

A CONSTRUCTIVE MULTI-WAY  
CIRCUIT PARTITIONING ALGORITHM  
BASED ON MINIMUM DEGREE  
ORDERING

A THESIS

SUBMITTED TO THE DEPARTMENT OF COMPUTER  
ENGINEERING AND INFORMATION SCIENCE  
AND THE INSTITUTE OF ENGINEERING AND SCIENCE  
OF BILKENT UNIVERSITY  
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS  
FOR THE DEGREE OF  
MASTER OF SCIENCE

By  
Omit V. Gatahyllrok  
September, 1994

THESIS  
QA  
165  
.C38  
1994

**A CONSTRUCTIVE MULTI-WAY  
CIRCUIT PARTITIONING ALGORITHM  
BASED ON MINIMUM DEGREE  
ORDERING**

A THESIS

SUBMITTED TO THE DEPARTMENT OF COMPUTER  
ENGINEERING AND INFORMATION SCIENCE  
AND THE INSTITUTE OF ENGINEERING AND SCIENCE  
OF BİLKENT UNIVERSITY  
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS  
FOR THE DEGREE OF  
MASTER OF SCIENCE

By

Ümit V. Çatalyürek

September, 1994

Ümit V. ÇATALYÜREK  
İstanbul, Türkiye

QA

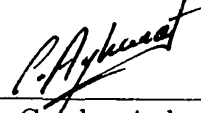
165

·C38

199+

B025558

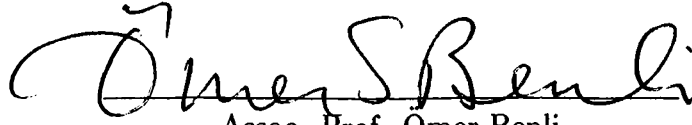
I certify that I have read this thesis and that in my opinion it is fully adequate, in scope and in quality, as a thesis for the degree of Master of Science.



---

Asst. Prof. Cevdet Aykanat (Advisor)

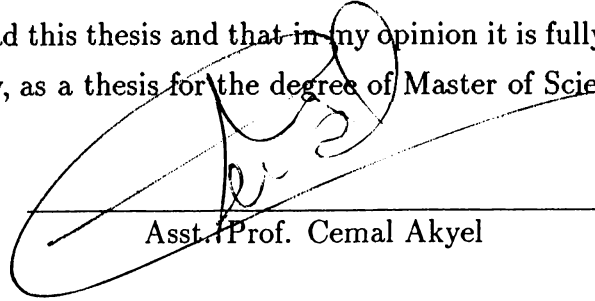
I certify that I have read this thesis and that in my opinion it is fully adequate, in scope and in quality, as a thesis for the degree of Master of Science.



---

Assoc. Prof. Omer Benli

I certify that I have read this thesis and that in my opinion it is fully adequate, in scope and in quality, as a thesis for the degree of Master of Science.



---

Asst. Prof. Cemal Akyel

Approved for the Institute of Engineering and Science:



---

Prof. Mehmet Baray  
Director of the Institute

## ABSTRACT

### A CONSTRUCTIVE MULTI-WAY CIRCUIT PARTITIONING ALGORITHM BASED ON MINIMUM DEGREE ORDERING

Ümit V. Çatalyürek

M.S. in Computer Engineering and Information Science

Advisor: Asst. Prof. Cevdet Aykanat

September, 1994

Circuit partitioning has many important applications in VLSI. Circuit partitioning problem can be most properly modeled as hypergraph partitioning. In this work, we propose a novel  $k$ -way hypergraph partitioning heuristic using the Minimum Degree (MD) ordering which is a well-known heuristic for reducing the amount of fills in the factorization of symmetric sparse matrices. The proposed algorithm operates on the dual graph of the given hypergraph. The algorithm grows node-clusters on the dual graph which induce cell-clusters with locally minimum net-cut sizes. The quotient graph concept, widely used in MD ordering, is exploited for the sake of efficient implementation. The proposed algorithm outperforms well-known heuristics, such as Kernighan-Lin (KL) based algorithms and Simulated Annealing, in terms of solution quality on various VLSI benchmark circuits. A nice property of the proposed algorithm is that its execution time reduces with increasing  $k$  as opposed to the existing iterative heuristics. It is even faster than the fast KL-based algorithms on the partitioning of the benchmark circuits for  $k \geq 16$ .

*Keywords:* Circuit Partitioning, Hypergraph Partitioning, Dual Graph, Minimum Degree Ordering, Quotient Graph

## ÖZET

### MINIMUM DERECE SIRALAMASINA DAYALI YAPICI ÇOK KISIMLI DEVRE PARÇALAMA ALGORİTMASI

Ümit V. Çatalyürek

Bilgisayar ve Enformatik Mühendisliği, Yüksek Lisans

Danışman: Yrd. Doç. Dr. Cevdet Aykanat

Eylül, 1994

Devre parçalamının geniş ölçekli tümleşik tasarımlarda bir çok önemli uygulaması vardır. Devre parçalama problemi en uygun şekilde hiperçizge parçalama olarak modellenenir. Bu çalışmada, yoğunluğu çok seyrek olan simetrik matrislerin faktörizasyonunda yaratılan eleman sayısını azaltmada çokça kullanılan Minimum Derece (MD) sıralama sezgisel metodunu kullanarak yeni bir  $k$ -kısımlı hiperçizge parçalama sezgisel algoritması öneriyoruz. Önerilen algoritma verilen hiperçizgenin karşıt çizgesi üzerinde çalışır. Önerilen algoritma karşıt çizgenin üzerinde çizge düğümlerini biraraya getirerek hiperçizgede yerel olarak minimum ağ-kesme miktarına sahip düğüm demetleri oluşturur. Algoritmanın daha hızlı çalışabilmesi için MD sıralamasında çokça kullanılan kümleştirimmiş çizge kavramı uygulanmıştır. Önerilen algoritma, bir çok standart test devrelerinde, elde edilen çözüm kalitesi açısından, Kernighan-Lin (KL) ve Simulated Annealing gibi çokça kullanılan sezgisel algoritmalarından çok daha iyi sonuçlar vermektedir. Algoritmamızın bir diğer önemli özelliği ise; daha önce önerilmiş metodların tersine, çalışma zamanının artan  $k$  değeriyle birlikte azalmasıdır. Hatta, önerilen algoritma hızlı olduğu bilinen KL-tipi algoritmalarından,  $k \geq 16$  değeri için, standart test devrelerinde daha hızlı çalışmaktadır.

*Anahtar Sözcükler:* Devre Parçalama, Hiperçizge Parçalama, Karşıt Çizge, Minimum Derece Sıralaması, Kümeleştirilmiş Çizge

## ACKNOWLEDGEMENTS

I would like to express my deep gratitude to my supervisor Dr. Cevdet Aykanat for his guidance, suggestions, and invaluable encouragement throughout the development of this thesis. I would like to thank Dr. Ömer Benli for reading and commenting on the thesis. I would also like to thank Dr. Cemal Akyel for reading and commenting on the thesis. I owe special thanks to Dr. Mehmet Baray for providing a pleasant environment for study. I am grateful to my family, my wife and my friends for their infinite moral support and help.



To my parents  
and  
my wife Gamze

# Contents

<b>1</b>	<b>Introduction</b>	<b>1</b>
<b>2</b>	<b>Circuit Partitioning and Previous Works</b>	<b>4</b>
2.1	Preliminaries . . . . .	4
2.2	Problem Definition . . . . .	6
2.3	Previous Works . . . . .	7
2.3.1	Iterative Algorithms . . . . .	7
2.3.2	Constructive Algorithms . . . . .	10
<b>3</b>	<b>Minimum Degree Ordering</b>	<b>12</b>
3.1	The Basic Algorithm . . . . .	13
3.2	Implementation with Quotient Graph Model . . . . .	14
<b>4</b>	<b>Circuit Partitioning Using MD Ordering</b>	<b>17</b>
4.1	Dual Graph . . . . .	18
4.2	Node Selection	19
4.3	Size and Valence and Calculations . . . . .	24
4.4	Graph Transformation . . . . .	26

4.5	More About Balancing . . . . .	28
4.6	Complexity Analysis . . . . .	30
4.6.1	Space Complexity Analysis . . . . .	31
4.6.2	Time Complexity Analysis . . . . .	33
<b>5</b>	<b>Experiments and Results</b>	<b>37</b>
5.1	Implementation . . . . .	37
5.2	Results . . . . .	39
<b>6</b>	<b>Conclusion</b>	<b>50</b>

## List of Figures

3.1	Basic Minimum Degree Algorithm . . . . .	14
4.1	A sample hypergraph $H$ and its dual graph $G$ . . . . .	19
4.2	Construction of dual quotient graph . . . . .	20
4.3	Elimination steps . . . . .	23
4.4	Reachable set calculation . . . . .	26
4.5	Degree update . . . . .	27
4.6	Algorithm for finding the cluster adjacency . . . . .	27
4.7	Update of valence and cluster size . . . . .	27
4.8	Quotient graph transformation . . . . .	29
4.9	First Fit Decreasing heuristic . . . . .	30
4.10	Balancing the partitions . . . . .	31
4.11	Main algorithm . . . . .	32
4.12	Node selection algorithm . . . . .	33

## List of Tables

5.1	Properties of test circuits. ( $p$ is the number of pins, $\sigma$ is standard deviation, avg is average.) . . . . .	38
5.2	Dual Graphs of Test Circuits . . . . .	38
5.3	Comparison of QMD-HP and QMD-BHP. ( $W_{max}$ and $W_{min}$ are the maximum and minimum part weights respectively, $\delta$ is unbalance ratio, %B.I. is the percent balance improvement in QMD-BHP.)	41
5.4	Cutsizes averages and standard deviations ( $\sigma$ ) for test circuits. . . . .	42
5.5	Minimum cutsizes for benchmark circuits. (Bold values are the best values in each row.) . . . . .	44
5.6	Maximum cutsizes for benchmark circuits. (Bold values are the best values in each row.) . . . . .	45
5.7	Stability Ratios (ratio of standard deviation to cutsize) for benchmark circuits.(Bold values are the best values in each row.)	46
5.8	Execution times for benchmark circuits (in seconds). (Bold values are the best values in each row.) . . . . .	47
5.9	Average (Avg) results for performances of algorithms. Averages were taken over all our test instances. ( $k$ is the number of parts. Bold values are the best values in each row.) . . . . .	49
5.10	Average percentage improvements of the proposed QMD-BHP algorithm with respect to SN and SA algorithms for different number of parts. Averages were taken over all our test instances. $k$ is the number of parts. . . . .	49

# Chapter 1

## Introduction

Divide and conquer strategy underlies in the solution of the hard problems. It is based on dividing the problem into small sub-problems contributing to the solution of the fundamental problem, hence this division reduces the search space. This strategy is mostly used in the combinatorial optimization problems and in VLSI layout design.

In VLSI layout design, electronic circuits are modeled as graphs (hypergraphs), in such a way that modules and interconnections in the circuits are represented as nodes and edges (nets), respectively. Divide and conquer strategy assists in solution of the layout design problem. The sub-problems are arised by dividing or partitioning the circuit into two or more parts by satisfying the some balance criteria. The total interconnections between these parts must also be minimized to have a better solution for the whole problem. In the literature, this partitioning problem is referred as *graph/hypergraph partitioning* or *mincut partitioning*.

The graph (hypergraph) partitioning problem is *NP-hard* [5]. Hence, heuristics giving suboptimal solutions in polynomial time are used to solve the problem. Known heuristic algorithms can be divided into two groups;

1. Iterative algorithms,
2. Constructive algorithms.

Iterative algorithms start with an initial solution, and try to *improve* this initial one, at each iteration, until a local optima is found. One of the most

popular heuristic is Kernighan-Lin [13] method. It is an iterative graph *bipartitioning* heuristic, in which each iteration contains a number of cell (module) *swaps* on the balanced partitions. Many of the subsequent algorithms are based on this heuristic. The same swap strategy is applied to hypergraph partitioning problem by Schweikert-Kernighan [23]. Fiduccia-Mattheyses [4] introduced a better data structure and *cell move* strategy. Their method works on unbalanced partitions given the lower and upper bound on the partition sizes. The time complexity of one pass (iteration) of the method is also reduced to linear in the size of circuit. Krishnamurty [17] extended the cell *gain* concept by introducing a look-ahead ability. However his method is also a bipartitioning method as Fiduccia-Mattheyses'. Sanchis [22] extended this formulation to the multiple-way partitioning.

There are many other heuristic approaches such as Simulated Evolution [14] and Simulated Annealing [15]. In general, Simulated Annealing (SA) has the best solution quality among all those known heuristics. Optimization in the parameters of SA and extensive empirical studies have been done by Johnson et. al. [9].

Kahng [11] introduced a constructive bisection algorithm based on the *intersection graph*  $G$  which is dual to the input hypergraph. However, his algorithm produces unbalanced partitions, since no weight information is kept in the intersection graph. Kamidoi et.al. [12] introduced a new constructive algorithm called *Weighted Hypergraph Bisection* (WHB) with the notion of *net-graph*. WHB is an extension of Kahng's method. It produces more balanced partitions than Kahng's method and comparable cutsizes results.

Although the treatment so far is mostly graph theoretic, motivation for the work is from the direct solution of sparse linear systems [21]. One phase of the direct solution is to find a new ordering for rows and columns of matrix, to reduce the *fill-in* in the forward and backward substitution phases, which is also a *NP-hard* problem [27]. *Minimum Degree Ordering*, proposed by Tinney[25], is the most popular heuristic algorithm. It works on the structure of the matrix, therefore it is a graph algorithm. Liu [18] shows that minimum degree ordering (MD) results in partitioning by node separator.

Proposed algorithm in this work is based on the minimum degree and *dual graph* of the input hypergraph. Our dual graph is similar to the intersection graph of Kahng. There is one node in the dual graph corresponding to each net

in the hypergraph. Two nodes in graph are connected only if the respective pair of nets have at least one cell in common in hypergraph. Given this definition node separator in the graph determines a cut in the hypergraph.

Our algorithm is based on the following well-known observation:

**Observation 1** *Assigning cells (modules) to the parts to minimize the cutsize is equal to assigning nets to parts (making them internal nets) in order to maximize the number of nets that are not in the cut.*

Proposed algorithm chooses a net to make internal by a heuristic based on the minimum degree ordering. However, instead of assigning the net to some part, it enlarges a cluster by adding the selected net. Therefore, it can also be considered as a clustering algorithm. We have also realized that, with a slightly different perspective, the partitioning problem can also be expressed by the clustering problem, i.e. if we allow to enlarge clusters up to a given partition size, we get a partitioning on the input. However, we may get more parts than required. As the current cut between clusters is realized, the problem is reduced to the number partitioning problem. This *NP-hard* problem can also be solved by a simple heuristic such as *First Fit Decreasing*.

The outline of the work is as follows, the next chapter gives some preliminaries about the graph/hypergraph partitioning problem and more detailed information about the previous works. Minimum degree ordering algorithm which is the basis of our algorithm is explained in the third chapter. The fourth chapter discusses the proposed algorithm. Empirical studies are given in the fifth chapter and the conclusions are presented in the last chapter.



## Chapter 2

# Circuit Partitioning and Previous Works

### 2.1 Preliminaries

Hypergraph  $H = (C, N)$  is defined as a set of cells  $C$  and a set of nets (hyperedges)  $N$  between those cells. Since in VLSI, circuits are modeled as hypergraphs, cell set  $C$  denotes the set of modules, and net set denote the interaction between modules. Every net  $n_i \in N$  is a subset of cells. The cells in a net are called *pins* or *terminals* of the net.

We say that cell  $c$  is *incident* to net  $n$  if  $c \in n$ , and two cells which are incident to same net are called *adjacent*, in other words, cells in a net are adjacent. Degree of a cell is denoted by  $deg(c)$  and it is the number of incident nets. In general, minimum node degree is assumed to be 1, i.e.  $deg(c_i) \geq 1$  for  $1 \leq i \leq |C|$ . We use the notation  $|C|$  as the cardinality of set  $C$ . Cells with degree zero are called as *isolated* cells, and they do not introduce problems in partitioning. The degree of a net is the number of pins (terminals). It is also assumed that every net contains at least two pins i.e.  $|n| \geq 2$ .

In a hypergraph the total number of pins  $p$  is defined as

$$p = \sum_{c \in C} deg(c) \tag{2.1}$$

or

$$p = \sum_{n \in N} |n|. \tag{2.2}$$

Graph  $G = (C, E)$  is a special case of hypergraph such that each edge contains exactly two terminals. Therefore, algorithms proposed for hypergraphs can work on graphs without modifications. Since our aim is to solve circuit partitioning problem and hypergraph models the circuit better than graphs, we will explain partitioning algorithms using hypergraph notation.

For hypergraph  $H = (C, N)$ , the weight function on cell set maps each cell to a positive integer, i.e. for each  $c \in C$ ,  $weight(c) \geq 1$ . We can think that this function maps each cell to its area in the layout. The cost function on net set is defined in the same manner, i.e. for each  $n \in N$ ,  $cost(n) \geq 1$ . Definitions of  $weight$  and  $cost$  function can be extended for a set. Let  $A \subset C$  and  $M \subset N$  then

$$weight(A) = \sum_{c \in A} weight(c)$$

$$cost(M) = \sum_{n \in M} cost(n).$$

Using this notation total weight of circuit can be expressed as  $weight(C)$  and total cost of nets is  $cost(N)$ .

$k$ -way partition of hypergraph  $H$  is defined as

**Definition 1**  $\mathcal{P} = \{P_1, P_2, \dots, P_k\}$  is  $k$ -way partition of hypergraph  $H$  if and only if the following three conditions hold:

- $P_i \subset C$  and  $P_i \neq \emptyset$  for  $1 \leq i \leq k$
- $\bigcup_{i=1}^k P_i = C$
- $P_i \cap P_j = \emptyset$  for  $1 \leq i < j \leq k$

When  $k = 2$  we call this partitioning as *bisection* or *bipartition*.

For a partition  $\mathcal{P}$ , a net  $n$  is said to be internal in partition  $P_i$  if and only if

$$\forall c \in n, c \in P_i$$

or

$$n \cap P_i = n.$$

The set of internal nets  $N_I$  is defined as  $N_I = \{n | n \text{ is internal net in a partition} \}$  or  $N_I = \{n | n \cap P_i = n \text{ for } n \in N \text{ and } P_i \in \mathcal{P}\}$  and the set of external nets  $N_E$  is defined as  $N_E = \{n | n \cap P_i \neq \emptyset \text{ and } n \cap P_i \neq n \text{ for } n \in N \text{ and } P_i \in \mathcal{P}\}$ . Cut size  $\mathcal{C}$  is defined as

$$\mathcal{C}(\mathcal{P}) = \sum_{n \in N_E} \text{cost}(n)$$

or

$$\mathcal{C}(\mathcal{P}) = \text{cost}(N_E).$$

Using different expression

$$\mathcal{C}(\mathcal{P}) = \text{cost}(N) - \text{cost}(N_I).$$

A partitioning is *balanced* if all parts have about the same weight. When all parts have exactly the same weight, we call this partitioning as *perfectly balanced*. Note that perfect balance is not possible in  $k$ -way partitioning if the total cell weight is not a multiple of  $k$ .

## 2.2 Problem Definition

Let  $\mathcal{N}$  be set of natural numbers. Given a hypergraph  $H = (C, N)$ , a weight function  $\text{weight} : C \rightarrow \mathcal{N}$ , a cost function  $\text{cost} : N \rightarrow \mathcal{N}$  let  $\mathcal{P}_i = \{P_1, P_2, \dots, P_k\}$  be a  $k$ -way partition as defined in Definition 1 satisfying the condition

$$\frac{W_{max} - W_{min}}{W_{max}} < \Delta$$

where  $W_{min}$  and  $W_{max}$  are minimum and maximum partition weights, respectively, and  $\Delta$  is predetermined *imbalance ratio*. Also let  $\Pi = \{\mathcal{P}_1, \mathcal{P}_2, \dots\}$  be the set of all feasible solutions.

**Question** Find a feasible solution (partition)  $\mathcal{P}$  that minimizes the cutsizes over all feasible solutions, or more formally;

$$\min_{\mathcal{P} \in \Pi} \mathcal{C}(\mathcal{P}) = \text{cost}(N) - \text{cost}(N_I).$$

This cost definition computes each external net once regardless of the number of parts which pins of the net distributed. Other cost definitions can be done using this number, such that let  $l$  be the number of parts which a net  $n$  in cut connects, then contribution of the net to the cut is  $(l - 1) \cdot \text{cost}(n)$ .

The hypergraph partitioning problem is *NP-hard* [5]. Although the graph is a special case of hypergraph in which each edge connects exactly two cells, it is also *NP-hard* problem. Any heuristic which solves the hypergraph partitioning problem can be used for graph partitioning problem, but graph partitioning heuristics need modifications to handle hypergraph partitioning.

Some other cost function definitions are also available in the literature. In the next section we will review the previous works on this problem in detail.

## 2.3 Previous Works

Available heuristic algorithms can be divided into two groups; *iterative* and *constructive* algorithms. Although there is a substantial amount of literature on the iterative approaches, the literature that addresses the constructive algorithms are rare and more recent. Now let us review the some of those heuristics:

### 2.3.1 Iterative Algorithms

#### Kernighan-Lin's Method :

This heuristic is a graph bipartitioning algorithm [13]. It works on the balanced partitions, starts with an initial partition (mostly random) and at each iteration the cutsize is reduced by a number of *cell swaps*. In order to get a balanced partition after a swap, all cells must be equally weighted. This scheme is not applicable for the current problems.

*Gain* of a swap is calculated as a reduction in the cutsize. All swap gains are computed and the cell pair with the largest gain is selected for swap. These two cells are tentatively interchanged and they are locked in their new partition in order to prevent the algorithm falling in an infinite loop. The cell swap gains of adjacent cells are recomputed since there may be a change due to current swap. The next largest swap gain cells are selected to swap next, and this loop goes until all cells are locked to complete a pass.

At the end of each pass the maximum prefix sum of gains (which must be positive) are calculated and the cell swaps whose gains are included in this prefix sum is done. If maximum prefix sum is not positive, this means that no

further improvements can be done and algorithm terminates. If it is positive, all cells have been unlocked and algorithm starts a new pass. Maximum prefix sum strategy allows the algorithm not to stuck in a local optima.

This algorithm is a bisection algorithm but, it can be also used for  $k$ -way partitioning using the heuristic recursively, if  $k$  is a power of 2. The time complexity of one pass is  $\mathcal{O}(n^2 \log n)$  where  $n$  is the number nodes in the graph. Empirical studies show that this heuristic results in poor cutsizes in very sparse graphs and in special type of graphs such as ladder graphs [1].

#### **Schweikert-Kernighan's Method :**

This heuristic [23], is the application of the swap strategy to hypergraph partitioning problem. Up to this work, graph model was used for hypergraph partitioning problems.

#### **Fiduccia-Mattheyses's Method :**

Fiduccia-Mattheyses (FM) [4] introduced the notion of cell *move*, as well as the new data structure, for the the hypergraph bisection algorithm. *Cell move gain* is computed as reduce in the cutsize and gains are put into *bucket list*. This reduces the time complexity of sorting nodes according to their gains to linear in the number of nodes and edges. That is, let  $p$  denotes the total number of pins, which is calculated as in the Equation 2.1, then the time complexity of one pass is  $\mathcal{O}(p)$ .

Hence the cell move strategy is used in this algorithm, and this heuristic can work on unbalanced partition. Given the lower and upper bound on the size of the parts, algorithm distinguishes the feasible and infeasible moves. It starts with an initial solution and it makes a number of cell move at each iteration. Same prefix sum strategy of Kernighan-Lin's method is also used as hill-climbing technique. Because of its ability of working on the unbalanced partition, many of the subsequent algorithms use the same balance criteria.

#### **Krishnamurty's Method :**

This heuristic [17] is an extension of FM's method. Look-ahead ability is added to the cell gain concept by considering the number of pins of a net in a part. Each node has a gain vector with size  $l$ , where  $l$  is the number of *levels*. First level gain is same as that in FM's method. Second level gain, shows the possible cut size reduction in the next move which follows the the current cell

move. If a net has 2 cells in a part ( $A$ ), and at least one cell in other part ( $B$ ), moving one of the two cells from  $A$  to  $B$  does not reduce the cut size, but it gives the chance to the other cell of this net, to reduce the cut. Therefore effect of this net to first level gain of those cells are 0 and effect to second level gains are the cost of this net.

#### Sanchis's Method :

Sanchis generalized the Krishnamurty's method to the multiple-way ( $k$ -way) circuit partitioning. Since there are more than one part which a cell can move, each part contains  $k - 1$  bucket list; one for each other part which a node can move. Hence,  $k$  must enter the run-time complexity of the algorithm. The time complexity of one pass is  $\mathcal{O}(l \cdot p \cdot k \cdot (\log k + G_{max} \cdot l))$ , where  $l$  is the number of levels and  $G_{max}$  is the size of buckets.

#### Simulated Annealing :

Simulated Annealing starts from a randomly chosen initial configuration, the configuration space is searched for the best solution using a probabilistic hill-climbing algorithm. In order to search, the neighborhood of a configuration must be defined. Neighborhood consists of all configurations which can be obtained by moving one node from a part to another part. At each iteration, one of the possible moves is chosen as a candidate move. Then decrease in the cutsize is calculated without changing the configuration. If candidate move decreases the cutsize, it is realized. If it increases the cutsize, then it is realized with a probability which decreases with the amount of increase in the total cutsize. Acceptance probabilities of moves that increase the cost are controlled by a temperature parameter  $T$  which is decreased using an annealing schedule. Hence, as the annealing proceeds, acceptance probabilities of uphill moves decrease. This method over performs, in the quality of cutsize, all the previous explained Kernighan-Lin based approaches [15]. However, its run-time is too large, this makes it impractical. Optimizations in parameters of Simulated Annealing, and extensive empirical studies have been done by Johnson et. al. [9].

#### Ratio Cut :

Wei-Cheng [26] [2] present a new heuristic called *Ratio Cut*. This is basically, Kernighan-Lin based hypergraph bisection heuristic with a new cost function. They put the balance criteria into the cost definition. Their method

gives highly uneven partitions.

### Hybrid Approaches :

It is known that Kernighan-Lin based algorithms perform poorly in very sparse graphs (hypergraphs) and in large graphs. To handle this problem a number of clustering methods have been proposed. Cong-Smith [3] introduced a clustering algorithm which works on the graphs. They convert the hypergraph to the graph by representing a  $r$ -terminal net by a  $r$ -clique. Then they use a heuristic algorithm to construct the clusters. The clustered graph is given as input to the Fiduccia-Mattheyses algorithm. Shin-Kin [24] proposed a clustering algorithm which works on hypergraphs, then a KL based heuristic is used to partition the clustered hypergraph.

## 2.3.2 Constructive Algorithms

Some of the constructive algorithms are based on eigenvector approaches such as Hadley et.al [7] and Hagen-Kahng [8].

As can be noticed, the clustering algorithms explained in the previous section can also be expressed as a constructive algorithm, if we allow to enlarge a cluster up to a partition size.

Two well known constructive heuristics are :

### Kahng's Method :

Kahng [11] presents a constructive hypergraph bisection algorithm which has a run-time complexity  $\mathcal{O}(n^2)$  where  $n$  is the number of nodes in the hypergraph. His algorithm constructs an *intersection graph* from the given hypergraph. Each node of the graph corresponds to a net in the hypergraph and two nodes in the graph are connected only if the respective pair of nets have at least one cell in common in hypergraph. Note that this intersection graph concept is similar to the dual graph concept exploited in this study (Section 4.1). His algorithm selects a seed node at random and finds the furthest node from it in the intersection graph. Then it uses *breadth-first search*, starting from those two nodes, to find an initial cut in the intersection graph. This initial cut in the intersection graph corresponds to a *partial bipartition* in the hypergraph. The partial bipartition is completed by a PLA folding based algorithm resulting in

a bipartition of the hypergraph. This method gives uneven partitions, since no size information is stored in the intersection graph.

### WHB :

Kamidoi et.al. [12] extend the Kahng's method, by introducing the *net-graph* where nets are also taken as nodes in addition to the cells of the hypergraph. The edge set of the netgraph contains the *incidence* information of the hypergraph. That is, if a cell in the hypergraph is incident to a net, then the corresponding cell-node in the netgraph is adjacent to the corresponding net-node. Their algorithm selects a random net-node as a seed and finds the furthest net-node to use as the second seed. Modified version of breadth-first search is then used to construct an initial cut. It takes care of the weight of the parts. Then, a heuristic is used to complete the cut into a bipartition of the hypergraph. Their algorithm also requires  $\mathcal{O}(n^2)$  computation time. Their cutsizes results are comparable with the Kahng's method. Their test data were sparse and random and they claim that algorithm WHB performs 14% better than FM.



## Chapter 3

# Minimum Degree Ordering

The motivation for this work is from the direct solution of large sparse linear systems. Let  $\mathbf{A}$  be a large  $n$ -by- $n$  sparse symmetric positive definite matrix. The direct solution of the linear system

$$\mathbf{Ax} = \mathbf{b}$$

involves factoring the matrix  $\mathbf{A}$  into  $\mathbf{LL}^T$ , where  $\mathbf{L}$  is the lower triangular Cholesky factor of  $\mathbf{A}$ . When  $\mathbf{A}$  is factored, it normally suffers some fill. Since  $\mathbf{PAP}^T$  is also symmetric and positive definite for any permutation matrix  $\mathbf{P}$ , we can instead solve the reordered system

$$(\mathbf{PAP}^T)(\mathbf{Px}) = \mathbf{Pb}.$$

The choice of  $\mathbf{P}$  can have a dramatic effect on the amount of fill that occurs during the factorization. Thus, it is standard practice to reorder the rows and columns of the matrix before performing the factorization.

The problem of finding a *best* ordering for  $\mathbf{A}$  in the sense of minimizing the fill is computationally intractable: an *NP-hard* problem [27]. We are, therefore, obliged to rely on heuristic algorithms. By far, the most popular fill-reducing scheme used is the Tinney's *Minimum Degree* (MD) algorithm [25], which corresponds to the Markowitz scheme [20] for unsymmetric matrices. This scheme is based on the following observation:

Suppose that  $i - 1$  rows/columns are selected for reordering. Note that this corresponds to determining the first  $i - 1$  rows/columns of the  $\mathbf{P}$  matrix. The number of non-zeros in the filled graph for those rows/columns is fixed. In order to reduce the number of non-zeros in the  $i$ -th row/column, it is intuitive

that in the sub-matrix remaining to be factored, the row/column with the fewest non-zeros should be selected as the  $i$ -th row/column. In other words, the scheme may be regarded as a method that reduces the fill of a matrix by a local minimization.

### 3.1 The Basic Algorithm

The MD algorithm can easily be described in terms of ordering a symmetric graph using the elimination graph model [21]. Generally, it works only with the zero/nonzero structure of a symmetric matrix and *simulates* in some manner the steps of symmetric Gaussian elimination. The zero/nonzero structure of a sparse symmetric matrix  $\mathbf{A}$  can easily be represented by a structure graph  $G^A = (V^A, E^A)$ . Each row/column  $i$  in a sparse matrix  $\mathbf{A}$  is associated with a node  $i$  in its structure graph. Two nodes  $i$  and  $j$  in the structure graph are connected (i.e.,  $\{i, j\} \in E^A$ ) only if  $a_{ij} \neq 0$  in the sparse matrix  $\mathbf{A}$ . Let  $G_0 = (V_0, E_0) = G^A$  be the structure graph of given matrix. We will use notation  $G_i$  to denote the  $i$ -th elimination graph which is obtained by the elimination of  $i$  nodes from the initial graph  $G_0$ .  $adj_{G_i}(x)$  denotes the adjacency list of the node  $x$  in the  $i$ -th elimination graph  $G_i$ . At each elimination step a node  $x_i$  in  $G_{i-1}$  is selected, and eliminated graph  $G_i$  is obtained from  $G_{i-1}$  by:

- deleting node  $x_i$  and its incident edges in  $G_{i-1}$ ,
- adding edges to graph so that nodes in  $adj_{G_{i-1}}(x_i)$  are pairwise adjacent in  $G_i$ .

Based on the transformation rule, we note that if a node  $v$  is not adjacent to  $x_i$  in  $G_{i-1}$

$$adj_{G_i}(v) = adj_{G_{i-1}}(v)$$

However, if  $v \in adj_{G_{i-1}}(x_i)$ , then we have

$$adj_{G_i}(v) = (adj_{G_{i-1}}(x_i) \cup adj_{G_{i-1}}(v)) - \{v, x_i\}.$$

Therefore, only the degree of a node in  $adj_{G_{i-1}}(x_i)$  may change after the elimination graph transformation from  $G_{i-1}$  to  $G_i$  due to the deletion of edges incident to  $x_i$  and possible addition of new edges joining nodes adjacent to

---

```

Function MinimumDegree
   $G_0 \leftarrow G^A$ 
   $i \leftarrow 1$ 
  while  $i \leq |V^A|$  do
    In the elimination graph  $G_{i-1} = (V_{i-1}, E_{i-1})$ ,
      choose a node  $x_i$  of minimum degree
    Form the new elimination graph  $G_i = (V_i, E_i)$ ,
      by eliminating the node  $x_i$  from  $G_{i-1}$ 
     $i \leftarrow i + 1$ 
endFunction

```

---

Figure 3.1. Basic Minimum Degree Algorithm

$x_i$ . Using the elimination graph model, the basic algorithm is presented in Figure 3.1.

Filled graph of  $G^A = (V^A, E^A)$  is defined as symmetric graph  $G^F = (V^F, E^F)$ , where  $\mathbf{F} = \mathbf{L} + \mathbf{L}^T$ . Obviously,  $V^F = V^A$ , and  $E^F$  consists of all edges in  $E^A$  and all filled edges during factorization. The filled graph  $G^F$  can easily be constructed from the sequence of elimination graphs using the following lemma,

**Lemma 3.1** *The edge  $\{x_i, x_j\} \in E^F$  if and only if  $\{x_i, x_j\} \in E$  or  $\{x_i, x_k\} \in E^F$  and  $\{x_k, x_j\} \in E^F$  for some  $k < \min\{i, j\}$ .*

### 3.2 Implementation with Quotient Graph Model

The characterizations of  $G_i$  (for  $i = 1, \dots, |V^A|$ ) and  $E^F$  can be directly computed in terms of the *original* graph  $G^A$ , using the *reachable set* concept. Let  $S$  be a subset of the node set and  $u \notin S$ . The node  $u$  is said to be *reachable* from a node  $y$  *through*  $S$  if there exist a path  $(y, v_1, \dots, v_k, u)$  from  $y$  to  $u$  for  $k \geq 0$  such that  $v_i \in S$  for  $1 \leq i \leq k$ .  $Reach(y, S)$  denotes the reachable set of  $y$  through  $S$ , and defined as

$$Reach(y, S) = \{u \notin S | u \text{ is reachable from } y \text{ through } S\}.$$

The edge set of the elimination graphs can be computed using the following theorem

**Theorem 3.1** [6] *The edge  $\{u, v\} \in E_i$  if and only if  $v \in \text{Reach}(u, S_i)$ , where  $S_i = \{x_1, \dots, x_i\}$  denotes the sequence of nodes eliminated in the first  $i$  steps of the MD algorithm.*

Hence, the adjacency set of a node  $u \notin S_i$  in  $G_i$  can be calculated by generating the reachable set of  $u$  through  $S_i$ , i.e.,

$$\text{adj}_{G_i}(u) = \text{Reach}_{G_0}(u, S_i) \quad \text{for } i = 1, \dots, |V^A|.$$

The only disadvantage of this *implicit* representation is the amount of work required to determine reachable sets can be large, especially at later stages of elimination. However, it has a small and predictable storage requirement. Note that the maximum amount of storage requirement is *unpredictable* in the explicit representation of elimination.

The *quotient graph* concept is introduced to reduce the amount of work to generate reachable sets. In quotient graphs, connected eliminated nodes are coalesced in order to shorten the length of paths to uneliminated nodes. Let  $G = (V, E)$  be a given graph and let  $\mathcal{P}$  be a  $p$ -way partition on its node set  $V$ :

$$\mathcal{P} = (V_1, \dots, V_p)$$

That is  $\bigcup_{k=1}^p V_k = V$  and  $V_i \cap V_j = \emptyset$  for  $i \neq j$ . We define the *quotient graph* of  $G$  with respect to  $\mathcal{P}$  to be the graph  $G/\mathcal{P} = (\mathcal{P}, \mathcal{E})$ , where  $\{V_i, V_j\} \in \mathcal{E}$  if and only if  $\text{adj}(V_i) \cap V_j \neq \emptyset$ . Here,  $\text{adj}(V_i) = \bigcup_{v \in V_i} \text{adj}_G(v)$ .

**Definition 2** *Let  $\mathcal{V}(S)$  denotes the set of connected components in the subgraph  $G(S)$ . Then the partitioning on the node set  $V$ ,*

$$\overline{\mathcal{V}}(S) = \mathcal{V}(S) \cup (V - S)$$

*uniquely defines the quotient graph  $G/\overline{\mathcal{V}}(S)$ .*

Hence, elimination graphs can be efficiently represented using quotient graphs according to the following theorem:

**Theorem 3.2** [6] For  $v \in V - S_i$ ,

$$\text{Reach}_G(v, S_i) = \text{Reach}_{Q_i}(v, \mathcal{V}(S_i))$$

where  $Q_i = G/\bar{V}(S_i) = (\bar{V}(S_i), \mathcal{E}_i)$ .

## Chapter 4

# Circuit Partitioning Using MD Ordering

The algorithm presented in this work is based on the MD ordering algorithm. Our algorithm uses quotient graph model for elimination, because of its storage advantage over the basic MD algorithm. Quotient graph model basically inherits the storage advantage of reachable sets model, and improves the run-time of this model by introducing *supernode* concept which is not more than coalescing the connected eliminated nodes. The proposed algorithm will be referred here as *Quotient Minimum Degree for Balanced Hypergraph Partitioning (QMD-BHP)* algorithm.

Section 4.1 presents the dual graph concept. The idea behind the node selection scheme in the dual graph and its correspondence to the original hypergraph are discussed in Section 4.2. Algorithms for size and valence computations needed in node selections are presented in Section 4.3. Section 4.4 contains discussion about the elimination graph transformations to be performed after node selections. An algorithm to improve the balance quality of the partition found by the QMD-HP algorithm is proposed and presented in Section 4.5. Finally, Section 4.6 discusses the computational complexity of the proposed QMD-BHP.

## 4.1 Dual Graph

Let hypergraph  $H = (C, N)$  be given where  $C$  is set the of cells, and  $N$  is the set of nets (hyperedges). Each  $n \in N$  is a subset of  $C$  which has a cardinality of at least two, i.e. each net connects two or more cells. There is one node in  $G$  corresponding to each net in  $H$ . Two nodes in  $G$  are connected if only if the respective pair of nets have at least one cell (pin) in common in  $H$ . Let  $pins(n)$  denotes the set of the cells incident to net  $n$ , and  $nets(c)$  denotes the set of the nets connected to cell  $c$ . Subscript  $H$  and  $G$  will be used to denote hypergraph and graph, respectively. For example,  $pins_H(n)$  will denote the pin-list of net  $n$  in hypergraph  $H$ , and  $pins_G(n)$  will denote the pin-list of the net associated with the node  $n$  in graph  $G$ . As will be explained later,  $pins_G(n)$  is a *dynamic* list, whereas  $pins_H(n)$  is a *static* list. We will skip subscript if it is clear from the context. Using this notation, formal definition of dual graph is as follows;

**Definition 3** *Dual graph of hypergraph  $H = (C, N)$  is a graph  $G = (N, E)$ , where  $N$ , net list of  $H$ , is the node set of  $G$ , and  $e = \{n_i, n_j\} \in E$  if and only if  $n_i, n_j \in N$ , for  $1 \leq i, j \leq |N|$ , such that  $i \neq j$  and  $pins_H(n_i) \cap pins_H(n_j) \neq \emptyset$ .*

Our dual graph has several attributes associated with each node. The attributes  $pins_G(n)$  and  $onepins(n)$  for each node  $n$  of  $G$  are defined as follows

- $onepins(n) = \{c | c \in pins_H(n) \text{ and } deg_H(c) = 1\}$ .
- $pins_G(n) = pins_H(n) - onepins(n)$ .

A cell is called a *one-pin cell* if its degree is one, i.e. it is connected to only one net in  $H$ . Note that only one-pin cells in  $H$  do not introduce any edges into  $G$ . Hence, these cells are excluded from the pin-lists of the respective nodes in  $G$ . One-pin cells connected to net  $n$  is denoted by  $onepins(n)$ . Hence,  $|pins_H(n)| = |pins_G(n)| + |onepins(n)|$ . Figure 4.1 illustrates a sample hypergraph  $H$  with 10 cells, 9 nets and its dual graph  $G$  with 9 nodes, 15 edges. The pin-lists of the nodes of the dual graph are illustrated in brackets. In this example,  $pins_H(n_8) = \{7, 8, 9\}$  whereas  $pins_G(n_8) = \{7, 8\}$  since cell 9 is a one-pin cell and hence  $onepins(n_8) = \{9\}$ .

By referring to Definition 2,  $\mathcal{Q} = G/\overline{\mathcal{N}}(S) = (\overline{\mathcal{N}}(S), \mathcal{E})$  denotes the dual quotient graph of hypergraph  $H$ . Hence,  $\mathcal{Q}_i = (\overline{\mathcal{N}}(S_i), \mathcal{E}_i)$  corresponds to

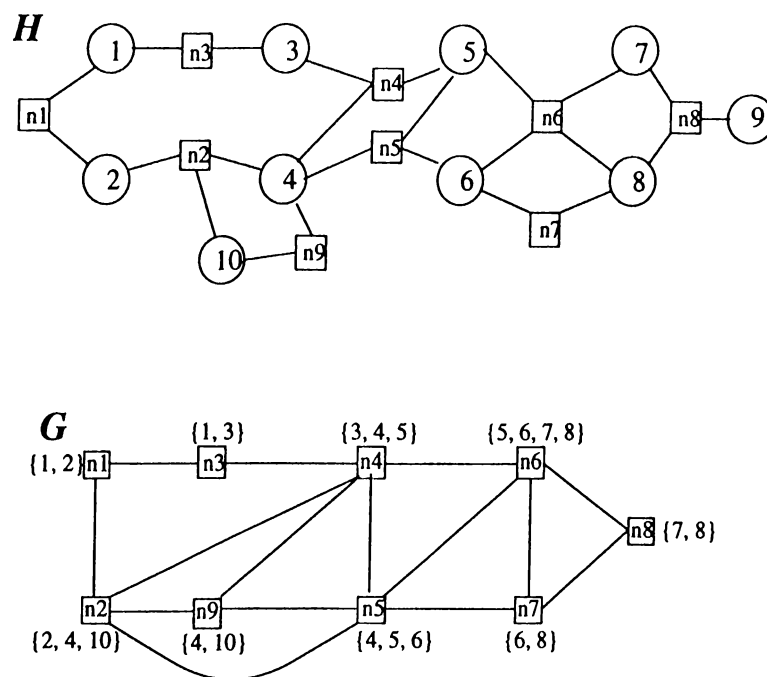


Figure 4.1. A sample hypergraph  $H$  and its dual graph  $G$

the  $i$ -th elimination graph  $G_i = (N_i, E_i)$  where  $S_i = \{n_1, \dots, n_i\}$  is the sequence of eliminated nodes in the first  $i$  steps of the MD algorithm. Figure 4.2 illustrates the pseudo-code for the dual graph construction algorithm. In this pseudo-code, attributes without subscript refer to the dual graph domain. Here,  $adj_{\mathcal{Q}}(n)$  refers to the set of nodes adjacent to node  $n$  in  $\mathcal{Q}$ , and  $deg_{\mathcal{Q}}(n)$  refers to the number of nodes in this set, i.e.  $deg_{\mathcal{Q}}(n) = |adj_{\mathcal{Q}}(n)|$ . Note that  $\overline{N}(\emptyset) = N$ , and hence  $\mathcal{Q}_0 = G_0$  initially. First outer *for-loop* in Figure 4.2, initializes the attributes of nodes of  $\mathcal{Q}$ . Second outer *for-loop* constructs the edge set  $\mathcal{E}$  of  $\mathcal{Q}$  and computes *onepins* attribute and initializes the pin-list of each node. Third outer *for-loop* initializes other node attributes to be used for node selection during the QMD algorithm.

## 4.2 Node Selection

We will discuss here only the partitioning of hypergraphs with unweighted cells and nets for the sake of clarity of the presentation. The proposed algorithm is applicable for hypergraphs with weighted cells and nets with minor modifications. In partitioning of a hypergraph with unweighted nets, the cutsizes  $\mathcal{C}(\mathcal{P})$



---

```

/* input : a hypergraph  $H = (C, N)$  */
/* output : a dual quotient graph  $\mathcal{Q}_0 = (\overline{\mathcal{N}}(\emptyset), \mathcal{E})$  */
Function ConstructDual( $H, \mathcal{Q}$ )
 $\mathcal{E} \leftarrow \emptyset$ 
for  $n = 1$  to  $|N|$  do
     $pins(n) \leftarrow \emptyset$ 
     $onepins(n) \leftarrow \emptyset$ 
for  $c = 1$  to  $|C|$  do
    if  $deg_H(c) = 1$  then
        let  $n$  be the only net incident to cell  $c$  (i.e.  $nets_H(c) = \{n\}$ )
         $onepins(n) \leftarrow onepins(n) \cup \{c\}$ 
    else
        for each  $n \in nets_H(c)$  do
             $pins(n) \leftarrow pins(n) \cup \{c\}$ 
        for each net pair  $\{n_i, n_j\}$  incident to cell  $c$  (i.e.  $n_i, n_j \in nets_H(c)$ ) do
             $\mathcal{E} \leftarrow \mathcal{E} \cup \{\{n_i, n_j\}\}$ 
for  $n = 1$  to  $|N|$  do
     $csize(n) \leftarrow |pins_H(n)|$ 
     $valence(n) \leftarrow deg_{\mathcal{Q}}(n)$ 
    for each  $m \in adj_{\mathcal{Q}}(n)$  do
        if  $(pins(m) \subseteq pins(n))$  and  $(onepins(m) = \emptyset)$  then
             $valence(n) \leftarrow valence(n) - 1$ 
endFunction

```

---

Figure 4.2. Construction of dual quotient graph

of a partition  $\mathcal{P}$  simplifies into

$$\mathcal{C}(\mathcal{P}) = |N_E| = |N| - |N_I|.$$

Hence, min-cut hypergraph partitioning becomes equivalent to maximizing the number  $|N_I|$  of internal nets. In the proposed algorithm, selecting a node  $n$  in  $G$  corresponds to making net  $n$  in  $H$  an internal net. Initially, in  $G_0 = G$ , all nodes are assumed to be separator nodes and there exists no *node-clusters*. Hence, in  $H_0$ , all nets are assumed to be external nets and there exists no *cell-clusters*.

Consider selecting a node  $n_i$  in the elimination quotient graph  $\mathcal{Q}_{i-1}$ . If there exists no previously selected node in the adjacency list of  $n_i$ , node  $n_i$  becomes a cluster node  $n_i$  in  $\mathcal{Q}_i$  representing the node-cluster  $N_{n_i}$  in  $G_i$ . Otherwise, node  $n_i$  combines with the cluster nodes in its adjacency list to become a cluster node in  $\mathcal{Q}_i$  representing the node-cluster

$$N_{n_i} = \left( \bigcup_{n_j \in \text{cadj}_{\mathcal{Q}_{i-1}}(n_i)} N_{n_j} \right) \cup \{n_i\} \quad (4.1)$$

in  $G_i$ , where  $\text{cadj}_{\mathcal{Q}_{i-1}}(n_i) = \mathcal{N}(S_{i-1}) \cap \text{adj}_{\mathcal{Q}_{i-1}}(n_i)$  represents the set of cluster nodes adjacent to  $n_i$  in  $\mathcal{Q}_{i-1}$ . Recall that  $S_{i-1} = \{n_1, \dots, n_{i-1}\}$  denotes the sequence of nodes selected during the first  $i-1$  steps, and  $\mathcal{N}(S_{i-1})$  denotes the overall set of cluster nodes in  $\mathcal{Q}_{i-1}$ . Note that cluster nodes in  $\text{cadj}_{\mathcal{Q}_{i-1}}(n_i)$  are removed from the cluster node set  $\mathcal{N}(S_i)$  during this transformation. In both cases, the new node-cluster  $N_{n_i}$  in  $G_i$  induces a new cell-cluster  $C_{n_i}$  in  $H_i$ . In the former case, the respective cell-cluster  $C_{n_i}$  in  $H_i$  contains only the pins of  $n_i$ , i.e.  $\text{pins}_H(C_{n_i}) = \text{pins}_H(n_i)$ . In the latter case, pins of net  $n_i$  combine with the pins of the cell-clusters in  $H$  corresponding to the cluster nodes in  $\text{cadj}_{\mathcal{Q}_{i-1}}$  to form a new cell-cluster  $C_{n_i}$ . That is,

$$\text{pins}_H(C_{n_i}) = \left( \bigcup_{n_j \in \text{cadj}_{\mathcal{Q}_{i-1}}(n_i)} \text{pins}_H(C_{n_j}) \right) \cup \text{pins}_H(n_i) \quad (4.2)$$

In this notation, each cell-cluster in  $H$  is labeled with the last net made internal in that cluster. Note that node-clusters and the respective cell-clusters constitute connected components in  $G$  and  $H$ , respectively. Figure 4.3 illustrates the elimination steps in the dual quotient graph of the sample hypergraph given in Figure 4.1. In this figure, each  $\mathcal{Q}_i$  is also associated with the respective  $H_i$  to illustrate the cell-cluster formation. Selection of  $n_8$  in  $\mathcal{Q}_1$  which forms the cell-cluster  $C_{n_8}$ , where  $\text{pins}_H(C_{n_8}) = \text{pins}_H(n_8) = \{7, 8\}$  is an example for the

former case. Selection of  $n_7$  in  $\mathcal{Q}_3$  which form the cell-cluster  $C_{n_7}$  such that  $pins_H(C_{n_7}) = pins_H(C_{n_8}) \cup pins_H(n_7) = \{6, 7, 8, 9\}$ , where  $n_8 \in cadj_{\mathcal{Q}_3}(n_7)$ , is an example for the latter case.

Consider the selection of a node  $n_i$  with the minimum degree in the elimination graph  $G_{i-1}$  to form a node-cluster  $N_{n_i}$  in  $G_i$ . This choice is a greedy choice in the hope that nodes with smaller degree will introduce less fills compared to the nodes with larger degrees. However, if the node-cluster  $N_{n_i}$  satisfies  $|N_{n_i}| + |adj_G(N_{n_i})| < |N|$ ,  $adj_G(N_{n_i})$  forms a separator for  $N_{n_i}$ . Furthermore, we have

**Theorem 4.1** [19]  $adj_{G_{i-1}}(n_i) = adj_G(N_{n_i})$  and  $deg_{G_{i-1}}(n_i) = |adj_G(N_{n_i})|$ .

Hence, the greedy choice in selecting node  $n_i$  in  $G_{i-1}$  also corresponds to a locally optimal choice in minimizing the node-separator size  $|adj_G(N_{n_i})|$ . Note that nodes in  $adj_G(N_{n_i})$  will either combine with cluster  $N_{n_i}$  to form new clusters or remain in the separator during the future node selections. In other words, they have no chance to be included in other node-clusters which will not contain  $N_{n_i}$ . Hence, local minimization of the separator size also has the desirable effect of even distribution of the remaining unselected nodes among the other node-clusters.

In the proposed algorithm, we grow node-/cell- clusters in  $G/H$  as connected-components similar to MD algorithm. However, the criteria for selecting a node  $n_i$  in  $G_{i-1}$  is the local minimization of the *net-cut* (*net-separator*) size of the cell-cluster  $C_{n_i}$  that will be induced by the node-cluster  $N_{n_i}$  to be formed upon selecting  $n_i$ . Let,

$$\begin{aligned} extnets_H(C_{n_i}) = \{n_j \in N \mid & pins_H(n_j) \cap pins_H(C_{n_i}) \neq \emptyset \\ & \wedge pins_H(n_j) - pins_H(C_{n_i}) \neq \emptyset\} \quad (4.3) \end{aligned}$$

represents the set of external nets (real net-cut) of cluster  $C_{n_i}$ . That is, an external net of a cell-cluster has at least one pin in that cluster and at least one pin outside that cluster. In a dual analogy to the MD algorithm, nets in  $extnets_H(C_{n_i})$  will either become the internal nets of the cluster that will contain  $C_{n_i}$  or remain in the net-cut in the future node/net selections in  $G/H$ . That is, they have no chance of becoming internal nets of cell-clusters which do not contain  $C_{n_i}$ . Hence, similar to the MD algorithm, local minimization of the net-cut size also has the desirable effect of even distribution of the remaining unselected nets as internal nets among the other cell-clusters.

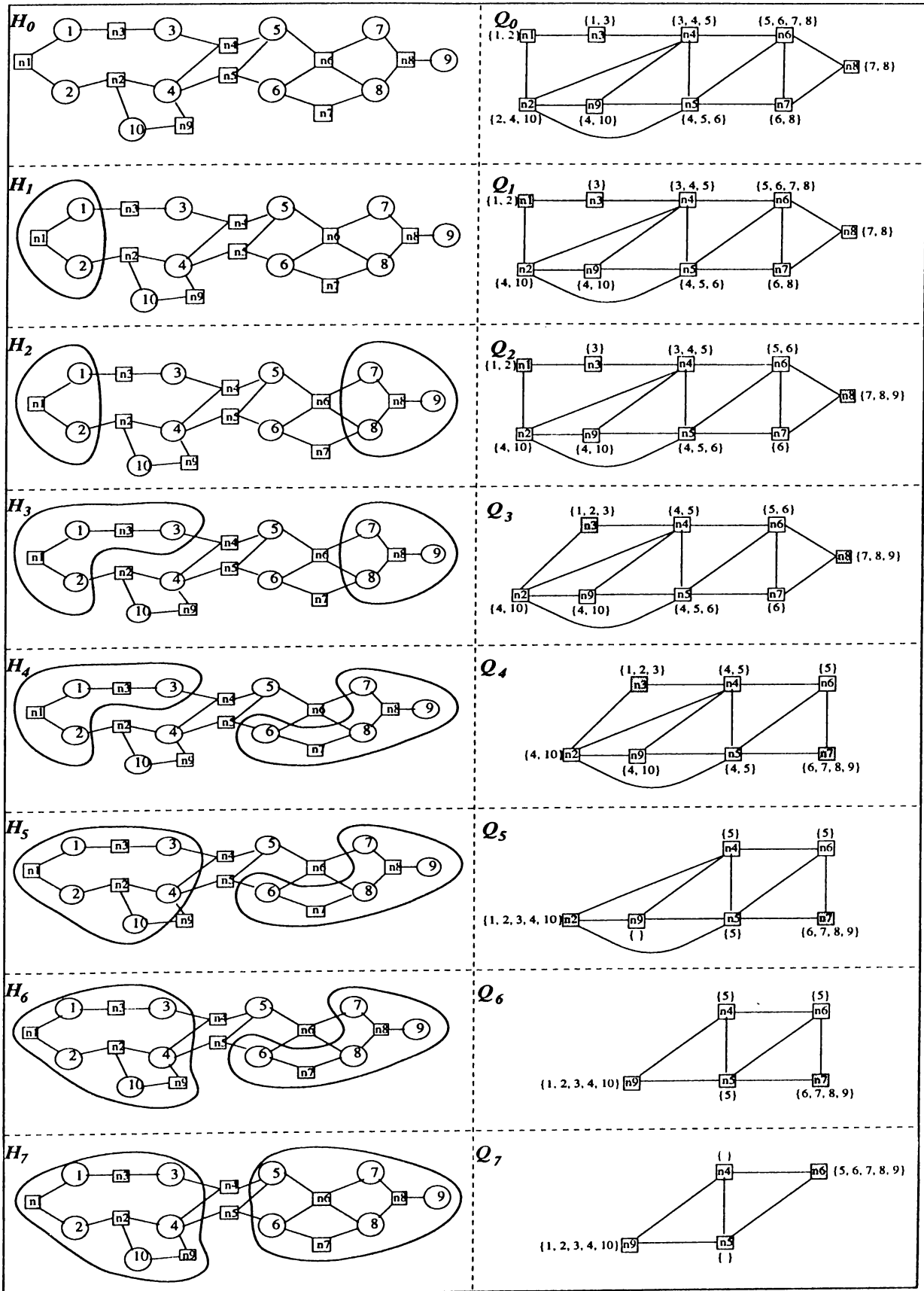


Figure 4.3. Elimination steps

The net-cut size of a cell-cluster that will be induced by the node-cluster to be formed upon selecting a node in  $G$  will be referred here as the *valence* of that node. However, large number of *ties* occur during node selections according to the valence values as in the MD algorithm. The selection of the next node with minimum valence from the candidate set is effectively determined by the initial ordering which essentially determines the way ties are resolved. In this work, we propose a tie-breaking strategy which enables growing balanced cell-clusters. In the proposed algorithm, when more than one node has the the same valence, the one with the minimum cluster size is selected first. Here, cluster size of a node in an elimination graph refers to the size of cell-cluster that will be induced by the node-cluster to be formed upon selecting that node. If the cells of the hypergraph are unweighted, the size of a cell-cluster is equal to the number of cells in that cluster. Our selection scheme does not allow a cluster to grow beyond a predetermined maximum part size. That is, unselected nodes whose cluster sizes exceed the indicated maximum part size are not considered during selections. The maximum part size is selected as  $\frac{|C|}{k} \cdot (1 + \frac{\Delta}{2})$  where  $\Delta$  is the imbalance ratio (Section 2.2) and  $\frac{|C|}{k}$  denotes the size of a part under perfect balance conditions.

### 4.3 Size and Valence and Calculations

Both stopping criteria for cluster expansion and tie-breaking criteria necessitate the cluster size (*csize*) computation for each unselected node. Here,  $csize(n)$  of an unselected node  $n$  in  $Q_i$  denotes the size of the cell-cluster  $C_n$  to be induced upon selecting  $n$ . The  $csize_{G_i}(n)$  attribute of an unselected node  $n$  in  $G_i$  can be computed by finding the cardinality of the pin-set in the right-hand side of Equation 4.2. Note that pin-sets of all cell-clusters induced by the cluster nodes in  $adj_{Q_i}(n)$  are disjoint sets. Hence, the cardinality of the pin-set represented with first set-union operation can easily be computed by a simple addition. However, net  $n$  shares at least one pin with each cell-cluster induced by the cluster nodes. Hence, all we need to compute is the number of new pins to be introduced by net  $n$  to the cell-cluster  $C_n$ . For the sake of efficiency of these computations, we maintain a *dynamic* pin-list ( $pins_{G_i}(n) = pins_{Q_i}(n)$ ) for each unselected node  $n$ . Upon selecting a node  $n_i$  in  $Q_{i-1}$ , pin-list of each unselected node  $n \in nadj_{Q_{i-1}}(n_i)$  is updated as

$$pins_{Q_i}(n) = pins_{Q_{i-1}}(n) - pins_{Q_i}(n_i) \quad (4.4)$$

where  $nadj_{\mathcal{Q}_{i-1}}(n_i) = adj_{\mathcal{Q}_{i-1}}(n_i) - cadj_{\mathcal{Q}_{i-1}}(n_i)$  denotes the set of unselected nodes adjacent to  $n_i$  in  $\mathcal{Q}_{i-1}$ . Hence,  $pins_{\mathcal{Q}_{i-1}}(n)$  denotes the subset of pins of net  $n$  (excluding those in  $onepins(n)$ ) which are not assigned to any cell-cluster in  $H_{i-1}$ . Thus,  $csize_{\mathcal{Q}_i}(n)$  of an unselected node can efficiently be computed as

$$csize_{\mathcal{Q}_i}(n) = \sum_{m \in cadj_{\mathcal{Q}_i}(n)} csize_{\mathcal{Q}_i}(m) + |pins_{\mathcal{Q}_i}(n)| + |onepins(n)|. \quad (4.5)$$

Initial computation of  $csize(n)$  is given in the last *for-loop* of Figure 4.2. Note that  $cadj$  set of each node is initially empty since there is no selected nodes yet. Therefore initial  $csize$  values contains the number of pins of the respective net. Upon selecting  $n_i$  in  $\mathcal{Q}_{i-1}$ , the cluster size of only those nodes in  $rchset_{\mathcal{Q}_{i-1}}(n_i)$  should be recomputed for  $\mathcal{Q}_i$ . Pseudo-code of size calculation is given in Figure 4.7. Maintaining dynamic pin-list for each node has other merits during valence computation as will be discussed later.

The node separator  $adj_G(N_{n_i})$  of the node-cluster  $N_{n_i}$  in  $G_i$  formed upon selecting node  $n_i$  in  $G_{i-1}$  already induces a net-cut (cut-separator) for the cell-cluster  $C_{n_i}$  in  $H$ . However, this induced net-cut may be an overestimation for the real net-cut of  $C_{n_i}$  in  $H$ . That is,

$$extnets_H(C_{n_i}) \subseteq adj_G(N_{n_i}). \quad (4.6)$$

Some of the unselected nets may directly become an internal net of the cell-cluster  $C_{n_i}$  upon selecting node  $n_i$  in  $G_i$ . This happens for an unselected node  $n_j \in adj_{G_i}(n_i)$  whenever  $pins_H(C_{n_i}) \supseteq pins_H(n_j)$ . We will refer to such nodes/nets as mass-elimination nodes/nets and define the mass-elimination node/net set of an uneliminated node  $n$  as

$$masselim(n) = adj_G(N_n) - extnets_H(C_n). \quad (4.7)$$

Nodes in the mass-elimination set of a node  $n$  can be eliminated together and included into  $N_n$  upon selecting  $n$ . Our implementation forces them to be selected following the selection of  $n$ . Note that they do not introduce any extra pins to the respective cell-cluster  $C_n$  in contrast to the standard node selections. Hence, the valence of a node  $n$  in the elimination graph  $\mathcal{Q}_i$  can be computed as

$$valence_{\mathcal{Q}_i}(n) = deg_{\mathcal{Q}_i}(n) - |masselim_{\mathcal{Q}_i}(n)|. \quad (4.8)$$

Here,  $deg_{\mathcal{Q}_i}(n)$  denotes the degree of node  $n$  in  $\mathcal{Q}_i$  which is the selection criteria in the original MD algorithm. Hence, valence computations necessitate finding the mass-elimination node set for each unselected node.

---

```

Function ReachSet(n)
rchset  $\leftarrow \emptyset$ 
for each v  $\in$  adj(n) do
    if deg(v)  $\geq 0$  then /* v is an uneliminated node */
        rchset  $\leftarrow$  rchset  $\cup$  {v}
    else /* v is a cluster node */
        for each u  $\in$  adj(v) - {n} do
            rchset  $\leftarrow$  rchset  $\cup$  {u}
return rchset
endFunction

```

---

Figure 4.4. Reachable set calculation

**Theorem 4.2** *An unselected node*  $m \in \text{masselim}_{\mathcal{Q}_i}(n)$  *if and only if*  $\text{onpins}(m) = \emptyset$  *and*  $\text{cadj}_{\mathcal{Q}_i}(m) \subseteq \text{cadj}_{\mathcal{Q}_i}(n)$  *and*

$$\text{either (i) } m \in \text{nadj}_{\mathcal{Q}_i}(n) \wedge \text{pins}_{\mathcal{Q}_i}(m) \subseteq \text{pins}_{\mathcal{Q}_i}(n)$$

$$\text{or (ii) } m \in \text{rchset}_{\mathcal{Q}_i}(n) \wedge m \notin \text{nadj}_{\mathcal{Q}_i}(n) \wedge \text{pins}_{\mathcal{Q}_i}(m) = \emptyset$$

Proof easily follows by noting the two facts. First, unassigned pins (cells) of node  $m$  should be assigned to  $C_n$  by the selection of node  $n$ . Second, the cell-clusters which contains the previously assigned pins of  $m$ , should be merged into  $C_n$  by the selection of node  $n$ .

The third *for-loop* in Figure 4.2 performs the initial valence computations. The second *for-loop* in Figure 4.7 performs the valence update for a node which is in the reachable set of the selected node. Note that, during the  $\mathcal{Q}_{i-1} \rightarrow \mathcal{Q}_i$  transformation, only the degree and mass-elimination node set of a node in the reachable set of node  $n_i$  should be considered for update (See pseudo-code in Figure 4.5 and 4.7).

#### 4.4 Graph Transformation

Let  $n_i$  be the eliminated node in the  $G_{i-1} \rightarrow G_i$  transformation and  $S_i = \{n_1, \dots, n_i\}$  be the sequence of the eliminated nodes. Recall that, at any step  $i$  of the algorithm, the node set  $\overline{\mathcal{N}}(S_{i-1})$  of  $\mathcal{Q}_{i-1}$  contains two types of

---

```

Function DegreeUpdate(rchset)
for each  $v \in rchset$  do
     $vrchset \leftarrow ReachSet(v)$ 
     $deg(v) \leftarrow |vrchset|$ 
endFunction

```

---

Figure 4.5. Degree update

---

```

Function ClusterAdj( $n$ )
 $cadj \leftarrow \emptyset$ 
for each  $v \in adj(n)$  do
    if  $deg(v) < 0$  then
         $cadj \leftarrow cadj \cup \{v\}$ 
return  $cadj$ 
endFunction

```

---

Figure 4.6. Algorithm for finding the cluster adjacency

---

```

Function UpdateValenceAndSize( $v$ )
 $vrchset \leftarrow ReachSet(v)$ 
 $vcadj \leftarrow ClusterAdj(v)$ 
 $csize(v) \leftarrow \emptyset$ 
for each  $u \in vcadj$  do
     $csize(v) \leftarrow csize(v) + csize(u)$ 
 $csize(v) \leftarrow csize(v) + |pins(v)| + |onepins(v)|$ 
 $valence(v) \leftarrow deg(v)$ 
for each  $u \in vrchset$  do
    if ( $onepins(u) = \emptyset$ ) and ( $pins(u) - pins(v) = \emptyset$ ) then
         $ucadj \leftarrow ClusterAdj(u)$ 
        if  $ucadj \subseteq vcadj$  then
             $valence(v) \leftarrow valence(v) - 1$ 
endFunction

```

---

Figure 4.7. Update of valence and cluster size



nodes; cluster nodes and uneliminated nodes denoted by the sets  $\mathcal{N}(S_{i-1})$  and  $N - S_{i-1}$ , respectively. Furthermore, the edge set  $\mathcal{E}_{i-1}$  contains two types of edges; edges between uneliminated node pairs and edges between cluster nodes and uneliminated nodes. Hence, given  $\mathcal{Q}_{i-1} = (\overline{\mathcal{N}}(S_{i-1}), \mathcal{E}_{i-1})$  with  $\mathcal{Q}_0 = G_0$ , the corresponding quotient graph transformation  $\mathcal{Q}_{i-1} \rightarrow \mathcal{Q}_i$  can be performed as follows: cluster nodes in  $cadj_{\mathcal{Q}_{i-1}}(n_i)$  are removed from the connected component set and the cluster node  $n_i$  is added as a new connected component. That is,

$$\mathcal{N}(S_i) = \mathcal{N}(S_{i-1}) - cadj_{\mathcal{Q}_{i-1}}(n_i) \cup \{n_i\} \quad (4.9)$$

and  $n_i$  is removed from the uneliminated node set. Recall that  $\mathcal{N}(S_i)$  denotes the set of connected components in the subgraph  $G(S_i)$  and  $\overline{\mathcal{N}}(S_i) = \mathcal{N}(S_i) \cup (N - S_i)$  represents the node set of  $\mathcal{Q}_i$ . Note that, node cluster  $N_{n_i}$  in  $G_i$  is represented with the cluster node  $n_i$  in  $\mathcal{Q}_i$ .

All uneliminated nodes in the node adjacency list of the cluster node in  $cadj_{\mathcal{Q}_{i-1}}(n_i)$  are connected to  $n_i$  if they were not in the set  $nadj_{\mathcal{Q}_{i-1}}(n_i) \cup \{n_i\}$ . All edges incident to cluster nodes in  $cadj_{\mathcal{Q}_{i-1}}(n_i)$  are removed from the edge set. That is,

$$\begin{aligned} \mathcal{E}_i = & \mathcal{E}_{i-1} \cup \left\{ \{n_l, n_i\} \mid n_l \in (nadj_{\mathcal{Q}_{i-1}}(n_k) - \{n_i\}) \text{ where } n_k \in cadj_{\mathcal{Q}_{i-1}}(n_i) \right\} \\ & - \left\{ \{n_k, n_l\} \mid n_k \in cadj_{\mathcal{Q}_{i-1}}(n_i) \text{ and } n_l \in nadj_{\mathcal{Q}_{i-1}}(n_k) \right\} \end{aligned} \quad (4.10)$$

The quotient graph representation enables the use of the edge slots of the nodes in  $cadj(n_i)$  for new edges to be added to the adjacency list of  $n_i$ . It is guaranteed that the number of such edge slots are larger than or equal to the number of edges to be added to the adjacency list of  $n_i$  during transformation [6]. Figure 4.8 illustrates the pseudo-code for quotient graph transformation where  $n$  denotes the selected node.

## 4.5 More About Balancing

Although the tie-breaking strategy of QMD-BHP enables growing balanced cell-clusters, there is no bound on the minimum cluster size when the algorithm terminates. That is, when the algorithm terminates, it is guaranteed that there will be no cluster whose size exceed the maximum part size, but the partition can still be infeasible according to the problem definition (Section 2.2). The

---

```

Function GraphTrans( $n, rchset, cadj$ )
 $adj(n) \leftarrow rchset$ 
for each  $v \in rchset$  do
     $adj(v) \leftarrow (adj(v) - cadj) \cup \{n\}$ 
endFunction

```

---

Figure 4.8. Quotient graph transformation

nice property of the algorithm is that the unselected nodes in the dual graph already induce cut-nets in the hypergraph. If those cut-nets are realized, the problem reduces to the  $k$ -way number partitioning problem on the sizes of cell-clusters and remaining unassigned unit-sized pins (cells) of the cut-nets. This *NP-hard* problem can also be solved by a well-known heuristic namely *First Fit Decreasing* (FFD). Note that the number of unselected nodes/nets is an overestimation for the real cutsizes. Due to the assignment of FFD, the real cutsizes can be smaller than the number of unselected nodes/nets.

Although FFD is a successful heuristic the resulting partition can also be infeasible. That is, the imbalance ratio of the resulting partition can be larger than the predetermined imbalance ratio ( $\Delta$ ). In order to get a smaller imbalance ratio, some of the cells which are assigned to the part with maximum size should be assigned to part with the minimum size, or parts should be rearranged, such that; the difference between the part with maximum size and part with minimum size is reduced. However, this process can increase the cutsizes. We propose a simple heuristic using the cutsizes overestimation property of the algorithm. We break the largest cluster into its components by making its representative node/net unselected. Recall that the representative net of a cell-cluster is the last net which was made internal to that cluster. Making a node/net unselected corresponds to breaking the respective cell-cluster into the cell-clusters whose representative nodes were adjacent to that node in the dual quotient graph during its selection. Hence, this provides more clusters with smaller sizes to FFD by introducing only one net to the net-cut. Recall that the number of unselected nodes in the dual graph is an upper case bound on the cutsizes at any step of the algorithm. The selection of the cell-cluster for breakdown is greedy. The heuristic chooses the largest cell-cluster hoping that it contains more sub-clusters than a cell-cluster with a smaller size. Only one breakdown cannot be sufficient to get a feasible partition. Therefore, this

---

```

Function FFD(clusterset)
for each partition p do
    partweight(p)  $\leftarrow$  0
while clusterset not empty do
    Take the largest cluster C from clusterset
    Put C into minimum weighted partition
for each unselected nodes n (i.e.  $\text{deg}(n) \geq 0$ ) do
    Let U be unassigned pins of n (i.e.  $U = \text{pins}(n) \cup \text{onepins}(n)$ )
    while U is not empty do
        Take a cell c from U
        Put the cell c into minimum weighted partition
    Find the minimum and maximum weighted partitions,  $W_{min}$  and  $W_{max}$  respectively
return  $(W_{max} - W_{min})/W_{max}$ 
endFunction

```

---

Figure 4.9. First Fit Decreasing heuristic

process should be repeated until a partition which has an imbalance ratio less than the predetermined value is found.

Pseudo-code for the FFD algorithm is given in Figure 4.9. Algorithm for the balancing process is given in the Figure 4.10. Outer-most *while-loop* checks if the imbalance ratio is satisfied or not. If it is not satisfied the largest cell-cluster *L* is selected for breakdown. Note that, the representative node/net can be a mass-elimination node. Therefore, mass-elimination nodes/nets are also introduced to the net-cut together with the node whose selection causes to mass elimination. The inner most *while-loop* checks if the representative is a mass-elimination node/net or not. If it is a mass-elimination node, it is introduced to the net-cut. Hence, balancing completes the discussion of the proposed algorithm, the main algorithm of QMD-BHP is also given in the Figure 4.11.

## 4.6 Complexity Analysis

Let  $H = (C, N)$  be given input circuit, and let  $Q = (\overline{N}(S), \mathcal{E})$  be the dual quotient graph of given input hypergraph as defined in the Definition 3. Let  $d_c$

---

```

Function BalancePartitions(clusterforest)
Let clusterset be the set of clusters formed due to elimination
 $\delta \leftarrow FFD(\textit{clusterset})$ 
while  $\delta > \Delta$  do
    Delete the largest cluster L from clusterset
    while C is the only child of L in clusterforest with same cluster size
         $L \leftarrow C$ 
    for each child C of L in the clusterforest do
        Add C into clusterset
     $\delta \leftarrow FFD(\textit{clusterset})$ 
endFunction

```

---

Figure 4.10. Balancing the partitions

and  $d_n$  be maximum node and net degree in hypergraph  $H$  respectively. And let  $d_v$  be the maximum node degree in dual quotient graph  $\mathcal{Q}$ .

#### 4.6.1 Space Complexity Analysis

- Input hypergraph is stored in two link-list arrays as described in [22]. It has space complexity  $\mathcal{O}(d_c \cdot |C| + d_n \cdot |N|)$
- Adjacency list representation is used to store dual quotient graph  $G$ . The number of edges in the graph  $|E| < \sum_{c \in C} \binom{\text{deg}_H(c)}{2}$  by definition of dual graph. The space complexity of the storing adjacency list representation is  $\mathcal{O}(|E|)$ . Empirical studies on the over 40 test circuit shows that  $|E| \leq \binom{\lceil \mu \rceil + 1}{2} \cdot |C|$ , where  $\mu$  is average cell degree in the hypergraph.
- Reach set *rchset* and cluster adjacency set *cadj* is stored in one-dimensional array with the size  $|N|$  which is the worst case boundary of these sets. Therefore space complexity of these arrays is  $\mathcal{O}(|N|)$ .
- A one-dimensional array is also used as marking array to compute set operations effectively, its size is exactly  $|N|$ . That is, its space complexity is also  $\mathcal{O}(|N|)$
- The worst case space complexity of *masselim* is  $\mathcal{O}(|N|)$ .

---

```

Function Main
Initialize selectheap
Construct  $\mathcal{Q}_0$  from given Hypergraph  $H$  by calling function ConstructDual
masselim  $\leftarrow \emptyset$ 
Initialize clusterforest with empty
for each node  $n \in \mathcal{Q}_0$  do
    Insert  $n$  into selectheap
i  $\leftarrow 1$ 
while  $i \leq |N|$  do
     $n \leftarrow \text{SelectNode}()$ 
    if  $n = -1$  then
        break while-loop
     $\text{deg}(n) \leftarrow -1$ 
     $\text{rchset} \leftarrow \text{ReachSet}(n)$ 
     $\text{cadj} \leftarrow \text{ClusterAdj}(n)$ 
    for each  $v \in \text{rchset}$  do
        delete  $v$  from selectheap
    for each  $u \in \text{cadj}$  do
        Add edge  $(n, u)$  into clusterforest
    Transform  $\mathcal{Q}_{i-1}$  into  $\mathcal{Q}_i$  by calling GraphTrans( $n, \text{rchset}, \text{cadj}$ )
    Update degrees of reachable nodes by calling DegreeUpdate( $\text{rchset}$ )
    for each  $v \in (\text{adj}(n) - \text{cadj})$  do
         $\text{pins}(v) \leftarrow \text{pins}(v) - \text{pins}(n)$ 
        if ( $\text{pins}(v) = \emptyset$ ) and ( $\text{onepins}(v) = \emptyset$ ) then
             $\text{vcadj} \leftarrow \text{ClusterAdj}(v)$ 
            if  $\{n\} = \text{vcadj}$  then
                put  $v$  into masselim
    for each  $v \in (\text{rchset} - \text{masselim})$  do
        Update valence and cluster size of  $v$  by calling UpdateValenceAndSize( $v$ )
        Insert  $v$  into selectheap
     $i \leftarrow i + 1$ 
BalancePartitions(clusterforest)
endFunction

```

---

Figure 4.11. Main algorithm

---

```

Function SelectNode
if masselim =  $\emptyset$  then
  repeat
    if selectheap not empty then
       $n \leftarrow \text{ExtractMinimum}(\text{selectheap})$ 
    else
       $n \leftarrow -1$ 
    until ( $\text{csize}(n) \leq \text{maxw}$ ) or ( $n = -1$ )
  else
    Take one node from masselim into  $n$  and delete it from set
  return  $n$ 
endFunction

```

---

Figure 4.12. Node selection algorithm

- *selectheap* has space complexity  $\mathcal{O}(|N|)$ .
- *clusterforest* has space complexity  $\mathcal{O}(|N|)$ .
- *onepins* and *pinlist* contains the cells which a net connected, therefore their total space complexity is  $\mathcal{O}(p)$  where  $p$  is the total number of pins which is defined in the Chapter 2.
- Since degree, valence and cluster sizes are the integer attributes of the nodes in the dual graph, their space complexity is  $\mathcal{O}(|N|)$ .

Therefore total space complexity of the algorithm is  $\mathcal{O}(d_c \cdot |C| + d_n \cdot |N| + |E|)$  or using  $p$ ; the number of pins in the hypergraph, it is  $\mathcal{O}(p + |E|)$

## 4.6.2 Time Complexity Analysis

Reading the input hypergraph has the time complexity  $\mathcal{O}(p)$ .

*ConstructDual* :

- First loop initializes pin-lists *pins* and *onepins* in  $\mathcal{O}(|N|)$ .

- Second loop constructs the pin-lists and dual graph; inside the loop pin-list are constructed in  $\mathcal{O}(d_c)$  and for each incident net pair (there are  $\mathcal{O}(d_c^2)$  pairs) an edge check is done in  $\mathcal{O}(d_v)$  and if no such edge exist, edge is constructed in constant time. Therefore total time complexity of the second loop is  $\mathcal{O}(|C| \cdot d_c^2 \cdot d_v)$ .
- Third loop determines the valences in  $\mathcal{O}(|N| \cdot d_c \cdot d_n)$  since subset operation can be done in  $\mathcal{O}(d_n)$  using a marker array.

Total time complexity of constructing dual graph is  $\mathcal{O}(|N| \cdot d_c \cdot d_n + |C| \cdot d_c^2 \cdot d_v)$ .

*SelectNode* :

- Extracting minimum from heap is  $\mathcal{O}(\log |N|)$ .
- Taking a node from *masselim* is  $\mathcal{O}(1)$ .
- Repeat-loop may executed  $|N|$  times in the worst case but, then *while-loop* in the main algorithm is terminated. Therefore

Total time complexity of this function is  $\mathcal{O}(\log |N|)$ .

*ReachSet* :

Main loop of this function is executed maximum  $d_v$  times. If the adjacent node is not eliminated before it is added to *rchset*, otherwise all adjacents of that eliminated node are added into *rchset*. In the early stages of the algorithm is obvious that there are not so much eliminated nodes in the graph, hence this function has the time complexity  $\mathcal{O}(d_v)$ , but at the later stages since there are more eliminated nodes, the worst case time complexity of this function is  $\mathcal{O}(d_v^2)$ .

*DegreeUpdate* :

Since the degree of each node in the *rchset* must be computed, and this is done by calling *ReachSet* for each node, the worst case complexity of this function is  $\mathcal{O}(d_v^4)$ . But it must be noticed that at the early stages this function has time the complexity  $\mathcal{O}(d_v^2)$ .

*UpdateValenceAndSize* :

- Reach set of the node which is being updated ( $v$ ) is calculated in  $\mathcal{O}(d_v^2)$ .
- Cluster size of the node is computed using adjacent cluster nodes in  $\mathcal{O}(d_v)$ .
- Each node, which is in the reach set of  $v$ , is checked if it goes to the mass-elimination set of  $v$  when the node  $v$  has been selected. This requires a set subtraction which can be done in  $\mathcal{O}(d_n)$  using a mark array, and a subset check in the cluster adjacency sets, since the cluster adjacency sets contains at most  $\mathcal{O}(d_v)$  items, this subset check can also be done with the time complexity  $\mathcal{O}(d_v)$  using again a marker array. Hence, this process must be done for each item in the reach set, this step has the time complexity  $\mathcal{O}(d_v^2 \cdot (d_v + d_n))$ .

The time complexity of updating the valence and the cluster size of a node is  $\mathcal{O}(d_v^2 \cdot (d_v + d_n))$  in the worst case. Again at the early stages this process has the time complexity  $\mathcal{O}(d_v \cdot (d_v + d_n))$ .

*GraphTrans* :

- Reach set of the selected node is placed in the adjacency list of the node in  $\mathcal{O}(d_v^2)$ .
- Adjacency set of each node in the *rchret* set is updated by deleting the cluster-nodes adjacent to selected node from the set and adding the selected node as the representative of the node-cluster. This has the time complexity  $\mathcal{O}(d_v^3)$ .

Therefore total time complexity of this function is  $\mathcal{O}(d_v^3)$ , again it must be noticed that at the early stages since reach set has  $\mathcal{O}(d_v)$  items, the time complexity of the function is  $\mathcal{O}(d_v^2)$ .

*FFD* :

Let set  $U$  denotes the unselected nodes,  $k$  denotes the number of parts, and  $s$  denotes the cardinality of cluster set.



- Initializing partition weights takes  $\mathcal{O}(k)$  times.
- Assigning the cluster to parts takes  $\mathcal{O}(s \cdot (s + k))$ .
- Finding the unselected nodes set  $U$  take  $\mathcal{O}(|N| \cdot d_n)$  and assigning the pins of these nodes to parts has the time complexity  $\mathcal{O}(d_n \cdot k)$ .

Therefore the time complexity of this algorithm is  $\mathcal{O}(|N| + |U| \cdot d_n \cdot k + s \cdot (s + k))$ .

*BalancePartitions :*

The main loop of this function depends the imbalance ratio  $\Delta$  if the required ratio is not achieved largest cluster is break down its components and *FFD* algorithm is executed again. In the worst case all clusters can be broken its components until original node set of the dual graph is constructed. Therefore in the worst case this algorithm has the time complexity  $\mathcal{O}(|N| \cdot (|N| \cdot d_n \cdot k + s \cdot (s + k)))$ . But in all our experiments this loop is newer executed more than 25 times.

*Main Algorithm :*

- All nodes in the reach set of selected node  $n$  is deleted from *selectheap* in  $\mathcal{O}(d_v^2 \cdot \log |N|)$  (in the worst case).
- All neighborhood nodes is placed in the *clusterforest* in  $\mathcal{O}(d_v)$ .
- Nodes which will go *masselim* is computed in  $\mathcal{O}(d_v \cdot (d_n + d_v))$ .
- Since the valence and the cluster size of all nodes in the reach set must be recalculated this takes  $\mathcal{O}(d_v^4 \cdot (d_v + d_n) + d_v^2 \cdot \log |N|)$ .

Since the main loop is executed  $|N|$  times in the worst case, worst case time complexity of the whole algorithm is  $\mathcal{O}(d_c \cdot (|N| \cdot d_n + |C| \cdot d_c \cdot d_v) + |N| \cdot (d_v^4 \cdot (d_v + d_n) + d_v^2 \cdot \log |N|))$ . It should be noticed that at the early stages of the algorithm since reach sets are in  $\mathcal{O}(d_v)$  then the time complexity of the algorithm is reduces to  $\mathcal{O}(d_c \cdot (|N| \cdot d_n + |C| \cdot d_c \cdot d_v) + |N| \cdot (d_v^2 \cdot (d_v + d_n) + d_v \cdot \log |N|))$ .

## Chapter 5

# Experiments and Results

This section deals with the performance evaluation of the proposed algorithm, compared with two well-known heuristics: Sanchis (SN) and Simulated Annealing (SA). Each algorithm is tested using 12 benchmark circuits from LayoutSynth92 standard cell suite and Partitioning93 test suite in ACM/SIGDA Design Automation Benchmarks (also known as MCNC Benchmarks). Characteristics of the test circuits are shown in the Table 5.1, characteristics of the corresponding dual graph are also summarized in Table 5.2.

### 5.1 Implementation

Two versions of the proposed algorithm have been implemented. First version does not contain the post balancing process, we call this version as Quotient Minimum Degree for Hypergraph Partitioning (QMD-HP) . Post processing has been included in the second version Quotient Minimum Degree for Balanced Hypergraph Partitioning (QMD-BHP). Since our algorithm is constructive, it produces same results in each run. To get a different solution, we have permuted the node and net numbers randomly. As expected this has resulted in different solutions. The imbalance ratio in the QMD-BHP  $\Delta$  is set to 0.20 to ensure that we have the same balance criteria with the compared algorithms SA and SN. Reading the circuit, permuting the net and node numbers and outputting the result are also included in the execution time of the algorithm.

Sanchis's multiple-way network partitioning algorithm is also implemented.

Table 5.1. Properties of test circuits. ( $p$  is the number of pins,  $\sigma$  is standard deviation, avg is average.)

name	$ C $	$ N $	$p$	cell degree			net degree		
				max.	avg.	$\sigma$	max.	avg.	$\sigma$
balu	701	702	2493	9	3.556	1.171	117	3.551	5.285
c1355	650	618	1745	5	2.685	0.757	11	2.824	1.312
c2670	924	860	2375	5	2.570	1.118	30	2.762	2.546
c3540	1038	1016	3131	5	3.016	0.887	23	3.082	2.336
c7552	2247	2140	6171	5	2.746	0.978	137	2.884	3.877
industry1	2271	2186	7731	9	3.404	1.124	318	3.537	9.016
primary2	3014	3029	11219	9	3.722	1.549	37	3.704	3.819
s838	495	460	1261	5	2.547	0.920	33	2.741	2.398
sioo	602	383	1771	4	2.942	0.466	128	4.624	7.417
struct	1888	1888	5375	4	2.847	0.604	16	2.847	1.793
test03	1607	1618	5807	54	3.614	1.752	225	3.589	8.462
test06	1752	1641	6638	6	3.789	1.233	388	4.045	11.701

Table 5.2. Dual Graphs of Test Circuits

name	$ N $	$ E $	node degree		
			max.	avg.	$\sigma$
balu	702	3175	308	9.046	13.491
c1355	618	1421	23	4.599	2.567
c2670	860	2292	53	5.330	4.989
c3540	1016	3334	53	6.563	4.716
c7552	2140	5982	250	5.591	7.053
industry1	2186	9064	532	8.293	15.404
primary2	3029	16200	131	10.697	11.559
s838	460	1154	65	5.017	4.733
sioo	383	1749	256	9.133	14.764
struct	1888	5084	32	5.386	3.700
test03	1618	8384	344	10.363	17.257
test06	1641	8313	769	10.132	24.079

In the implementation of gain arrays we have used the  $l$  levels of one-dimensional bucket array, each bucket array consisting of  $2p + 1$  entries where  $p$  is the maximum cell degree in the circuit. At level 1 there will be one bucket array indexed from  $-p$  to  $p$ . Each of the entries in this array points a bucket array at level 2, entries of the bucket arrays in the second level also points to bucket arrays in the level 3, and so on. At the last level entries are pointers to gain nodes. This is the same data structure with [22].

The bounds on the size of parts are given as  $W_P - 0.1 \cdot W_P \leq W_i \leq W_P + 0.1 \cdot W_P$  where  $W_i$  denotes the size of part  $i$  and  $W_P$  is the perfect load balance computed as  $W_P = \frac{\text{Total cell weight of circuit}}{\text{Number of Parts}}$ . Run-time again contains the reading the input circuit and generating the random initial partitioning and outputting the result.

Our Simulated Annealing implementation based on the cooling schedule in [9] and follows the guidelines supplied in [9, 10, 16] for multiple-way partitioning. The starting temperature was set to 10 where the acceptance rate was 90%. The termination condition was met when either the acceptance rate was less than 2%, or the same cutsizes was encountered  $|C|/2$  times. The penalty function which allows the infeasible partitions is not used to ensure that each algorithm we compared selects a move in the same way.

All algorithms were implemented in C programming language, and experiments were carried out on a Sun Sparc 10 Workstation.

## 5.2 Results

Table 5.3 compares the unbalanced version of proposed algorithm (QMD-HP) with the balanced version (QMD-BHP), where  $k$  is the number of parts and  $\delta$  is the imbalance ratio. The number in the parenthesis in the cut column of QMD-BHP gives the ratio of the cut to cut found by QMD-HP. The number in the parenthesis in the Time column is calculated in the same manner. Both algorithms start without permuting the node and net numbers, i.e. QMD-BHP differs only in the last balancing procedure, whereas QMD-HP does not have any balancing post process. Times are almost the same, however there are some cases which run-time of QMD-BHP is gradually less than QMD-HP, this can only be explained by our operating system. Since it is a multi-user system with virtual paging, this time difference can be caused by page-swap.

The last column is the percentage improvement in the balance, it is calculated as;

$$Bal.Imp = \frac{\delta_{QMD-HP} - \delta_{QMD-HBP}}{\delta_{QMD-HP}} \cdot 100.$$

As can be seen in table; QMD-BHP shows the great improvement in the balance. However, in the most of the runs, there is no change in the cut since QMD-HP has also produced balanced partitioning in those test cases. On the average QMD-BHP produces 5% more cutsizes, in other words, QMD-HP over-performs the balanced version QMD-BHP by 4% in the cutsize. The question is “what is important ? Cutsize or balance ?”, since we will compare algorithm with the SN and SA our results must be balanced to be fare.

Table 5.4 shows the cutsize averages and standard deviation of the 4 programs (in fact SN-L1 and SN-L3 is the same algorithm with different parameters) in the 12 test circuits with partition number  $k$  varying from 2 to 32. SN-L1 is the Sanchis’s algorithm with the first level gain, it can be considered as multi-way Fiduccia Mattheyses’s algorithm. SN-L3 is the Sanchis’s algorithm with the three level of gains. Selection of the level 3 is based on the average net degree of our test data. Average net degree is about 3 nearly in all data. Since level concept of Sanchis’s algorithm based on the net degree, setting the level parameter larger than 3 does not effect the quality of the cutsize so much, but it increases the space requirement of the algorithm with the running time.

Each test, for each  $k$  value for each test circuit, 100 run have been done for the algorithms SN-L1, SN-L2, and QMD-BHP. Since we have 12 test circuits and 6 different  $k$  values, 7200 runs have been done for each of the programs. For each test cases 10 SA run have been done, because of its high running time requirement. We could not put the results of  $k = 32$  runs for SA because of time limitations.

The numbers in the parenthesis are the ratio of the cutsize of the respective algorithm to the cut size found by SN-L1. The improvement can be computed as  $1 - \text{cut-ratio}$ . For example, when  $k = 4$  for test circuit balu, SN-L3 shows  $1 - 0.77 = 0.23 = 23\%$  improvement, where SA shows  $1 - 0.44 = 0.56 = 56\%$  improvement and QMD-BHP has  $1 - 0.40 = 0.60 = 60\%$  improvement in the cutsize. Bold numbers indicate the best values in each row.

The minimum and the maximum cutsizes achieved in the test runs have been listed in the Table 5.5 and Table 5.6, respectively.  $k$  denotes the number

Table 5.3. Comparison of QMD-HP and QMD-BHP. ( $W_{max}$  and  $W_{min}$  are the maximum and minimum part weights respectively,  $\delta$  is unbalance ratio, %B.I. is the percent balance improvement in QMD-BHP.)

name	k	QMD-HP					QMD-BHP					
		$W_{min}$	$W_{max}$	$\delta$	Cut	Time	$W_{min}$	$W_{max}$	$\delta$	Cut	Time	%B.I.
balu	2	328	373	12.1	38	5.7	328	373	12.1	38 (1.00)	5.7 (1.00)	0.00
	4	159	211	24.6	66	4.7	159	183	13.1	78 (1.18)	4.8 (1.02)	46.78
	6	87	152	42.8	78	4.6	107	130	17.7	92 (1.18)	4.5 (0.98)	58.63
	8	74	126	41.3	81	4.2	82	99	17.2	103 (1.27)	4.3 (1.02)	58.39
	16	41	49	16.3	133	3.3	41	49	16.3	133 (1.00)	3.3 (1.00)	0.00
	32	21	22	4.5	160	3.0	21	22	4.5	160 (1.00)	3.0 (1.00)	0.00
c1355	2	292	358	18.4	29	0.6	292	358	18.4	29 (1.00)	0.6 (1.00)	0.00
	4	151	197	23.4	45	0.5	159	172	7.6	47 (1.04)	0.5 (1.00)	67.63
	6	108	109	0.9	53	0.5	108	109	0.9	53 (1.00)	0.5 (1.00)	0.00
	8	78	98	20.4	53	0.5	79	96	17.7	55 (1.04)	0.5 (1.00)	0.00
	16	39	43	9.3	59	0.4	39	43	9.3	59 (1.00)	0.5 (1.25)	0.00
	32	20	21	4.8	77	0.4	20	21	4.8	77 (1.00)	0.5 (1.25)	0.00
c2670	2	462	462	0.0	28	1.4	462	462	0.0	28 (1.00)	1.5 (1.07)	0.00
	4	231	231	0.0	46	1.3	231	231	0.0	46 (1.00)	1.3 (1.00)	0.00
	6	147	181	18.8	57	1.3	147	181	18.8	57 (1.00)	1.3 (1.00)	0.00
	8	111	142	21.8	63	1.2	112	127	11.8	65 (1.03)	1.3 (1.08)	45.90
	16	57	58	1.7	84	1.2	57	58	1.7	84 (1.00)	1.1 (0.92)	0.00
	32	28	30	6.7	113	1.0	28	30	6.7	113 (1.00)	1.1 (1.10)	0.00
c3540	2	514	524	1.9	94	6.1	514	524	1.9	94 (1.00)	6.0 (0.98)	0.00
	4	237	304	22.0	123	4.5	252	264	4.5	140 (1.14)	4.4 (0.98)	79.38
	6	156	240	35.0	147	3.7	164	196	16.3	146 (0.99)	3.8 (1.03)	53.35
	8	129	130	0.8	163	3.2	129	130	0.8	163 (1.00)	3.2 (1.00)	0.00
	16	64	65	1.5	187	2.5	64	65	1.5	187 (1.00)	2.6 (1.04)	0.00
	32	31	37	16.2	219	2.0	31	37	16.2	219 (1.00)	2.1 (1.05)	0.00
c7552	2	1099	1148	4.3	34	8.7	1099	1148	4.3	34 (1.00)	8.2 (0.94)	0.00
	4	552	591	6.6	58	9.1	552	591	6.6	58 (1.00)	8.0 (0.88)	0.00
	6	356	463	23.1	102	7.6	358	444	19.4	114 (1.12)	7.7 (1.01)	16.19
	8	257	321	19.9	140	7.1	257	321	19.9	140 (1.00)	6.9 (0.97)	0.00
	16	134	176	23.9	207	6.2	134	167	19.8	212 (1.02)	6.2 (1.00)	17.19
	32	70	71	1.4	267	5.4	70	71	1.4	267 (1.00)	5.5 (1.02)	0.00
industry1	2	1135	1136	0.1	37	40.3	1135	1136	0.1	37 (1.00)	39.4 (0.98)	0.00
	4	508	660	23.0	99	34.6	532	660	19.4	100 (1.01)	33.8 (0.98)	15.79
	6	357	459	22.2	143	35.2	358	438	18.3	144 (1.01)	32.5 (0.92)	17.81
	8	271	327	17.1	177	32.1	271	327	17.1	177 (1.00)	30.7 (0.96)	0.00
	16	139	163	14.7	254	28.9	139	163	14.7	254 (1.00)	29.0 (1.00)	0.00
	32	70	72	2.8	337	26.3	70	72	2.8	337 (1.00)	27.3 (1.04)	0.00
primary2	2	1216	1798	32.4	297	221.1	1497	1517	1.3	321 (1.08)	205.1 (0.93)	95.93
	4	625	1004	37.7	409	78.9	732	792	7.6	420 (1.03)	73.1 (0.93)	79.93
	6	431	648	33.5	420	59.6	468	576	18.8	436 (1.04)	56.2 (0.94)	44.01
	8	362	412	12.1	445	48.1	362	412	12.1	445 (1.00)	45.5 (0.95)	0.00
	16	188	189	0.5	500	32.8	188	189	0.5	500 (1.00)	33.3 (1.02)	0.00
	32	94	95	1.1	583	23.8	94	95	1.1	583 (1.00)	24.2 (1.02)	0.00
s838	2	234	261	10.3	31	0.6	234	261	10.3	31 (1.00)	0.6 (1.00)	0.00
	4	109	156	30.1	46	0.5	114	139	18.0	50 (1.09)	0.5 (1.00)	40.30
	6	76	112	32.1	53	0.5	78	91	14.3	56 (1.06)	0.6 (1.20)	55.56
	8	54	79	31.6	58	0.5	56	68	17.6	55 (0.95)	0.5 (1.00)	44.24
	16	30	35	14.3	69	0.5	30	35	14.3	69 (1.00)	0.5 (1.00)	0.00
	32	14	18	22.2	95	0.4	15	18	16.7	108 (1.14)	0.5 (1.25)	25.00
sioo	2	294	308	4.5	25	2.5	294	308	4.5	25 (1.00)	2.5 (1.00)	0.00
	4	132	162	18.5	25	2.5	132	162	18.5	25 (1.00)	2.6 (1.04)	0.00
	6	96	110	12.7	25	2.6	96	110	12.7	25 (1.00)	2.5 (0.96)	0.00
	8	66	96	31.3	25	2.6	74	78	5.1	31 (1.24)	2.5 (0.96)	83.59
	16	33	63	47.6	25	2.5	37	44	15.9	31 (1.24)	2.7 (1.08)	66.59
	32	17	20	15.0	86	1.5	17	20	15.0	86 (1.00)	1.5 (1.00)	0.00
struct	2	788	1100	28.4	41	6.9	904	984	8.1	57 (1.39)	6.8 (0.99)	71.34
	4	395	534	26.0	84	6.0	431	498	13.5	97 (1.15)	5.7 (0.95)	48.31
	6	251	454	44.7	92	5.7	279	336	17.0	131 (1.42)	5.2 (0.91)	62.06
	8	194	312	37.8	128	5.1	229	248	7.7	138 (1.08)	5.0 (0.98)	79.74
	16	92	148	37.8	165	4.2	114	122	6.6	195 (1.18)	4.6 (1.10)	82.67
	32	46	74	37.8	237	3.9	56	64	12.5	268 (1.13)	3.8 (0.97)	66.96
test03	2	765	842	9.1	92	33.1	765	842	9.1	92 (1.00)	31.5 (0.95)	0.00
	4	341	484	29.5	155	25.6	368	445	17.3	158 (1.02)	24.3 (0.95)	41.43
	6	244	362	32.6	169	24.0	252	312	19.2	191 (1.13)	23.0 (0.96)	41.00
	8	195	237	17.7	189	21.9	195	237	17.7	189 (1.00)	20.6 (0.94)	0.00
	16	98	115	14.8	241	17.0	98	115	14.8	241 (1.00)	16.9 (0.99)	0.00
	32	49	54	9.3	294	15.0	49	54	9.3	294 (1.00)	13.3 (0.89)	0.00
test06	2	827	925	10.6	67	49.5	827	925	10.6	67 (1.00)	47.5 (0.96)	0.00
	4	342	542	36.9	85	46.7	438	438	0.0	93 (1.09)	45.8 (0.98)	100.00
	6	290	295	1.7	92	45.3	290	295	1.7	92 (1.00)	44.2 (0.98)	0.00
	8	171	270	36.7	106	45.8	206	246	16.3	122 (1.15)	44.0 (0.96)	55.65
	16	108	113	4.4	154	39.7	108	113	4.4	154 (1.00)	38.1 (0.96)	0.00
	32	54	55	1.8	225	34.5	54	55	1.8	225 (1.00)	31.0 (0.90)	0.00

Table 5.4. Cutsizes averages and standard deviations ( $\sigma$ ) for test circuits.

name	k	SN-L1		SN-L3		SA		QMD-BHP	
		avg.	$\sigma$	avg.	$\sigma$	avg.	$\sigma$	avg.	$\sigma$
balu	2	35.8	8.2	36.3 (1.01)	9.0	33.2 (0.93)	7.8	36.4 (1.02)	3.8
	4	172.3	9.2	133.3 (0.77)	18.1	76.2 (0.44)	15.3	69.2 (0.40)	4.7
	6	203.4	9.2	163.4 (0.80)	9.2	125.6 (0.62)	12.3	88.7 (0.44)	2.0
	8	224.3	9.9	171.9 (0.77)	5.6	146.2 (0.65)	4.2	102.7 (0.46)	3.1
	16	260.9	7.6	183.4 (0.70)	3.9	166.2 (0.64)	3.2	133.6 (0.51)	1.4
	32	280.3	5.8	195.5 (0.70)	5.1	--	--	160.0 (0.57)	0.0
c1355	2	37.1	7.2	34.3 (0.92)	8.2	34.6 (0.93)	7.5	29.0 (0.78)	0.0
	4	98.4	6.7	78.4 (0.80)	6.0	68.4 (0.70)	2.9	46.5 (0.47)	0.5
	6	114.5	4.3	92.0 (0.80)	6.1	77.6 (0.68)	4.2	53.0 (0.46)	0.0
	8	123.0	4.9	100.3 (0.82)	6.4	80.4 (0.65)	2.6	55.0 (0.45)	0.0
	16	138.9	5.9	115.2 (0.83)	5.1	90.0 (0.65)	2.3	59.0 (0.42)	0.0
	32	161.6	8.4	128.3 (0.79)	5.0	--	--	77.0 (0.48)	0.0
c2670	2	48.4	9.6	55.2 (1.14)	11.7	44.8 (0.93)	5.3	30.7 (0.63)	5.1
	4	132.2	9.4	117.8 (0.89)	8.3	77.4 (0.59)	2.4	46.2 (0.35)	0.6
	6	163.6	11.8	131.9 (0.81)	7.9	85.8 (0.52)	3.9	57.1 (0.35)	1.6
	8	179.9	11.7	139.9 (0.78)	8.2	85.8 (0.48)	1.2	63.4 (0.35)	1.2
	16	224.5	10.4	160.7 (0.72)	8.7	99.6 (0.44)	4.9	84.2 (0.38)	1.9
	32	258.4	9.6	185.6 (0.72)	8.0	--	--	112.0 (0.43)	1.8
c3540	2	83.9	16.2	86.2 (1.03)	17.7	77.2 (0.92)	9.7	99.3 (1.18)	9.7
	4	245.8	11.4	203.8 (0.83)	21.2	144.8 (0.59)	3.7	133.3 (0.54)	8.4
	6	290.5	9.8	243.7 (0.84)	14.3	183.0 (0.63)	7.3	155.6 (0.54)	6.1
	8	311.7	11.9	262.9 (0.84)	12.8	197.4 (0.63)	7.6	166.5 (0.53)	5.4
	16	364.9	11.2	295.5 (0.81)	9.9	230.0 (0.63)	2.8	196.6 (0.54)	3.4
	32	409.6	10.3	319.1 (0.78)	7.9	--	--	223.3 (0.55)	2.7
c7552	2	44.4	15.5	47.6 (1.07)	15.8	83.6 (1.88)	4.6	32.3 (0.73)	4.8
	4	373.9	26.1	230.9 (0.62)	27.8	171.4 (0.46)	8.3	66.6 (0.18)	9.8
	6	488.3	20.1	303.8 (0.62)	37.1	219.2 (0.45)	7.7	114.3 (0.23)	11.7
	8	540.2	20.3	352.5 (0.65)	36.9	259.4 (0.48)	10.2	146.2 (0.27)	10.1
	16	648.6	22.0	442.0 (0.68)	34.0	342.2 (0.53)	15.9	203.6 (0.31)	5.5
	32	721.8	16.9	509.1 (0.71)	17.9	--	--	265.5 (0.37)	5.2
industry1	2	58.8	27.7	69.2 (1.18)	29.1	71.2 (1.21)	16.8	36.5 (0.62)	7.9
	4	423.0	28.9	293.7 (0.69)	39.1	184.8 (0.44)	19.8	111.1 (0.26)	12.1
	6	518.3	19.4	378.2 (0.73)	39.9	263.4 (0.51)	25.3	153.3 (0.30)	15.0
	8	569.4	17.9	406.2 (0.71)	32.4	293.2 (0.51)	10.7	176.3 (0.31)	9.6
	16	660.6	17.1	485.6 (0.74)	21.4	392.4 (0.59)	13.5	259.9 (0.39)	7.5
	32	735.4	15.8	536.2 (0.73)	18.1	--	--	332.5 (0.45)	3.5
primary2	2	282.3	40.9	259.6 (0.92)	43.9	226.0 (0.80)	23.2	299.5 (1.06)	26.4
	4	802.0	29.4	617.9 (0.77)	38.4	424.2 (0.53)	33.0	401.8 (0.50)	14.9
	6	938.5	25.9	716.9 (0.76)	39.7	508.0 (0.54)	15.8	433.9 (0.46)	13.0
	8	1009.5	23.4	777.4 (0.77)	35.4	565.8 (0.56)	21.3	455.6 (0.45)	12.8
	16	1156.1	18.4	891.2 (0.77)	27.7	714.3 (0.62)	30.6	503.6 (0.44)	8.6
	32	1257.3	14.6	965.2 (0.77)	22.8	--	--	578.1 (0.46)	5.2
s838	2	26.0	5.7	23.3 (0.90)	4.0	22.2 (0.85)	6.5	30.6 (1.18)	3.9
	4	84.1	7.2	65.2 (0.78)	8.6	46.8 (0.56)	2.3	47.2 (0.56)	2.5
	6	107.1	6.8	81.8 (0.76)	8.3	61.0 (0.57)	1.7	53.9 (0.50)	2.2
	8	119.5	6.4	90.8 (0.76)	7.5	68.4 (0.57)	3.1	57.9 (0.48)	2.4
	16	145.6	5.0	107.8 (0.74)	5.4	88.6 (0.61)	3.4	68.7 (0.47)	1.0
	32	164.2	5.2	122.2 (0.74)	5.2	--	--	107.2 (0.65)	1.3
sioo	2	44.5	10.4	25.2 (0.57)	0.8	29.8 (0.67)	2.4	25.0 (0.56)	0.0
	4	94.1	8.3	63.8 (0.68)	7.0	68.8 (0.73)	3.6	25.0 (0.27)	0.0
	6	119.2	5.1	76.0 (0.64)	5.6	83.0 (0.70)	6.0	25.0 (0.21)	0.0
	8	138.1	6.5	83.0 (0.60)	5.7	92.6 (0.67)	2.5	31.0 (0.22)	0.0
	16	176.5	7.6	92.9 (0.53)	3.3	94.6 (0.54)	0.5	31.0 (0.18)	0.0
	32	200.3	6.9	96.3 (0.48)	1.7	--	--	86.0 (0.43)	0.0
struct	2	56.0	8.9	55.2 (0.99)	12.8	67.2 (1.20)	17.0	43.4 (0.78)	9.0
	4	290.3	16.7	284.9 (0.98)	23.2	130.0 (0.45)	12.2	94.7 (0.33)	8.7
	6	365.0	22.6	344.8 (0.94)	19.1	160.6 (0.44)	9.6	120.0 (0.33)	7.1
	8	436.7	31.4	371.1 (0.85)	17.4	180.0 (0.41)	11.2	140.5 (0.32)	6.7
	16	625.2	38.3	409.0 (0.65)	16.9	259.4 (0.41)	10.1	195.7 (0.31)	4.6
	32	839.0	34.4	453.1 (0.54)	25.9	--	--	270.1 (0.32)	4.4
test03	2	112.6	22.6	115.5 (1.03)	24.4	89.8 (0.80)	6.8	83.8 (0.74)	11.8
	4	333.9	20.4	288.3 (0.86)	27.2	157.4 (0.47)	9.8	160.9 (0.48)	14.9
	6	397.0	17.9	339.0 (0.85)	21.0	226.8 (0.57)	7.5	177.8 (0.45)	13.8
	8	434.8	14.3	365.2 (0.84)	19.3	250.8 (0.58)	10.6	203.4 (0.47)	9.7
	16	511.1	13.9	410.1 (0.80)	12.6	321.4 (0.63)	14.4	242.7 (0.47)	4.6
	32	567.4	13.2	441.3 (0.78)	9.0	--	--	302.8 (0.53)	4.7
test06	2	90.4	12.1	82.0 (0.91)	13.1	81.8 (0.90)	4.1	68.6 (0.76)	2.7
	4	296.5	22.7	244.8 (0.83)	22.3	151.2 (0.51)	11.1	92.8 (0.31)	4.6
	6	362.1	19.0	289.2 (0.80)	18.9	173 (0.48)	14.5	95.5 (0.26)	5.2
	8	400.3	16.5	314.8 (0.79)	15.3	191.8 (0.48)	12.4	123.5 (0.31)	4.2
	16	459.5	11.5	350.0 (0.76)	10.9	248.8 (0.54)	14.9	154.1 (0.34)	7.0
	32	498.3	9.3	378.5 (0.76)	11.6	--	--	229.0 (0.46)	6.4

of parts. The numbers in the parenthesis, are again ratio of the minimum (maximum) cut of the respective algorithm to the minimum (maximum) cut found by the SN-L1. Out of 72 test cases QMD-BHP found best minimum in 61 circuit. It fails to find best minimum mostly in bipartitioning problem. Again out 72 test cases QMD-BHP found best maximum in 64 test cases.

Another quality measurement is *Stability Ratio* of an algorithm which can be found by computing the ratio of standard deviation to the average cutsizes. These results are listed in the Table 5.7, each entries in this table corresponds the stability ratio of the algorithm, i.e.

$$StabilityRatio = \frac{Standard\ deviation}{Average}.$$

Bold numbers indicate the best values in each row and  $k$  is the number of parts.

The average execution times of the algorithms in seconds are listed in the Table 5.8.  $k$  is the number of parts and each value in the parenthesis gives the ratio of the average execution time of the respective algorithm to the execution time of SN-L1. Execution times include reading circuit, constructing the random initial partitioning (permuting the cell, and net number in QMD-BHP), and outputting the results as well as the algorithm itself. For these test cases SA runs approximately 1017 times slower than SN-L1 for  $k = 2, 4, 6, 8, 16$  (note that  $k = 32$  case is not included. This number will be considerably higher if we able to add  $k = 32$  case) and proposed algorithm QMD-BHP runs 3.63 times slower than SN-L1. The interesting thing in this table, proposed algorithm's run-time decline as the  $k$ , number of parts, become large. Best example of this case is circuit primary2. The run-time of the 4-way partitioning is less than the half of the run-time of 2-way partitioning. However, the run-time of the other algorithms increases as the  $k$  increases. In our implementation of SN algorithm, the running time of the algorithm increases in  $\mathcal{O}(k^2)$  (Numbers in the table does not increase exactly with  $k^2$  because runtime includes reading input, generating random initial partitioning, etc.).

Table 5.9 summarizes all of the previous tables for different  $k$  values. For each  $k$  value, QMD-BHP is superior in the average cutsizes. It shows 52.3% improvement over the SN-L1. However, SA has only 35.5% improvement over the SN-L1. For  $k = 2$  SN algorithm produces better results than SA algorithm. Cutsizes quality of the SA and QMD-BHP is better when number of parts is bigger than 2.

For minimum cutsizes, QMD-BHP is again winner in general. However, for



Table 5.5. Minimum cutsizes for benchmark circuits. (Bold values are the best values in each row.)

name	k	SW-L1	SW-L3	SA	QMD-BHP
balu	2	<b>27</b>	<b>27</b> (1.00)	<b>27</b> (1.00)	30 (1.11)
	4	147	90 (0.61)	<b>54</b> (0.37)	59 (0.40)
	6	184	123 (0.67)	106 (0.58)	<b>85</b> (0.46)
	8	203	157 (0.77)	140 (0.69)	<b>92</b> (0.45)
	16	234	171 (0.73)	160 (0.68)	<b>130</b> (0.56)
	32	266	183 (0.69)	--	<b>160</b> (0.60)
c1355	2	21	<b>19</b> (0.90)	22 (1.05)	29 (1.38)
	4	80	65 (0.81)	64 (0.80)	<b>46</b> (0.58)
	6	102	79 (0.77)	73 (0.72)	<b>53</b> (0.52)
	8	113	83 (0.73)	78 (0.69)	<b>55</b> (0.49)
	16	128	104 (0.81)	86 (0.67)	<b>59</b> (0.46)
	32	145	118 (0.81)	--	<b>77</b> (0.53)
c2670	2	25	<b>23</b> (0.92)	39 (1.56)	24 (0.96)
	4	110	92 (0.84)	73 (0.66)	<b>43</b> (0.39)
	6	137	115 (0.84)	80 (0.58)	<b>55</b> (0.40)
	8	154	121 (0.79)	84 (0.55)	<b>59</b> (0.38)
	16	199	139 (0.70)	92 (0.46)	<b>82</b> (0.41)
	32	233	167 (0.72)	--	<b>109</b> (0.47)
c3540	2	58	60 (1.03)	69 (1.19)	73 (1.26)
	4	217	128 (0.59)	141 (0.65)	<b>110</b> (0.51)
	6	268	201 (0.75)	176 (0.66)	<b>141</b> (0.53)
	8	288	234 (0.81)	184 (0.64)	<b>156</b> (0.54)
	16	342	268 (0.78)	225 (0.66)	<b>190</b> (0.56)
	32	384	300 (0.78)	--	<b>216</b> (0.56)
c7552	2	21	<b>21</b> (1.00)	76 (3.62)	23 (1.10)
	4	301	185 (0.61)	159 (0.53)	<b>49</b> (0.16)
	6	435	214 (0.49)	208 (0.48)	<b>91</b> (0.21)
	8	485	262 (0.54)	243 (0.50)	<b>120</b> (0.25)
	16	586	353 (0.60)	326 (0.56)	<b>192</b> (0.33)
	32	683	455 (0.67)	--	<b>250</b> (0.37)
industry1	2	20	<b>19</b> (0.95)	48 (2.40)	30 (1.50)
	4	343	181 (0.53)	153 (0.45)	<b>99</b> (0.29)
	6	467	256 (0.55)	219 (0.47)	<b>115</b> (0.25)
	8	516	318 (0.62)	278 (0.54)	<b>161</b> (0.31)
	16	624	430 (0.69)	375 (0.60)	<b>243</b> (0.39)
	32	700	492 (0.70)	--	<b>324</b> (0.46)
primary2	2	183	<b>176</b> (0.96)	182 (0.99)	244 (1.33)
	4	709	526 (0.74)	388 (0.55)	<b>364</b> (0.51)
	6	869	621 (0.71)	487 (0.56)	<b>406</b> (0.47)
	8	962	693 (0.72)	535 (0.56)	<b>423</b> (0.44)
	16	1104	816 (0.74)	681 (0.62)	<b>483</b> (0.44)
	32	1217	910 (0.75)	--	<b>563</b> (0.46)
s838	2	17	<b>16</b> (0.94)	16 (0.94)	23 (1.35)
	4	66	43 (0.65)	43 (0.65)	<b>42</b> (0.64)
	6	90	60 (0.67)	58 (0.64)	<b>50</b> (0.56)
	8	100	71 (0.71)	64 (0.64)	<b>54</b> (0.54)
	16	133	97 (0.73)	85 (0.64)	<b>66</b> (0.50)
	32	150	109 (0.73)	--	<b>104</b> (0.69)
sioo	2	25	<b>25</b> (1.00)	25 (1.00)	25 (1.00)
	4	74	43 (0.58)	64 (0.86)	<b>25</b> (0.34)
	6	104	61 (0.59)	73 (0.70)	<b>25</b> (0.24)
	8	118	67 (0.57)	88 (0.75)	<b>31</b> (0.26)
	16	162	78 (0.48)	94 (0.58)	<b>31</b> (0.19)
	32	179	93 (0.52)	--	<b>86</b> (0.48)
struct	2	42	<b>33</b> (0.79)	36 (0.86)	34 (0.81)
	4	251	234 (0.93)	121 (0.48)	<b>76</b> (0.30)
	6	266	300 (1.13)	145 (0.55)	<b>99</b> (0.37)
	8	362	328 (0.91)	163 (0.45)	<b>121</b> (0.33)
	16	556	363 (0.65)	245 (0.44)	<b>180</b> (0.32)
	32	753	395 (0.52)	--	<b>261</b> (0.35)
test03	2	62	<b>68</b> (1.10)	83 (1.34)	60 (0.97)
	4	287	208 (0.72)	142 (0.49)	<b>126</b> (0.44)
	6	337	283 (0.84)	212 (0.63)	<b>153</b> (0.45)
	8	382	303 (0.79)	239 (0.63)	<b>182</b> (0.48)
	16	475	367 (0.77)	307 (0.65)	<b>232</b> (0.49)
	32	541	415 (0.77)	--	<b>292</b> (0.54)
test06	2	66	<b>63</b> (0.95)	75 (1.14)	67 (1.02)
	4	236	177 (0.75)	137 (0.58)	<b>86</b> (0.36)
	6	314	221 (0.70)	153 (0.49)	<b>89</b> (0.28)
	8	365	282 (0.77)	170 (0.47)	<b>117</b> (0.32)
	16	431	321 (0.74)	231 (0.54)	<b>141</b> (0.33)
	32	474	352 (0.74)	--	<b>218</b> (0.46)

Table 5.6. Maximum cutsizes for benchmark circuits. (Bold values are the best values in each row.)

name	k	SW-L1	SW-L3	SA	QMD-BHP
balu	2	61	61 (1.00)	45 (0.74)	45 (0.74)
	4	200	168 (0.84)	96 (0.48)	82 (0.41)
	6	228	179 (0.79)	144 (0.63)	94 (0.41)
	8	248	185 (0.75)	151 (0.61)	112 (0.45)
	16	282	192 (0.68)	169 (0.60)	136 (0.48)
	32	292	209 (0.72)	--	160 (0.55)
c1355	2	55	53 (0.96)	42 (0.76)	29 (0.53)
	4	113	96 (0.85)	71 (0.63)	47 (0.42)
	6	126	111 (0.88)	85 (0.67)	53 (0.42)
	8	134	115 (0.86)	84 (0.63)	55 (0.41)
	16	166	128 (0.77)	92 (0.55)	59 (0.36)
	32	183	140 (0.77)	--	77 (0.42)
c2670	2	69	84 (1.22)	54 (0.78)	41 (0.59)
	4	158	137 (0.87)	80 (0.51)	47 (0.30)
	6	204	154 (0.75)	92 (0.45)	65 (0.32)
	8	206	163 (0.79)	87 (0.42)	69 (0.33)
	16	252	180 (0.71)	105 (0.42)	90 (0.36)
	32	281	205 (0.73)	--	118 (0.42)
c3540	2	141	133 (0.94)	96 (0.68)	123 (0.87)
	4	269	242 (0.90)	151 (0.56)	152 (0.57)
	6	313	272 (0.87)	197 (0.63)	167 (0.53)
	8	345	292 (0.85)	206 (0.60)	180 (0.52)
	16	395	320 (0.81)	233 (0.59)	206 (0.52)
	32	432	339 (0.78)	--	231 (0.53)
c7552	2	106	89 (0.84)	90 (0.85)	42 (0.40)
	4	435	304 (0.70)	182 (0.42)	97 (0.22)
	6	550	397 (0.72)	229 (0.42)	139 (0.25)
	8	585	426 (0.73)	273 (0.47)	166 (0.28)
	16	724	505 (0.70)	366 (0.51)	217 (0.30)
	32	757	543 (0.72)	--	277 (0.37)
industry1	2	166	139 (0.84)	98 (0.59)	77 (0.46)
	4	471	377 (0.80)	210 (0.45)	169 (0.36)
	6	562	447 (0.80)	296 (0.53)	191 (0.34)
	8	612	491 (0.80)	309 (0.50)	211 (0.34)
	16	725	536 (0.74)	416 (0.57)	288 (0.40)
	32	788	567 (0.72)	--	341 (0.43)
primary2	2	406	369 (0.91)	249 (0.61)	352 (0.87)
	4	867	700 (0.81)	468 (0.54)	434 (0.50)
	6	1009	797 (0.79)	523 (0.52)	467 (0.46)
	8	1061	856 (0.81)	596 (0.56)	487 (0.46)
	16	1211	948 (0.78)	741 (0.61)	524 (0.43)
	32	1292	1009 (0.78)	--	589 (0.46)
s838	2	50	38 (0.76)	31 (0.62)	40 (0.80)
	4	101	86 (0.85)	50 (0.50)	53 (0.52)
	6	122	101 (0.83)	63 (0.52)	58 (0.48)
	8	133	107 (0.80)	73 (0.55)	63 (0.47)
	16	160	124 (0.78)	94 (0.59)	72 (0.45)
	32	176	135 (0.77)	--	110 (0.63)
sioo	2	67	28 (0.42)	31 (0.46)	25 (0.37)
	4	111	80 (0.72)	73 (0.66)	25 (0.23)
	6	132	92 (0.70)	88 (0.67)	25 (0.19)
	8	155	93 (0.60)	95 (0.61)	31 (0.20)
	16	196	99 (0.51)	95 (0.48)	31 (0.16)
	32	214	102 (0.48)	--	86 (0.40)
struct	2	86	94 (1.09)	87 (1.01)	91 (1.06)
	4	331	330 (1.00)	154 (0.47)	117 (0.35)
	6	424	388 (0.92)	172 (0.41)	148 (0.35)
	8	525	404 (0.77)	191 (0.36)	169 (0.32)
	16	713	446 (0.63)	271 (0.38)	207 (0.29)
	32	928	508 (0.55)	--	282 (0.30)
test03	2	169	189 (1.12)	102 (0.60)	131 (0.78)
	4	384	349 (0.91)	168 (0.44)	195 (0.51)
	6	443	385 (0.87)	232 (0.52)	208 (0.47)
	8	466	404 (0.87)	270 (0.58)	226 (0.48)
	16	539	441 (0.82)	343 (0.64)	255 (0.47)
	32	598	464 (0.78)	--	313 (0.52)
test06	2	124	137 (1.10)	87 (0.70)	73 (0.59)
	4	341	299 (0.88)	170 (0.50)	103 (0.30)
	6	409	326 (0.80)	187 (0.46)	116 (0.28)
	8	450	357 (0.79)	205 (0.46)	138 (0.31)
	16	487	381 (0.78)	275 (0.56)	175 (0.36)
	32	521	433 (0.83)	--	246 (0.47)

Table 5.7. Stability Ratios (ratio of standard deviation to cutsizes) for benchmark circuits. (Bold values are the best values in each row.)

name	k	SM-L1	SM-L3	SA	QMD-BHP
balu	2	0.229	0.248	0.235	<b>0.104</b>
	4	<b>0.053</b>	0.136	0.201	0.068
	6	0.045	0.056	0.098	<b>0.023</b>
	8	0.044	0.033	<b>0.029</b>	0.030
	16	0.029	0.021	0.019	<b>0.010</b>
	32	0.021	0.026	-	<b>0.000</b>
c1355	2	0.194	0.239	0.217	<b>0.000</b>
	4	0.068	0.077	0.042	<b>0.011</b>
	6	0.038	0.066	0.054	<b>0.000</b>
	8	0.040	0.064	0.032	<b>0.000</b>
	16	0.042	0.044	0.026	<b>0.000</b>
	32	0.052	0.039	-	<b>0.000</b>
c2670	2	0.198	0.212	<b>0.118</b>	0.166
	4	0.071	0.070	0.031	<b>0.013</b>
	6	0.072	0.060	0.045	<b>0.028</b>
	8	0.065	0.059	<b>0.014</b>	0.019
	16	0.046	0.054	0.049	<b>0.023</b>
	32	0.037	0.043	-	<b>0.016</b>
c3540	2	0.193	0.205	0.126	0.098
	4	0.046	0.104	<b>0.026</b>	0.063
	6	<b>0.034</b>	0.059	0.040	0.039
	8	0.038	0.049	0.039	<b>0.032</b>
	16	0.031	0.034	<b>0.012</b>	0.017
	32	0.025	0.025	-	<b>0.012</b>
c7552	2	0.349	0.332	<b>0.055</b>	0.149
	4	0.070	0.120	<b>0.048</b>	0.147
	6	0.041	0.122	<b>0.035</b>	0.102
	8	<b>0.038</b>	0.105	0.039	0.069
	16	0.034	0.077	0.046	<b>0.027</b>
	32	0.023	0.035	-	<b>0.020</b>
industry1	2	0.471	0.421	0.236	<b>0.216</b>
	4	<b>0.068</b>	0.133	0.107	0.109
	6	<b>0.037</b>	0.105	0.096	0.098
	8	<b>0.031</b>	0.080	0.036	0.054
	16	<b>0.026</b>	0.044	0.034	0.029
	32	0.021	0.034	-	<b>0.011</b>
primary2	2	0.145	0.169	0.103	<b>0.088</b>
	4	<b>0.037</b>	0.062	0.078	<b>0.037</b>
	6	<b>0.028</b>	0.055	0.031	0.030
	8	<b>0.023</b>	0.046	0.038	0.028
	16	<b>0.016</b>	0.031	0.043	0.017
	32	0.012	0.024	-	<b>0.009</b>
s838	2	0.219	0.172	0.293	<b>0.127</b>
	4	0.086	0.132	<b>0.049</b>	0.053
	6	0.063	0.101	<b>0.028</b>	0.041
	8	0.054	0.083	0.045	<b>0.041</b>
	16	0.034	0.050	0.038	<b>0.015</b>
	32	0.032	0.043	-	<b>0.012</b>
sioo	2	0.234	0.032	0.081	<b>0.000</b>
	4	0.088	0.110	0.052	<b>0.000</b>
	6	0.043	0.074	0.072	<b>0.000</b>
	8	0.047	0.069	0.027	<b>0.000</b>
	16	0.043	0.036	0.005	<b>0.000</b>
	32	0.034	0.018	-	<b>0.000</b>
struct	2	<b>0.159</b>	0.232	0.253	0.207
	4	<b>0.058</b>	0.081	0.094	0.092
	6	0.062	<b>0.055</b>	0.060	0.059
	8	0.072	<b>0.047</b>	0.062	0.048
	16	0.061	0.041	0.039	<b>0.024</b>
	32	0.041	0.057	-	<b>0.016</b>
test03	2	0.201	0.211	<b>0.076</b>	0.141
	4	<b>0.061</b>	0.094	0.062	0.093
	6	0.045	0.062	<b>0.033</b>	0.078
	8	<b>0.033</b>	0.053	0.042	0.048
	16	0.027	0.031	0.045	<b>0.019</b>
	32	0.023	0.020	-	<b>0.016</b>
test06	2	0.134	0.160	0.050	<b>0.039</b>
	4	0.077	0.091	0.073	<b>0.050</b>
	6	<b>0.052</b>	0.065	0.084	0.054
	8	0.041	0.049	0.065	<b>0.034</b>
	16	<b>0.025</b>	0.031	0.060	0.045
	32	<b>0.019</b>	0.031	-	0.028

Table 5.8. Execution times for benchmark circuits (in seconds). (Bold values are the best values in each row.)

name	k	SM-L1	SM-L3	SA	QMD-BHP
balu	2	<b>0.63</b>	0.80 (1.27)	61.22 (97.17)	5.28 (8.38)
	4	<b>0.86</b>	1.51 (1.76)	587.58 (683.23)	4.83 (5.62)
	6	<b>1.47</b>	2.52 (1.71)	1200.98 (816.99)	4.48 (3.05)
	8	<b>1.62</b>	3.66 (2.26)	1584.56 (978.12)	4.35 (2.69)
	16	3.75	12.19 (3.25)	4579.68 (1221.25)	<b>3.34</b> (0.89)
	32	9.77	51.06 (5.23)	--	<b>3.06</b> (0.31)
c1355	2	<b>0.49</b>	0.60 (1.22)	161.12 (328.82)	0.60 (1.22)
	4	<b>0.63</b>	0.93 (1.48)	565.60 (897.78)	<b>0.52</b> (0.83)
	6	<b>0.86</b>	1.57 (1.83)	1058.70 (1231.05)	<b>0.50</b> (0.58)
	8	1.15	2.38 (2.07)	1395.08 (1213.11)	<b>0.50</b> (0.43)
	16	3.50	7.72 (2.21)	3016.04 (861.73)	<b>0.48</b> (0.14)
	32	9.59	26.69 (2.78)	--	<b>0.48</b> (0.05)
c2670	2	<b>0.74</b>	0.80 (1.08)	240.12 (324.49)	1.40 (1.89)
	4	<b>0.90</b>	1.15 (1.28)	1046.72 (1163.02)	1.30 (1.44)
	6	1.49	2.14 (1.44)	1873.30 (1257.25)	<b>1.26</b> (0.85)
	8	2.14	3.66 (1.71)	3049.84 (1425.16)	<b>1.26</b> (0.59)
	16	5.91	11.43 (1.93)	6135.12 (1038.09)	<b>1.15</b> (0.19)
	32	16.33	43.76 (2.68)	--	<b>1.13</b> (0.07)
c3540	2	<b>1.13</b>	1.14 (1.01)	584.76 (517.49)	6.66 (5.89)
	4	<b>1.53</b>	2.23 (1.46)	1507.28 (985.15)	4.69 (3.07)
	6	<b>1.95</b>	3.46 (1.77)	2491.46 (1277.67)	3.73 (1.91)
	8	<b>2.85</b>	5.42 (1.90)	3497.48 (1227.19)	3.23 (1.13)
	16	7.11	15.81 (2.22)	8413.82 (1183.38)	<b>2.43</b> (0.34)
	32	21.41	61.91 (2.89)	--	<b>2.08</b> (0.10)
c7552	2	<b>3.11</b>	3.36 (1.08)	565.02 (181.68)	8.68 (2.79)
	4	<b>5.10</b>	5.39 (1.06)	3742.70 (733.86)	8.22 (1.61)
	6	<b>7.02</b>	9.33 (1.33)	10692.46 (1523.14)	7.68 (1.09)
	8	9.20	13.09 (1.42)	17971.80 (1953.46)	<b>7.14</b> (0.78)
	16	22.75	42.18 (1.85)	46001.22 (2022.03)	<b>6.32</b> (0.28)
	32	59.51	145.53 (2.45)	--	<b>5.61</b> (0.09)
industry1	2	<b>3.30</b>	3.48 (1.05)	526.50 (159.55)	38.39 (11.63)
	4	<b>4.59</b>	6.14 (1.34)	4932.18 (1074.55)	32.89 (7.17)
	6	<b>6.35</b>	9.83 (1.55)	11504.72 (1811.77)	31.38 (4.94)
	8	<b>8.51</b>	15.43 (1.81)	19825.56 (2329.68)	30.39 (3.57)
	16	<b>22.62</b>	44.39 (1.96)	55720.86 (2463.34)	26.58 (1.18)
	32	58.28	180.58 (3.10)	--	<b>25.18</b> (0.43)
primary2	2	<b>5.68</b>	5.52 (0.97)	913.96 (160.91)	234.88 (41.35)
	4	<b>7.66</b>	9.40 (1.23)	3565.84 (465.51)	105.54 (13.78)
	6	<b>9.86</b>	14.34 (1.45)	18406.74 (1866.81)	70.79 (7.18)
	8	<b>13.76</b>	21.91 (1.59)	37304.18 (2711.06)	56.47 (4.10)
	16	<b>34.36</b>	64.89 (1.89)	99078.47 (2883.54)	37.10 (1.08)
	32	76.68	233.49 (3.04)	--	<b>27.11</b> (0.35)
s838	2	<b>0.37</b>	0.41 (1.11)	87.58 (236.70)	0.63 (1.70)
	4	0.57	0.80 (1.40)	352.32 (618.11)	<b>0.55</b> (0.96)
	6	0.78	1.33 (1.71)	604.66 (775.21)	<b>0.54</b> (0.69)
	8	1.03	2.07 (2.01)	830.38 (806.19)	<b>0.50</b> (0.49)
	16	2.71	6.73 (2.48)	1949.48 (719.37)	<b>0.49</b> (0.18)
	32	8.01	25.57 (3.19)	--	<b>0.51</b> (0.06)
sioo	2	<b>0.43</b>	0.43 (1.00)	42.70 (99.30)	2.59 (6.02)
	4	<b>0.68</b>	0.74 (1.09)	487.22 (716.50)	2.56 (3.76)
	6	<b>0.94</b>	1.10 (1.17)	806.24 (857.70)	2.56 (2.72)
	8	<b>1.30</b>	1.59 (1.22)	1113.86 (856.82)	2.59 (1.99)
	16	3.54	4.85 (1.37)	2470.66 (697.93)	<b>2.63</b> (0.74)
	32	10.26	19.57 (1.91)	--	<b>1.53</b> (0.15)
struct	2	<b>1.88</b>	2.33 (1.24)	155.28 (82.60)	7.02 (3.73)
	4	<b>2.70</b>	3.15 (1.17)	415.06 (153.73)	5.95 (2.20)
	6	<b>4.38</b>	5.36 (1.22)	697.88 (159.33)	5.53 (1.26)
	8	5.41	7.46 (1.38)	950.30 (175.66)	<b>5.31</b> (0.98)
	16	16.19	24.25 (1.50)	2108.76 (130.25)	<b>4.43</b> (0.27)
	32	64.36	90.38 (1.40)	--	<b>4.02</b> (0.06)
test03	2	<b>2.28</b>	2.40 (1.05)	864.48 (379.16)	36.39 (15.96)
	4	<b>3.38</b>	4.09 (1.21)	2947.66 (872.09)	26.23 (7.76)
	6	<b>4.50</b>	6.09 (1.35)	6693.38 (1487.42)	21.72 (4.83)
	8	<b>7.16</b>	9.87 (1.38)	11201.86 (1564.51)	18.70 (2.61)
	16	16.56	30.17 (1.82)	21761.30 (1314.09)	<b>15.89</b> (0.96)
	32	45.91	108.67 (2.37)	--	<b>13.25</b> (0.29)
test06	2	<b>2.36</b>	2.45 (1.04)	145.24 (61.54)	47.70 (20.21)
	4	<b>3.19</b>	3.74 (1.17)	3237.82 (1014.99)	44.20 (13.86)
	6	<b>4.16</b>	5.66 (1.36)	8134.66 (1955.45)	44.30 (10.65)
	8	<b>5.73</b>	8.26 (1.44)	13701.64 (2391.21)	42.77 (7.46)
	16	12.99	23.51 (1.81)	23867.50 (1837.37)	37.56 (2.89)
	32	35.56	91.20 (2.56)	--	<b>32.15</b> (0.90)

bipartitioning problem SN-L3 finds the best results. QMD-BHP results for bipartitioning problem are 14.8% worse than SN-L1. In general, SN-L3 produces 24.7% more favorable results than SN-L1 and QMD-BHP has 45.6% better results than SN-L1.

Timing results of the algorithms are normalized according to the SN-L1, for different  $k$  values. For example, SN-L3 runs 1.09 times slower than SN-L1 for  $k = 2$  and 2.80 times slower for  $k = 32$ . QMD-BHP is 10.07 times slower than SN-L1 for  $k = 2$  but only 0.24 times faster for the  $k = 32$ . Run-time reduction of the QMD-BHP can be easily seen from this table. Run-time of SA increases from 219.12 times to 11364.36 times when  $k$  increases from 2 to 16. Fastest program for  $k = 2, 4, 6, 8$  is SN-L1. However, for  $k = 16, 32$  QMD-BHP is the fastest algorithm. It is expected that for much bigger  $k$  values QMD-BHP will be always fastest algorithm because of its nature.

Table 5.10 summarizes percentage improvement of the QMD-BHP algorithm with respect to compared algorithms for the each  $k$  value. QMD-BHP overperforms the all compared algorithms for each value of  $k$ . However, improvement percentage is higher for  $k > 2$ . QMD-BHP performs 16.31% better than SN-L1 for bipartitioning but 52.47% better for 32-way partitioning. Again QMD-BHP results 10.3% better cutsizes than SA for  $k = 2$  and 30.4% for 16-way partition.

Table 5.9. Average (Avg) results for performances of algorithms. Averages were taken over all our test instances. ( $k$  is the number of parts. Bold values are the best values in each row.)

$k$	SN-L1	SN-L3	SA	QMD-BHP
Avg Improvements (%) in Avg Cutsizes wrt SN-L1				
2	0.00	2.87	-0.22	<b>16.31</b>
4	0.00	20.86	46.23	<b>61.20</b>
6	0.00	21.95	44.14	<b>62.26</b>
8	0.00	23.53	44.31	<b>61.41</b>
16	0.00	27.25	43.10	<b>60.30</b>
32	0.00	29.22	-	<b>52.47</b>
Avg Improvements (%) in Minimum Cutsizes wrt SN-L1				
2	0.00	3.76	-42.37	-14.87
4	0.00	30.21	41.03	<b>59.00</b>
6	0.00	27.40	41.27	<b>60.52</b>
8	0.00	27.23	40.94	<b>60.02</b>
16	0.00	29.69	40.90	<b>58.62</b>
32	0.00	30.02	-	<b>50.21</b>
Avg Ratio (%) of Standard Deviation to Avg Cutsizes				
2	22.72	21.93	15.35	<b>11.14</b>
4	6.52	10.09	7.20	<b>6.12</b>
6	4.67	7.35	5.64	<b>4.60</b>
8	4.38	6.11	3.90	<b>3.37</b>
16	3.46	4.12	3.48	<b>1.88</b>
32	2.84	3.28	-	<b>1.16</b>
Avg Ratio of Avg Running Times to those of SN-L1				
2	1.00	1.09	219.12	10.07
4	1.00	1.30	781.54	5.17
6	1.00	1.49	1251.65	3.31
8	1.00	1.68	1469.35	2.24
16	1.00	2.03	1364.36	<b>0.76</b>
32	1.00	2.80	-	<b>0.24</b>

Table 5.10. Average percentage improvements of the proposed QMD-BHP algorithm with respect to SN and SA algorithms for different number of parts. Averages were taken over all our test instances.  $k$  is the number of parts.

$k$	SN-L1	SN-L3	SA
2	16.3	11.9	10.3
4	61.2	51.2	26.8
6	62.3	51.9	32.2
8	61.4	49.9	30.4
16	60.3	46.1	30.4
32	52.5	32.3	

## Chapter 6

### Conclusion

In this work, we have proposed a new coarse-grained constructive multiple-way hypergraph partitioning algorithm. We have transformed the mincut problem to maximal internal net problem, by dualizing the input hypergraph to graph. Hence, instead of minimizing the external nets in the hypergraph, we try to maximize internal net by selecting appropriate nets in the dual graph. The selection scheme based on the well-known ordering heuristic from direct solution of sparse linear systems.

Efficient implementation of the proposed algorithm have been done using the *quotient graphs*. Hence, the space complexity of the basic minimum degree algorithm is unknown till the end of the algorithm, we have used the quotient graph which has known space complexity. In fact, using this model the net selection can be done in-place.

An interesting property of the proposed algorithm is the fact that it is a  $k$ -way partitioning algorithm. However, its run-time is not proportional with the  $k$  where it is the case in most of the previous algorithms. For example, in the direct  $k$ -way partitioning Kernighan-Lin algorithm, each pass contains  $k \cdot \log k$  as multiplier. Hence, the running time increases as the number of parts  $k$  increases. Proposed algorithm shows the significant reduction in run-time when  $k$  increases.

Two versions of the proposed algorithm have been implemented; QMD-HP and QMD-BHP. QMD-HP produces good partitioning in terms of the cutsize, however in some cases it may produce unbalanced partitions. We have added balancing post process based on the a number partitioning heuristic in the

second version QMD-BHP. The imbalance ratio between the parts can be given as a parameter to this process.

The performance of the proposed algorithm QMD-BHP is evaluated in comparison with two well-known heuristics; Simulated Annealing (SA) and Sanchis's Method (SN) for 12 benchmark circuits. Since the number of levels in the Sanchis's Method is a parameter, we have run the algorithm with two different parameters; 1 and 3. Level 1 can be considered direct multiple-way Fiduccia-Mattheyses algorithm. Level 3 is chosen since it is an appropriate value for our test cases, hence in all test circuits average net degree is about 3.

Test results show that proposed algorithm over-performs the compared algorithms. On the average, the cutsizes results of QMD-BHP is 26.0% better than the SA algorithm, and it is 52.3% better than the SN-L1 algorithm. For bigger  $k$  values, the quality of the cutsize found by QMD-BHP is much better than the bipartitioning cutsizes, i.e. the results of 2-way partitioning problems are only 10.3% better than SA, however in 16-way partitioning results are 30.4% better than SA.

Timing results show that the slowest algorithm is SA. For the  $k$  value varying from 2 to 16, it is approximately 1017 times slower than SN-L1. For these test circuits, QMD-BHP is approximately 3.63 times slower than SN-L1. However, it should be noticed that, running time of QMD-BHP reduces when the number of part  $k$  increases. It is 10.7 times slower than SN-L1 for 2-way partitioning, but it is 0.24 times faster when the  $k = 32$ . Our algorithm is a bottom-up constructive algorithm, and due to nature of the algorithm it terminates early for large  $k$  values, therefore, this is not a surprising result.

For the future work, algorithm can be modified to reduce its run-time by changing the calculation of the *valence* value, i.e., instead of recalculating the valence value of the each node in the reach set of the selected node, an update mechanism can be found. *Incomplete degree update* mechanism and *uneliminated supernode* concept can also be adopted to speed-up the algorithm. Proposed algorithm can also be used in the placement problem in VLSI, and in the mapping problem for parallel programming.



## Bibliography

- [1] T. N. Bui, C. Heigham, C. Jones, and F. T. Leighton. Improving the performance of the Kernighan-Lin and simulated annealing graph bisection algorithms. In *Proceedings of the 26th ACM/IEEE Design Automation Conference*, pages 775–778, 1989.
- [2] C.-K. Cheng and Y.-C. Wei. An improved two-way partitioning algorithm with stable performance. *IEEE Transactions on Computer-Aided Design*, 10(12):1502–1511, December 1991.
- [3] J. Cong and M'L. Smith. A parallel bottom-up clustering algorithm with applications to circuit partitioning in vlsi design. In *Proceedings of the 30th ACM/IEEE Design Automation Conference*, pages 755–760, 1993.
- [4] C.M. Fiduccia and R.M. Mattheyses. A linear-time heuristic for improving network partitions. In *Proceedings of the 19th ACM/IEEE Design Automation Conference*, pages 175–181, 1982.
- [5] M. R. Garey and D. S. Johnson. *Computers and Intractability*. W.H. Freeman and Co., New York, New York, 1979.
- [6] J. A. George and J. W. H. Liu. *Computer solution of large sparse positive definite systems*. Prentice-Hall, 1981.
- [7] S. W. Hadley, B. L. Mark, and A. Vanelli. An efficient eigenvector approach for finding netlist partitions. *IEEE Transactions on Computer-Aided Design*, 11(7):885–892, July 1992.
- [8] L. Hagen and A. B. Kahng. New spectral methods for ratio cut partitioning and clustering. *IEEE Transactions on Computer-Aided Design*, 11(9):1074–1085, Sep 1992.

- [9] D. S. Johnson, C. R. Aragon, L. A. McGeoch, and C. Schevon. Optimization by simulated annealing: An experimental evaluation; part I, graph partitioning. *Operations Research*, 37(6):865–892, Nov 1989.
- [10] D. S. Johnson, C. R. Aragon, L. A. McGeoch, and C. Schevon. Optimization by simulated annealing: An experimental evaluation; part II, graph coloring and number partitioning. *Operations Research*, 39(3):378–406, May 1991.
- [11] A. B. Kahng. Fast hypergraph partition. In *Proc. 26th Design Automation Conference*, pages 762–766, 1989.
- [12] Y. Kamidoi, S. Wakabayashi, J. Miyao, and N. Yoshida. A fast heuristic algorithm for hypergraph bisection. Technical report, Faculty of Engineering, Hiroshima University, 1991.
- [13] B.W. Kernighan and S. Lin. An efficient heuristic procedure for partitioning graphs. Technical Report 2, The Bell System Technical Journal, Feb 1970.
- [14] R. M. King and P. Banerjee. Esp : Placement by simulated evolution. *IEEE Transactions on Computer-Aided Design*, 8(3):245–256, March 1989.
- [15] S. Kirkpatrick, C. D. Gelatt Jr., and M. P. Vecchi. Optimization by simulated annealing. *Science*, 220(4598):671–680, May 1983.
- [16] S. Kirkpatrick, C. D. Gelatt Jr., and M. P. Vecchi. Optimization by simulated annealing. *Science*, 220(4598):671–680, May 1983.
- [17] B. Krishnamurthy. An improved min-cut algorithm for partitioning VLSI networks. *IEEE Transactions on Computers*, 33(5):438–446, May 1984.
- [18] J. W. H. Liu. A graph partitioning algorithm by node separator. *ACM Transactions on Mathematical Software*, 15(3):198–219, Sep 1989.
- [19] J. W. H. Liu. The role of elimination trees in sparse factorization. *SIAM J. Matrix Anal. App.*, 11(1):134–172, Jan 1990.
- [20] H. M. Markowitz. The elimination form of the inverse and its application to linear programming. *Management Sci.*, 3:255–269, 1957.

- [21] D. J. Rose. *Graph Theory and Computing*, chapter A graph-theoretic study of the numerical solution of sparse positive definite systems of linear equations, pages 183–217. Academic Press, 1972.
- [22] L. A. Sanchis. Multiple-way network partitioning. *IEEE Transactions on Computers*, 38(1):62–81, Jan 1989.
- [23] D. G. Schweikert and B. W. Kernighan. A proper model for the partitioning of electrical circuits. In *Proceedings of the 9th ACM/IEEE Design Automation Conference*, pages 57–62, 1972.
- [24] H. Shin and C. Kim. A simple yet effective technique for partitioning. *IEEE Transactions on VLSI Systems*, 1(3):380–386, Sep 1993.
- [25] W. F. Tinney and J. W. Walker. Direct solution of sparse network equations by optimally ordered triangular factorization. In *Proc. IEEE*, volume 55, pages 1801–1809, 1967.
- [26] Y.-C. Wei and C.-K. Cheng. Ratio cut partitioning for hierarchical designs. *IEEE Transactions on Computer-Aided Design*, 10(7):911–921, July 1991.
- [27] M. Yannakis. Computing the minimum fill-in is np-complete. *SIAM J. Algebraic Discrete Methods*, 2:77–79, 1981.