approaches to VLSI circuit partitioning. *IEEE Trans. on CAD*, 19:534–549, 2000.

[34] C. M. Fiduccia and R. M. Mattheyses. A linear time heuristic for improving network partitions. In *Proc. DAC*, pages 175–181, 1982.

[35] S. W. Hadley, B. L. Mark, and A. Vannelli. An efficient eigenvector approach for finding netlist partitions. *IEEE Trans. on CAD*, 11(7):885–892, 1992.

[36] L. W. Hagen, D. J. Huang, and A. B. Kahng. On implementation choices for iterative improvement partitioning methods. In *Proc. DATE*, pages 144–149, 1995.

[37] L. W. Hagen and A. B. Kahng. New spectral methods for ratio cut partitioning and clustering. *IEEE Trans. on CAD*, 11(9):1074–1085, 1992.

[38] S. Hauck and G. Borriello. An evaluation of bipartitioning techniques. *IEEE Trans. on CAD*, 16(8):849–866, 1997.

[39] B. Hu and M. Marek-Sadowska. Fine granularity clustering for large scale placement problems. In *Proc. ISPD*, pages 67–74, 2003.

[40] B. Hu and M. Marek-Sadowska. Multilevel fixed-point-addition-based VLSI placement. *IEEE Trans. on CAD*, 24(8):1188–1203, 2005.

[41] B. Hu, Y. Zeng, and M. Marek-Sadowska. mFAR: Fixed-points-addition-based VLSI placement algorithm. In *Proc. ISPD*, pages 239–241, 2005.

[42] A. B. Kahng. Futures for partitioning in physical design. In *Proc. ISPD*, pages 190–193, 1998.

[43] A. B. Kahng, S. Reda, and Q. Wang. Architecture and details of a high quality, large-scale analytical placer. In *Proc. ICCAD*, pages 890–897, 2005.

[44] A. B. Kahng and Q. Wang. A faster implementation of APlace. In *Proc. ISPD*, pages 218–220, 2006.

[45] G. Karypis, R. Aggarwal, V. Kumar, and S. Shekhar. Multilevel hypergraph partitioning: Application in VLSI domain. In *Proc. DAC*, pages 526–529, 1997.

[46] G. Karypis, R. Aggarwal, V. Kumar, and S. Shekhar. Multilevel hypergraph partitioning: Application in VLSI domain. *IEEE Trans. on VLSI Systems*, 7(1):69–79, 1999.

[47] G. Karypis and V. Kumar. Multilevel k-way hypergraph partitioning. In *Proc. DAC*, pages 343–348, 1999.

[48] A. Kennings and K. Vorwerk. Force-directed methods for generic placement. *IEEE Trans. on CAD*, 25(10):2076–2087, 2006.

[49] B. W. Kernighan and S. Lin. An efficient heuristic procedure for partitioning graphs. *The Bell System Technical Journal*, 49(2):291–307, 1970.

[50] J. Li and L. Behjat. A connectivity based clustering algorithm with applications to VLSI circuit partitioning. *IEEE Trans. on CAS II*, 53(5):384–388, 2006.

[51] J. Li and L. Behjat. Net cluster: A net-reduction based clustering preprocessing algorithm. In *Proc. ISPD*, pages 200–205, 2006.

[52] J. Li, L. Behjat, and J. Huang. An effective clustering algorithm for mixed-size placement. In *Proc. ISPD*, pages 111–118, 2007.

[53] J. Li, L. Behjat, and A. Kennings. Net cluster: A net-reduction-based clustering preprocessing algorithm for partitioning and placement. *IEEE Trans. on CAD*, 26(4):669–679, 2007.

[54] Q. Liu and M. Marek-Sadowska. A study of netlist structure and placement efficiency. In *Proc. ISPD*, pages 198–203, 2004.

[55] P. H. Madden. Reporting of standard cell placement results. In *Proc. ISPD*, pages 30–35, 2001.

[56] G. Moore. Cramming more components into integrated circuits. *Electronics*, 38(8), 1965.

[57] G. Nam. ISPD 2006 placement contest: Benchmark suite and results. In *Proc. ISPD*, page 167, 2006.

[58] G. Nam, C. J. Alpert, P. Villarrubia, B. Winter, and M. Yildiz. The ISPD2005 placement contest and benchmark suite. In *Proc. ISPD*, pages 216–219, 2005.

[59] G.-J. Nam, S. Reda, C. J. Alpert, P. G. Villarrubia, and A. B. Kahng. A fast hierarchical quadratic placement algorithm. *IEEE Trans. on CAD*, 25(4):678–691, 2006.

[60] G. Parthasarathy, M. Marek-Sadowska, A. Mukherjee, and A. Singh. Interconnect complexity-aware FPGA placement using Rent's rule. In *Proc. SLIP*, pages 115–121, 2001.

[61] J. Roy, D. Papa, S. Adya, H. Chan, A. Ng, F. Lu, and I. Markov. Capo: Robust and scalable open-source min-cut floorplacer. In *Proc. ISPD*, pages 224–226, 2005.

[62] Y. Saab. An effective multilevel algorithm for bisecting graphs and hypergraphs. *IEEE Trans. on Computers*, 53(6):641–652, 2004.

[63] S. M. Sait and H. Youssef. *VLSI Physical Design Automation, Theory and Practice.* World Scientific, 1999.

[64] L. A. Sanchis. Multi-way network partitioning. *IEEE Transactions on Computers*, 38(1):62–81, 1989.

[65] L. A. Sanchis. Multi-way network partitioning with different cost functions. *IEEE Transactions on Computers*, 42(12):1500–1504, 1993.

[66] C. Sechen and A. Sangiovanni-Vincentelli. The TimberWolf placement and routing package. *IEEE Journal of Solid-State Circuits*, 20(2):510–522, 1985.

[67] N. Selvakkumaran and G. Karypis. Multi-objective hypergraph partitioning algorithms for cut and maximum subdomain degree minimization. In *Proc. ICCAD*, pages 726–733, 2003.

[68] K. Shahookar and P. Mazumder. VLSI cell placement techniques. *ACM Computing Surveys*, 23(2):143–220, 1991.

[69] N. A. Sherwani. *Algorithms for VLSI Physical Design Automation, Third Edition.* Kluwer Academic Publishers, Massachusetts, USA, 1999.

[70] A. Singh and M. Marek-Sadowska. Efficient circuit clustering for area and power reduction in FPGAs. In *Proc. FPGA*, pages 59–66, 2002.

[71] P. Spindler and F. M. Johannes. Fast and robost quadratic placement combined with an exact linear net model. In *Proc. ICCAD*, pages 179–186, 2006.

[72] T. Taghavi, X. Yang, and M. Sarrafzadeh. Dragon2005: Large-scale mixed sized placement tool. In *Proc. ISPD*, pages 245–247, 2005.

[73] A. Vannelli. An adaptation of the interior point method for solving the global routing problem. *IEEE Trans. on CAD*, 10(2), 1991.

[74] N. Viswanathan and C. C.-N. Chu. FastPlace: Efficient analytical placement using cell shifting, iterative local refinement and a hybrid net model. In *Proc. ISPD*, pages 26–33, 2004.

[75] N. Viswanathan, M. Pan, and C. Chu. FastPlace 3.0: A fast multilevel quadratic placement algorithm with placement congestion control. In *Proc. ASPDAC*, pages 135–140, 2007.

[76] K. Vorwerk and A. Kennings. Mixed-sized placement via line search. In *Proc. ICCAD*, pages 899–904, 2005.

[77] M. Wang, X. Yang, and M. Sarrafzadeh. Dragon2000: Standard-cell placement tool for large industry circuits. In *Proc. ICCAD*, pages 260–263, 2000.