Optimum Logic Encoding and Layout Wiring for VLSI Design: A Graph-Theoretic Approach

by

Chuan-Jin Shi

A thesis
presented to the University of Waterloo
in fulfilment of the
thesis requirement for the degree of
Doctor of Philosophy
in
Computer Science

Waterloo, Ontario, Canada, 1993

Abstract

This thesis addresses two problems in VLSI design: constrained via minimization—which aims at minimizing the number of vias between routing layers— and constrained logic encoding—a problem fundamental to the design of synchronous, and hazard-free asynchronous, circuits. We show that these two problems have the same combinatorial structure, which can be captured by a new graph-theoretic model, called signed hypergraph. They can be formulated as two new optimization problems, namely maximum balance and minimum covering, related to a balance property of signed hypergraphs.

On the theoretical side, we establish a structural characterization of balanced signed hypergraphs. We then prove that both maximum balance and minimum covering are NP-complete. We present an integer linear programming formulation for maximum balance of signed hypergraphs, and a polynomial-size linear programming formulation for the case of planar signed graphs. We show that maximum balance in a planar signed hypergraph reduces to the minimum hypergraph T-join in its planar dual. We address the problem of modeling signed hypergraphs by real-weighted hypergraphs or graphs. We settle a conjecture of Lengauer which states that a clique is a best approximate model for a hyperedge, even if dummy vertices are allowed. We present a local search algorithm for the maximum balance problem, with one pass running in linear time. We describe a simple greedy peeling heuristic for minimum covering. We prove that greedy peeling has a guaranteed performance bound for solving a class of VLSI optimization problems of the so-called cluster-cover structure.

On the practical side, our work on constrained via minimization breaks new ground for the case of k-way splits ($k \leq 3$) with a compact reduction to graph T-joins and a polynomial-size linear programming formulation. For the case of multi-way splits ($k > 3$), it provides a direct and efficient local search for timing-driven layer assignment and an optimal modeling scheme for good approximation algorithms. For logic synthesis, we present a unified approach to optimum state assignment for synchronous and hazard-free asynchronous circuit design. We have implemented our results as two experimental CAD tools. As demonstrated on a set of industry benchmarks, our tools outperform existing tools in terms of both solution quality and CPU time.

Acknowledgments

First, I would like to thank my supervisor, Prof. John Brzozowski. Without his many constructive comments, his high standards of clarity and rigor, his patient guidance and encouragement, this thesis would not have been possible. I am also deeply grateful to him for many other things beyond the thesis. I would like to quote the following words by Carl-Johan Seger: *I hope I will be able to use this experience of how supervision* **should** *be done the day I will supervise my own students.*

I would like to thank the other members of my examining committee, Professors Ernest Kuh, Anna Lubiw, Ian Munro, and Tony Vannelli. Prof. Kuh stimulated my initial interest in performance-driven layout techniques, when I was a graduate student at Fudan University working on timing analysis. Prof. Vannelli's recognition of my initial research effort in this subject and many stimulating discussions were important to the evolution of this study into its current shape. Professors Lubiw and Munro provided many instructive discussions regarding the graph-theoretical and combinatorial-optimization aspects of this research.

I thank members of the MAVERIC group, in particular Prof. Jo Ebergen, Igor Benko, Peter Mayo, and Radu Negulescu, for valuable criticism on my initial ideas during our group meetings. I am especially grateful to Prof. Ebergen who taught me both the theory of computation and the science of programming.

I thank Dr. F. Barahona and Prof. E. Tardos for their help, via email, on graph T-joins, and Dr. M. Lomonosov for his advice on bipartite graphs. I am grateful to Krishna Padayachee for many helpful discussions on graph T-joins, and to Prof. W. Cunningham, Prof. U.S.R. Murty, and Dr. S. Gao for discussions on signed hypergraphs.

I thank Professors Jiri Vlach and Jim Barby for their helpful support during my stay at Waterloo. I wish to thank Prof. René David, whose influence was one of the factors which helped me choose the University of Waterloo for my PhD study.

I would like to reserve my deepest thanks to my parents and my family, especially to my wife, Tracey Luo: her constructive comments led to the results in Chapter 3.

Finally, I would like to acknowledge the financial support from the Information Technology Research Centre of Ontario through a fellowship and a grant, the Province of Ontario through an Ontario Graduate Scholarship, and the Natural Sciences and Engineering Research Council of Canada through a research assistantship under grant OGP0000871.

To my parents,

You-Nian Shi and Lian-Fang Zhang

Table of Contents

List of Tables

List of Figures

.

List of Abbreviations

- ASIC: Application-Specific Integrated Circuits
- BIST: Built-In Self-Test
- CAD: Computer-Aided Design
- CUT: Circuit Under Test
- FPGA: Field-Programmable Gate Array
- FSM: Finite State Machine
- MCNC: Microelectronics Center of North Carolina
- NP: Nondeterministic Polynomial
- PCB: Printed Circuit Board
- PLA: Programmable Logic Array
- PLS: Polynomial-time Local Search
- RC: Resistor and Capacitor
- VLSI: Very Large Scale Integrated Circuits

Chapter 1

Introduction

In this chapter we describe the motivations for this work, summarize our results, and outline the organization of this thesis.

1.1 Two VLSI Synthesis Problems

Our research has been motivated by the following two problems in very large scale integrated (VLSI) circuit design.

The Optimal Layer Assignment Problem. Given a collection of nets after physical placement and routing, each net consisting of wire segments that electrically connect a set of terminals, find an assignment of wire segments to two layers so that the number of vias is minimized.

The Constrained Encoding Problem. Given a set $S = \{s_1, ..., s_m\}$ of m *states*, find an encoding $\boldsymbol{\alpha}$ of S into a set $\{\boldsymbol{\alpha}(s_1), ..., \boldsymbol{\alpha}(s_m)\}$ of m binary k-tuples (k-bit vectors), in such a way that all the constraints (defined below) are satisfied and k is minimized. A *constraint*, also known as a *(partial) dichotomy*, requires that a subset P of S be distinguished from a disjoint subset Q of S by at least one bit, i.e., that bit must have the value 0 for all the states in P and 1 for all the states in Q, or vice versa.

The optimal layer assignment problem was first addressed by Hashimoto and Stevens in the design of printed circuit boards (PCB) [60]. Considerable progress has been made on this problem in the past two decades. On the practical side, owing to the importance of via minimization in improving circuit yield and reliability, researchers in computer-aided design (CAD) have developed a number of heuristics, for channel routing [60], knock-knee routing [157], and general gridless routing [110]. On the theoretical side, it has been established that a special case of layer assignment is an application of planar max cut and matching theory [6, 23, 87, 117, 141]. Although the complexity issue of the general case has perplexed computer scientists for quite a while, the problem was finally proved to be NP-complete [24, 100, 111]. Nevertheless, a precise mathematical model and mathematical tools were still lacking for the general problem. As a consequence, the complexity analysis was rather involved, the problem was understood in a rather ad hoc fashion, and the heuristics were not robust. CAD tools on via minimization are not yet common in the marketplace.

The constrained encoding problem was first studied by Tracey in 1966 [146] and was once considered well solved [148]. The problem arises in many contexts of the synthesis of asynchronous finite state machines (FSMs), as pointed out by Tracey and Unger in the 1960s, and by a number of other researchers later on [1, 143]. However, because the commercial activities in the design of sequential circuits are dominated by the synchronous (clock-based) design style, constrained encoding has not been considered a real problem for CAD engineers. This scenario has changed very recently for two reasons. The first is a discovery that the optimum synthesis of synchronous finite state machines is related to constrained encoding [123, 158]. The second is a remarkable revival of interest in the asynchronous design style [18, 25]. The methods developed in the 1960s—when tens of variables were considered large—are no longer practical for problems arising from VLSI design.

The development of concise mathematical models, the study of their basic structures, the search for efficient solution methods that can handle large problems and additional requirements, and the development of practical CAD tools, for these two VLSI synthesis problems, are the scope of this thesis.

1.2 A Graph-Theoretic Framework

A central concept in this thesis is a notion of *signed hypergraph.* Like a graph, a hypergraph consists of vertices and edges; in a hypergraph, however, each edge may be incident with more than two vertices. A signed hypergraph is a hypergraph in which each edge-vertex incidence is assigned a polarity ($+$ or $-$). Thus we may talk about the set of vertices positively (negatively) incident with an edge, and about the set of edges positively (negatively) incident with a vertex.

An edge e of a signed hypergraph is said to be *balanced* by a bipartition if all the vertices positively incident with e are in one block of the bipartition, and all the vertices negatively incident with e are in the other block. We define the following two optimization problems.

1. The *maximum balance* problem is to find a bipartition such that the number of balanced edges is maximized.
2. The *minimum covering* problem is to find a minimal number of bipartitions such that each edge is balanced by at least one bipartition.

Two fundamental observations, as will be established later in this thesis, are that

- the optimal layer assignment problem can be exactly and easily formulated as the maximum balance problem in a "planar" signed hypergraph, and

- the constrained encoding problem turns out to be the minimum covering problem.

Therefore, the notion of signed hypergraph provides an abstract and unified graph-theoretic framework for studying the two VLSI synthesis problems.

The notion of signed hypergraph turns out to be a new and unexplored concept in graph theory. Therefore, a main focus of this thesis is to study certain theoretical properties of signed hypergraphs, and to investigate how to solve the maximum balance problem and the minimum covering problem effectively and efficiently.

1.3 Results of this Thesis

The nature of this research is to bridge the gap between theory and practice. Some theoretical aspects of signed hypergraphs will be investigated, in order to reveal certain combinatorial structure that can lead to efficient and reliable algorithms for solving real VLSI synthesis problems. However, no attempt will be made to study theoretical properties that appear to have no immediate applications. On the other hand, the two practical problems will be treated completely and precisely, and the algorithms developed will be implemented and evaluated with respect to real industrial benchmarks. Physical details and system-level issues in VLSI synthesis are outside the scope of this thesis.

The results in this thesis can be interpreted both from the theoretical side in terms of signed hypergraphs, and from the practical side in terms of VLSI synthesis. On the theoretical side, we obtain the following results:

1. We give a structural characterization of the edge balance property. We show that there exists a bipartition that balances all the edges if and only if a signed hypergraph is free of negative cycles. (A negative cycle is a cycle that involves an odd number of negative edge-vertex incidences.)

2. We prove that both the maximum balance problem and the minimum covering problem are NP-complete. We also give a clear boundary that separates out the cases solvable in polynomial time.
3. We pay special attention to the maximum balance problem in a planar signed hypergraph, which corresponds to a formulation of the optimal layer assignment problem. For the case of planar signed graphs, we reduce maximum balance to minimum graph T-join, which is solved in polynomial time. We also show how to reduce maximum balance in a planar signed hypergraph to the so-called hypergraph T-join, based on which we develop a pseudo-polynomial-time algorithm.
4. Given a signed hypergraph, we define the cost of a bipartition, or cut, to be the total weight associated with all unbalanced edges. We study the following question: Given a signed hypergraph, does there exist a weighted hypergraph, or a weighted signed graph, or a weighted graph that has the same set of vertices and the same cut property? By "the same cut property", we mean that the same cut in the two graphs has the same cost. We show that there is the following "cut hierarchy" of graphs:

$$\begin{aligned}
& \text{positive-weighted signed hypergraph} \\
\equiv\ & \text{critical signed hypergraph} \\
\equiv\ & \text{real-weighted hypergraph} \\
\supset\ & \text{positive-weighted hypergraph} \\
\supset\ & \text{real-weighted signed graph} \\
\equiv\ & \text{positive-weighted signed graph} \\
\equiv\ & \text{real-weighted graph} \\
\supset\ & \text{positive-weighted graph.}
\end{aligned}$$

   Here, critical signed hypergraphs are a class of signed hypergraphs as defined in Chapter 4.2. By $\equiv$, we mean that the graph in either side can be converted to the graph in the other side with the same cut property. By $\supset$, we mean that the graph in the left side can be converted to the graph in the right side with the same cut property, but the reverse is not true.

Furthermore, we generalize the notion of cut-equivalence to min-cut equivalence. This is motivated by adding "dummy" vertices. We define the cost of a cut to be the minimal cost among a set of cuts that differ only in dummy vertices. We show the existence of the following "min-cut hierarchy":

$$\begin{aligned} & \text{positive-weighted signed hypergraph} \\ \equiv\ & \text{critical signed hypergraph} \\ \equiv\ & \text{real-weighted hypergraph} \\ \supseteq\ & \text{real-weighted signed graph} \\ \equiv\ & \text{positive-weighted signed graph} \\ \equiv\ & \text{real-weighted graph} \\ \supseteq\ & \text{positive-weighted hypergraph} \\ \supset\ & \text{positive-weighted graph.} \end{aligned}$$

Here, for $\supseteq$, we are able to show a transformation of the graph on the left side to the graph on the right side with the same cut property, but we do not know whether the reverse is true or not.

5. We show that, for an arbitrary hypergraph, there exists no positive-weighted graph that has the same min-cut property. This reveals the inherent difficulty of formulation of the optimum layer assignment problem by existing graph concepts. We settle a conjecture of Lengauer [91] on hypergraph modeling: there is no min-cut model that can have a smaller approximation error than that of a complete graph. By using mathematical programming, we derive a planar min-cut model for hyperedges with degree no greater than 8. This leads to good approximation algorithms for maximum balance of a planar signed hypergraph.
6. We present a local search heuristic to solve the maximum balance problem in a general signed hypergraph. The heuristic is a generalization of the work of Fiduccia and Matheyses on netlist partitioning, with one pass running in linear

time.

7. We describe a simple heuristic, called greedy peeling, for solving the minimum covering problem. We prove that the greedy peeling heuristic is most likely a best-possible approximation algorithm. Furthermore, we show that greedy peeling yields guaranteed performance bounds for partial constrained encoding—a variation of the minimum covering problem.
8. We present a mathematical programming formulation of the maximum balance problem. In the planar case, we derive a polynomial-size linear programming formulation. We propose a novel technique that reduces the number of inequalities significantly.

Our ultimate goal is to search for efficient and reliable algorithms for solving the optimal layer assignment problem and the constrained encoding problem. To this end, the theory and heuristics developed in this thesis have been implemented and evaluated on industrial benchmarks. On the practical side, we have obtained the following results:

9. It is the first time that the optimal layer assignment problem for two-layer routing is studied in a precise mathematical framework. This leads to a simple proof of its NP-completeness, and a novel reduction to planar T-joins; the reduction reveals the more essential structure of the optimal layer assignment problem.
   Using the model of signed hypergraph, we have developed a program, called *POLAR2*, for *P*erformance-driven *O*ptimal *L*ayer *A*ssignment of two-layer *R*outings. *POLAR2* accepts a signed hypergraph description of a two-layer routing. It attempts to find an optimal solution, by using either an *exact* method based on mathematical programming, or a *fast* heuristic method based on local search. *POLAR2* has been tested on several practical routing examples, and has demonstrated its superior performance in searching for optimum solutions and in handling various practical constraints.

10. It is the first time that a graph-theoretic framework has been established for studying the constrained encoding problem. A package, called *ENCORE*, has been developed for sequential logic synthesis. The core of *ENCORE* is the greedy peeling heuristic for minimum covering. *ENCORE* has been applied to a variety of practical problem instances, including a number of examples in the literature, and industrial benchmarks from the Microelectronics Center of North Carolina (MCNC). *ENCORE*'s performance is considerably better than that of other tools.

1.4 Organization of the Thesis

The thesis is intended to be useful to CAD engineers who develop computer programs for VLSI layout synthesis and VLSI logic synthesis, and to computer scientists and mathematicians who are interested in pursing further some theoretic issues arising from this study. To achieve this dual goal, we present theoretical results in a formal and abstract way enriched by intuitive examples. We describe practical applications, omitting physical details but focusing on pertinent information that defines the problems.

Our research involves graph theory and combinatorial optimization [12, 13, 51], mathematical programming and polyhedral combinatorics [64, 88, 98, 119], algorithm design and NP-complexity theory [3, 48], VLSI layout design [91, 155], and VLSI sequential logic synthesis [17, 149]. However, for the most part, the thesis is self-contained.

This thesis is organized as follows. In the first six chapters, we concentrate on the development of concepts and tools within the framework of signed hypergraphs. In the last two chapters, we return to the two practical problems, and show how to formulate and to solve them in the framework established. More specifically, in Chapter 2, we formally introduce the notion of signed hypergraph and then focus on certain theoretical aspects of signed hypergraphs, including the structural theorem, the duality theorem, and the proof of NP-completeness of the maximum balance and minimum covering problems. In Chapter 3, we study the maximum balance problem in planar signed hypergraphs. Chapter 4 explores the cut hierarchy of signed hypergraphs and settles the problem of modeling signed hypergraphs by graphs. In Chapter 5, we describe a local search heuristic for solving the maximum balance problem in general signed hypergraphs. Chapter 6 is devoted to a greedy heuristic for minimum covering and the analysis of its performance bounds. In Chapter 7, we apply the mathematical programming approach. Chapter 8 presents an application to the optimal layer assignment problem. In Chapter 9, we describe how various sequential logic synthesis problems are related to constrained encoding, and then describe the *ENCORE* program for sequential logic synthesis. Chapter 10 summarizes the thesis and describes some open problems.

Chapter 2

Theory of Signed Hypergraphs

In this chapter, we formally define the notions of signed hypergraph and balance. We present characterizations of the balance property of signed hypergraphs, describe some related optimization problems, and investigate their complexity.

2.1 Terminology and Notation

A *signed hypergraph* H is an ordered triple* $(V(H), E(H), \psi_H)$ — or simply (V, E, ψ), if H is understood — consisting of a set V of elements, called *vertices*, a set E, disjoint from V, of elements called *edges*, and an *incidence function* $\psi : V \times E \rightarrow \{-1, 0, 1\}$.

The *incidence matrix* of a signed hypergraph $H = (V, E, \psi)$ is a $|V| \times |E|$ matrix

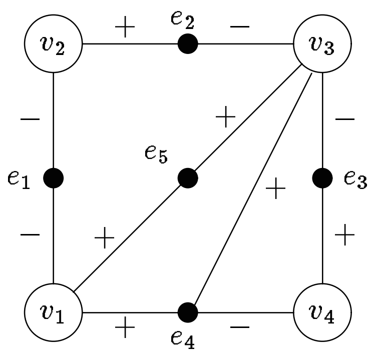
$$\boldsymbol{\psi} = (\psi_{ij})$$

where $\psi_{ij} = \psi(v_i, e_j)$, and $|V|$ ($|E|$) denotes the number of vertices (edges) of H. Clearly, each signed hypergraph has a unique incidence matrix, and each $(0, \pm 1)$-matrix corresponds to an incidence matrix of a signed hypergraph.

*We follow here the notation of Bondy and Murty [13] because the use of triples provides a simple way of specifying signed hypergraphs.

The *dual* $H^* = (V^*, E^*, \psi^*)$ of a signed hypergraph $H = (V, E, \psi)$ is a signed hypergraph where $V^* = E$, $E^* = V$, and ψ^* is the transpose of ψ. Consequently, $(H^*)^* = H$.

If e is an edge and v a vertex such that $\psi(v, e) \neq 0$, then v is said to be *incident with* e and vice versa. More specifically, if $\psi(v, e) = 1$, v is *positively* incident with e; if $\psi(v, e) = -1$, v is *negatively* incident with e. The incidence function permits an edge and a vertex to meet only once. Thus no edge can connect a vertex to itself; i.e., "self-loops" are not allowed in signed hypergraphs. Two or more vertices are said to be *adjacent* if they are incident with the same edge. The *degree* $d(v)$ of a vertex v in H is the number of edges of H incident with v. We denote by $\triangle_V$ the maximum degree of the vertices of H. The *degree* $d(e)$ of an edge e in H is the number of vertices of H incident with e. We denote by $\triangle_E$ the maximum degree of the edges of H.



(a)

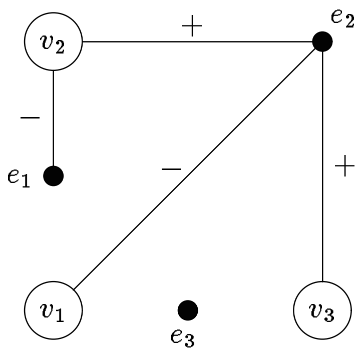
$$\begin{array}{ccccc|c}
e_1 & e_2 & e_3 & e_4 & e_5 & \\
\left(\begin{array}{ccccc} -1 & 0 & 0 & 1 & 1 \\ -1 & 1 & 0 & 0 & 0 \\ 0 & -1 & -1 & 1 & 1 \\ 0 & 0 & 1 & -1 & 0 \end{array}\right) & & & & & \begin{array}{c} v_1 \\ v_2 \\ v_3 \\ v_4 \end{array}
\end{array}$$

(b)

Figure 2.1: (a) A signed hypergraph H; (b) its incidence matrix.

A signed hypergraph H degenerates to a *signed graph* if $d(e) = 2$ for every edge $e \in E$. It degenerates to a *hypergraph* if $\psi(v, e) \in \{0, 1\}$ for all $v \in V$ and $e \in E$. Finally, a signed hypergraph H degenerates to a *graph* if $\psi(v, e) \in \{0, 1\}$ and $d(e) = 2$, for all $v \in V$ and $e \in E$. We usually denote a (signed) graph by G, and write $V(G)$, $E(G)$, and ψ_G, etc.

A signed hypergraph is shown in Fig. 2.1, where a circle represents a vertex, and a small solid circle (called *edge node*) with several line segments attached to it represents an edge. The graph obtained by treating edge nodes in the same way as the vertices of H is called the *underlying graph* of H. Figures 2.2 and 2.3 are two more examples of signed hypergraphs, where the signed hypergraph in Fig. 2.3 is the dual of the signed hypergraph in Fig. 2.2.

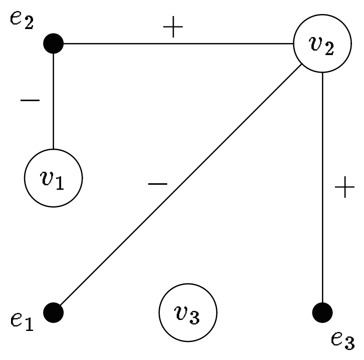


(a)

$$\begin{array}{c|c} e_1 \quad e_2 \quad e_3 & \\ \begin{pmatrix} 0 & -1 & 0 \\ -1 & 1 & 0 \\ 0 & 1 & 0 \end{pmatrix} & \begin{array}{c} v_1 \\ v_2 \\ v_3 \end{array} \end{array}$$

(b)

Figure 2.2: A signed hypergraph H_1; (b) its incidence matrix.



(a)

$$\begin{array}{c|c} e_1 \quad e_2 \quad e_3 & \\ \begin{pmatrix} 0 & -1 & 0 \\ -1 & 1 & 1 \\ 0 & 0 & 0 \end{pmatrix} & \begin{array}{c} v_1 \\ v_2 \\ v_3 \end{array} \end{array}$$

(b)

Figure 2.3: A signed hypergraph H_2; (b) its incidence matrix.

The underlying graph of a signed hypergraph has the property that all the vertices are partitioned into two subsets (the set of hypergraph vertices and the set of edge nodes) such that every edge connects one vertex from each subset; such a graph is called *bipartite* in graph theory. Each signed hypergraph has a unique underlying bipartite graph, and each bipartite graph corresponds to the underlying graph of a signed hypergraph. Thus we may use three terminologies — signed hypergraphs, $(0, \pm 1)$ matrices, and ± 1 (edge-) weighted bipartite graphs — interchangeably.

A graph is said to be *planar* if it is possible to embed it in a plane so that no two edges intersect. A signed hypergraph is *planar* if its underlying graph is planar. For example, signed hypergraphs in Fig. 2.1, Fig. 2.2, and Fig. 2.3 are planar.

A signed hypergraph H' is a *subhypergraph* of H (written $H' \subseteq H$) if $V(H') \subseteq V(H)$, $E(H') \subseteq E(H)$, and $\psi_{H'}$ is the restriction of $\psi(H)$ to $V(H') \times E(H')$. For example, H' defined by $V(H') = \{v_1, v_2, v_3\}$, $E(H') = \{e_1, e_3, e_4\}$, and

$$\psi_{H'} = \begin{pmatrix} -1 & 0 & 1 \\ -1 & 0 & 0 \\ 0 & -1 & 1 \end{pmatrix}$$

is a subhypergraph of H defined in Fig. 2.1. Note that, in terms of the incidence matrix, the restriction of ψ_H to $V(H') \times E(H')$ is obtained by crossing out those rows of ψ_H that are not in $V(H')$ and those columns of ψ_H that are not in $E(H')$. A *spanning subhypergraph* H' of H is a subhypergraph with $V(H') = V(H)$.

Suppose that V' is a nonempty subset of V. A *subhypergraph of* $H = (V, E, \psi)$ *induced by vertex set* V' is the subhypergraph whose vertex set is V' and whose edge set is the set of those edges of H that are incident only with vertices in V'; it is denoted by $H(V')$. We also say that $H(V')$ is a *vertex-induced subhypergraph* of H. For example, the subhypergraph of H in Fig. 2.1 induced on the vertex set $V' = \{v_1, v_2, v_3\}$ is the triple (V', E', ψ') where $E' = \{e_1, e_2, e_5\}$ and

$$\psi' = \begin{pmatrix} -1 & 0 & 1 \\ -1 & 1 & 0 \\ 0 & -1 & 1 \end{pmatrix}.$$

Note that the vertex-induced subhypergraph $H(V - V')$ is the subhypergraph obtained from H by deleting the vertices in V' together with all their incident edges.

Suppose that E' is a nonempty subset of E. A *subhypergraph of* $H = (V, E, \psi)$ *induced by edge set* E' is the subhypergraph whose vertex set is the set of vertices incident with edges in E' and whose edge set is E'; it is denoted by $H(E')$. We also say that $H(E')$ is an *edge-induced subhypergraph* of H. The spanning subhypergraph with edge set $E - E'$, written as $H(E - E')$, is the subhypergraph obtained from H by deleting the edges in E'. For example, the subhypergraph $H(E - \{e_4\})$ of H in Fig. 2.1 is the triple (V', E', ψ') defined below:

$$\begin{aligned} V' &= \{v_1, v_2, v_3, v_4\}, \\ E' &= \{e_1, e_2, e_3, e_5\}, \\ \psi' &= \begin{pmatrix} -1 & 0 & 0 & 1 \\ -1 & 1 & 0 & 0 \\ 0 & -1 & -1 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix}. \end{aligned}$$

Let $q \geq 1$ be an integer. A *path of length* q is defined to be a sequence

$$(v_1, e_1, v_2, e_2, ..., v_q, e_q, v_{q+1})$$

such that

$$\forall i : 1 \leq i \leq q : v_i \neq v_{i+1}, \psi(v_i, e_i) \neq 0, \text{ and } \psi(v_{i+1}, e_i) \neq 0.$$

The *sign* of a path is equal to

$$\prod_{i=1}^{q} \psi(v_i, e_i)\psi(v_{i+1}, e_i).$$

A *positive (negative) path* is a path with positive (negative) sign. A *cycle* is a path in which $v_{q+1} = v_1$; note that the length of any cycle is necessarily greater than 1. An *odd (even) cycle* is a cycle with odd (even) length. For example, for the signed hypergraph of Fig 2.1, the path $(v_1, e_1, v_2, e_2, v_3)$ is negative. Cycles $(v_1, e_1, v_2, e_2, v_3, e_5, v_1)$ and $(v_1, e_4, v_3, e_2, v_2, e_1, v_1)$ are negative, whereas the cycle

$$(v_1, e_1, v_2, e_2, v_3, e_5, v_1, e_4, v_3, e_2, v_2, e_1, v_1)$$

is positive.

A *bipartition* π of H is a separation of V into a pair of subsets, say (V_1, V_2), such that $V_1 \cup V_2 = V$ and $V_1 \cap V_2 = \emptyset$. Here V_1 and V_2 are called *blocks* of π. The block containing vertex v will be denoted by $\pi(v)$. An edge e is *balanced* by a bipartition π if the following condition is satisfied:

$$\forall v_1, v_2 \in V, \quad \psi(v_1, e) = \psi(v_2, e) \neq 0 \text{ implies } \pi(v_1) = \pi(v_2), \tag{2.1}$$

$$\psi(v_1, e) = -\psi(v_2, e) \neq 0 \text{ implies } \pi(v_1) \neq \pi(v_2). \tag{2.2}$$

For example, in Fig. 2.1, bipartition $(\{v_1, v_2\}, \{v_3, v_4\})$ balances edges e_1 and e_2, but not edges e_3, e_4 and e_5; bipartition $(\{v_1, v_2, v_3\}, \{v_4\})$ balances edges e_1, e_3, e_4 and e_5, but not edge e_2.

A bipartition is said to *balance* a signed hypergraph if it balances all of its edges. A signed hypergraph is said to be *balanced* if there exists a bipartition that balances all the edges. For example, bipartition $(\{v_1, v_2, v_3\}, \{v_4\})$ balances the subhypergraph $H(E - \{e_2\})$ of H in Fig. 2.1; hence this subhypergraph is balanced.

2.2 Fundamental Characterizations

In this section, we present some characterizations of the balance property of signed hypergraphs.

Proposition 2.2.1 *Every subhypergraph of a balanced signed hypergraph is balanced.*

Proof. Every edge of the subhypergraph is balanced by the bipartition that balances the original signed hypergraph. $\square$

Theorem 2.2.1 (Structure Theorem)
A signed hypergraph is balanced if and only if it is free of negative cycles.

Proof. We first prove the only if part. Suppose that a signed hypergraph H is balanced; then there exists a bipartition π such that, for each edge $e \in E(H)$, $\psi(v_1, e) = \psi(v_2, e) \neq 0$ implies that v_1 and v_2 are in the same block of π, and $\psi(v_1, e) = -\psi(v_2, e) \neq 0$ implies that v_1 and v_2 are in different blocks. We say that a path is *cut* by π if there exists a path segment of the form $v_i e_i v_{i+1}$ such that v_i and v_{i+1} belong to different blocks of π. Then the sign of any cycle in H may be determined by counting the number of times the cycle is cut by π. Since a cycle can only be cut by a bipartition an even number of times, every cycle must be positive.

We prove the if part by construction. Suppose that we are given a signed hypergraph H that is free of negative cycles. We construct a bipartition $\pi = (X, Y)$ as follows: Arbitrarily select a vertex v and assign it to the block X. Since H is free of negative cycles, for each pair of vertices v and v', all paths joining v and v' have the same sign. Search H in a depth-first manner and calculate the sign of any path from v to v'. If the sign is positive, then v' is in X, otherwise v' is in Y.

We claim that all the edges in H are balanced by π. By construction, for each edge $e \in E(H)$ which connects vertices v_1 and v_2, $\psi(v_1, e) = \psi(v_2, e) \neq 0$ implies that the path joining v_1 and v_2 is positive, and thus that v_1 and v_2 are in the same block of π. If $\psi(v_1, e) = -\psi(v_2, e) \neq 0$, then the path joining v_1 and v_2 is negative, and v_1 and v_2 are in different blocks of π. Therefore edge e is balanced by π, and H is a balanced signed hypergraph. $\square$

The construction used in the proof above gives a linear-time algorithm for checking whether a signed hypergraph is balanced. We state this observation as a corollary.

Corollary 2.2.1 *Checking the balance of a signed hypergraph takes linear time.*

Theorem 2.2.2 (Duality Theorem)
The dual of a balanced signed hypergraph is balanced.

Proof. By Theorem 2.2.1, a signed hypergraph is balanced if and only if it is free of negative cycles. Since a signed hypergraph and its dual have the same underlying bipartite graph, cycles in a signed hypergraph have a one-to-one correspondence with cycles in its dual. Thus the dual of a balanced signed hypergraph is also free of negative cycles, and it is therefore balanced. □

2.3 The Maximum Balance Problem and its Complexity

In this section, we study the following problem: Given a signed hypergraph, find a bipartition such that the number of balanced edges is maximized. This is called the *maximum balance problem*. It is a natural graph-theoretic formulation of the optimal layer assignment problem of two-layer routings in VLSI layout synthesis [132]. In fact, it was this reason that motivated us to introduce signed hypergraphs and to study their structural properties.

The main result of this section is the NP-completeness of the maximum balance problem; more precisely, we establish a clear boundary between polynomial-time solvable cases and the general NP-complete case. In addition, the complexity of the "dual" of the maximum balance problem and the complexity of the maximum balance problem in a *planar* signed hypergraph are investigated.

Before we prove the NP-completeness results, we first make the following observations. Given a signed hypergraph H, the maximum balance problem is that of finding a bipartition π such that the number of balanced edges is maximized. From the definition of balance, the spanning subhypergraph of H with the set of edges balanced by π is balanced. In other words, after the set E' of edges that are not balanced by π is removed, the remaining signed hypergraph $H(E - E')$ is balanced. By Theorem 2.2.1, $H(E - E')$ is free of negative cycles. By Corollary 2.2.1, the construction for $H(E - E')$ of a bipartition π that balances all the edges in $E - E'$ takes linear time. Therefore, an alternative formulation of the maximum balance problem, which is of the same time complexity as the original, is as follows:

> Given a signed hypergraph $H = (V, E, \psi)$, find a set $E' \subseteq E$ with minimum cardinality such that $H(E - E')$ is free of negative cycles.

Following the terminology of Yannakakis [159], we call this the *edge-deletion balance* problem for a signed hypergraph. The corresponding decision problem is: Given H and an integer $k \geq 0$, does there exist a set E' of k edges such that $H(E - E')$ is free of negative cycles?

A key to our proof of the NP-completenesses of the maximum balance problem and its variants is provided by Lemmas 2.3.1 and 2.3.2, which state some properties of the following construction: Given a graph $G = (V, E, \psi)$, construct a signed hypergraph H with the same set of vertices and the same set of edges. For each pair of vertices v_i and v_j in G joined by an edge e in E, set $\psi_H(v_i, e) = 1$ and $\psi_H(v_j, e) = -1$ in H. We refer to this as the *sign construction.* The sign construction takes linear time. Since $d(e) = 2$ for every $e \in E$ in G, H so constructed is in fact a signed graph; i.e., $\triangle_E = 2$. An example of the sign construction is given in Fig. 2.4.

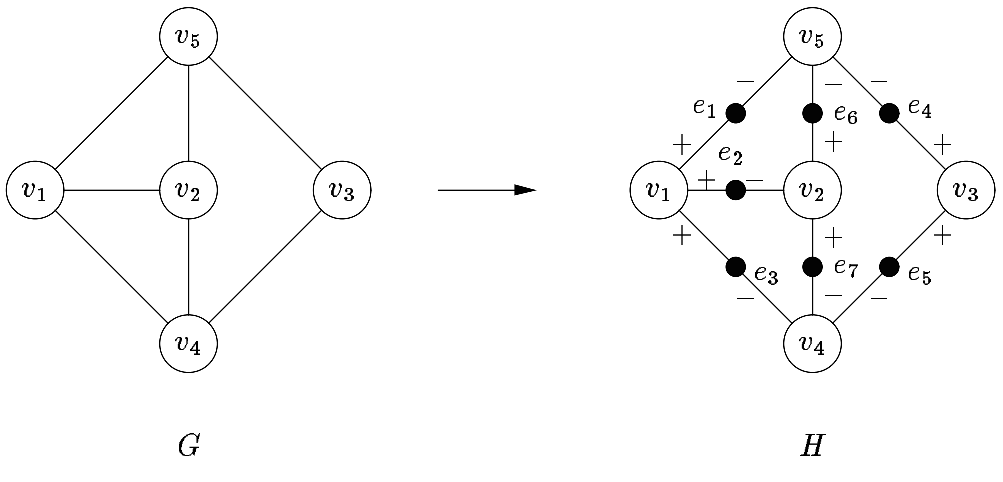


Figure 2.4: An illustration of the sign construction.

Lemma 2.3.1 *Let G be a graph, and H any corresponding signed hypergraph obtained by the sign construction. Then there exists a set E' of k edges in G such that $G(E-E')$ is bipartite if and only if there exists a set of k edges in H such that $H(E-E')$ is free of negative cycles.*

Proof. Let e be an arbitrary edge in H, and let v_1 and v_2 be two vertices incident with e. By the construction of H, we always have $\psi(v_1, e)\psi(v_2, e) = -1$. Therefore the sign of any cycle in H is determined by the number of edges in the cycle: It is positive if the cycle is even, and it is negative otherwise. Thus H is free of negative cycles, if and only if G is free of odd cycles. It is well known that G is free of odd cycles if and only if G is bipartite [12]. □

Similarly, we have the following result:

Lemma 2.3.2 *Let G be a graph, and H any corresponding signed hypergraph obtained by the sign construction. Then there exists a set V' of k vertices in G such that $G(V-V')$ is bipartite if and only if there exists a set of k vertices in H such that $H(V-V')$ is free of negative cycles.*

Theorem 2.3.1 *The edge-deletion balance problem for a signed hypergraph $H = (V, E, \psi)$ is NP-complete even if (a) $\triangle_E = 2$ and $\triangle_V = 3$, and (b) $\triangle_E = 3$ and $\triangle_V = 2$.*

Proof. We first show that the edge-deletion balance problem is in NP. Given a set E' of k edges, checking whether $H(E-E')$ is free of negative cycles is equivalent to checking whether a bipartition exists that balance all the edges in $E-E'$; this can be done in linear time by Corollary 2.2.1.

The NP-hardness of the edge-deletion balance problem for signed hypergraphs with $\triangle_E = 2$ and $\triangle_V = 3$ is proved by showing that the problem can be reduced in polynomial time from the following problem, which was proved to be NP-complete by Garey, Johnson and Stockmeyer [49], and Yannakakis [159]:

Edge-Deletion Bipartite Problem:

Instance: A graph $G = (V, E, \psi)$ with $d(v) = 3$ for every vertex $v \in V$, and an integer k.

Question: Does there exist a set E' of k edges such that $G(E - E')$ is bipartite?

Given a graph $G = (V, E, \psi)$ with $d(v) = 3$ for every vertex $v \in V$, we use the sign construction to generate a signed hypergraph H. Since $d(v) = 3$ for every $v \in V$ in G, the signed hypergraph H so constructed has $\triangle_E = 2$ and $\triangle_V = 3$. By Lemma 2.3.1, there exists a set E' of k edges in G such that $G(E - E')$ is bipartite if and only if there exists a set of k edges in H such that $H(E - E')$ is free of negative cycles. Therefore the problem for a signed hypergraph with $\triangle_E = 2$ and $\triangle_V = 3$ is NP-hard.

To prove the NP-hardness of the maximum balance problem for a signed hypergraph with $\triangle_E = 3$ and $\triangle_V = 2$, we use the "dual" of the edge-deletion bipartite problem — the vertex-deletion bipartite problem, which was proved to be NP-complete by Choi, Nakajima and Rim [24]:

Vertex-Deletion Bipartite Problem:

Instance: A graph $G = (V, E, \psi)$ with $d(v) = 3$ for every vertex $v \in V$, and an integer k.

Question: Does there exist a set V' of k vertices such that $G(V - V')$ is bipartite?

Given a graph with $d(v) = 3$ for every vertex $v \in V$, we apply the sign construction to generate a signed hypergraph $(V(H), E(H), \psi_H)$. We then construct the dual H^* of H. Since $d(v) = 3$ for every vertex $v \in V$, H^* is such that $\triangle_V = 2$ and $\triangle_E = 3$.

We claim that there exists a set V' of k vertices in G such that $G(V - V')$ is bipartite if and only if there exists a set E' of k edges in H^* such that $H^*(E - E')$ is balanced. By Lemma 2.3.2, there exists a set V' of k vertices in G such that $G(V - V')$

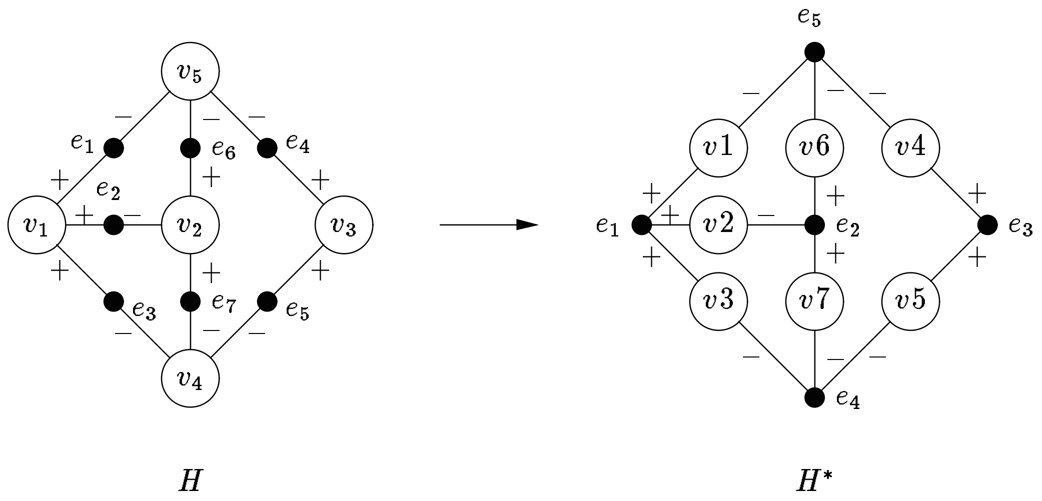


Figure 2.5: An example of the dual construction.

is bipartite if and only if there exists a set V' of k edges in H such that $H(V - V')$ is balanced. See Fig. 2.4. By Theorem 2.2.2, there exists a set V' of k vertices in H such that $H(V - V')$ is balanced, if and only if there exists a set E' of k edges in H^* such that $H^*(E - E')$ is balanced. See Fig. 2.5. Thus the claim is indeed true. Both the sign construction and the dual construction take linear time. □

The "dual" of the edge-deletion balance problem is the *vertex-deletion balance* problem:

> Given a signed hypergraph $H = (V, E, \psi)$, find a set $V' \subseteq V$ with minimum cardinality such that $H(V - V')$ is balanced.

As a corollary of Theorems 2.2.2 and 2.3.1, we have the following result.

Corollary 2.3.1 *The vertex-deletion balance problem for a signed hypergraph is NP-complete even if (a) $\triangle_E = 2$ and $\triangle_V = 3$, and (b) $\triangle_E = 3$ and $\triangle_V = 2$.*

To complete our analysis, we consider the cases that are not known to be NP-complete. For the general signed hypergraph, it is easy to see that the case with $\triangle_E =$

2 and $\triangle_V = 2$ can be solved in linear time. This leads to a complete understanding of the complexity of the maximum balance problem in a general signed hypergraph, as illustrated in Fig. 2.6.

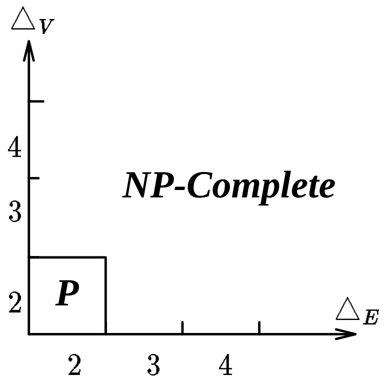


Figure 2.6: Complexity of the maximum balance problem.

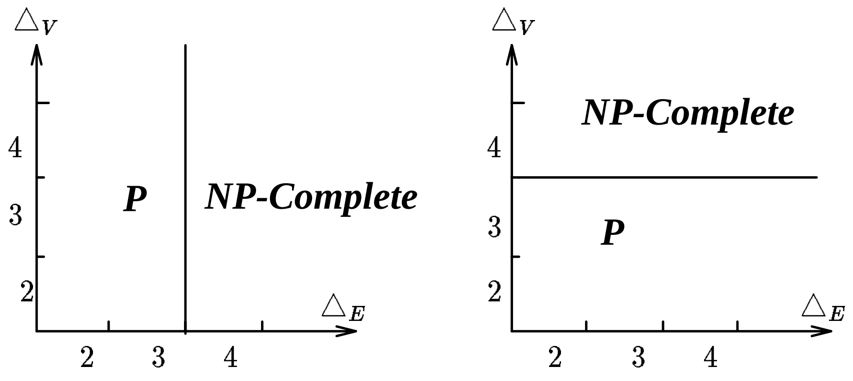
In the rest of this section, we consider the effect of planarity on the complexity of the maximum balance problem. This is motivated by the fact that the layer assignment problem of integrated circuit layout gives rise to signed hypergraphs that are usually planar.

According to Choi, Nakajima and Rim [24], the vertex-deletion bipartite problem when restricted to a planar graph is still NP-complete when $\triangle_V > 3$. By using the same constructions and arguments as in the proof of Theorem 2.3.1, and noting that the dual of a planar signed hypergraph is planar, we have the following result:

Theorem 2.3.2 *The edge-deletion balance problem is NP-complete for a planar signed hypergraph with* $\triangle_E > 3$.

Corollary 2.3.2 *The vertex-deletion balance problem is NP-complete for a planar signed hypergraph with* $\triangle_V > 3$.

For the maximum balance problem of planar signed hypergraphs with $\triangle_E \leq 3$, we will develop polynomial-time algorithms in the next chapter. A complete picture of the complexity results for maximum balance in planar signed hypergraphs is in Fig. 2.7.



(a) edge deletion (b) vertex deletion

Figure 2.7: Complexity of planar maximum balance.

2.4 The Minimum Covering Problem and its Complexity

In this section, we consider the following problem: Given a signed hypergraph, find a minimal number of bipartitions such that each edge is balanced by at least one bipartition. This is called the *minimum covering problem.* It is a natural graph-theoretic formulation of the constrained encoding problem arising in various contexts of VLSI logic synthesis (see Chapter 9). The main result of this section is the NP-completeness of the minimum covering problem.

We introduce a notion of set decomposition. If E is a set, a *decomposition* of E is a set $\{E_i,\ i = 1, ..., k\}$ of subsets E_i of E such that $\cup_{i=1}^{k} E_i = E$. If E is a set of edges of a signed hypergraph H, and $\{E_i,\ i = 1, ..., k\}$ is a decomposition of E, then $\{H(E_i),\ i = 1, ..., k\}$ is also called a *decomposition* of H.

By Theorem 2.2.1, the subhypergraph induced by the set of edges that are balanced by a bipartition is balanced, i.e., free of negative cycles. Thus an alternative formulation of the minimum covering problem is as follows:

> Given a signed hypergraph $H = (V, E, \psi)$, find a decomposition $\{H_i, i = 1, ..., k\}$ of H such that (a) H_i is balanced and (b) k is minimized.

This is the *balanced subhypergraph decomposition problem.* If $k = 1$, the problem degenerates to that of testing whether a signed hypergraph H is balanced, which is solvable in linear time. If $k = |E|$, the problem can also be solved in linear time by simply choosing each edge as a signed hypergraph.

Theorem 2.4.1 *The balanced subhypergraph decomposition problem for a signed hypergraph is NP-complete.*

Proof. We first show that the balanced subhypergraph decomposition problem is in NP. Given a signed hypergraph H and a set $\{H_i,\ i = 1, ..., k\}$, we can verify in time polynomial in the size of the problem, i.e., in the size of H plus $\{H_i,\ i = 1, ..., k\}$, whether H_i is balanced for all i and whether $\{H_i,\ i = 1, ..., k\}$ is indeed a decomposition of H.

To prove the NP-hardness, we show that the problem is polynomial-time reducible from the following known NP-complete problem [48]:

Graph K-Colorability Problem

Instance:	A graph $G = (V, E, \psi)$, and an integer $2 < K < \|V\|$.
Question:	Does there exist a mapping $f : V \rightarrow \{1, ..., K\}$ such that, for each edge e incident with vertices v_i and v_j, $f(v_i) \neq f(v_j)$?

If K is restricted so that $K = 2^k$, $1 < k < \log_2 |V|$, this is the graph 2^k-colorability problem. Since the graph K-colorability problem remains NP-complete for any fixed K—for example, the graph 3-colorability problem is NP-complete— the graph 2^k-colorability problem is also NP-complete.

We now show that the graph 2^k-colorability problem is equivalent to the following problem:

Bipartite Subgraph Decomposition Problem

Instance: A graph $G = (V, E, \psi)$ and an integer k, $1 < k < \log_2 |V|$.

Question: Does there exist a decomposition $\{G_i, i = 1, ..., k\}$ of G such that G_i is bipartite for all i?

In other words, a graph is 2^k-colorable, if and only if it can be *decomposed* into k bipartite subgraphs.

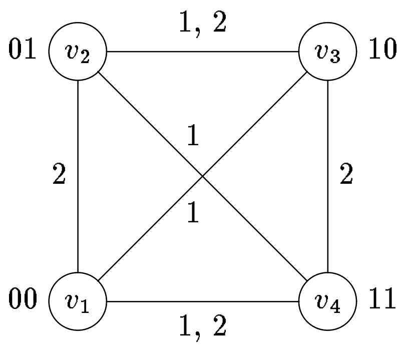


Figure 2.8: A four-coloring of a graph.

First, suppose that a graph G is decomposable into a set G_i, $i = 1, ...k$, of bipartite subgraphs, and that (V_i^-, V_i^+) is a bipartition of G_i such that no edge connects a vertex in V_i^- to a vertex in V_i^+. We then assign to any vertex v a k-bit binary number with the ith bit determined as follows: It is 1 if v is in V_i^+, and it is 0 otherwise. Since any edge e is contained in at least one bipartite subgraph, the binary numbers assigned in this way to the two vertices incident to e differ in at least one bit. This leads to a valid 2^k-coloring. For example, in Fig. 2.8, graph G is decomposed into two bipartite subgraphs: G_1 induced by those edges labeled by 1, and G_2 induced by those edges labeled by 2. Subgraph G_1 is bipartite with respect to bipartition $(\{v_1, v_2\}, \{v_3, v_4\})$. Thus 0 is assigned to v_1 and v_2, and 1 to v_3 and v_4 for the first bit. Subgraph G_2 is bipartite with respect to bipartition $(\{v_1, v_3\}, \{v_2, v_4\})$.

Thus 0 is assigned to v_1 and v_3, and 1 to v_2 and v_4 for the second bit. This results in a four-coloring of graph G: (v_1, v_2, v_3, v_4) *to* $(00, 01, 10, 11)$. In summary, if G is decomposable into k bipartite subgraphs, then it is 2^k-colorable.

Conversely, suppose that a graph G is 2^k-colorable. The 2^k colors can be represented by k-bit binary numbers; Then the two numbers assigned to two vertices incident to an edge differ in at least one bit. We say that the edge is *separated by that bit*. In Fig. 2.8, the bits that separate an edge are marked: for example, the edge that joins v_2 and v_3 is separated by bits 1 and 2. All the edges separated by the same bit induce a subgraph; thus there is a total of k subgraphs. According to the definition of coloring, each edge is separated by at least one bit, and hence contained in at least one subgraph. Therefore, the set of k subgraphs is a decomposition of the original graph. From the construction, the bit value assigned to consecutive vertices in every cycle must alternate between 0 and 1. That means that every cycle in each subgraph is of even length; therefore each subgraph is bipartite.

Altogether, we have shown that a graph is 2^k-colorable, if and only if it can be decomposable into k bipartite subgraphs. Since the graph 2^k-colorability problem is NP-complete, so is the bipartite subgraph decomposition problem. Now we only need to show that the bipartite subgraph decomposition problem can be transformed into the balanced subhypergraph decomposition problem in polynomial time. This follows immediately from the sign construction of a signed hypergraph from a given graph, and from Lemma 2.3.1. $\square$

2.5 Signed Graphs and Balanced $(0, \pm 1)$ Matrices

To our knowledge, the notion of signed hypergraph has not previously appeared in the literature. However, the equivalent concepts of $(0, \pm 1)$ matrices and of ± 1 weighted (or called directed) bipartite graphs are important concepts in the area of mathematical programming. In the graph-theoretical community, there have been studies on such topics as signed graphs [59]. In this section, we provide a brief summary of these concepts. Our emphasis is on their relations with signed hypergraphs.

Harary's Signed Graphs

In the study of certain phenomena in social psychology, Harary [59] conceived the notion of signed graph and it balance. A signed graph G consists of a set V of vertices together with two disjoint subsets E^+ and E^- of the set of unordered pairs of vertices. The elements of the sets E^+ and E^- are called positive edges and negative edges respectively. A cycle of a signed graph is positive if the number of negative edges involved is even; otherwise it is negative. A signed graph is balanced if all of its cycles are positive. Harary proved the following theorem: A signed graph G is balanced if and only if its vertex set V can be partitioned into two disjoint subsets V_1 and V_2 in such a way that each positive edge of G joins two vertices of the same subset and each negative edge joins two vertices of different subsets.

Clearly, our notions of signed hypergraph and the sign of a cycle, and our structure theorem generalize nh counterparts above. One slight difference is that we define the concept of balance in terms of bipartitions instead of cycles. This is because the VLSI synthesis problems that we are interested in are naturally stated in terms of bipartitions. The absence of negative cycles is thus a structural property of these problems.

We make several remarks. First, our proof of the structure theorem is simple and provides a linear-time algorithm for balance testing. Second, as shown in Chapter 4, problems defined over signed graphs can be transformed exactly to problems in terms of weighted graphs. In this sense, signed graphs have the same "expressive" power as ordinary graphs. Signed hypergraphs, however, provide more expressive power than ordinary graphs, as we shall see in Chapter 4. Finally, we note some recent work by Zaslavsky on characterization [163] and orientation embedding [164] of signed graphs, by Hoede on related marked graphs [62], and by Boros, Crama and

Hammer on the relation between maximum balance of signed graphs and quadratic 0-1 optimization [14].

Restricted Unimodularity and Balanced $(0, \pm 1)$ Matrices[†]

A matrix is totally od (TU) if every square submatrix has determinant $0, \pm 1$. It is well known that integer programs whose constraint matrices have the TU property can be solved optimally by relaxing the integrality restriction. This is important since integer programs are NP-complete, whereas their relaxations (linear programs) are polynomial-time solvable.

A very simple sufficient condition for total unimodularity is called restricted total unimodularity. A $(0, \pm 1)$ matrix ψ is restricted totally od if and only if the corresponding ± 1 edge-weighted bipartite graph G of ψ is free of "negative" cycles. Here a "negative" cycle is one with the sum of its weights of all involved edges congruent to 2 modulo 4 according to Conforti and Rao [29]. Alternatively, Yannakakis [160] defined the sign of a cycle to be the product of the signs of its edges, with the sign of an edge determined as follows: Suppose that G is bipartite with respect to bipartition (V^+, V^-), where V^+ and V^- are two sets of vertices corresponding to all the edges and all the vertices, respectively. Then an edge is directed from a vertex v_i in V^+ to a vertex v_j in V^- if $\psi_{ij} = 1$, and to a vertex v_i in V^+ from a vertex v_j in V^- if $\psi_{ij} = -1$. Such a directed bipartite graph is called a *matrix digraph* by Yannakakis. Let C be a cycle; we traverse C in one direction, assign a "sign" $+1$ to an edge e if e has the same direction, and assign a "sign" -1 to e if e has the opposite direction.

Instead of associating signs with edges in ± 1 edge-weighted bipartite graph G as above, we can equivalently associate signs to "path segments" in the corresponding signed hypergraph as follows: The sign of a path segment $v_i e_j v_{i+1}$ is -1 times the product of $\psi(v_i, e_j)$ and $\psi(v_{i+1}, e_j)$. This is similar to our definition in Section 2.1, whether the sign of a path segment $v_i e_j v_{i+1}$ is 1 times the product of $\psi(v_i, e_j)$ and

[†]The author thanks Anna Lubiw for bringing this work to his attention.

$\psi(v_{i+1}, e_j)$. But this apparently slight difference leads to very distinct results: balance testing is much simpler than restricted-unimodularity testing. As shown in Theorem 2.2.1, for balance testing, the absence of negative cycles can be checked by finding a bipartition, which is essentially one pass of depth-first search. For unimodularity testing, the absence of "negative" cycles needs an examination of the entire cycle space, a much more involved process [29, 160].

Unimodularity testing for a signed graph G can be done by balance testing of its "negation" $\overline{G}$. The negation $\overline{G}$ of G is the same as G except that the sign of each edge is the negation of the original sign (here we use nh notion). It follows from Theorem 2.2.1 that, balance testing of $\overline{G}$ amounts to testing whether the set of vertices can be partitioned into two subsets so that any two vertices joined by a positive edge in $\overline{G}$ are in the same subset and any two vertices joined by a negative edge in $\overline{G}$ are in different subsets. In terms of G, this amounts to testing whether the set of vertices can be partitioned into two subsets so that any two vertices joined by a negative edge are in the same subset and any two vertices joined by a positive edge are in different subsets. In terms of the incidence matrix ψ of G, which has two nonzero entries in each column, ψ is totally od if and only if the set of rows can be partitioned into two subsets, so that for every column with two 1's or two -1's, one nonzero entry is in each subset, and for every column with an 1 and a -1, both nonzero entries are in the same subset. This is a well-known result in mathematical programming [64].

Very recently, Conforti and Cornuéjols [30] reported that balanced $(0, \pm 1)$ matrices defined by Truemper [147] are a superclass of totally unimodular matrices. A *balanced* $(0, \pm 1)$ matrix is a $(0, \pm 1)$ matrix in which for every submatrix with exactly two nonzero entries per row and per column, the sum of the entries is a multiple of 4. In terms of the corresponding ± 1 edge-weighted bipartite graph, a $(0, \pm 1)$ matrix is balanced if and only if the sum of the weights of the edges in every chordless cycle is a multiple of 4. The problem of how to recognize such balanced $(0, \pm 1)$ matrices remains open.

2.6 Summary

In this chapter, we introduced the notion of *signed hypergraph*. We defined the concepts of *edge-balance* and *negative cycle*. We gave characterizations of the edge-balance property: the structure theorem and the duality theorem. We introduced two optimization problems: maximum balance and minimum covering. The two problems are natural models of two VLSI synthesis problems respectively; however, the graph-theoretic formulations have not been explored in the past. The main scope of this thesis is the study of the complexity, solution methods, and applications of these problems.

The complexity issues were another focus of this chapter. We established that both the maximum balance problem and the minimum covering problem are NP-complete. Moreover, we have shown that the maximum balance problem remains NP-complete even for planar signed hypergraphs.

The notion of signed hypergraph was first introduced by us in the study of the VLSI via minimization problem [137]. It happens to be a generalization of the notion of signed graph introduced by Harary in the mid 1950s for studying certain phenomena in social physiology [59]. The structure theorem is thus a generalization of Harary's theorem of signed graphs. The proof given here indeed provides a simple proof of Harary's original theorem.

Chapter 3

Maximum Balance in Planar Signed Hypergraphs

One motivation of studying the maximum balance problem comes from the need to design efficient algorithms for the optimum layer assignment problem in VLSI layout design, since the former is a rigorous graph-theoretic formulation of the latter. There are four practical aspects of the formulation that may be helpful in designing efficient algorithms. First, optimum layer assignment gives rise to signed hypergraphs that are planar. Second, most of the edges in the resulting signed hypergraph connect only two vertices. In other words, hyperedges are rare. Third, all hyperedges have weight 1, and all edges with degree two have small integer weights, which are usually 1. Finally, the number of edges that must be removed to balance the resulting signed hypergraph is usually significantly smaller than the total number of edges.

This chapter is devoted to algorithm development for the maximum balance problem in planar signed hypergraphs. In particular, we will reduce the maximum balance problem in a planar signed hypergraph to a so-called hypergraph T-join problem in its planar dual. Although the notion of hypergraph T-join is new, its restriction—graph T-join—has been well-studied in combinatorial optimization. There are polynomial-time algorithms for finding a minimum T-join. Such algorithms can be generalized to hypergraph T-joins to yield pseudo-polynomial-time algorithms.

3.1 Planar Duals and Marked Hypergraphs

In this section, we show that the maximum balance problem in a planar signed hypergraph can be simplified by exploiting planarity. Specifically, we show that the problem reduces to the so-called *minimum matching set* problem in its planar dual.

Consider a planar signed hypergraph embedded in the plane*. Such a planar signed hypergraph partitions the plane into a number of connected regions, called *faces.* Figure 3.1(a) shows a planar signed hypergraph H with four faces, f_1, f_2, f_3, and f_4. We also refer to the cycle that forms the boundary of a given face as a face. For example, f_2 in Fig. 3.1(a) corresponds to the cycle $v_1 e_1 v_2 e_2 v_3 e_5 v_1$.

A *positive (negative) face* is a face (i.e., a cycle) with positive (negative) sign. In Fig. 3.1(a), faces f_1 and f_2 are negative, whereas faces f_3 and f_4 are positive.

Each planar signed hypergraph has exactly one unbounded face, called the *exterior face*; in Fig. 3.1, f_1 is the exterior face. All other faces are called *interior faces.* We say that the exterior face *encloses* a set of interior faces. Similarly, we also say that a cycle encloses a set of faces. For example, cycle $v_1 e_1 v_2 e_2 v_3 e_4 v_1$ encloses faces f_2 and f_3.

Proposition 3.1.1 *For a planar signed hypergraph H, the sign of a cycle is equal to the product of the signs of all the faces that are enclosed by the cycle.*

Proof. We view a planar signed hypergraph as its underlying bipartite graph; each edge in this bipartite graph is associated with either $+1$ or -1. Given a cycle

*In this thesis, when we speak of a planar signed hypergraph, we mean a planar signed hypergraph embedded in the plane. Such an embedding is normally given in the applications we have in mind [132].

that encloses a set of faces, we refer to edges in the cycle as *boundary edges*, and to edges in the faces but not in the cycle as *internal edges.* Since each internal edge is used by exactly two faces, the product of the signs of all the faces enclosed by the cycle is equal to the product of the signs of boundary edges, i.e., the sign of the cycle. □

For example, in Fig. 3.1(a), cycle $C = v_1e_1v_2e_2v_3e_4v_1$ encloses two faces f_2 and f_3. It is easy to verify that the sign of C is equal to the product of the signs of f_2 and f_3, which is negative. The sign of f_1 is equal to the product of the signs of f_2, f_3 and f_4. This leads to the following proposition:

Proposition 3.1.2 *A planar signed hypergraph has an even number of negative faces.*

Proof. By Proposition 3.1.1, the sign of the exterior face is determined by the number of negative faces enclosed: it is positive if it encloses an even number of negative faces, otherwise negative. Thus the total number of negative faces is even. □

A face is said to be *incident* with the vertices and edges in its boundary. When an edge is incident with more than one face, we say that the edge *separates* the faces incident with it. For example, in Fig. 3.1(a), edge e_1 separates faces f_1 and f_2, and edge e_4 separates faces f_1, f_3 and f_4. It can be seen that the number of faces separated by an edge is less than or equal to the degree of the edge.

Now we consider how to balance a planar signed hypergraph H. This is equivalent to having $H(E - E_1)$ free of negative faces, since H is planar; hence we are interested in faces and edges. Further, by Proposition 3.1.1, the sign of the resulting face after the removal of an edge is the product of the signs of all the faces separated by the edge, i.e., it is independent of the sign of the edge. This gives rise to the notion of planar dual.

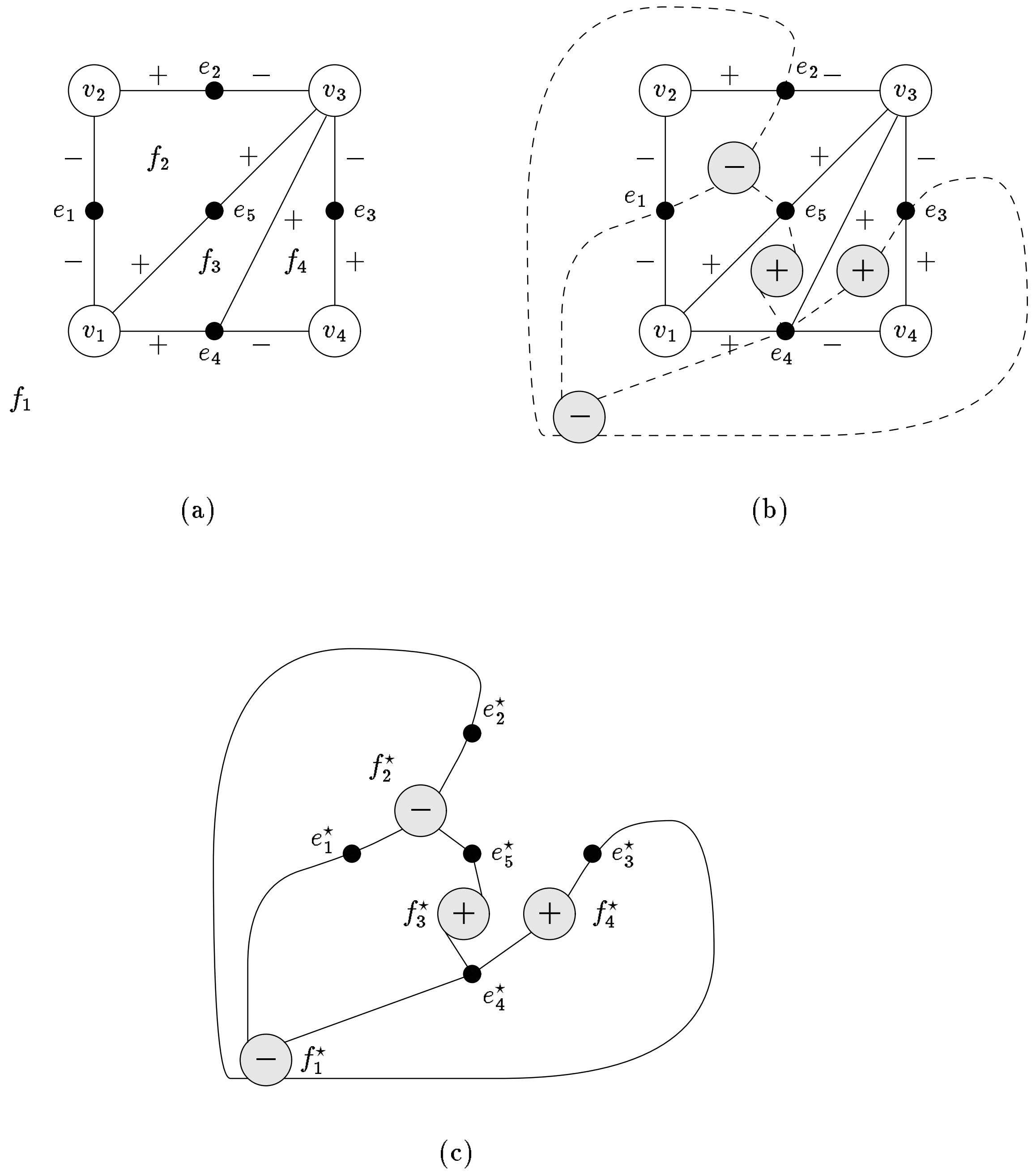


Figure 3.1: Planar dual of a planar signed hypergraph H.

Let H be a planar signed hypergraph; the *planar dual* $H^\star$ of H is defined as follows: corresponding to each face f of H there is a vertex $f^\star$ of $H^\star$; Corresponding to edge e of H that separates two or more faces, there is an edge $e^\star$ of $H^\star$; a set of vertices in $H^\star$ are joined by edge $e^\star$ if and only if their corresponding faces are separated by edge e in H. Associated with each vertex $f^\star$ in $H^\star$ is a sign that is the sign of its corresponding face f in H.

For example, the planar dual $H^\star$ of the planar signed hypergraph H in Fig. 3.1(a) is shown in Fig. 3.1(c).

One can verify that edges in H incident with only one face are not of interest. For example, the planar dual $H_1^\star$ of the planar signed hypergraph H_1 in Fig. 3.2(a) is shown in Fig. 3.2(b) (signs are omitted here). We note that edges e_6 and e_{10} in H_1 do not have counterparts in $H_1^\star$.

The planar dual of a planar signed hypergraph is a hypergraph in which each vertex is associated with either a positive sign or a negative sign. Such hypergraphs are called *marked hypergraphs*, following Beineke and Harary [9], who invented the notion of *marked graphs* in the modeling of relations between persons in psychology. In a marked hypergraph, vertices associated with a positive (negative) sign are called *positive (negative) vertices.*

The marked hypergraph as the planar dual of a planar signed hypergraph has several properties. First, it is planar. Second, it is connected, i.e., for every pair of vertices u and v, there exists a path with u and v as end-vertices. Furthermore, by Proposition 3.1.2, it has an even number of negative vertices.

Before we get into the formulation of the maximum balance problem of a planar signed hypergraph H in terms of its planar dual $H^\star$, we describe the following result regarding bounds on solutions to our problem.

Proposition 3.1.3 *Let G be a planar signed graph with n faces and k negative faces.*

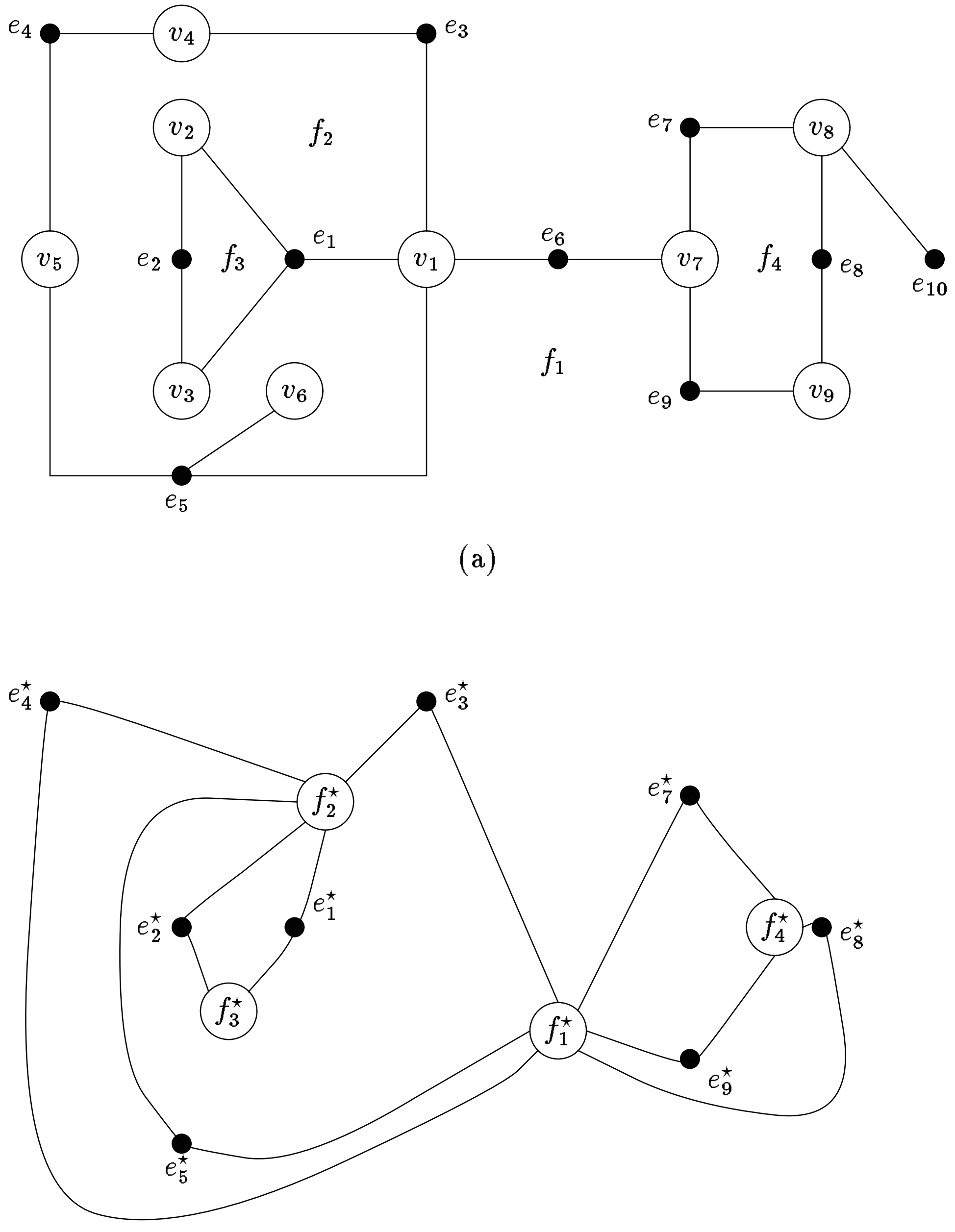


Figure 3.2: Planar signed hypergraph, H_1, and its planar dual.

Let m be the minimum number of edges needed to remove in order to balance H. Then $\frac{1}{2}k \leq m \leq n$.

Proof. We first prove that $m \leq n$. Clearly, we can delete one edge from each face to make all the faces positive. By Proposition 3.1.1, all the cycles in G are positive. Let E' be such a set of deleted edges; then $m = |E'| \leq n$ and $H(E - E')$ is balanced.

We now prove that $m \geq \frac{1}{2}k$. Since each edge is adjacent with at most two faces, the deletion of an edge will make at most 2 negative faces positive. There are k negative faces. So the number of edges needed to remove in order to balance G is at least $\frac{1}{2}k$. $\square$

Since the optimum layer assignment problem gives rise to a signed hypergraph with a majority of edges having degree 2, the bounds above have practical implications. In terms of layer assignment, m is the number of real vias, and n (the number of faces) is related to the number of potential vias ($|E|$) through the Euler formula:

$$n = \text{ the number of potential vias } - \text{ the number of clusters (vertices) } + 2.$$

Practically, in order to achieve a global via reduction, the number of potential vias chosen is usually much greater than the number of real vias m [133]. Therefore, k is also much smaller than n. This observation will be used later in Section 3.3 in finding a practically efficient algorithm for optimum layer assignment.

Now we consider how to formulate the maximum balance problem of a planar signed hypergraph H in its planar dual $H^\star$. We need to remove a set E_1 of edges such that $H(E - E_1)$ is free of negative faces. By Lemma 3.1.1, any negative face f can be eliminated if and only if it is merged with an odd number of negative faces. We assume that E_f is the set of edges that need to be removed for eliminating f, and that $E_f^\star$ is the corresponding set of edges in $H^\star$. Then $H^\star(E_f^\star)$ is a connected subhypergraph that contains an even number of negative vertices; we call such a subhypergraph a *matching component*. Given a marked hypergraph $H = (V, E, \psi)$

with an even-cardinality set $T \subseteq V$ of vertices, a *matching set* M is a set of edges such that (a) $H(M)$ consists of a set of matching components, and (b) $T \subseteq V(H(M))$. For example, consider $H^\star$ in Fig. 3.1 with $T = \{f_1^\star, f_2^\star\}$. Then $\{e_1^\star\}$, $\{e_2^\star\}$, $\{e_4^\star, e_5^\star\}$, and any subset of edges that contains any of the three subsets above, are matching sets of $\langle H^\star, T\rangle$.

Therefore, the maximum balance problem in a planar signed hypergraph is equivalent to the following problem: Given a planar marked hypergraph $H = (V, E, \psi)$ with $T \subseteq V$ consisting of an even number of negative vertices, find a matching set having minimum cardinality. We will call this the *minimum matching set* problem in a planar marked hypergraph.

By Theorem 2.3.2, the maximum balance problem in a planar signed hypergraph is NP-complete; we thus have the following complexity result.

Theorem 3.1.1 *The minimum matching set problem is NP-complete if $\Delta_E > 3$, even for planar marked hypergraphs.*

3.2 Reduction to Hypergraph T-Joins

In this section, we first generalize a known graph-theoretic notion of T-join to marked hypergraphs. We then observe that a T-join in a marked hypergraph is a matching set, and thus show that the minimum matching set problem reduces to the minimum hypergraph T-join problem. The significance of this reduction lies in the fact that the minimum graph T-join problem can be solved efficiently.

We recall the notion of graph T-join [97]. Let $G = (V, E, \psi)$ be a graph and let $T \subseteq V$ be of even cardinality. A *graph T-join* of $\langle G, T\rangle$ is defined to be a subset of the edges that meets each vertex of T an odd number of times and that meets each vertex of $V - T$ an even number of times. Figure 3.3 illustrates two examples of

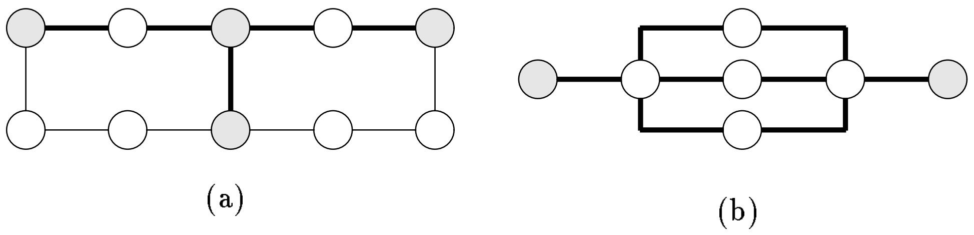


Figure 3.3: Examples of graph T-joins.

graph T-joins, where shaded cycles represent the vertices in T, and edges represented by bold-faced lines form a T-join. This graphical convention is adopted from now on for illustrating T-joins. We note that, if we are given a graph $G = (V, E, \psi)$ and $T = V$, then a T-join of $\langle G, T \rangle$ that meets each vertex exactly once is known as *perfect matching*—a concept of fundamental importance in graph theory and combinatorial optimization [13, 97].

We now consider how to generalize the notion of T-join to marked hypergraphs. Let $H = (V, E, \psi_H)$ be a marked hypergraph with $T \subseteq V$ of even cardinality, and $G = (V \cup E, F, \psi_G)$ be its underlying bipartite graph. A graph T-join of $\langle G, T \rangle$ is a set $F_1 \subseteq F$ of edges that meets each vertex of T in G an odd number of times and that meets each vertex of $(V \cup E) - T$ in G an even number of times. A *hypergraph T-join* of $\langle H, T \rangle$ is defined to be $E \cap V(G(F_1))$, i.e., the subset of edges in H that are "used" by a T-join in the underlying graph of H. A T-join in the underlying graph is called the *underlying graph T-join* of a hypergraph T-join. For example, in the marked hypergraph in Fig. 3.1, all hypergraph T-joins are $\{e_1^\star\}$, $\{e_2^\star\}$, $\{e_4^\star, e_5^\star\}$, $\{e_3^\star, e_4^\star, e_5^\star\}$, $\{e_1^\star, e_2^\star, e_4^\star, e_5^\star\}$, and $\{e_1^\star, e_2^\star, e_3^\star, e_4^\star, e_5^\star\}$.

We make a remark on our definition of hypergraph T-join. On the one hand, when $H = (V, E, \psi)$ is a graph, our definition degenerates to the known notion of graph T-join. On the other hand, a hypergraph T-join may meet a vertex in T an even number of times: For example, $\{e_3^\star, e_4^\star, e_5^\star\}$ is a hypergraph T-join, which meets $f_1^\star$ twice. It may also meet a vertex in $V - T$ an odd number of times: For example, $\{e_4^\star, e_5^\star\}$ is a hypergraph T-join that meets $f_4^\star$ once.

Now we are ready to make some observations about the relation between T-joins and matching sets in a marked hypergraph.

Proposition 3.2.1 *A T-join is a matching set.*

Proof. We first note that the subhypergraph induced by a hypergraph T-join contains the set of all vertices (except those vertices that represent hypergraph edges) in the subgraph G induced by the underlying graph T-join. Since all the negative vertices are contained in G, it is sufficient to show that G consists of a set of connected components each containing an even number of negative vertices. By the definition of graph T-join, all negative vertices have odd degree and all positive vertices have even degree. It is well known that, in any graph, the number of vertices of odd degree is even [13]; thus the number of negative vertices in each component is even. $\square$

Proposition 3.2.2 *A minimal matching set is a T-join.*

Proof. Let $H = (V, E, \psi)$ be a marked hypergraph $T \subseteq V$ of even cardinality, and let M be a minimal matching set. Consider the underlying graph G of $H(M)$. There exists a set of graph T-joins of $\langle G, T \rangle$. Each such graph T-join, in turn, defines a hypergraph T-join M'. By Proposition 3.2.1, any hypergraph T-join is a matching set. Since M is a minimal matching set, M' must be the same as M. Therefore M is a T-join. $\square$

It follows immediately from Propositions 3.2.1 and 3.2.2 that the minimum matching set problem is reduced to the following problem: Given a hypergraph $H = (V, E, \psi)$ and $T \subseteq V$ of even cardinality, find a hypergraph T-join having minimum cardinality. We will call this the *minimum hypergraph T-join* problem.

For example, consider the marked hypergraph $H^\star$ in Fig. 3.1 with $T = \{f_1^\star, f_2^\star\}$. Optimal solutions to the minimum hypergraph T-join problem are $\{e_1^\star\}$ and $\{e_2^\star\}$.

Since the minimum matching set problem in a planar marked hypergraph is NP-complete, we have the following complexity result.

Theorem 3.2.1 *The minimum hypergraph T-join problem is NP-complete if $\Delta_E > 3$, even for planar marked hypergraphs.*

3.3 On Hypergraph T-Join Algorithms

We have shown that the maximum balance problem in a planar signed hypergraph reduces to the minimum hypergraph T-join problem in its planar dual. This observation leads to two important results. First, when $\Delta_E \leq 3$, the maximum balance problem reduces to the minimum graph T-join problem in a planar graph. There are many efficient polynomial algorithms available for finding minimum graph T-joins; for examples, see Barahona [7], Edmonds and Johnson [42], and Sebö [128]. In Section 3.1, we have observed that the number of T vertices is much smaller than the total number of vertices for the problems arising from optimum layer assignment. We thus prefer a simple graph T-join algorithm due to Edmonds and Johnson [42]: First calculate the distances between all the pairs of vertices in T, then construct a complete weighted graph with T as the vertex set, with weights determined by shortest distances, and finally find the minimum weighted perfect matching in the complete weighted graph. Since the best known algorithms for minimum weighted perfect matching run in $O(|T|^3)$ time, the total cost is $O(|V||T| + |T|^3)$. Since $|T|$ is significantly smaller than $|V|$ for the optimum layer assignment problem, our algorithm practically improves the best known algorithm for optimum layer assignment, which runs in $O(|V|^{\frac{3}{2}} \log |V|)$ [87]. Also, the algorithm in [87] is difficult to implement.

Next, the observation above gives rise to pseudo-polynomial algorithms for the minimum hypergraph T-join problem, which is NP-complete for $\Delta_E > 3$. The basic idea is to introduce additional vertices for hyperedges with degree greater than 3.

The resulting graph G is similar to the underlying graph. We denote the set of additional vertices by S and call them S-vertices. Let G be a graph and $T \subseteq V(G)$ and $S \subseteq V(G)$ where $|T|$ is even and $T \cap S = \emptyset$. We would like to find a graph T-join with the minimum cost, where the cost is calculated as the number of distinct edges used by a graph T-join, except that all the edges incident with the same S-vertex are counted as 1 (corresponding to one hyperedge). We need to find all $|S|$ distinct smallest graph T-joins, and then to re-calculate the cost of each graph T-join based on its usage of vertices incident to S-vertices. The k-th smallest graph T-join can be found in a similar way to finding the k-th shortest path. Although this extension is naive, it works well for the optimum layer assignment problem, which gives rise to signed hypergraphs with only few hyperedges.

3.4 Summary

In this chapter, we have shown that the maximum balance problem in a planar signed hypergraph reduces to a generalized version of a well-known graph-theoretic problem in its planar dual. That is finding a minimum graph T-join, a problem known to be polynomially solvable. Its generalization, the minimum hypergraph T-join problem, has been solved here in pseudo-polynomial time by extending graph T-join algorithms.

Chapter 4

Modeling of Signed Hypergraphs

We consider a weighted signed hypergraph where each edge e is associated with a real number w_e. The maximum balance problem in a weighted signed hypergraph is to find a bipartition such that its cost—the total weight associated with all the unbalanced edges—is minimized. We consider the (weighted) maximum balance problem in this chapter, since it generalizes several known graph-theoretic problems as summarized in Table 4.1*.

Table 4.1: Degenerate cases of maximum balance.

$\psi = \{0,1\}$	$\Delta_E > 2$	$\Delta_E = 2$
w_e real	hypergraph partitioning	max cut
$w_e \geq 0$	netlist partitioning	min cut

When $\psi \in \{0,1\}$, the maximum balance problem degenerates to the hypergraph bipartitioning problem [12]. Further, if $w_e \geq 0$ for all edges e, the problem is known as netlist partitioning, an important problem in VLSI design [45, 150]. When $\psi \in \{0,1\}$

*Some practical motivations for considering the weighted version of the problem will be presented in Chapters 8 and 9.

and $\Delta_E = 2$, the maximum balance problem degenerates to the maximum cut (or max cut) problem [141]. Further, if $w(e) \geq 0$ for all edges e, the problem is traditionally called the minimum cut (or min cut) problem [64].

One natural question is whether it is possible to represent the maximum balance problem in a general signed hypergraph by one of the known graph-theoretic problems above. If this were the case, then we could make use of a rich body of known theoretic and algorithmic results, and even some existing software packages.

In the previous chapter, we have developed a polynomial-time algorithm for the maximum balance problem in planar signed graphs. Noticing that the case of $\Delta_E = 3$ can be reduced to the case of $\Delta_E = 2$, an interesting question is whether there exists a similar reduction for a signed hypergraph with $\Delta_E > 3$. Such a reduction does not exist unless $P = NP$, since we have shown that the maximum balance problem in planar signed hypergraphs is NP-complete for $\Delta_E > 3$. As a consequence, the best we can expect are approximate reductions. One motivation of this chapter is to look for a good approximate reduction that gives rise to efficient approximation algorithms.

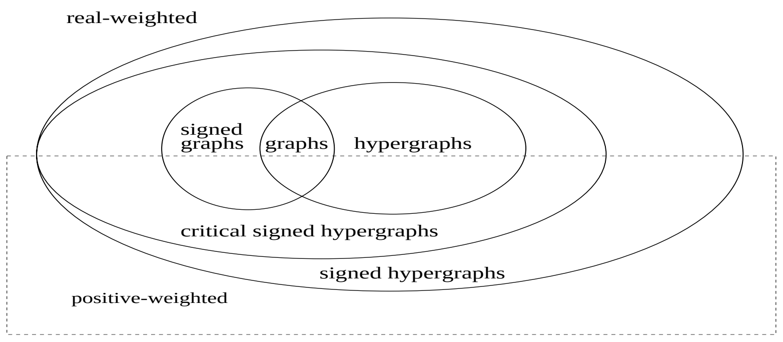


Figure 4.1: Structure hierarchy of signed hypergraphs.

In general, we are interested here in the expressive power of signed hypergraphs. We have encountered four types of graphs, namely, signed hypergraphs, hypergraphs, signed graphs, and graphs. The introduction of signed hypergraphs, hypergraphs, and signed graphs provides a more efficient way for representing and solving certain classes of practical problems. The structure hierarchy of these "graphs" is illustrated in Fig. 4.1. (Critical signed hypergraph are defined in Section 4.2.) The question is whether we really need all of them from the theoretical point of view. The study in this chapter is thus partly inspired by, and also parallels, the researches by theoretical computer scientists on the computational power of various computing machines, such as finite state automata, push-down automata, Turing machines [63]. The property of signed hypergraphs that of interest to us here is the bipartition that has the minimal cost. Since bipartitions are often called cuts in graph theory, the goal of this chapter is to study the *cut hierarchy* of signed hypergraphs.

4.1 Notion of Min-Cut Modeling

Let $H = (V, E, \psi)$ be a signed hypergraph. Let $\pi = (V^+, V^-)$ be a cut (bipartition) of V. We use $cost(\pi, H)$ to denote the cost of cut π, i.e., the number of unbalanced edges. If there is a weight associated with each edge in H, then $cost(\pi, H)$ is the total weight associated with all the unbalanced edges of π.

A *restriction* of π to V_a, $V_a \subseteq V(H)$, denoted by $\pi \downarrow V_a$, is obtained from π by removing those vertices not belonging to V_a. For example, let $\pi = (\{v_1, v_2, v_4\}, \{v_3\})$, $V_a = \{v_1, v_2, v_3\}$, and $V_b = \{v_1, v_2, v_4\}$; then $\pi \downarrow V_a = (\{v_1, v_2\}, \{v_3\})$, $\pi \downarrow V_b = (\{v_1, v_2, v_4\}, \emptyset\})$.

Let $H_1 = (V_1, E_1, \psi_1)$ be a subhypergraph of H. Let $H_2 = (V_2, E_2, \psi_2)$ be a signed hypergraph such that $V_2 \supseteq V_1$. The *replacement* of H_1 in H by H_2 is a signed hypergraph H' obtained from H by removing all the edges in E_1 and inserting all the edges in E_2. Figure 4.2 illustrates an example of replacement. For every cut π of H, there exists a set of cuts in H' whose restrictions are π.

Let H_1 and H_2 be two signed hypergraphs with $V(H_2) \supseteq V(H_1)$. Let H be a

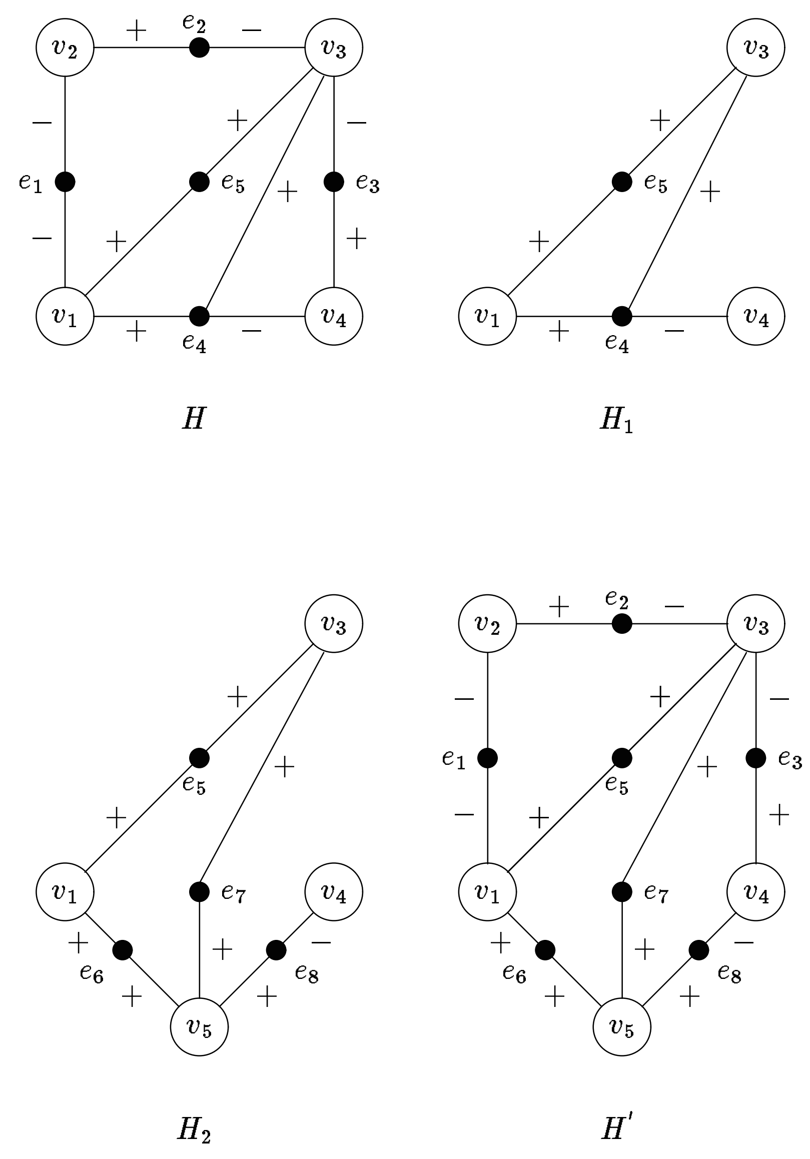


Figure 4.2: Illustration of replacement.

signed hypergraph that contains H_1 as a subhypergraph. Let H' be the replacement of H_1 in H by H_2. Then signed hypergraph H_2 is said to be a *min-cut* model of H_1 if, for every signed hypergraph H and every pair of cuts π_1 and π_2 of H,

$$cost(\pi_1, H) < cost(\pi_2, H) \quad \text{implies}$$
$$min\{cost(\pi_1', H') | \pi_1' \downarrow V = \pi_1\} < min\{cost(\pi_2', H') | \pi_2' \downarrow V = \pi_2\},$$

and

$$cost(\pi_1, H) = cost(\pi_2, H) \quad \text{implies}$$
$$min\{cost(\pi_1', H') | \pi_1' \downarrow V = \pi_1\} = min\{cost(\pi_2', H') | \pi_2' \downarrow V = \pi_2\}.$$

In particular, if $V_2 = V_1$, then H_2 is said to be a *cut model* of H_1. Vertices in V_1 are called *active* vertices, and those in $V_2 - V_1$ are *dummy* vertices.

Table 4.2: Cuts in a signed hypergraph H.

cut	unbalanced edges	cost
$(\{v_1\}, \{v_2, v_3, v_4\})$	e_1, e_2, e_3, e_4, e_5	5
$(\{v_2\}, \{v_1, v_3, v_4\})$	e_1, e_3, e_4	3
$(\{v_3\}, \{v_1, v_2, v_4\})$	e_4, e_5	2
$(\{v_4\}, \{v_1, v_2, v_3\})$	e_2	1
$(\{v_1, v_2\}, \{v_3, v_4\})$	e_3, e_4, e_5	3
$(\{v_1, v_3\}, \{v_2, v_4\})$	e_1	1
$(\{v_1, v_4\}, \{v_2, v_3\})$	e_1, e_2, e_4, e_5	4
$(\{v_1, v_2, v_3, v_4\}, \emptyset)$	e_2, e_3, e_4	3

For example, for the signed hypergraph H in Fig. 4.2, there are 8 cuts as listed in Table 4.2. For the signed hypergraph H' in Fig. 4.2, there are 16 cuts as listed in Table 4.3. In Table 4.3, the second, the third, and the fourth columns, list all 16 cuts in H', their unbalanced edges, and their costs, respectively. The first column gives the cuts restricted to $V(H)$. The fifth column lists the min-cost associated with all the cuts that have the same restrictions. Clearly, the min-cost associated with each restricted cut has the same cost as the original cut in Table 4.3. Therefore H_2 is a min-cut model of H_1, where vertex v_5 is a dummy vertex.

Table 4.3: Cuts in a signed hypergraph H'.

cut restricted to $V(H)$	cut in H'	unbalanced edges	cost	min-cost
$(\{v_1\},\{v_2,v_3,v_4\})$	$(\{v_1\},\{v_2,v_3,v_4,v_5\})$	e_1,e_2,e_3,e_5,e_6,e_8	6	5
	$(\{v_1,v_5\},\{v_2,v_3,v_4\})$	e_1,e_2,e_3,e_5,e_7	5	
$(\{v_2\},\{v_1,v_3,v_4\})$	$(\{v_2\},\{v_1,v_3,v_4,v_5\})$	e_1,e_3,e_8	3	3
	$(\{v_2,v_5\},\{v_1,v_3,v_4\})$	e_1,e_3,e_6,e_7	4	
$(\{v_3\},\{v_1,v_2,v_4\})$	$(\{v_3\},\{v_1,v_2,v_4,v_5\})$	e_5,e_7,e_8	3	2
	$(\{v_3,v_5\},\{v_1,v_2,v_4\})$	e_5,e_6	2	
$(\{v_4\},\{v_1,v_2,v_3\})$	$(\{v_4\},\{v_1,v_2,v_3,v_5\})$	e_2	1	1
	$(\{v_4,v_5\},\{v_1,v_2,v_3\})$	e_2,e_6,e_7,e_8	4	
$(\{v_1,v_2\},\{v_3,v_4\})$	$(\{v_1,v_2\},\{v_3,v_4,v_5\})$	e_3,e_5,e_6,e_8	4	3
	$(\{v_1,v_2,v_5\},\{v_3,v_4\})$	e_3,e_5,e_7	3	
$(\{v_1,v_3\},\{v_2,v_4\})$	$(\{v_1,v_3\},\{v_2,v_4,v_5\})$	e_1,e_6,e_7,e_8	4	1
	$(\{v_1,v_3,v_5\},\{v_2,v_4\})$	e_1	1	
$(\{v_1,v_4\},\{v_2,v_3\})$	$(\{v_1,v_4\},\{v_2,v_3,v_5\})$	e_1,e_2,e_5,e_6	4	4
	$(\{v_1,v_4,v_5\},\{v_2,v_3\})$	e_1,e_2,e_5,e_7,e_8	5	
$(\{v_1,v_2,v_3,v_4\},\emptyset)$	$(\{v_1,v_2,v_3,v_4\},\{v_5\})$	e_2,e_3,e_6,e_7	4	3
	$(\{v_1,v_2,v_3,v_4,v_5\},\emptyset)$	e_2,e_3,e_8	3	

Figure 4.3 gives an example of a cut model. Unmarked edges are assumed to have weight 1. It can be verified that any cut in H has the same cost as the corresponding

cut in H'. For example, for the cut $(\{v_1\}, \{v_2, v_3, v_4\})$, the cost in H is 5 as shown in Table 4.2. In H', the unbalanced edges are e_1, e_2, e_3, e_7 and e_8 for a total cost of $1 + 1 + 1 + \frac{3}{2} + \frac{1}{2} = 5$.

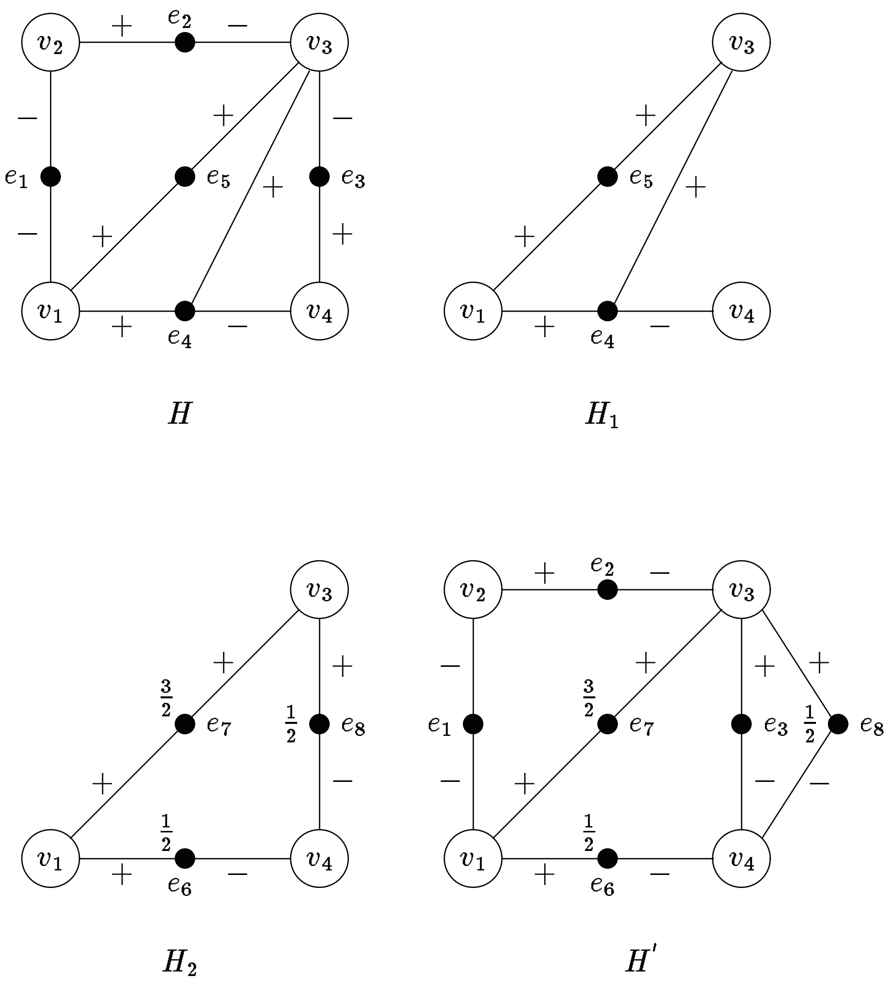


Figure 4.3: Illustration of a cut-model.

The following proposition follows immediately from the definition of min-cut model:

Proposition 4.1.1 *Let H_1 and H_2 be two signed hypergraphs with $V(H_2) \supseteq V(H_1)$. Then H_2 is a min-cut model of H_1 if and only if, for every pair of cuts π_1 and π_2 in*

H_1,

$$min\{cost(\pi_1', H_2)|\pi' \downarrow V(H_1) = \pi_1\} - min\{cost(\pi_2', H_2)|\pi' \downarrow V(H_1) = \pi_2\}$$
$$= cost(\pi_1, H_1) - cost(\pi_2, H_1).$$

Finding the min-cut model for a specific graph is not of much interest. More interesting is an algorithm that takes H_1 as the input and outputs the min-cut model H_2. Thus we can speak of asymptotic growth of min-cut modeling in the same way as we do for algorithms [3, 33]. In this sense, we say that H_2 is a *polynomial* min-cut model of H_1, if H_2 is a min-cut model of H_1 and the size of H_2 is polynomial in the size of H_1.

4.2 Modeling of Signed Hypergraphs

In this section, we first show that, for any signed hypergraph, there exists a real-weighted hypergraph as its cut model. This construction may be exponential; the question whether there exists a polynomial cut model remains open. We then present a simple construction of a polynomial cut model for a special class of signed hypergraphs, called "critical". For a special class of "critical" signed hypergraphs—(unsigned) hypergraphs, we describe a construction that gives rise to real-weighted graphs as polynomial min-cut models.

Theorem 4.2.1 *For any signed hypergraph, there exists a real-weighted hypergraph as its cut model.*

Proof. It is sufficient to show that there exists a cut model for any edge in a signed hypergraph. Let H_1 be a signed hypergraph consisting of only edge e with a set V of k vertices. We may assume that $k \geq 3$, since the case of $k < 3$ is trivial. Our cut model will be a real-weighted hypergraph H_2; we will assume that H_2 contains all the edges that correspond to subsets of V with two or more vertices. The total number of edges in H_2 is thus

$$\sum_{i=2}^{k} C_k^i = 2^k - k - 1,$$

where C_k^i denotes the number of distinct combinations of i elements chosen from a set of k elements. Associated with each edge in H_2 is a real weight whose value is to be determined; hence there are $2^k - k - 1$ variables in total. By Proposition 4.1.1, if we can find a weighting such that the difference between the costs of any two cuts in H_2 is the same as the difference between the costs of the two corresponding cuts in H_1, then there exists a cut model. Since the total number of possible cuts on H_2 is 2^{k-1}, there are $2^{k-1} - 1$ equations to be satisfied. Because the number of variables is always greater than the number of equations for $k \geq 3$, such a weighting is possible. The theorem is thus proved. □

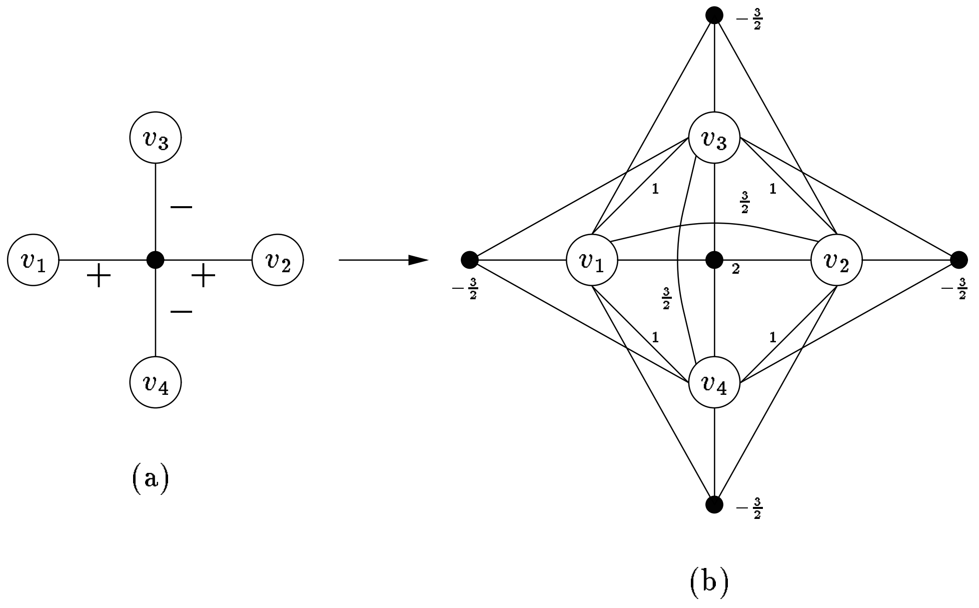


Figure 4.4: The cut model of an edge with degree four.

To illustrate the proof used for Theorem 4.2.1, we consider cut modeling of a signed hyperedge with degree four ($k = 4$). We assume its cut model as illustrated in Fig. 4.4(b): there are $2^k - k - 1 = 11$ edges, each joining a subset of two or more vertices. Clearly, if we choose edge weights in such a way that bipartition $(\{v_1, v_2\}, \{v_3, v_4\})$ has cost zero, and all the others have cost one, then the conditions in Proposition 4.1.1 are satisfied. We thus have a system of $2^{k-1} = 8$ equations, as listed below:

$$\begin{array}{c} (\{v_1\},\{v_2,v_3,v_4\}) \\ (\{v_2\},\{v_1,v_3,v_4\}) \\ (\{v_3\},\{v_1,v_2,v_4\}) \\ (\{v_4\},\{v_1,v_2,v_3\}) \\ (\{v_1,v_2\},\{v_3,v_4\}) \\ (\{v_1,v_3\},\{v_2,v_4\}) \\ (\{v_1,v_4\},\{v_2,v_3\}) \\ (\{v_1,v_2,v_3,v_4\},\emptyset) \end{array} \begin{pmatrix} 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 & 1 & 0 & 1 & 1 & 0 & 1 & 1 \\ 0 & 1 & 0 & 0 & 1 & 1 & 1 & 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 0 & 1 & 1 & 0 & 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 & 1 & 0 & 1 & 1 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 & 0 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 \end{pmatrix} \begin{pmatrix} w_{12} \\ w_{13} \\ w_{14} \\ w_{23} \\ w_{24} \\ w_{34} \\ w_{123} \\ w_{124} \\ w_{134} \\ w_{234} \\ w_{1234} \end{pmatrix} = \begin{pmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 0 \\ 1 \\ 1 \\ 1 \end{pmatrix}.$$

One can verify that the following weighting is a solution:

$$w_{12} = w_{34} = \frac{3}{2}, \quad w_{13} = w_{14} = w_{23} = w_{24} = 1,$$

$$w_{123} = w_{124} = w_{134} = w_{234} = -\frac{3}{2}, \qquad w_{1234} = 2.$$

The construction above may require an exponential number of edges with nonzero weights. We do not know whether there exists a solution with $O(k)$ nonzeros to the set of $(2^{k-1} - 1)$ equations needed in the construction. However, for a class of signed hypergraphs, called "critical", we are able to give a polynomial cut model.

An edge e is said to be *critical* if there exists at most one edge-vertex incidence that is different from all the other incidences for this edge. A signed hypergraph that consists of only critical edges is called a *critical signed hypergraph.*

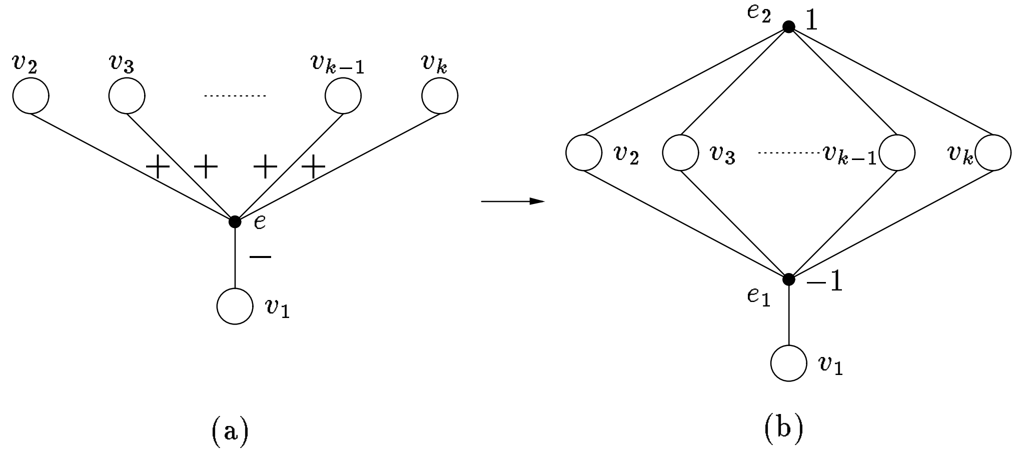


Figure 4.5: Cut modeling of a critical edge.

Any critical edge with only positive or negative edge-vertex incidences is replaced by an (unsigned) hyperedge with the same weight. The cut model for the other case of critical edges is illustrated in Fig. 4.5. It uses one edge weighted -1 to join all the vertices, and one edge weighted 1 to join the vertices with the same sign. Clearly, in Fig. 4.5(a), the cut $(\{v_1\}, \{v_2, ..., v_k\})$ has cost 0, and all other cuts have cost 1. In Fig. 4.5(b), the cut $(\{v_1\}, \{v_2, ..., v_k\})$ has cost -1, and all other cuts have cost 0. Therefore, by Proposition 4.1.1, Fig. 4.5(a) and Fig. 4.5(b) are cut models of each other. Further, these cut models are polynomial.

Any signed hypergraph with $\Delta_E = 3$ is critical; it can be replaced by a real-weighted hypergraph cut model with size polynomial in terms of the original signed hypergraph. A special case is the class of signed graphs. As a result, signed graphs do not have more expressive power than real-weighted graphs under the cut model.

Conversely, for any real-weighted hypergraph, we can construct a positive-weighted critical signed hypergraph as its cut model: for a positive-weighted edge, we replace it by an edge whose edge-vertex incidences are the same (either 1 or -1); for a negative-weighted edge, we use the reverse of the construction illustrated in Fig. 4.5. Each time a negative-weighted edge with degree k is replaced by a signed edge with degree k and another negative-weighted edge with degree $k-1$, as illustrated in Fig. 4.6. It can be verified that

- in Fig. 4.6(a), the cut that partitions all the vertices into the same block has cost 0, all other cuts have cost -1.
- in Fig. 4.6(b), the cut that partitions all the vertices into the same block has cost 1, all other cuts have cost 0.

Therefore, by Proposition 4.1.1, Fig. 4.6(a) and Fig. 4.6(b) are cut models of each other. We then use this construction for the edge of degree $k-1$ in Fig. 4.5(b). Recursive applications of the construction k times give rise to a critical signed hypergraph that is a cut model of the original negative-weighted edge. Since there are k edges in the resulting critical signed hypergraph, the construction above yields a polynomial cut model.

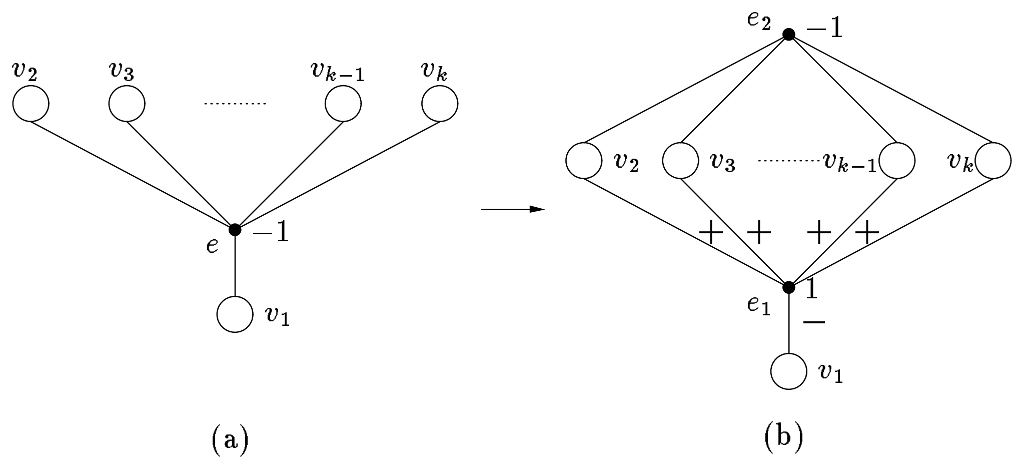


Figure 4.6: Cut modeling of a negative-weighted edge.

As the result of the two constructions above, we have the following result:

Theorem 4.2.2 *Real-weighted hypergraphs and positive-weighted critical signed hypergraphs are polynomial cut models of each other.*

Further, by Theorems 4.2.1 and and 4.2.2, we have the following corollary:

Corollary 4.2.1 *For any signed hypergraph, there exists a positive-weighted critical signed hypergraph as its cut model.*

To complete our discussion over the relation between signed hypergraphs and critical signed hypergraphs, we introduce an operation of flipping. For an edge (a vertex) in a signed hypergraph, a *flipping* operation switches all the positive incidences with all the negative ones for that edge (vertex). Figure 4.7 illustrates an example of edge flipping. An interesting property of flipping is that the maximum balance problem is invariant under flipping.

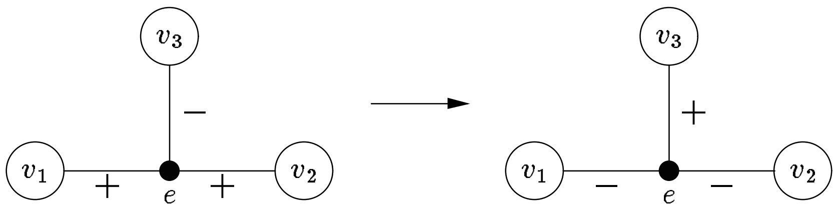


Figure 4.7: Illustration of edge flipping.

Proposition 4.2.1 *There exist signed hypergraphs that are not transformable into critical under flipping.*

Proof. We prove this by an example: Consider a signed hypergraph H with 4 vertices and 2^4 edges—all the edges with degree 4, i.e., all the edges with no 0's and any choice of ± 1. Clearly H contains non-critical edges. Edge flipping does not change whether an edge is critical. Vertex flipping leaves H invariant. $\square$

To conclude this section, we describe a construction that leads to real-weighted graphs as polynomial min-cut models for hypergraphs. The construction is due to

Ihler, Wagner, and Wagner [66], and is illustrated in Fig. 4.8 for a hyperedge of degree k ($k = 4$). The active vertices are v_1 to v_k. The dummy vertices are p_1 to p_k and n_1 to n_k. Edges from v_i to n_i are weighted $N = -(k-2)/k$; edges from v_i to p_i are weighted $P = 1$. All other edges have a small positive weight $1/k$. It can be verified that

- the min cut that partitions all active vertices into one block also partitions all dummy vertices into the same block; thus it has cost 0,
- every other min cut (with respect to active vertices) *always* unbalances all the edges weighted N, balances all the edges weighted P, and unbalances $k(k-1)$ edges weighted p. Therefore the cost of each min cut is $kN + k(k-1)p = 1$.

By Proposition 4.1.1, this construction gives rise to a polynomial min-cut model.

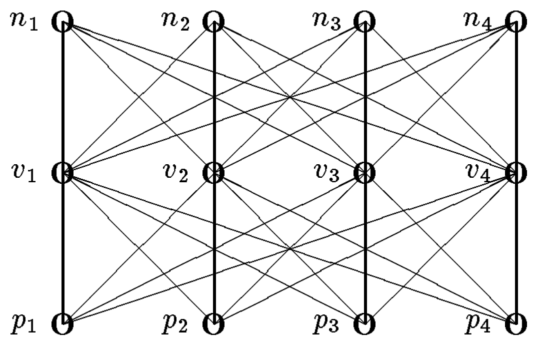


Figure 4.8: Min-cut modeling of a hyperedge by a real-weighted graph.

4.3 Min-Cut Model with Positive Weights

In this section, we consider min-cut modeling in which only positive weights are allowed. We show that there is no positive-weighted hypergraph as a min-cut model for a signed hypergraph and that there is no positive-weighted graph as a min-cut model for a hypergraph. We then study the problem of modeling a hyperedge by a positive-weighted graph, where weights are chosen to minimize the approximation error. We prove that a complete graph (called clique) is indeed the best graph structure for a min-cut model of a hyperedge. This settles a conjecture by Lengauer [66, 91]. We then compare several schemes for choosing edge weights.

Theorem 4.3.1 *There does not exist a positive-weighted hypergraph as a min-cut model for a signed hypergraph.*

Proof. It is sufficient to show that there does not exist a positive-weighted graph as a min-cut model for a signed graph. By Theorem 4.2.1, for any signed graph there exists a real-weighted graph as its cut model. So we only need to show there is no positive-weighted graph as a min-cut model for a real-weighted graph. We consider an edge e joining v_i and v_j with weight -1. The cost of cut $(\{v_i\}, \{v_j\})$ is -1. The cost of cut $(\{v_i, v_j\}, \emptyset)$ is 0. Any positive-weighted graph G (with dummy vertices) that involves vertices v_i and v_j will induce positive costs for cuts whose restrictions are $(\{v_i\}, \{v_j\})$, and zero min-cut cost for cuts whose restrictions are $(\{v_i, v_j\}, \emptyset)$. By Proposition 4.1.1, G could not be a min-cut model of the original edge e. Thus this theorem is proved. $\square$

Theorem 4.3.2 *There exists no positive-weighted graph as a min-cut model for a hyperedge of degree k, $k > 3$. Furthermore, the best approximate min-cut model with positive weights is a clique.*

Proof. Suppose that there exists a min-cut model, which is a positive-weighted graph G with a set A of k active vertices and with a set D of an arbitrary number of dummy vertices. For each bipartition (S, T) of A, we introduce a pair (s, t) of vertices with s connecting to all the vertices in the S group and t connecting to all the vertices in the T group. In total, we need to introduce 2^k such pairs.

Because G is a graph with positive-weighted edges, the well-known max-flow/min-cut theorem applies [64]. Thus for each (s, t) pair, the minimum cut separating s from

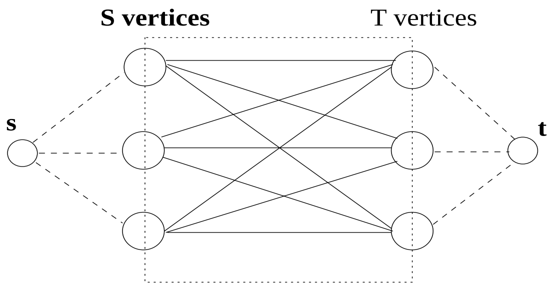


Figure 4.9: An illustration of a flow network.

t is equal to the maximum flow between s and t. Any flow network with source s and target t, where s connects a set of S vertices and t connects a set of T vertices, is equivalent to a complete bipartite flow network (ignoring (s,t)) as illustrated in Fig. 4.9. In summary, for all (s,t) pairs, the resulting flow network is exactly a complete graph (clique) involving $S \cup T = A$ vertices.

We need to determine edge weights (called capacities in network theory [64]) in such a way that the flow between each (s,t) pair is exactly one. Again, we apply the max-flow/min-cut theorem. The corresponding problem is to determine edge weights in a clique so that every cut has cost 1. One can easily verify that this is impossible for $k > 3$. Hence the theorem is proved. $\square$

Now we consider how to choose edge weights for a clique so as to minimize the approximation error. Since all edges are symmetric, they are assumed to have the same weight. We have two alternatives for choosing weights. The first one is to enable the cost of bisection of the clique to be 1 and the costs of all the other cuts to be as close to 1 (from below) as possible; this is a best *underestimation* of the original hyperedge. The solutions will provide a lower bound on the cut cost of the original hyperedge. The other one is to have the cost of separating one vertex from all the others to be 1, and the cost of all the others as close to 1 (from above) as possible; this is a best *overestimation* of the original hyperedge.

Now we consider how to measure the approximation error. We assume a to be the edge weight. Let s_i denote the approximation error, measuring how far the weight of cut i is away from 1, let s_∞ denote the maximum approximation error, let p_∞ denote the probability of having the maximum error, and let $\overline{s}$ denote the mean error. There are $m = 2^{k-1}$ cuts for a hyperedge of degree k. Then we have the following results:

- Underestimation:

$$\begin{aligned} s_\infty &= 1-(k-1)a, \\ p_\infty &= \frac{k}{2^{k-1}}, \\ \overline{s} &= \frac{1}{2^k}\sum_{i=1}^{k-1}(1-i(k-i)a)C_k^i. \end{aligned}$$

- Overestimation:

$$\begin{aligned} s_\infty &= \lceil\frac{k}{2}\rceil\lfloor\frac{k}{2}\rfloor a-1, \\ p_\infty &= \frac{1}{2^{k-1}}C_k^{\lfloor\frac{k}{2}\rfloor}, \\ \overline{s} &= \frac{1}{2^k}\sum_{i=1}^{k-1}(i(k-i)a-1)C_k^i. \end{aligned}$$

There have been three schemes for choosing a, as proposed by Stevens and vanCleemput [142], by Xiong and Kuh [157], and by Vannelli and Hadley [150]. The first two schemes were developed in the study of the via minimization problem, and were chosen intuitively.

The weighting of Stevens and vanCleemput is determined in such a way that the cost of the cut that bipartitions the clique into two groups with one group containing only one vertex is exactly the same for all k with $2 \leq k \leq 8$. The case $k \leq 8$ is considered, because it corresponds to the most commonly used grid-based routing methodologies [142]. The weights of any other cuts for $k \geq 3$ are thus larger than the weight assigned to $k = 2$. Therefore this is an overestimation of the original hyperedge.

The weighting chosen by Xiong and Kuh is $\frac{1}{k(k-1)}$, which is an underestimation of the original hyperedge.

The weight used by Vannelli and Hadley is $(\lceil\frac{k}{2}\rceil\lfloor\frac{k}{2}\rfloor)^{-1}$, where $\lfloor\frac{k}{2}\rfloor$ $(\lceil\frac{k}{2}\rceil)$ is the smallest (largest) integer greater (less) than or equal to $\frac{k}{2}$. It is also an underestimation of the original hyperedge; however, it was calculated using various linear programming models to minimize the approximation errors [150].

Table 4.4: Comparison of four weighting schemes.

	Stevens & vanCleemput				Xiong & Kuh				Vannelli & Hadley				fan model		
k	a	s_∞	p_∞	$\overline{s}$	a	s_∞	p_∞	$\overline{s}$	a	s_∞	p_∞	$\overline{s}$	s_∞	p_∞	$\overline{s}$
3	$\frac{1}{2}$	0	1	0	$\frac{1}{3}$	$\frac{1}{3}$	1	0	$\frac{1}{2}$	0	1	0	0	1	0
4	$\frac{1}{3}$	$\frac{1}{3}$	$\frac{3}{7}$	$\frac{1}{7}$	$\frac{1}{6}$	$\frac{1}{2}$	$\frac{4}{7}$	$\frac{3}{7}$	$\frac{1}{4}$	$\frac{1}{4}$	$\frac{4}{7}$	$\frac{1}{7}$	$\frac{1}{2}$	$\frac{2}{7}$	$\frac{1}{7}$
5	$\frac{1}{4}$	$\frac{1}{2}$	$\frac{2}{3}$	$\frac{1}{3}$	$\frac{1}{10}$	$\frac{3}{5}$	$\frac{1}{3}$	$\frac{7}{15}$	$\frac{1}{6}$	$\frac{1}{3}$	$\frac{1}{3}$	$\frac{1}{9}$	$\frac{3}{5}$	$\frac{2}{15}$	$\frac{1}{5}$
6	$\frac{1}{5}$	$\frac{4}{5}$	$\frac{10}{31}$	$\frac{15}{31}$	$\frac{1}{15}$	$\frac{2}{3}$	$\frac{6}{31}$	$\frac{17}{31}$	$\frac{1}{9}$	$\frac{4}{9}$	$\frac{6}{31}$	$\frac{13}{93}$	$\frac{7}{10}$	$\frac{2}{31}$	0.23
7	$\frac{1}{6}$	1	$\frac{5}{9}$	$\frac{7}{9}$	$\frac{1}{21}$	$\frac{5}{7}$	$\frac{2}{21}$	$\frac{31}{63}$	$\frac{1}{12}$	$\frac{1}{2}$	$\frac{2}{21}$	$\frac{1}{9}$	$\frac{3}{4}$	$\frac{2}{63}$	0.24
8	$\frac{1}{7}$	$\frac{9}{7}$	$\frac{35}{127}$	$\frac{129}{127}$	$\frac{1}{28}$	$\frac{3}{4}$	$\frac{8}{127}$	$\frac{63}{127}$	$\frac{1}{16}$	$\frac{9}{16}$	$\frac{8}{127}$	$\frac{15}{127}$	$\frac{15}{19}$	$\frac{2}{127}$	0.24

Table 4.4 gives a comparison among four different schemes (the last one will be described in the next section) for hyperedges of degree ≤ 8. Note that the weights determined by the model of Stevens and vanCleemput have been normalized with respect to the weight assigned to a 2-way split (420). Several observations can be drawn from this table.

1. For all schemes, $k = 3$ can be modeled exactly.

2. The weighting of Stevens and vanCleemput is better than that of Xiong and Kuh for $k \leq 5$ but is worse for $k > 5$.

3. Among three schemes, the weighting by Vannelli and Hadley is the best for any k in terms of $\bar{s}$, s_∞ and the possibility p_∞ of being s_∞.

4.4 Planar Min-Cut Model with Positive Weights

One motivation for studying the problem of modeling of signed hypergraphs by known graph-theoretic notions of hypergraphs and graphs is to develop efficient approximate algorithms. In particular, we are interested in the maximum balance problem in planar signed hypergraphs. The use of the clique as the min-cut model for a hyperedge with degree exceeding three will lead to nonplanar graphs. This is not desirable, since the maximum balance problem in a planar signed graph can be solved in polynomial time, whereas the maximum balance problem in a general signed graph is NP-complete. In this section, we study planarity preserving min-cut modeling.

The graph structure we choose to model a hyperedge is called a *fan graph*. In Fig. 4.10, we give the fan graphs used to model a hyperedge with k end-vertices for $k \leq 8$. The edge weights are determined by mathematical programming to minimize the estimation error. One can see from the table that the maximum error by the fan model is close to that of Xiong and Kuh, but the possibility of being such a worst case is lower. Moreover, the mean error is less than that of Xiong and Kuh, as well as that of Stevens and vanCleemput.

As an example, mathematical programming formulation for modeling a hyperedge with four vertices is given below:

$$\textit{minimize} \qquad f = \sum_{l=1}^{7}(s_l - s)^2 + \sum_{l=1}^{7} s_l$$

subject to

$$a_{12} + a_{13} + a_{14} + s_1 \qquad\qquad = 1$$

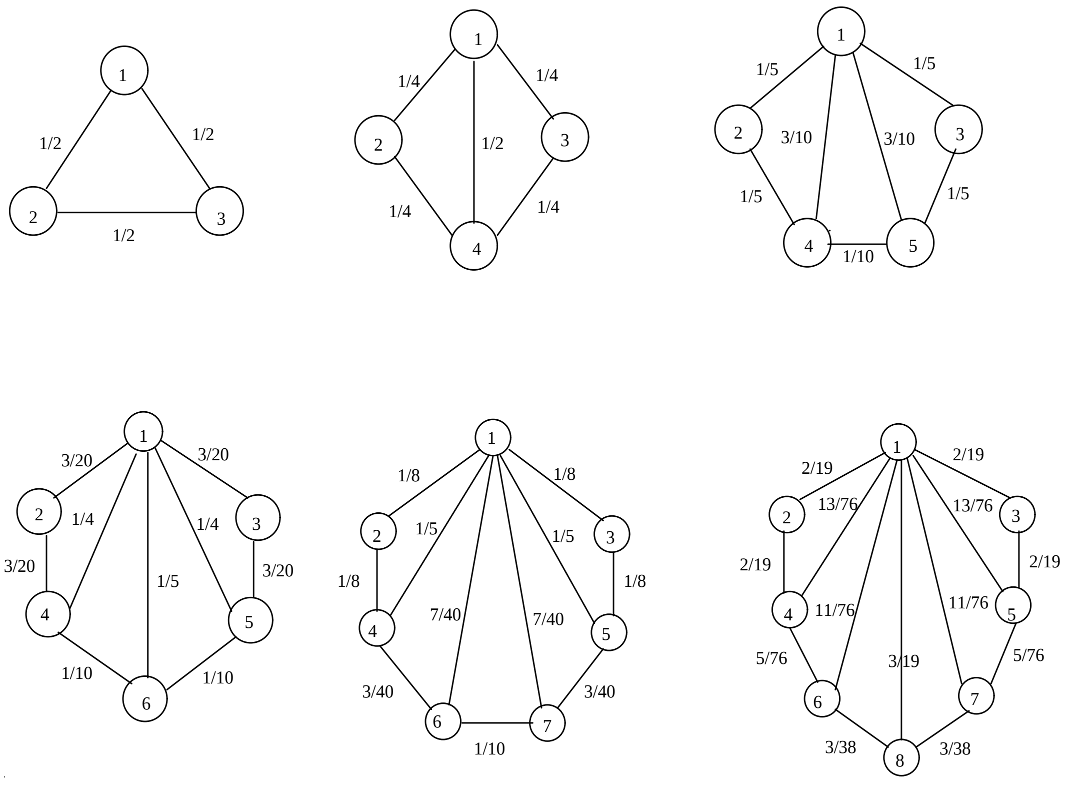


Figure 4.10: Planar graph approximations of hyperedges with degree $k \leq 8$.

$$\begin{aligned}
a_{12} + a_{24} + s_2 &= 1\\
a_{13} + a_{34} + s_3 &= 1\\
a_{14} + a_{24} + a_{34} + s_4 &= 1\\
a_{13} + a_{14} + a_{24} + s_5 &= 1\\
a_{12} + a_{14} + a_{34} + s_6 &= 1\\
a_{12} + a_{24} + a_{13} + a_{34} + s_7 &= 1\\
s_1 + s_2 + s_3 + s_4 + s_5 + s_6 + s_7 - 7s &= 0
\end{aligned}$$

The following solution to the problem above is obtained by using MINOS, a mathematical programming package [105]:

$$a_{12} = a_{13} = a_{24} = a_{34} = \frac{1}{4}, \quad a_{14} = \frac{1}{2}$$
$$s_1 = s_4 = s_5 = s_6 = s_7 = 0, \quad s_2 = s_3 = \frac{1}{2}$$

4.5 Summary and Open Problems

Our contributions in this chapter are twofold. On the theoretical side, our results lead to the *cut hierarchy* of Fig. 4.11. Two classes of "graphs" are cut-equivalent if they are cut models of each other. One class A of "graphs" *contains* another class B, if "graphs" in class A are cut models of "graphs" in class B. If dummy vertices are allowed, then we have the "min-cut" hierarchy as illustrated in Fig. 4.12.

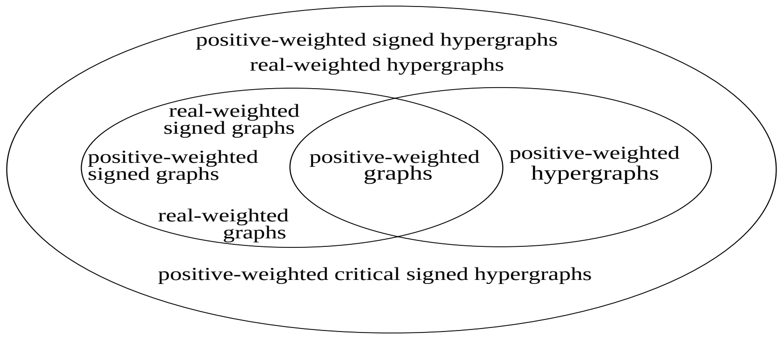

positive-weighted signed hypergraphs
real-weighted hypergraphs
real-weighted signed graphs
positive-weighted signed graphs
positive-weighted graphs
positive-weighted hypergraphs
real-weighted graphs
positive-weighted critical signed hypergraphs

Figure 4.11: Cut hierarchy of signed hypergraphs.

On the practical side, we proved that the clique is indeed the best approximate min-cut model with positive weights for a hyperedge even if dummy vertices are allowed. This settles a conjecture by Lengauer [91, 66], which further confirms theoretically the effectiveness of the use of the clique model in netlist partitioning, as done by Hadley, Mark and Vannelli [56].

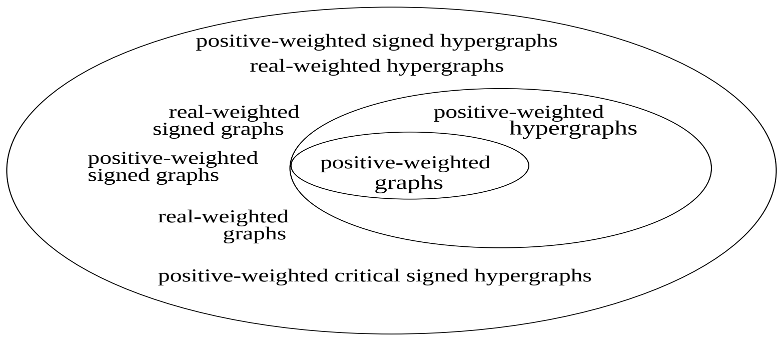

positive-weighted signed hypergraphs
real-weighted hypergraphs
real-weighted signed graphs
positive-weighted hypergraphs
positive-weighted signed graphs
positive-weighted graphs
real-weighted graphs
positive-weighted critical signed hypergraphs

Figure 4.12: Min-cut hierarchy of signed hypergraphs.

A planarity preserving min-cut model was presented for the first time. Because its weights are determined by mathematical programming so as to minimize the approximation error, the proposed planarity preserving min-cut model gives rise to good polynomial time approximation algorithms for the maximum balance problem in a planar signed hypergraph. This result is of practical relevance, since hyperedges with $\Delta_E > 4$ are only few in the problem instances resulting from optimal layer assignment [23].

On the other hand, the results obtained in this chapter indicate that there is generally no good way to model a signed hypergraph by graphs with positive weights. This provides one reason why we shall develop robust heuristics that can work directly on signed hypergraphs, which will be the scope of the next chapter. The same observation has been made on network partitioning [91, 150].

There are several open problems. The first one is the existence of a real-weighted hypergraph as a polynomial cut model for a signed hypergraph (Section 4.2). We conjecture that such a polynomial model does not exist. That means, the cut model of a signed hypergraph might have to be exponential. Therefore, the introduction of signed hypergraphs might lead to *exponential speedup of efficiency* over existing "graphs", which have significant impact on algorithm design. The second problem is the existence of an analytic expression for the problem of planar min-cut modeling. In addition, it is also interesting to see if the fan graph is the best planar graph structure for modeling a hyperedge.

Chapter 5

A Local Search Heuristic for Maximum Balance

In Chapter 3, we presented algorithms for the maximum balance problem in planar signed hypergraphs, which, in turn, solve the optimum layer assignment problem, since the maximum balance problem in planar signed hypergraphs is a precise graph-theoretic formulation of the optimum layer assignment problem. In this chapter, we consider the case of general signed hypergraphs. This study is motivated by several considerations. First, a signed hypergraph is not necessarily planar. Second, with the ever increasing demand in high-performance circuit design, it is desirable to consider timing constraints during layer assignment; such constraints may give rise to non-planar signed hypergraphs [23]. Finally, the study here provides a basis to the development of efficient heuristics for the minimum covering problem, as explored in the next chapter.

Recall from Chapter 2 that the maximum balance problem is NP-complete; it is therefore very unlikely that there exist polynomial-time algorithms for finding *optimum* solutions. One paradigm to attack NP-complete problems is *by approximation*, i.e., approximating a general case of the problem by special cases that can be solved polynomially. This usually leads to guaranteed efficient algorithms for finding *guaranteed good* solutions, though not necessarily optimal. Indeed, the result of approximation of signed hypergraphs by planar signed graphs in Section 4.4, combined with the algorithm developed in Chapter 3, provides an efficient approximation algorithm for the maximum balance problem in planar signed hypergraphs. However, it is not easy to see how to apply this paradigm to the general case, since the problem in general signed graphs is NP-complete also. Therefore, this chapter focuses on a more generally applicable paradigm—local search.

Local search is based on what is perhaps the oldest optimization method—trial and error. In Section 5.1, we describe the paradigm of local search and its adaptation to our problem. In Section 5.2, we work through an example, to illustrate further the basic idea, and to examine where computations are consumed. We then show in Section 5.3 how to speed up those computations by using suitable data structures and incremental techniques. In Section 5.4, we present a technique that can help move out of local optima, based on the property of signed hypergraphs. Some theoretical aspects of local search are discussed in Section 5.5.

5.1 Local Search Paradigm and ENCORE Neighborhood

Local search is a general paradigm for solving optimization problems. An optimization problem P consists of

1. a set of possible inputs,
2. a group of constraints which specifies the property that the solution must satisfy. The set of possible inputs that satisfies all the constraints is usually called the feasible solution set, denoted by F.
3. a cost mapping: $c : F \to R$, which associates with each feasible solution, $f \in F$, a real cost. (Here R is the set of real numbers.)

The maximization problem is to find an $f \in F$ for which

$$c(f) \geq c(y) \quad \text{for all } y \in F.$$

Such a solution f is called a *globally optimal solution*. Given a *neighborhood mapping*, $\mathrm{N} : F \rightarrow 2^F$, the set $\mathrm{N}(f)$, $f \in \mathrm{N}(f)$, is the *neighborhood* of f. The solution f is a *local optimum with respect to N* if

$$c(f) \geq c(y) \quad \text{for all } y \in \mathrm{N}(f).$$

Alternatively, given point $f \in F$, we define $gain(f) = c(t) - c(f)$ where $t \in \mathrm{N}(f)$ such that

$$c(t) \geq c(y) \quad \text{for all } y \in \mathrm{N}(f).$$

We denote such a point t by $next(f)$. A solution $f \in F$ is a *local optimum with respect to N* if

$$gain(f) \leq 0.$$

The paradigm of local search for solving the maximization problem P is now shown in Fig. 5.1. We start at some initial feasible solution $f \in F$ and then search for a best solution $next(f)$ in its neighborhood. As long as a best neighborhood *improves* the current solution, i.e., $gain(f) > 0$, we adopt it and repeat the neighborhood search from the new solution; we stop when we reach a local optimum. The process of searching for a new solution from a current solution is called one *pass* of local search.

We may rephrase an optimization problem with a superimposed neighborhood structure as a *local search optimization* problem stated below: given an input instance x, find a locally optimal solution. We refer to the use of local search paradigm in Fig. 5.1 to a local search optimization problem as *standard local search algorithm.*

In the rest of this section, we describe how to apply this paradigm to the maximum balance problem. The problem input is a signed hypergraph H. Any arbitrary

```
LOCAL_SEARCH
1   $f$ = some initial starting point in $F$
2   **while** $gain(f) > 0$
3       **do**  $f = next(f)$
4   **return** $f$
```

Figure 5.1: Paradigm of local search.

bipartition π of H is a feasible solution. If we denote the number of vertices in H by n, then the feasible set will the the set of all the 2^n bipartitions. The cost here is the number of balanced edges. We will show in Chapter 8 how to obtain a feasible solution when there exist constraints imposed on the maximum balance problem.

The most critical part is the choice of a neighborhood mapping N. A neighborhood should be easy to search, i.e., enable an efficient calculation of $gain(\pi)$ and $next(\pi)$, at the same time rich enough to assure the quality of the local optima obtained. A simple choice would be the set of points within Hamming distance 1. The process of searching for a best neighbor amounts to searching for a vertex which, if moved from its current group to its complementary group, will lead to maximum gain in terms of the number of balanced edges. For convenience, we call this the *Hamming neighborhood*.

We implement a more sophisticated neighborhood, called the ENCORE neighborhood: a neighboring bipartition is obtained from a current bipartition by a *sequence* of (at most n) moves; at each move, a vertex of maximum gain that has not been moved since the beginning of the sequence is chosen to move. This neighborhood generalizes, and explores a much larger neighborhood than, the simple Hamming neighborhood. Any local optimum with respect to the ENCORE neighborhood is certainly a local optimum with respect to the Hamming neighborhood; the converse does not hold, therefore, the use of the ENCORE neighborhood is expected to produce better local optima. Indeed, the ENCORE neighborhood is inspired by the success of the work of Kernighan and Lin [77], and Fiduccia and Mattheyses [45], for graph/network partitioning.

The key part is how to perform the search for $next(\pi)$ and the calculation of $gain(\pi)$ under the ENCORE neighborhood. We will look at an example first in the next section, and give a careful implementation later. We will show finally that the ENCORE neighborhood search takes only linear time, in terms of the size of the problem instance, i.e., the nonzero entries in the incidence matrix of a signed hypergraph. For simplicity, we will use ENCORE search to refer to the ENCORE neighborhood search.

5.2 ENCORE Search: An Illustrative Example

Consider the following example. Let $H = (V, E, \psi)$ where $V = \{v_1, ..., v_5\}$, $E = \{e_1, ..., e_4\}$, and ψ is described by the following matrix:

$$\psi^T = \begin{pmatrix} 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ -1 & 0 & 0 & -1 \\ 0 & 1 & -1 & -1 \end{pmatrix}.$$

Here the transpose ψ^T of the incidence matrix is used for notational convenience. Let $\mathbf{x}^{(0)} = (1,1,1,1,1)^T$ as the "seed", i.e., an initial bipartition. We want to find such a bipartition, $\mathbf{x} \in \{-1,1\}^n$, in the ENCORE neighborhood of $\mathbf{x}^{(0)}$ that balances as many edges as possible.

If we use the value 0 to denote "don't cares", then we may speak of "ternary" ({-1,0,1}) vectors. Any edge e_j is balanced by two "ternary" bipartitions: One is equal to the *jth* column of ψ, and the other is the negation of that column. We denote these bipartitions by $\mathbf{e}_j$ and $-\mathbf{e}_j$, respectively.

In order to decide which component of the present bipartition $\mathbf{x}$ should be changed to get closer to an optimal solution, we define the *direction matrix* $\mathbf{\Delta} = (\delta_{ij})_{m\times n}$, where $\delta_{ij} = x_i e_{ij}$. If x_i agrees with e_{ij}—the *ith* component of $\mathbf{e}_j$—then δ_{ij} is equal to 1; if x_i disagrees with e_{ij}, then δ_{ij} is equal to -1; finally, if s_i is not in e_j, then $e_{ij} = 0$ and δ_{ij} is also 0. Therefore the number of -1 entries in each column of Δ reflects how far $\mathbf{x}$ is from bipartition $\mathbf{e}_j$; we denote this number by d_j^+ and call it the *distance* from $\mathbf{x}$ to $\mathbf{e}_j$. Similarly, the number d_j^- of 1 entries in column j of Δ is the distance from $\mathbf{x}$ to $-\mathbf{e}_j$. Each entry in $\mathbf{\Delta}$ has the following meaning:

$$\delta_{ij} = \begin{cases} 1, & \text{if changing } x_i \text{ takes } \mathbf{x} \text{ away from } \mathbf{e}_j \text{ (closer to } -\mathbf{e}_j\text{) by 1,} \\ -1, & \text{if changing } x_i \text{ takes } \mathbf{x} \text{ closer to } \mathbf{e}_j \text{ (away from } -\mathbf{e}_j\text{) by 1,} \\ 0, & \text{otherwise.} \end{cases}$$

The shortest distance, called d_j, from $\mathbf{x}$ to a bipartition that balances edge e_j is $d_j = min\{d_j^+, d_j^-\}$. If $d_j \neq 0$, then edge e_j is not balanced by $\mathbf{x}$. We will denote by $\mathbf{d}^+$ the distance vector $(d_1^+, ..., d_m^+)$, and treat $\mathbf{d}^-$ and $\mathbf{d}$ similarly. The number c of balanced edges is the number of zero components of $\mathbf{d}$.

Initialization: The direction matrix for initial bipartition $\mathbf{x}^{(0)}$ is simply $\mathbf{\Delta}^{(0)} = \psi^{\mathbf{T}}$. The calculation of $\mathbf{d}^+$, $\mathbf{d}^-$, and $\mathbf{d}$, and c is illustrated below.

$$\overset{\mathbf{x}^{(0)}}{\begin{pmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \end{pmatrix}} \qquad \mathbf{\Delta}^{(0)} = \begin{pmatrix} 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ -1 & 0 & 0 & -1 \\ 0 & 1 & -1 & -1 \end{pmatrix}$$

$$\begin{aligned} \mathbf{d}^+ &= \begin{pmatrix} 1 & 0 & 1 & 2 \end{pmatrix} \\ \mathbf{d}^- &= \begin{pmatrix} 1 & 3 & 2 & 2 \end{pmatrix} \\ \mathbf{d} &= \begin{pmatrix} 1 & 0 & 1 & 2 \end{pmatrix} \qquad \boxed{\text{c} = 1} \end{aligned}$$

We consider which component of $\mathbf{x}$ should be changed to get as close to an optimal solution as possible. For this purpose we introduce the *gain matrix* $\mathbf{G} = (g_{ij})_{m\times n}$ where

$$g_{ij} = \begin{cases} 1, & \text{if the change of } x_i \text{ changes } e_j \text{ from unbalanced to balanced,} \\ -1, & \text{if the change of } x_i \text{ changes } e_j \text{ from balanced to unbalanced,} \\ 0, & \text{otherwise.} \end{cases}$$

Matrix $\mathbf{G}$ can be obtained by inspection of $\mathbf{d}$ and $\mathbf{\Delta}$. There are three situations: If $d_j = 0$—as in the case of d_2—then edge e_j is balanced by $\mathbf{x}$. If we change any component of $\mathbf{x}$ corresponding to a vertex joined in e_j, the number c of balanced edges will decrease by 1. Therefore $g_{12} = g_{22} = g_{52} = -1$. If $d_j = 1$—as in the case of d_3—there exists a component of $\mathbf{x}$ (here x_5) which, if changed, will cause e_j to become balanced. Therefore the corresponding entry (g_{53}) in $\mathbf{G}$ is 1. In the special case where $d_j = 1$ and $|e_j| = 2$—as in the case of e_1—both entries are equal to 1. Thus $g_{21} = g_{41} = 1$. If $d_j > 1$—as in the case of d_4—edge e_j will remain unbalanced no matter which component of $\mathbf{x}$ changes. Thus all the entries in the corresponding column (here column 4) are 0. The above calculations result in the matrix $\mathbf{G}^{(0)}$ shown below. The sum of the entries in row i of $\mathbf{G}$ is the total gain obtained by complementing x_i, which forms a *gain vector* γ. In our example, the fourth component has the largest gain; hence, we select it as the component to be changed.

$$\mathbf{G}^{(0)} = \begin{pmatrix} 0 & -1 & 0 & 0 \\ 1 & -1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & -1 & 1 & 0 \end{pmatrix} \quad \overset{\gamma^{(0)}}{\begin{pmatrix} -1 \\ 0 \\ 0 \\ 1 \\ 0 \end{pmatrix}} \quad \boxed{\text{select } x_4}$$

Move 1: We change the fourth component of $\mathbf{x}^{(0)}$, and the new bipartition is $\mathbf{x}^{(1)} = (1, 1, 1, -1, 1)^T$. This is the first *move*. The new direction matrix $\Delta^{(1)}$ is the same as $\Delta^{(0)}$ except that the $4th$ row is the negation of the original row. The fourth component of $\mathbf{x}$ is "locked" after the move, in the sense that it will not be changed again during the search for the first bipartition. Hence we do not need to check the gain entries corresponding to this component; such entries will be shaded. All of the calculations connected with Move 1 are compactly summarized below.

$$\mathbf{x}^{(1)} = \begin{pmatrix} 1 \\ 1 \\ 1 \\ -1 \\ 1 \end{pmatrix} \qquad \mathbf{\Delta}^{(1)} = \begin{pmatrix} 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 \\ 0 & 1 & -1 & -1 \end{pmatrix}$$

$$\begin{aligned} \mathbf{d}^{+} &= \begin{pmatrix} 0 & 0 & 1 & 1 \end{pmatrix} \\ \mathbf{d}^{-} &= \begin{pmatrix} 2 & 3 & 2 & 3 \end{pmatrix} \\ \mathbf{d} &= \begin{pmatrix} 0 & 0 & 1 & 1 \end{pmatrix} \qquad \boxed{\text{c} = 2} \end{aligned}$$

$$\mathbf{G}^{(1)} = \begin{pmatrix} 0 & -1 & 0 & 0 \\ -1 & -1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ & & & \\ 0 & -1 & 1 & 1 \end{pmatrix} \qquad \gamma^{(1)} = \begin{pmatrix} -1 \\ -2 \\ 0 \\ \\ 1 \end{pmatrix} \qquad \boxed{\text{select } x_5}$$

Move 2: According to $\gamma^{(1)}$, we change x_5. The computed results are illustrated below. All the entries in the new gain vector $\gamma^{(2)}$ are negative. This means that the corresponding components of $\mathbf{x}$ are either locked or, if changed, will cause more edges to be unbalanced. Our strategy, therefore, is to terminate the neighborhood search.

Thus we obtain $\mathbf{x} = (1,1,1,-1,-1)^T$, which balances edges e_1, e_3 and e_4.

$$\overset{\mathbf{x}^{(2)}}{\begin{pmatrix} 1 \\ 1 \\ 1 \\ -1 \\ -1 \end{pmatrix}} \qquad \boldsymbol{\Delta}^{(2)} = \begin{pmatrix} 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 \\ 0 & -1 & 1 & 1 \end{pmatrix}$$

$$\begin{aligned} \mathbf{d}^+ &= \begin{pmatrix} 0 & 1 & 0 & 0 \end{pmatrix} \\ \mathbf{d}^- &= \begin{pmatrix} 2 & 2 & 3 & 4 \end{pmatrix} \\ \mathbf{d} &= \begin{pmatrix} 0 & 1 & 0 & 0 \end{pmatrix} \qquad \boxed{\text{c} = 3} \end{aligned}$$

$$\mathbf{G}^{(2)} = \begin{pmatrix} 0 & 0 & -1 & -1 \\ -1 & 0 & -1 & 0 \\ 0 & 0 & 0 & -1 \\ & & & \\ & & & \end{pmatrix} \quad \overset{\gamma^{(2)}}{\begin{pmatrix} -2 \\ -2 \\ -1 \\ \\ \\ \end{pmatrix}} \qquad \boxed{\text{Stop}}$$

In the description above, the direction and gain matrices are introduced for descriptive convenience. What we really need are the distance and gain vectors. We can divide the computational cost associated with each move into three parts:

- **Task 1:** computation of the distance vectors.
- **Task 2:** calculation of the gain vector.
- **Task 3:** selection of a component in the gain vector that has the maximum value.

In the next section, we will see how this idea can be implemented so that its worst-case computational cost of ENCORE *SEARCH* is $O(p)$, where p is the number of nonzero elements in the incidence matrix.

5.3 Efficient Implementation of ENCORE Search

We now describe an efficient implementation of the basic ideas of the previous section. Efficiency is achieved by using suitable data structures for Tasks 1 and 3, and an incremental approach for Task 2, so that all three tasks are performed in $O(p)$ time.

5.3.1 Calculation of Distance Vectors

We use sparse-matrix techniques to store the incidence matrix ψ. (In other words, we use the adjacency-list data structure to represent a signed hypergraph.) We maintain an array of edges, in which each entry is a linked list of pointers to vertices in that edge. We also keep an array of vertices, in which each entry is a linked list of pointers to edges involving that vertex. These two arrays permit efficient traversal of nonzero entries by row or by column. We also keep track of the values of $\mathbf{d}^+$ and $\mathbf{d}^-$ for each edge. After changing x_i, we update the distance values as follows:

$$\begin{aligned} d_j^+ &= d_j^+(old) + \delta_{ij}, \\ d_j^- &= d_j^-(old) - \delta_{ij}. \end{aligned}$$

The number of operations needed for computing the new distances is thus $|E_i|$. The total number of operations needed to maintain distance vectors for one move is $\sum_{i=1}^{m} |E_i| = p$. The initialization of $\mathbf{d}^+$ and $\mathbf{d}^-$ takes p operations. Therefore the total cost for Task 1 is $2p$, i.e., $O(p)$.

Proposition 5.3.1 *Using the sparse-matrix data structures, the calculation of distance vectors in one pass of local search takes $O(p)$ time.*

5.3.2 Initialization of Gain Vectors

Now we consider how to calculate the gain vector for a given bipartition $\mathbf{x}$. All distances can be obtained in $O(p)$ time. Only edges with the shortest distances of 0 or 1 contribute to the gain vector; such edges are said to be $\mathbf{x}$-*sensitive*. The calculation of the gain vector checks for each edge to see if it is $\mathbf{x}$-sensitive. If so, all the vertices in that edge may be checked to determine whether to add 1, remove 1, or do nothing for that corresponding entry. In the worst case, all the edges are $\mathbf{x}$-sensitive, and all the vertices in each edge need to be checked. Therefore $O(p)$ operations are needed to build up the gain vector for a given $\mathbf{x}$.

Proposition 5.3.2 *Computing the gain vector for a given bipartition takes $O(p)$ time.*

5.3.3 Incremental Gain Updating

Knowing the gain vector for $\mathbf{x}$, we calculate the gain vector for the new $\mathbf{x}$ obtained by changing x_i. An efficient approach is to perform incremental updating, i.e., to modify only that part of the gain vector that is affected by the move.

To illustrate this idea, we refer to Move 1 in our introductory example. Both the distance and gain vectors for $\mathbf{x}^{(0)}$ are known. We change x_4, and all the nonzero entries in the fourth row of $\mathbf{\Delta}^{(0)}$ (corresponding to x_4) are negated. These entries are δ_{41} and δ_{44}. Therefore, only the distances for edges in E_4 are changed (here $E_4 = \{e_1, e_4\}$). As a consequence, only the first and the fourth columns of $\mathbf{G}$ may be changed. Since γ_i is equal to the sum of the entries in the *ith* row of $\mathbf{G}$, the new γ can be obtained from the old γ by first subtracting the first and the fourth columns of $\mathbf{G}^{(0)}$ and then adding the first and the fourth columns of $\mathbf{G}^{(1)}$. Thus, the calculation of contributions to the gain vector is required only for the edges in E_i.

Gain updating is required only when an edge in E_i is sensitive in the present or the previous moves. Due to the fact that each component is locked after the move, an edge e_j can be sensitive ($d_j = 1$ or $d_j = 0$) during the search for one bit assignment only a constant number of times. The reason is as follows: A moved component either agrees or disagrees with e_j. Consider the first case. Since the component is locked after the move, it will disagree with e_j forever. This means that d_j^+ will be greater than 0 forever. Similarly, for the second case, we can conclude that d_j^- will be greater than 0 forever. Thus the number of times that d_j is 0 is bounded. If there are two moves involving two components that originally agree with e_j, then, after the move, the two components will disagree with e_j forever, i.e., $d_j^+ > 1$. Similar reasoning shows that the number of times that d_j is 1 is also bounded. Formally, we have the following result.

Proposition 5.3.3 *For each edge, gain updating is required only a constant number of times during one pass of local search.*

A detailed case analysis reveals that three gain updatings are sufficient [136]. Since the number of updatings for each edge is bounded, we have the following proposition.

Proposition 5.3.4 *For each edge, all gain updating takes $O(p)$ time during one pass of local search.*

5.3.4 Data Structures for Gain Vectors

Now we describe a data structure, denoted by B, for the gain vector. It shall support the following operations:

- INSERT(B, i, g_i) inserts a component i with g_i in B.
- DELETE(B, i) deletes component i from B.

- UPDATE(B, i, q) updates g_i by q, i.e., lets $g_i = g_i + q$ and moves it to the appropriate place in B.
- MAX(B) returns the component of $\mathbf{x}$ with maximum gain, or NIL if B is empty.

We use a *bucket list*, as illustrated in Fig. 5.2. The range of the bucket list goes from

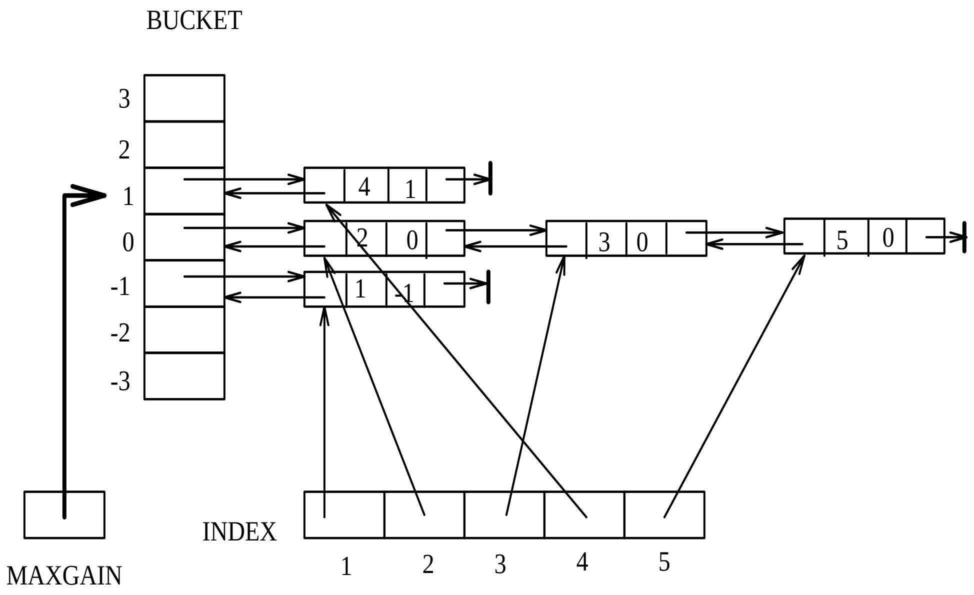


Figure 5.2: Bucket list structure (before Move 1).

$-l$ to l, where $l = max\{|E_i|, 1 \leq i \leq m\}$. The jth entry of the bucket list contains a doubly-linked list of unlocked components with gain currently equal to j. An additional array INDEX is used to maintain pointers for direct access to each component in the bucket list. Whenever a component is locked, we remove it from the bucket list and set the corresponding INDEX to *NIL*. A MAXGAIN pointer is maintained to keep track of the bucket having a component of highest gain. This pointer is updated by decrementing it whenever its bucket is found to be empty, and resetting it to a higher bucket whenever a component moves to a bucket above MAXGAIN. With the bucket-list data structure, all the operations above except MAX(B) take $O(1)$ time.

Proposition 5.3.5 *Operation* MAX *takes* $O(p)$ *time in total in one pass of local search.*

Proof. Operation MAX is performed at most n times. Since this is done by accessing pointer MAXGAIN, it is sufficient to examine how much work is needed to maintain MAXGAIN. Pointer MAXGAIN may be affected by INSERT, DELETE and UPDATE. Whenever a component moves to a bucket above MAXGAIN, MAXGAIN is simply reset to a higher bucket. Operation INSERT is invoked m times; therefore $O(p)$ time is needed for maintaining MAXGAIN due to this operation. Whenever a bucket of maximum gain is found to be empty, we need to decrement MAXGAIN until we find the next non-empty bucket. This may happen when operation DELETE is invoked, or when UPDATE is needed to decrease the gain ($q < 0$). The number of times DELETE is invoked is at most n, and each time there are at most $2e$ empty buckets, where $e = max\{|E_i|, 1 \leq i \leq m\}$. Therefore $O(p)$ time is needed for maintaining MAXGAIN due to DELETE operations. When UPDATE is invoked to decrease the gain, i.e., q is -1, at most one bucket may be found to be empty. When UPDATE is invoked to increase the gain, the current maximum gain is compared with the new gain to determine MAXGAIN. By Proposition 5.3.4, the total time needed for maintaining MAXGAIN due to UPDATE is $O(p)$. Therefore, the total time needed for maintaining MAXGAIN is $O(p)$. $\square$

5.3.5 Formal Description

ENCORE_SEARCH: Lines 1 and 2 initialize the seed $\mathbf{x}$. Line 3 invokes the procedure COMPUTE_GAIN to calculate the distances and to construct B under $\mathbf{x}$. The return value of COMPUTE_GAIN is the number of unbalanced edges. In line 4, the component of highest gain is returned with the aid of the operation MAX on B. If the gain value is positive, Lines 5 to 9 perform one move, that is, lock component i, remove the gain of component i from B, invoke the procedure UPDATE_GAIN to update B and the related distances due to the change of x_i, change x_i, and finally calculate the number u of unbalanced edges. Lines 5 to 9 repeat until either there is no positive-gain component, or all components are locked, i.e., B is empty.

```
ENCORE_SEARCH($H$)
 1    for $i \leftarrow 1$ to $m$
 2      do $x_i \leftarrow 1$
 3    $u \leftarrow$ COMPUTE_GAIN($H, \mathbf{x}$, B)
 4    while gain of MAX(B) $> 0$ and $u > 0$
 5      do  lock $x_i$
 6          DELETE(B, $i$)
 7          UPDATE_GAIN($H, i, \mathbf{x}$, B)
 8          $x_i \leftarrow -x_i$
 9          $u \leftarrow u -$ MAXGAIN
10    return x
```

COMPUTE_GAIN: Line 1 initializes the number u of unbalanced edges to n. Lines 2 and 3 initialize the temporary array γ to zero. Lines 4 to 23 form the main body for calculating the distances u and the gain vector γ. Lines 24 and 25 build B according to the calculated gain vector.

```
COMPUTE_GAIN($H$, $\mathbf{x}$, B)
1   $u \leftarrow n$
2   for $i \leftarrow 1$ to $m$
3       do $\gamma_i \leftarrow 0$
4   for each $e_j \in E$
5       do $d_j^+ \leftarrow d_j^- \leftarrow 0$
6          for each $x_i$ contained in $e_j$
7              do $\delta_{ij} \leftarrow x_i e_{ij}$
8                 if $\delta_{ij} = -1$
9                    then $d_j^+ \leftarrow d_j^+ + 1$
10                   else $d_j^- \leftarrow d_j^- + 1$
11      $d_j \leftarrow min\{d_j^+, d_j^-\}$
12      if $d_j = 0$
13         then $u \leftarrow u - 1$
14              for each $x_i$ in $e_j$
15                  do $\gamma_i \leftarrow \gamma_i - 1$
16      if $d_j = 1$
17         then if $|E_i| = 2$
18                 then for each $x_i$ in $e_j$
19                          do $\gamma_i \leftarrow \gamma_i + 1$
20                 else for each $x_i$ in $e_j$
21                          do if $x_i$ is $x_i$-sensitive
22                                then $\gamma_i \leftarrow \gamma_i + 1$
23                                     break
24  for $i \leftarrow 1$ to $m$
25     do INSERT(B, $i$, $\gamma_i$)
26  return $u$
```

```
UPDATE_GAIN($H, i, \mathbf{x}$, B)
1    for each $e_j$ in $E_i$
2      do $\delta_{ij} \leftarrow x_i e_{ij}$
3         if (d == 0 or 1) MODIFY_GAIN($-1, e_j, d_j$, B)
4         $d_j^+ \leftarrow d_j^+ + \delta_{ij}$
5         $d_j^- \leftarrow d_j^- - \delta_{ij}$
6         if (d == 0 or 1) MODIFY_GAIN($1, e_j, d_j$, B)
```

MODIFY_GAIN: Given an edge e_j, the distance d_j, and a parameter q, the procedure MODIFY_GAIN(f, e_j, d_j, B) works as follows. If $q = 1$, the edge's contribution is added to B; if $q = -1$, the contribution is removed from B.

```
MODIFY_GAIN($q, e_j, d_j$, B)
 1    if $d_j = 0$
 2      then for each $x_i$ in $e_j$
 3              do UPDATE(B, $i, -q$)
 4      else if $d_j = 1$
 5             then if $|E_i| = 2$
 6                    then for each unlocked $x_i$ in $e_j$
 7                           do UPDATE(B, $i, q$)
 8                    else for each unlocked $x_i$ in $e_j$
 9                           do if $x_i$ is $x_i$-sensitive
10                                then UPDATE(B, $i, q$)
11                                     break
```

5.3.6 Time Complexity

Theorem 5.3.1 *The running time of* ENCORE_SEARCH *is* $O(p)$.

Proof. Initialization in Lines 1 and 2 takes m time. By Proposition 5.3.2, Line 3 takes $O(p)$ time. By Proposition 5.3.5, Line 4 takes $O(p)$ time. In the worst case, Lines 5 to 9 is repeated m times. Each time the procedure UPDATE is invoked, it takes $O(|E_i|)$ operations to do lines 1, 2, 4, 5. According to Proposition 5.3.3, the number of calls to MODIFY_GAIN for edge e_j during the entire ENCORE_SEARCH is a constant, say t_1. Each call to MODIFY_GAIN for edge e_j takes $|e_j|$ operations. Therefore the number of operations required by line 7 in ENCORE_SEARCH is $\sum_{i=1}^{m} O(|E_i|) + \sum_{j=1}^{n}(t_1|e_j|) = O(p) + t_1 p = O(p)$. Hence the time complexity of algorithm ENCORE_SEARCH is $O(p)$. $\Box$

5.4 Hierarchical Refinement Towards Global Optima

The solutions produced by algorithm LOCAL_SEARCH are, in general, not optimal, but locally optimal with respect to the neighborhood. In this section, we present a refinement technique that makes use of the properties of signed hypergraphs to improve the solution quality.

We first consider an example as illustrated in Fig. 5.3(a). There are five vertices and four edges. The current bipartition is $(\{4\}, \{2, 3, 6, 8\})$ as illustrated by a dashed line. The edge incident with vertices 8 and 3 is not balanced. It is easy to verify that no further move of any vertex can have a positive gain; i.e., increase the number of balanced edges. However, as illustrated in Fig. 5.3(d), if vertices 3 and 4 are interchanged, then the number of balanced edges increases by 1.

This example can be generalized as follows. Let $H = (V, E)$ be a signed hypergraph, and V_1 be a nonempty subset of V. Let E_1 be all the edges that are only incident with vertices in V_1, i.e., all the end-vertices of such edges are in V_1. The pair (V_1, E_1) is a *sub-hypergraph* of H induced on V_1.

We consider the following property of sub-hypergraphs. Let $\pi = (V^+, V^-)$ be a

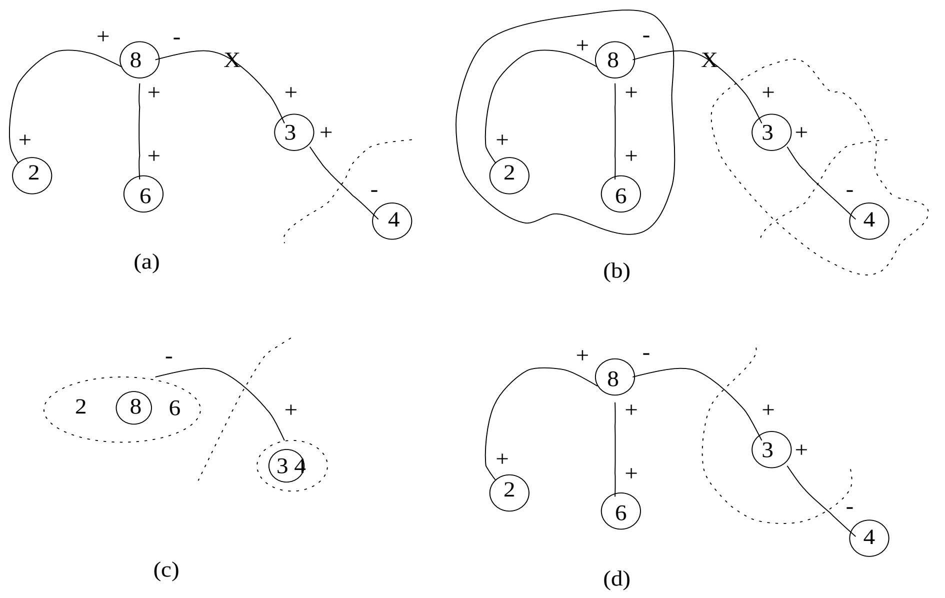


Figure 5.3: Illustration of further refinement of local search.

bipartition of V. All the vertices V_1 of sub-hypergraph H_1 are partitioned into the two groups by π. If we interchange all the vertices of V_1 in V^+ with all the vertices of V_1 in V^-, and denote the resulting bipartition as π', then the number of balanced edges by π' is the same as that of π. This is because all the balanced edges in E_1 remain balanced, all the unbalanced edges in E_1 remain unbalanced, and finally, all the edges not in E_1 remain as they were.

Suppose that V is partitioned into a set of subsets of vertices. For convenience, each subset is said to be a *cluster*. All the vertices in a cluster are partitioned to the two groups by π. If we interchange vertices in the two groups for a cluster, all the edges in the sub-hypergraph induced on this cluster will keep their balance status. However, edges with end-vertices in the different clusters may change their balance status. Hence we have a new optimization problem, that is to classify all the clusters of V to two categories —interchanged or not interchanged—so as to maximize the number of balanced edges. Since we do not need to consider the edges with end-vertices in a cluster, this problem can be formulated as the maximum balance problem in a reduced signed hypergraph. Let $H^r = (V^r, E^r)$ be such a reduced signed hypergraph, the construction of H^r involves the following steps:

1. Perform LOCAL_SEARCH on H, and denote the bipartition obtained by $\pi = (V^+, V^-)$.
2. Partition V into a set of clusters.
3. For each cluster, choose either the vertices in V^+ or those in V^- as reference vertices, and the other group as non-reference vertices.
4. Represent each cluster by a vertex in V^r.
5. If an edge e in H is incident with vertices in more than one cluster, construct an edge e^r in H^r incident with the corresponding vertices in V^r. If e is incident with sign s on a reference vertex of V, then e^r is incident with the corresponding vertex in V^r with the same sign. Otherwise, e^r is incident with the corresponding vertex in V^r with the complementary sign.

A simple way to partition V into a set of clusters is to define a maximal set of vertices that are incident with balanced edges to be a cluster. In other words, we shrink all the balanced edges and merge the vertices incident with each balanced edge to be one vertex.

Consider Fig. 5.3(a). All the vertices are partitioned into two clusters {2, 8, 6} and {3, 4}, as illustrated in Fig. 5.3(b). With vertices 8 and 3 chosen as reference vertices, the resulting reduced signed hypergraph is shown in Fig. 5.3(c).

Now LOCAL_SEARCH can be performed on H_r in an attempt to further increase the number of balanced edges. This process is repeated until the reduced signed hypergraph is stable, i.e., no further reduction of the cost is possible. Finally, we back-trace the process to obtain the final bipartition. The reader can verify that applying LOCAL_SEARCH on H_r in Fig. 5.3(c) yields the final bipartition as shown in Fig. 5.3(d).

There are several observations.

1. The algorithm takes $u \leq m$ iterations to converge, where m is the number of edges in a signed hypergraph.
2. Let $E_i \subseteq E$ be the set of balanced edges obtained by the ith run of LOCAL_SEARCH, and let $|E_i|$ be the cardinality of E_i. Then $E_1 \subset E_2 \subset \ldots \subset E_{u-1} \subseteq E_u$, and thus $|E_1| < |E_2| < \ldots < |E_{u-1}| \leq |E_u|$.
3. Each iteration takes $O(p)$ time in the worst case.

The refinement technique described above can also be used to enlarge the ENCORE neighborhood. Each time, instead of adopting a new solution in the neighborhood of current solution, we may use the refinement technique above to find a solution of even better quality.

5.5 Theoretical Aspects of Local Search

Having described practical considerations in applying local search to our problem, we turn now to some of its theoretical aspects. Surprisingly, although local search is conceptually simple and practically successful—it converges quickly with quite satisfactory solutions—very little has been known theoretically about its running time and solution quality. In fact, neither theoretical nor practical tools are well established yet. In this section, we summarize some preliminary observations in order to illustrate where the difficulties are.

5.5.1 Complexity of Local Search

We have shown that one iteration of our local search algorithm takes linear time; i.e., we are able to determine in linear time whether a solution is locally optimal and to find a better neighbor if it is not. The question is how many passes are needed for convergence to a local optimum, or, more generally, what is the complexity of finding locally optimal solutions.

It is not difficult to see that, for a signed hypergraph with *unweighted, or small-integer-weighted, edges*, the local search algorithm converges in $O(m)$ passes, where m is the number of edges. This is because the value of the objective function, which is bounded by $O(m)$, is always decreased in each pass. In practice, the number of required passes seems independent of the number of edges; usually only a few passes are sufficient.

On the other hand, if there are arbitrary weights associated with edges in a signed hypergraph, we can construct an instance of the problem—a signed hypergraph with certain weights associated with edges—with a specific initial bipartition, such that the standard local search algorithm takes an exponential number of passes to converge*. Moreover, it seems very unlikely that there exist other, possibly noniterative, algorithms that could do better in finding a local optimum that would be produced by the standard algorithm starting from a given initial bipartition. This problem is called the *standard algorithm problem* [127]. We will prove in Theorem 5.5.1 that the standard algorithm problem is NP-hard. However, the hardness in finding the precise local optimum by the standard algorithm does not imply hardness in finding *some* local optimum. In fact, the complexity of the local search problem—given a signed hypergraph, find some locally optimal solution $s \in F$—is still open.

For a given signed hypergraph H with a rational edge weighting w, there always exists a local optimum; further the local optimality can be verified in linear time. This fact can be rephrased in the following predicate form:

$$P(H, w) = (\text{There exists a bipartition } \pi \text{ such that } Q(H, w, \pi)),$$

*A proof will be given in Theorem 5.5.1.

where

$$Q(H, w, \pi) = (\text{for any bipartition } \pi' \text{ such that } \pi' \in \mathrm{N}(\pi),\ c(\pi) \geq c(\pi')),$$

and $Q(H, w, \pi)$ can be verified in polynomial time. Clearly $P(H, w)$ is true for all signed hypergraphs with arbitrary weightings. This is an *existentially polynomial (EP)* theorem according to the terminology of Edmonds [20]. Then, from Edmonds' conjecture-schemata (which are supported by numerous examples from various disciplines), it follows that there should exist a polynomial-time algorithm for finding such a π [20].

From a series of recent results of Johnson, Krentel, Papadimitriou, Schäffer, and Yannakakis [70, 82, 114, 116, 127], we can show that the complexity of our local search problem is in PLS (for Polynomial-time Local Search), a complexity class somewhere "between" P and NP. Further, it is PLS-complete, and thus as hard as any such problem.

A local search problem P is in the class PLS [70], if there exist polynomial-time algorithms

1. to compute an initial feasible solution,
2. to compute the cost c for a given solution,
3. either to determine that a solution is locally optimal or to find a better solution in N.

The class of PLS-complete problems includes the following well-known local search problems:

- the Kernighan-Lin heuristic for the graph partitioning problem [70, 77],
- the Lin-Kernighan heuristic for the traveling salesman problem [116],
- MAX CUT with the Hamming neighborhood; this is called the 1-*opt* MAX-CUT.

Moreover, for all the problems above, the standard algorithm takes exponential time in the worst case, and the standard algorithm problem is NP-hard.

In the following, we show that our problem is PLS-complete by giving a "PLS-reduction" from 1-*opt* MAX-CUT to it. A *PLS-reduction* from problem A in PLS to another problem B in PLS is defined in terms of two polynomially computable functions f and g. Given an instance x of A, f computes an instance $f(x)$ of B such that, for any local optimum s of $f(x)$, $g(s)$ is a local optimum of x. Since there are two polynomial functions involved, PLS-completeness proofs tend to be very complicated. In fact, the PLS-reduction for the Lin-Kernighan heuristic for the traveling salesman problem is ten-page long [116]. Fortunately, owing to previous results, the proof for our problem is not that involved.

If a function g is a one-to-one mapping between local optima of the target problem to local optima of the source, then a PLS-reduction is said to be *tight*[†]. For a tight PLS-reduction, if the standard algorithm for the source problem takes exponential time in the worst case, so does the algorithm for the target problem. If the standard algorithm problem for the source problem is NP-hard, so is the standard algorithm problem for the target problem.

Theorem 5.5.1 *The local search optimization problem for the maximum balance problem with the* ENCORE *neighborhood is PLS-complete. The standard algorithm for the local search optimization problem takes exponential time in the worst case. The corresponding standard algorithm problem is NP-hard.*

Proof. The ENCORE heuristic is in the class PLS. Now we can give a PLS-reduction from $1-opt$ MAX-CUT. This reduction is straightforward, since the Hamming neighborhood is a restricted ENCORE neighborhood, and since MAX CUT is

[†]A more formal definition can be obtained in [127].

a restricted version of the maximum balance problem. Our problem is thus PLS-complete. Further, this "restriction reduction" is tight. Therefore, it follows that the standard algorithm for our local search problem takes exponential time in the worst case, and that the corresponding standard algorithm problem is NP-hard. □

5.5.2 Quality of Local Search

The solutions obtained by local search are lower bounds to globally optimal solutions. We are interested in knowing how far they are away from global optima, or, more generally, what are upper bounds on global optima.

Generating upper bounds appears hard for the maximum balance problem except for some degenerate cases. For the problem of graph bipartitioning (with positive edge weights), there are two graph-theoretic approaches to generating such bounds. One, as noted first by Kernighan and Lin [77], is by using the max-flow/min-cut theorem of Ford and Fulkerson [46]. A minimum cut separating two designated vertices can be obtained by flow techniques in $O(n^3)$ time, where $n = |V|$ [33]. A global minimum cut can be found by using cut-tree techniques in $O(n^4)$ time [64].

The other approach, developed on the basis of a theory called the basis of graph spectra [35], yields even tighter bounds. The idea is to formulate the $n \times n$ *adjacency matrix* $A = (a_{ij})$ of a graph, with $a_{ij} = w_{ij}$ where w_{ij} is the weight associated with the edge joining v_i and v_j, and then to use the second smallest eigenvalue as a bound [40]. This approach has been extended and applied very successfully by Vannelli and his associates [56, 150] to netlist partitioning, and by Hagen and Kahn [58] to netlist partitioning with the ratio-cut objective [90, 154].

Both approaches above are based on the assumption of non-negative weights. From the last chapter, we know that negative weights are generally unavoidable if we want to approximate the maximum balance problem in a signed hypergraph by the maximum cut problem in a graph. Therefore, it seems difficult to extend these two approaches to our problem.

For many combinatorial optimization problems, the mathematical programming approach can be used to generate good bounds. The idea is to formulate a combinatorial optimization problem as an integer linear programming problem, and then to use the solutions given by linear relaxation (permitting nonintegral solutions) as bounds. However, this approach does not help for the maximum balance problem, since the linear relaxation of its 0-1 integer programming formulation (See Chapter 7) provides no useful information: its solution is a vector with all components being $\frac{1}{2}$ and the value of the objective function equals the total number of edges. We note that an application of a new lower bound given recently by Boros, Crama and Hammer [14] for unconstrained quadratic 0-1 optimization appears promising.

5.6 Summary

In this chapter, we presented an application of local search to solving the maximum balance problem in general signed hypergraphs. Our main contribution here is a careful implementation of local search, which is a generalization of the work of Fiduccia and Mattheyses on netlist partitioning. We have proved that the worst-case computational cost of one pass of the heuristic grows linearly with the size of signed hypergraph. We also presented a refinement technique that makes use of the properties of signed hypergraphs to further improve the solution quality. We also proved that our local search heuristic is PLS-complete; i.e., as hard as any such problem. The problem of whether we could have a polynomial algorithm for finding a local optimum still constitutes a "family treasure" [20].

Chapter 6

A Greedy Peeling Heuristic for Minimum Covering

Having focused on the maximum balance problem in the previous three chapters, we now turn to the minimum covering problem. In addition to the problem itself, we are also interested in the following variation: Given a signed hypergraph H and a fixed integer k, find k bipartitions that balance as many edges in H as possible. We call this variation the *partial covering* problem.

There are two motivations for studying the partial covering problem. First, the partial covering problem with $k = 1$ degenerates to the maximum balance problem*. Thus we hope that the design of minimum covering algorithms could benefit from maximum balance algorithms developed in the previous chapters. Second, as described in Chapter 8, partial covering itself is of practical importance to the optimum synthesis of synchronous logic circuits.

Given the NP-completeness of both the partial and minimum covering problems, and also given the large size of the problem instances we need to solve, we will focus on

*In this sense, the partial covering problem is called the *maximum k-balance* problem.

finding an efficient algorithm that can produce good approximate solutions. A simple heuristic, which solves both the partial covering problem and the minimum covering problem in a unified way, is as follows: First find a bipartition π that balances as many edges as possible, then remove the edges balanced by π from H, and repeat this process k times unless H is empty, i.e., contains no edges. This heuristic, called *greedy peeling*, solves the minimum covering problem, when k is set to a sufficiently large number.

The greedy peeling heuristic can be implemented very efficiently. The bounded version of the local search heuristic developed in the previous chapter can be used for maximum balance. The resulting implementation of greedy peeling runs in $O(kp)$ time, where p is the size of the signed hypergraph.

The issue we may doubt is the quality of the solution generated by greedy peeling. Before we describe its empirical evaluation in Chapter 9, we investigate this issue theoretically in this chapter. We will show that, under the assumption that the maximum balance problem can be solved exactly, the greedy peeling heuristic has a guaranteed (constant) performance bound for partial covering, and has a performance bound $\log p$ for minimum covering.

We notice that there exists a class of optimization problems arising in a variety of VLSI applications, which have the same structure as the problem we are considering here. Various heuristics have been developed for each problem, and have achieved remarkable success in practice; however, no theoretical justification is available yet. Our theoretical analysis turns out to be applicable to all these heuristics. This chapter is organized as follows: In Section 6.1, we introduce a mathematical framework that captures the underlying structure of a class of VLSI optimization problems. In Section 6.2, we describe two general paradigms, namely prime covering and greedy peeling, for solving the optimization problems of this structure. We then focus on deriving performance bounds of greedy peeling for partial covering and exact covering in Sections 6.3 and 6.4. In Section 6.5, we apply our theoretical results to justify some experimental results reported in the literature.

6.1 The Cluster-Cover Framework

Let E be a finite ground set of *elements*. A *set system* is a pair (E, F) with $F \subseteq 2^E$.

A *subset-closed system*[†] is a set system such that

1. $\emptyset \in F$,
2. $Y \in F$ and $X \subseteq Y$ implies $X \in F$.

A *superset-closed system* is a set system such that

1. $E \in F$,
2. $Y \in F$ and $X \supseteq Y$ implies $X \in F$.

The first property is called the *nonemptiness* property. The second property is called the *hereditary* property.

We are now ready to present an abstract framework, called *cluster-cover*, for a class of combinatorial optimization problems. We are given a set E of *ground elements* and a problem-specific *(compatibility) predicate* $\Pi(P)$ with a single parameter $P \subseteq E$. A subset P of E is called a *cluster* if it satisfies the predicate. Furthermore, the predicate Π must be such that the set $\mathcal{P}$ of clusters forms a subset-closed system $(E, \mathcal{P})$. The *maximum k-cluster problem* is as follows: Given an integer k, find at most k clusters that contain as many ground elements as possible. When $k = 1$, this becomes the *maximum cluster* problem.

Given a ratio α, $0 < \alpha \leq 1$, an *α-cover* is a set of clusters that contains at least the fraction α of the ground elements. Clearly the set—denoted by $\mathcal{C}$—of covers forms a superset-closed system $(\mathcal{P}, \mathcal{C})$. The *minimum α-cover* problem is as follows: Given a ratio α, $0 < \alpha \leq 1$, find an α-cover that contains as few clusters as possible. When $\alpha = 1$, this becomes the *minimum cover* problem, or the *exact cover* problem.

[†]A subset-closed system is known as an *independence system* according to Korte and Lovász [80].

An illustration of this structure is given in Fig. 6.1: The ground set E is $\{a, b, c\}$. The clusters are {a,b}, {b,c}, and all their subsets (including the empty set). Note that {a,c} and {a,b,c} are excluded by the problem-specific predicate (an example is given in Problem 2 described later in this section). Then given $\alpha = \frac{2}{3}$, the set of α-covers is $\mathcal{C} = \{C_1, ..., C_5, ...\}$, where $C_1 = \{\{a\}, \{b\}\}$, $C_2 = \{\{a\}, \{c\}\}$, $C_3 = \{\{b\}, \{c\}\}$, $C_4 = \{\{a, b\}\}$, $C_5 = \{\{b, c\}\}$, and the remaining covers are the super-sets of the first five covers. The empty cluster is ignored, since it does not contribute anything useful to the problem we are interested in.

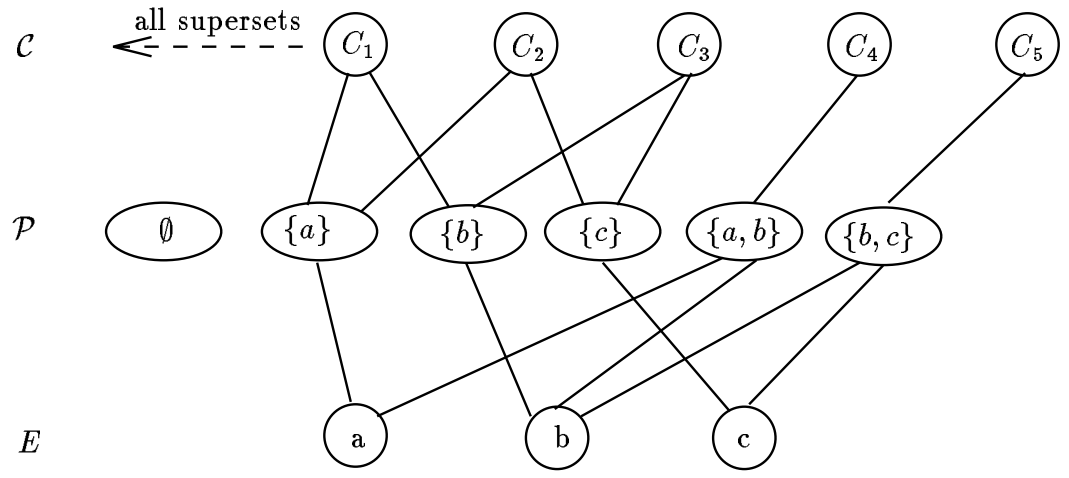


Figure 6.1: Structure illustrating the cluster-cover optimization problems.

The maximum k-cluster problem and the minimum α-cover problem are closely related. On the one hand, setting k to a sufficiently large number in the maximum k-cluster problem solves the minimum α-cover problem. On the other hand, the minimum α-cover problem with a sufficiently small α is equivalent to the maximum k-cluster problem. However, both problems have independent practical relevance.

On the theoretical side, the cluster-cover framework described above relates to two important concepts studied by mathematicians. One is a set-cover: Given a ground set, and a set of subsets of ground elements, the set-cover problem is to find the minimum number of subsets that cover all the ground set. Our framework of cluster-cover differs from set-cover in two aspects: (1) the set of subsets is implicitly given by a problem-specific predicate, and (2) the set of subsets is subset-closed. The other related concept is a matroid [80]. A matroid is a subset-closed system $(E, \mathcal{P})$ that obeys the combinatorial aspect of the Steinitz exchange principle:

- If $X, Y \in \mathcal{P}$ and $|X| < |Y|$ then there exists a $y \in Y - X$ such that $X \cup \{y\} \in \mathcal{P}$.

On the practical side, the cluster-cover framework abstracts a class of optimization problems arising in a variety of VLSI applications. Some of them have been studied extensively and are well understood, some are less explored, and some have been posed only very recently. In the rest of this section, we describe these problems in an informal way. We refer the interested readers to the literature for more precise definitions and more detailed motivations.

Problem 1: Combinational Logic Minimization [17, 121]

Combinational logic minimization is a problem of minimizing the cost of the circuit used to implement a Boolean function. A Boolean function f is usually *(incompletely) specified* by its *on-set*, the set of input values $\mathbf{x} \in \{0,1\}^n$ such that $f(\mathbf{x}) = 1$, and its *off-set*, the set of input values such that $f(\mathbf{x}) = 0$. Each element in the on-set (off-set) is called a *minterm*. The set of ground elements are the on-set. A cluster is a set of minterms that can be "covered" by a product term (called "cube" in the corresponding Boolean space) that does not contain any minterm in the off-set. Such a product term is called an *implicant* in logic synthesis. The exact cover problem is of practical interest, and has been studied extensively [17, 121]. For example, given the on-set $\{x_1x_2, x_1\overline{x_2}\}$ and the off-set $\{\overline{x_1}x_2\}$, $\{x_1x_2, x_1\overline{x_2}\}$ is a cluster, which can be "covered" by an implicant $\{x_1\}$. This cluster constitutes the minimum cost cover.

Problem 2: Constrained Encoding [135, 136]

This is one of the two problems we are studying in this thesis. A ground element is a dichotomy. For example, $a = (\{1,2\},\{3\})$, $b = (\{2\},\{4\})$, and $c = (\{1\},\{2,3\})$ are three dichotomies, where $\{1,...,4\}$ are states. Two dichotomies are *compatible* if no two states appearing in one subset of one dichotomy appear in two different subsets of the other dichotomy. For example, a and b, and b and c, are compatible; but a and c are not. A set of mutually compatible dichotomies forms a *cluster*, which in turn can be represented by another dichotomy. For example, a and b can be represented by $(\{1,2\},\{3,4\})$. We say that the new dichotomy *covers* all the dichotomies in the cluster. The cluster-cover structure for this example has been illustrated in Fig. 6.1. The maximum cluster problem is to find a dichotomy that covers as many dichotomies as possible. The minimum cover problem is to find a minimum number of dichotomies that cover all the given "ground" dichotomies. Both the maximum cluster and the minimum cover problems are of practical interest to sequential logic synthesis, as described in Chapter 9.

Problem 3: Multi-Layer Topological Planar Routing [31]

Consider Fig. 6.2 with two rows of terminals marked by numbers. Terminals marked by the same number form a *net*. A ground element is a net. Two nets are *compatible* if they can be routed in a plane without intersecting with each other. For example, nets 1, 2 and 3 (solid) in Fig. 6.2 can be routed in one plane; they form a cluster. Nets 4 and 5 (dashed) can be routed in another plane; they form another cluster. The maximum cluster problem is to find a maximal set of nets that can be routed in one plane. The minimum cover problem is to find the minimum number of planes such that all the nets can be routed without intersecting with each other. Both problems are of practical interest, and have been studied [31].

Problem 4: Application Timing for Delay-Fault Testing [67]

Consider a combinational circuit in Fig. 6.3 with two primary inputs (I1 and I2), two primary outputs (O1 and O2), and five gates (G1 to G5) interconnected by wires. Associated with each gate and wire is a *delay*: for example, gate G1 has delay 0, and the wire from input I1 to gate G1 has delay 10. Delay-fault testing involves an application of a sequence of test patterns at primary inputs. We denote by T_i the specific time of applying a signal to a primary input i. For example, we may have

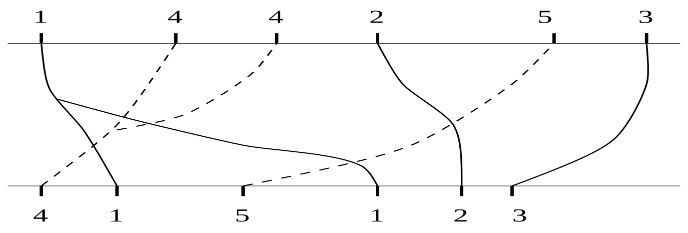


Figure 6.2: An example of multi-layer topological planar routing.

the so-called *application timing assignment* $T_{I1} = 0$ and $T_{I2} = 5$ for a test pattern. For a specific timing assignment of a test pattern, we denote by T_j the latest time for signals to arrive at output j from all the inputs. (e.g., $T_{O1} = T_{O2} = 25$.) For a path from an input i to an output j, the time allowed for signal propagation is $T_j - T_i$, which may be different from the total delay value associated with that path; this difference is called the *slack of that path.* (e.g., path I1→G1→G4→O1 is the only one with a nonzero slack (5).) We define the *delay slack* for each delay as the minimum of the path slacks of all input-to-output paths that include this delay. (e.g., the wire delay between G1 and G4 has a nonzero slack (5).)

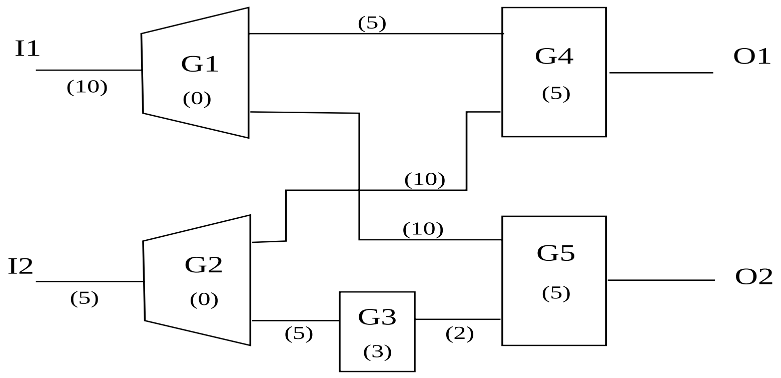


Figure 6.3: Simple example to illustrate application timing.

Under application timing assignment $T_{I1} = 0$ and $T_{I2} = 5$, only the wire delay between G1 and G4 has a nonzero slack (5), whereas under application timing assignment $T_{I1} = T_{I2} = 0$, $T_{O1} = 20$, $T_{O2} = 25$, path I2→G2→G3→G5→O2 has a nonzero slack (5). Also the wire delay between G2 and G3 and that between G3 and G5 have a nonzero slack (5). The total sum of delay slacks is 10.

A key observation established in [67] is that delay slacks affect the size of the delay fault detectable by any test set. In order to improve the quality of delay-fault testing, it is desirable to find an application timing assignment so as to keep delay slacks as small as possible. However, the slack of a delay d cannot be made arbitrarily small; there exists a *slack lower bound* that is the length of a longest input-to-output path minus the length of a longest input-to-output path including delay d.

The *multiple test application timing assignment* problem is as follows[‡]: Given a ratio, α, $0 < \alpha \leq 1$, find the smallest number of application timing assignments such that, for the ratio α of all the delays, there exists at least one timing assignment for each delay that enables the delay to achieve its slack lower bound. The *single test application timing assignment* problem is to find an application timing assignment such that as many delays as possible achieve their slack lower bounds.

A key to casting the test application timing assignment problems into our cluster-cover framework is the recognition of a compatible cluster. A subset of delays in the graph forms a compatible cluster, if all the delays in the subset can achieve their slack lower bounds under a single input time assignment; this is called a *consistent-tight set (subgraph)* by Iyengar and Vijayan [67].

Problem 5: Monitoring-Logic Design for BIST Enhancement [53]

Built-in self-test (BIST) is a technique widely used in VLSI testing. A typical BIST structure is shown in the upper part of Fig. 6.4, which consists of a test generator, a combinational circuit under test (CUT), and a signature analyzer. The test

[‡]The formulation given in [67] is slightly different from ours. However, their algorithm does indeed solve the problem in our formulation.

generator produces a test input sequence, which is applied to the CUT. The signature analyzer then compresses the output sequence of the CUT into a signature, and compares it against the correct one. If they are different, then the CUT is faulty. However, if the CUT is faulty, we may still obtain a correct signature because of the data compression. This phenomenon is called *aliasing*.

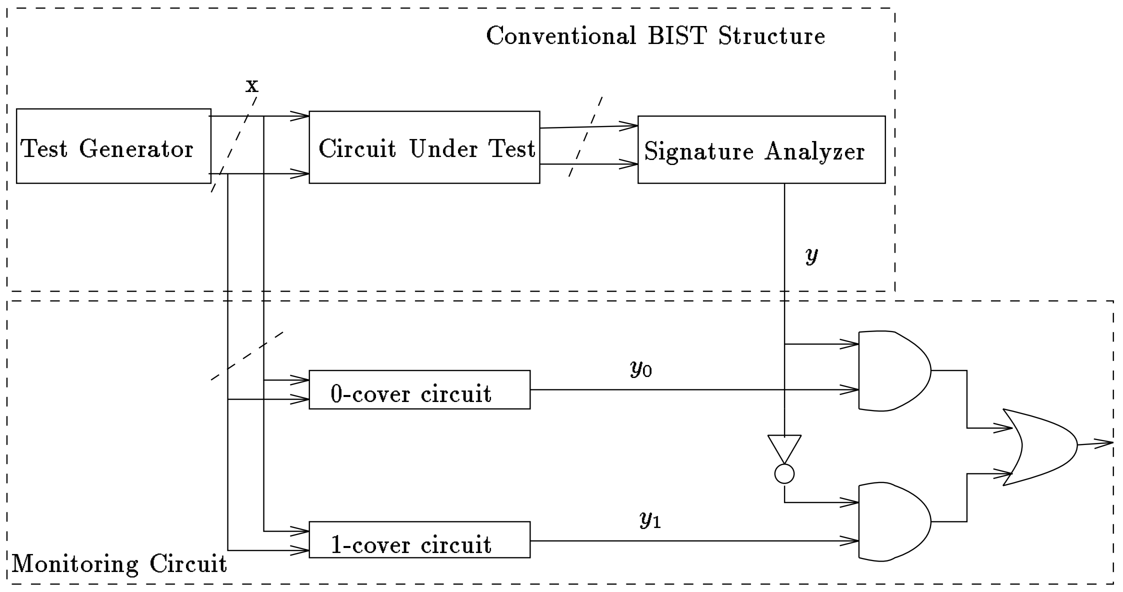


Figure 6.4: An augmented BIST structure.

To overcome aliasing, Gössel and Jürgensen [53] proposed an improved BIST structure as shown in Fig. 6.4, where an error-detection circuit is used to monitor the faults undetectable because of aliasing. The error-detection circuit consists of a "0-cover circuit" y_0 of the correct function f_0 such that $y_0(\mathbf{x}) = 1$ implies $f_0(\mathbf{x}) = 0$, and a "1-cover circuit" y_1 of f_0 such that $y_1(\mathbf{x}) = 1$ implies $f_0(\mathbf{x}) = 1$. To monitor those faults that cause a function $f_i, i = 1, ..., l$ being realized, instead of f_0, we need to have

1. the off-set of y_0 is the on-set of f_0,
2. the off-set of y_1 is the off-set of f_0,

3. the on-sets of y_0 and y_1 are such that, for each $f_i, i = 1, ..., l$, there exists at least one input pattern $\mathbf{x}$ for which the output $y_0(\mathbf{x})f_i(\mathbf{x}) + y_1(\mathbf{x})\overline{f_i(\mathbf{x})}$ of the error detection circuit is 1.

The last condition can be rephrased with the following concept: If $f_i(\mathbf{x}) \neq f_0(\mathbf{x}) = 0$, then we say that $\mathbf{x}$ *0-distinguishes* fault f_i, fault f_i is *0-distinguishable* under input pattern $\mathbf{x}$, and $\mathbf{x}$ is a *0-distinguishing* pattern. If $f_i(\mathbf{x}) \neq f_0(\mathbf{x}) = 1$, then we say that $\mathbf{x}$ *1-distinguishes* fault f_i, fault f_i is *1-distinguishable* under input pattern $\mathbf{x}$, and $\mathbf{x}$ is a *1-distinguishing* pattern. Then condition (3) can be replaced by the following three conditions:

3. each $f_i, i = 1, ..., l$, is either 0-distinguished or 1-distinguished by at least one pattern $\mathbf{x}$,
4. $y_0 = 1$ for all 0-distinguishing patterns,
5. $y_1 = 1$ for all 1-distinguishing patterns.

The new logic minimization problem arising from monitoring-logic design for BIST enhancement is as follows: Given $f_i, i = 0, ..., l$, design a minimal-cost two-output circuit (y_0 and y_1) that satisfies (1)-(5).

We consider an example of Gössel and Jürgensen, as shown in Table 6.1, with three undetectable faults f_1, f_2, and f_3 of the correct function f_0. Fault f_3 is 0-distinguishable under 000; fault f_1 is 0-distinguishable under 001 and 1-distinguishable under both 110 and 011; fault f_2 is 0-distinguishable under 001.

Before formulating this problem in our cluster-cover framework, we make the following observation. An input pattern $\mathbf{x}$ can 0-distinguish a set of faults, or 1-distinguish them, but not both. Thus $y_0(\mathbf{x}) = 1$ implies $y_1(\mathbf{x}) = 0$, and $y_1(\mathbf{x}) = 1$ implies $y_0(\mathbf{x}) = 0$. According to the terminology in logic synthesis (See Problem 1), the on-set of y_0 is the set of 0-distinguishing patterns, and the on-set of y_1 is the set of 1-distinguishing patterns. Therefore, the set of implicants of y_0 is disjoint from that of y_1. The structure of our problem can be illustrated in Fig. 6.5: Each minterm

Table 6.1: A simple example to illustrate BIST enhancement.

test pattern $\mathbf{x}$	f_0	f_1	f_2	f_3
000	0	0	0	1
100	0	0	0	0
010	0	0	0	0
110	1	0	1	1
001	0	1	1	0
101	1	1	1	1
011	1	0	1	1
111	1	1	1	1

may distinguish a set of faults; each implicant covers a set of minterms; a cover is a set of implicants.

Because the relation between faults and minterms can be easily obtained from the given table in linear time, the problem structure can be simplified so that it fits into our cluster-cover framework. The set of faults constitutes the ground set. A cluster is a set of faults that can be distinguished by minterms that can be covered by an implicant. A cover is a set of clusters that covers all the faults. The monitoring-logic minimization problem considered by Gössel and Jürgensen can be formulated as the minimum cover problem, i.e., find a cover that contains as few clusters as possible.

The cluster-cover framework for our example is shown in Fig. 6.6. The off-set of y_0 is {110,101,011,111}. The off-set of y_1 is {000,100,010,001}. Faults f_1, f_2, and f_3 form a y_0-cluster, because all they are 0-distinguishable by a set of minterms that can be represented by an implicant $\bar{x}_1\bar{x}_2$. The minimum cost cover is $\bar{x}_1\bar{x}_2$.

An interesting extension of the work of Gössel and Jürgensen is to study the maximum k-cluster problem. We may often have a limited area that can be used for monitoring logic. Then the problem is to design a monitoring circuit using this area

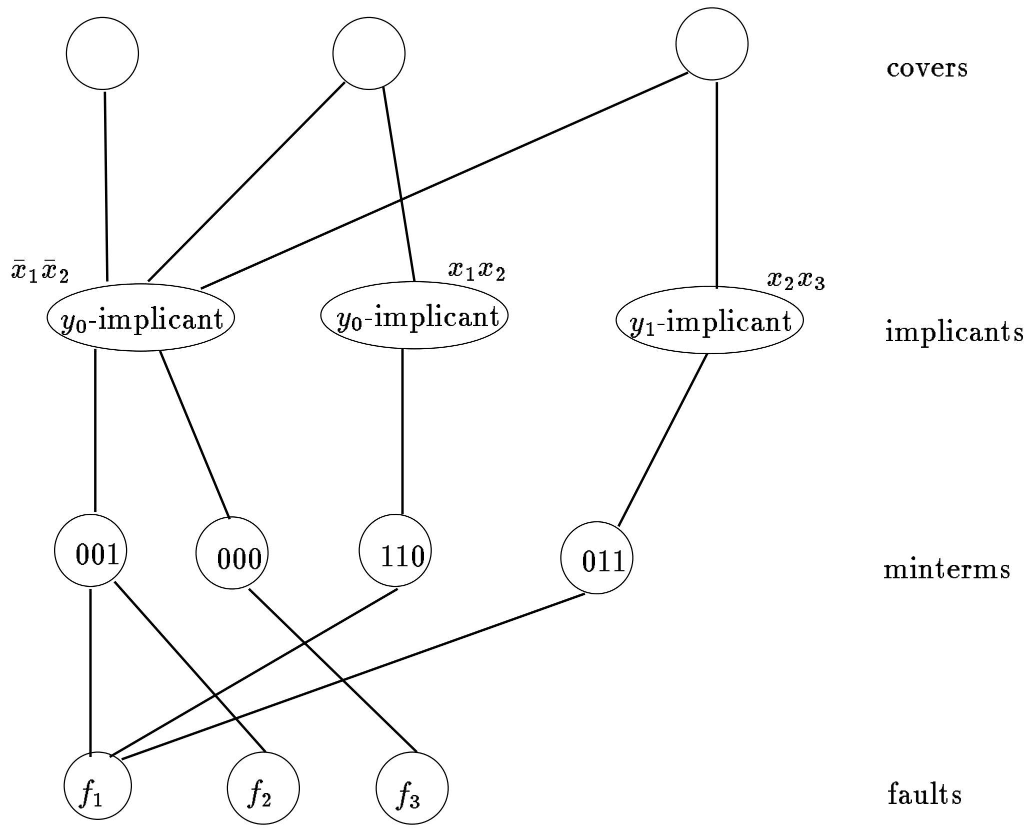


Figure 6.5: Three-level hierarchy for monitoring-logic.

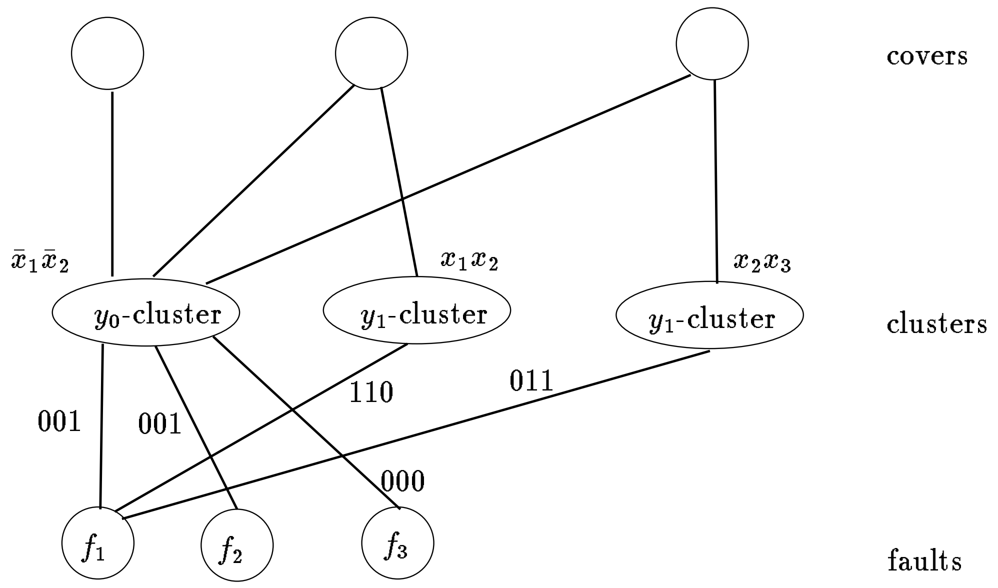


Figure 6.6: A cluster-cover framework for the monitoring-logic example.

in such a way to cover as many faults as possible. This is of more interest to the ASIC industry, since the gate-array or FPGA designs usually have some spare area.

Our cluster-cover framework leads to a precise formulation of the monitoring-logic minimization problem for BIST enhancement. Two paradigms described in the next section will provide, for the first time, both an exact algorithm and a complete heuristic for monitoring-logic minimization.

6.2 Prime Covering versus Greedy Peeling

In this section, we describe two basic paradigms, namely prime covering and greedy peeling, for solving the optimization problems that have the cluster-cover structure. We then categorize the heuristics developed previously for each optimization problem listed in the previous section.

6.2.1 Prime Covering

Prime covering is an enumeration-based approach to solving the minimum α-cover problem. The approach consists of two stages, corresponding to the two-level hierarchies of the problem structure as illustrated in Fig. 6.1. In the first stage, the set of clusters is generated. Because the set of clusters constitutes a subset-closed system, only maximal clusters—for which the addition of one more ground element will violate the predicate—need to be constructed. A maximal cluster is called a *prime cluster*.

With the set of constructed prime clusters, the second stage of prime covering is to solve the standard set-cover problem by the branch-and-bound method. Branch-and-bound is a process of successive partitioning of the solution space. A cluster is picked up—this is called *branching*—and then we examine the problem, assuming that the cluster is in the minimum cover, and then assuming that the cluster is not in the minimum cover. *Bounding* refers to generating lower bounds that can be used to prune the search space effectively.

To obtain a good lower bound for the objective function at each branch, either the compatibility graph or the conflict graph can be used. A compatibility (conflict) graph has vertices corresponding to ground elements. There is an edge joining two vertices if the corresponding two ground elements are compatible (not compatible), i.e., in (not in) one cluster. The number of vertices in the maximum independent set in the compatibility graph, which equals to the number of vertices in the maximum clique in the conflict graph, is a lower bound for the size of the minimum cover. In practice, finding the maximum clique is easier than finding the maximum independent set [121].

The determination of the branching cluster is based on a basic intuition that the ground elements participating in fewer clusters are the ones "hard" to cover. To measure this "hardness", each ground element is given as its weight the reciprocal of the number of clusters it participates in. The weight of a cluster is the total weight of all the ground elements in the cluster. The cluster of maximum weight which is also in the independent set is chosen as the branching cluster.

Prime covering is basically an exact algorithm for optimum solutions. It can be converted into a greedy heuristic by taking the first leaf visited as the solution and using no backtracking. The result obtained is a lower bound for the minimum cover.

The set-cover problem is NP-hard [48]. In the worst case, the number of branches needed for finding an optimum solution is exponential in terms of the number of prime clusters. The no-backtracking version of the algorithm or even greedy peeling can be used for set-covering, as is done in combinational logic minimization; however, the number of prime clusters can be exponential in terms of the number of ground elements. For example, as reported in the constrained encoding problem, the number of primes is prohibitedly large [123]. Furthermore, in most cases the construction of prime clusters is time-and-space consuming. Therefore, prime covering is usually used as a paradigm for finding optimum solutions to relatively small problems for which the clusters can be easily constructed.

6.2.2 Greedy Peeling

Greedy peeling solves both the minimum α-cover problem and the maximum k-cluster problem in a unified manner. As a one-stage approach, it constructs directly the clusters needed in the final solution (minimum cover). An informal description of greedy peeling with application to the minimum covering problem was provided at the beginning of this chapter. A more abstract description in terms of the cluster-cover framework is given in Fig. 6.7. Initially a ground set E and an integer k are given. The initial solution is an empty cover C (line 2). The key of greedy peeling is to use a subroutine that can solve the maximum l-cluster problem ($l \leq k$, typically $l = 1$ is used). The algorithm iterates $\lceil \frac{k}{l} \rceil$ times (lines 3 to 6): at each iteration, the subroutine is first invoked to find a cluster C_i and the subset of ground elements covered by C_i (substep: *solve*); then the ground elements covered by C_i are removed from E (substep: *peel-off*); and finally the newly found cluster C_i is added to the final solution (substep: *augment*). The algorithm returns the computed cover C and the remainder of the ground set consisting of the elements not covered by C.

```
GREEDY_PEELING($E, k$)
1   $E' \leftarrow E$
2   $C \leftarrow \{\ \}$
3   for $i = 1$ to $\lceil \frac{k}{l} \rceil$ do
4       $(C_i, E_i) \leftarrow$ solution of maximum $l$-cluster on $E'$    /* solve */
5       $E' \leftarrow E' - E_i$                                          /* peel off */
6       $C \leftarrow C \cup C_i$                                         /* augment */
7   return $(C, E')$
```

Figure 6.7: Paradigm of greedy peeling.

The greedy peeling heuristic can be applied to the second stage of prime covering for solving the set-cover problem, as we mentioned in the previous subsection. However, the chief advantage of greedy peeling, in contrast to prime covering, is to avoid the complicated and time-consuming process of cluster generation for the cluster-cover optimization problems.

The main disadvantage of greedy peeling, in contrast to prime covering, is that it is not obvious how to approach optimum solutions. However, for several degenerate cases, greedy peeling is proven to produce optimum solutions:

1. The maximum l-cluster problem can be solved exactly and the resulting subset-closed system consists of $\lceil \frac{k}{l} \rceil$ disjoint clusters.
2. The maximum l-cluster problem can be solved exactly, and the resulting subset-closed system obeys the combinatorial aspect of the Steinitz exchange princi-

ple:

- If $X, Y \in \mathcal{P}$ and $|X| < |Y|$ then there exists a $y \in Y - X$ such that $X \cup \{y\} \in \mathcal{P}$.

The subset-closed system with the property above is called a *matroid*. Matroid theory has established that the greedy algorithm *always* produces an optimal solution [33].

We now describe two general techniques to improve the solution quality of greedy peeling. One is to build local search on top of greedy peeling, which gives rise to *iterative greedy peeling*. It works as follows: Suppose that we have found a solution by greedy peeling after k iterations. The kth iteration was introduced, since there exists a non-empty set P of ground elements that has not been covered by the first $k - 1$ iterations. The set P thus appeared to be "hard" to cover. Therefore, we start a new run of greedy peeling by "insisting" that the first iteration of greedy peeling must peel off P. This can be accomplished by using a modified local search heuristic. The other technique is to prevent the peeling from being too greedy by imposing certain problem-specific criteria; for example, balance criteria in constrained encoding [136, 158].

6.2.3 Taxonomy of Cluster-Covering Heuristics

In this subsection, we summarize various heuristics developed previously for some VLSI optimization problems of the cluster-cover structure in two categories: prime covering and greedy peeling. We refer the interested readers to the original literature for details of those heuristics.

- **Prime Covering:** This category includes various algorithms for combinational logic optimization, such as [17, 121], the constrained encoding algorithm by Yang and and Ciesielski [158], the constrained encoding algorithm by Saldanha, Villa, Brayton, and Sangiovanni-Vincentelli [123], and the constrained encoding algorithm by Devedas and Newton [38].

- **Greedy Peeling:** This category includes the constrained encoding algorithm by Shi and Brzozowski [136], the topological planar routing algorithm by Cong, Hossain, and Sherwan [31], the application timing algorithm by Iyengar and Vijayan [67], and the monotoring-logic optimization algorithm by Gössel and Jürgensen [53].

In summary, our results in this section are two-fold. First, two paradigms are abstracted out in an attempt to capture the generally-applicable ingredients from various heuristics targeted for each individual application. They can be used as general guidelines for developing both exact and heuristic algorithms for new optimization problems that have the cluster-cover structure. Second, the taxonomy of cluster-covering heuristics provides additional insights into solving each specific problem. On one hand, we can use the paradigm of prime covering to develop exact algorithms for those problems which have been previously solved only by greedy peeling. On the other hand, we can apply the paradigm of greedy peeling to develop efficient hueristics for solving large-size problems that were previously unsolvable by prime covering.

6.3 Performance of Greedy Peeling for Partial Covering

In this section, we analyze the performance of greedy peeling for partial covering[§]. We define the performance ratio γ of greedy peeling as follows:

$$\gamma = \frac{\text{the number of ground elements covered by } k \text{ clusters using greedy peeling}}{\text{maximal number of ground elements that can be covered by } k \text{ clusters}}.$$

[§]This work is inspired by the work of Cong, Hossain, and Sherwan [31]. We realized that their initial analysis for topological planar routing can be generalized to a class of problems of the cluster-cover structure.

Our main result is as follows:

Theorem 6.3.1 *Suppose that the maximum l-cluster problem can be solved with a performance ratio ϵ; then the performance ratio of greedy peeling for the maximum k-cluster problem is*

$$\gamma \geq 1 - (1 - \frac{\epsilon}{\lceil \frac{k}{l} \rceil})^{\lceil \frac{k}{l} \rceil},$$

where $k \geq l$.

We first prove the following lemma.

Lemma 6.3.1 *Let $\mathcal{P}_k^*$ be the set of ground elements of maximal cardinality that could be covered by k clusters. Let P_i be the set of ground elements covered by greedy peeling at the ith iteration, $1 \leq i \leq \lceil \frac{k}{l} \rceil$. Then we have:*

$$|P_i| \geq \frac{\epsilon}{\lceil \frac{k}{l} \rceil}(|\mathcal{P}_k^*| - \sum_{j=1}^{i-1} |P_j|).$$

Proof. Suppose that $P_j \subset E$, $j < i$, is the set of ground elements covered by the jth iteration of greedy peeling. Then, at the end of the $(i-1)$th iteration of greedy peeling, the set of remaining ground elements that needs to be covered is:

$$P' = E - \cup_{j=1}^{i-1} P_j. \tag{6.1}$$

We wish to find out how the set of ground elements in P' covered by the ith iteration of greedy peeling relates to $\mathcal{P}_k^*$.

We first observe that, after the $(i-1)$ iterations of greedy peeling, the remaining part of $\mathcal{P}_k^*$ in P' is

$$\mathcal{P}_k^* \cap P' \supseteq \mathcal{P}_k^* - \cup_{j=1}^{i-1} P_j. \tag{6.2}$$

On the other hand, without loss of generality, we can assume that $\mathcal{P}_k^* = \cup_{i=1}^{\lceil \frac{k}{l} \rceil} P_i^*$, where $P_i^* \subseteq E$ is a subset of ground elements that can be covered by l clusters, and $P_i^* \cap P_j^* = \emptyset$ if $i \neq j$. Then,

$$\mathcal{P}_k^* \cap P' = (\cup_{i=1}^{\lceil \frac{k}{l} \rceil} P_i^*) \cap P' = \cup_{i=1}^{\lceil \frac{k}{l} \rceil} (P_i^* \cap P'). \tag{6.3}$$

Combining (6.2) and (6.3), we have the following inequality:

$$\cup_{i=1}^{\lceil \frac{k}{l} \rceil} P_i^* \cap P' \supseteq \mathcal{P}_k^* - \cup_{j=1}^{i-1} P_j.$$

By the pigeonhole principle, there must exist a x, $1 \leq x \leq \lceil \frac{k}{l} \rceil$ such that

$$|P_x^* \cap P'| \geq \frac{1}{\lceil \frac{k}{l} \rceil}(|\mathcal{P}_k^*| - \sum_{j=1}^{i-1} |P_j|).$$

Because each iteration of greedy peeling chooses a maximal set of ground elements that can be covered by l clusters within performance ratio ϵ, the set chosen by the ith iteration of greedy peeling is

$$P_i \geq \epsilon(P_x^* \cap P').$$

Therefore

$$|P_i| \geq \frac{\epsilon}{\lceil \frac{k}{l} \rceil}(|\mathcal{P}_k^*| - \sum_{j=1}^{i-1} |P_j|).$$

□

Now the proof of Theorem 6.3.1 consists of some algebraic manipulations.

Proof of Theorem 6.3.1. Let

$$w_i = \frac{\epsilon}{\lceil \frac{k}{l} \rceil}(|\mathcal{P}_k^*| - \sum_{j=1}^{i-1} w_j)$$

for $i > 0$ with $w_0 = 0$. Then we have

$$w_i = (\frac{\epsilon}{\lceil \frac{k}{l} \rceil})(1 - \frac{\epsilon}{\lceil \frac{k}{l} \rceil})^{i-1} |\mathcal{P}_k^*|.$$

Thus

$$\sum_{i=1}^{\lceil \frac{k}{l} \rceil} w_i = (1 - (1 - \frac{\epsilon}{\lceil \frac{k}{l} \rceil})^{\lceil \frac{k}{l} \rceil})|\mathcal{P}_k^*|.$$

So the performance ratio is:

$$\gamma \geq 1 - (1 - \frac{\epsilon}{\lceil \frac{k}{l} \rceil})^{\lceil \frac{k}{l} \rceil}.$$

Hence, the theorem is proved. □

We are now ready to make some observations.

- When $\lceil \frac{k}{l} \rceil = 1$, then $\gamma = \epsilon$ holds.
- Note that γ is a decreasing function. Since

$$lim_{x \to \infty}(1 - \frac{a}{x})^x = (\frac{1}{e})^a,$$

  γ is bounded by $1 - (\frac{1}{e})^\epsilon$. When $\epsilon = 1$, then γ is bounded by $1 - \frac{1}{e} = 0.632$.
- The derivative of the performance ratio with respect to ϵ is

$$\frac{\partial \gamma}{\partial \epsilon} \geq (1 - \frac{\epsilon}{\lceil \frac{k}{l} \rceil})^{(\lceil \frac{k}{l} \rceil - 1)}.$$

  If $\lceil \frac{k}{l} \rceil = 1$, then $\frac{\partial \gamma}{\partial \epsilon} = 1$. If $\lceil \frac{k}{l} \rceil \to \infty$, then $\frac{\partial \gamma}{\partial \epsilon} \geq e^{-\epsilon}$.
- The derivative of the performance ratio with respect to $\lceil \frac{k}{l} \rceil$ is

$$\frac{\partial \gamma}{\partial \lceil \frac{k}{l} \rceil} \geq \frac{\epsilon}{(\lceil \frac{k}{l} \rceil)^2}(1 - \frac{\epsilon}{\lceil \frac{k}{l} \rceil})^{\lceil \frac{k}{l} \rceil} ln(1 - \frac{\epsilon}{\lceil \frac{k}{l} \rceil}).$$

6.4 Performance of Greedy Peeling for Exact Covering

In this section, we analyze the performance of greedy peeling for exact covering¶. The main result is as follows.

¶This work is an application of the result by Johnson [69], Lovász [96] and Chvátal [26]. Their results on set-covering are known for a long time; however, such results have not been applied to the analysis of VLSI heuristics before.

Theorem 6.4.1 *The performance ratio of greedy peeling for the minimum covering problem is $\frac{1}{\log|P|}$, where P is the largest cluster.*

Proof. Let the set E of ground elements to be covered be $\{1, ..., m\}$, and let the given set of clusters be P_j, $j = 1, ..., n$. Let t denote the number of iterations required by greedy peeling for solving the minimum covering problem. Let P_j^r denote the set P_j at the beginning of iteration r, $r = 1, ..., t$; for typographical simplicity, we denote the size of P_j^r by w_j^r.

We assume that the cluster picked up by the greedy peeling heuristic at iteration r is P_r^r.

At iteration r, all the elements in P_r^r are peeled off from all the other clusters. Thus, each $i \in E$ belongs to precisely one of the sets P_r^r with $r = 1, 2, ..., t$. For this r associated with element i, the price, y_i, paid for covering element i can be written as

$$y_i = \frac{c_r}{w_r^r}.$$

From the peeling-off property of greedy peeling, we have $P_j \cap P_r^r = P_j^r - P_j^{r+1}$. Thus

$$\begin{aligned}\sum_{i=1}^{m} a_{ij} y_i &= \sum_{r=1}^{t} \Big(\sum_{i \in P_j \cap P_r^r} y_i \Big) \\ &= \sum_{r=1}^{t} \frac{(w_j^r - w_j^{r+1}) c_r}{w_r^r} \\ &= \sum_{r=1}^{s} \frac{(w_j^r - w_j^{r+1}) c_r}{w_r^r},\end{aligned}$$

where $s \leq t$ is the largest superscript such that $w_j^s > 0$.

From the greedy property of greedy peeling,

$$\frac{c_r}{w_r^r} \leq \frac{c_j}{w_j^r},$$

we have,

$$\sum_{i=1}^{m} a_{ij} y_i \quad \leq \quad c_j \sum_{r=1}^{s} \frac{(w_j^r - w_j^{r+1}) c_r}{w_j^r}.$$

Noticing that,

$$\sum_{r=1}^{s} \frac{w_j^r - w_j^{r+1}}{w_j^r} \leq \sum_{r=1}^{s} (H(w_j^r) - H(w_j^{r+1})) = H(w_j^1),$$

where

$$H(x) = \sum_{j=1}^{x} \frac{1}{j},$$

and

$$w_j^1 = |P_j| = \sum_{i=1}^{m} a_{ij},$$

we have,

$$\sum_{i=1}^{m} a_{ij} y_i \leq c_j H(\sum_{i=1}^{m} a_{ij})$$

for all j such that

$$\sum_{i=1}^{m} y_i = \sum_{j \in J^*} c_j.$$

Let $x = (x_j)$ be the binary incidence vector $x = (x_j)$ for an arbitrary cover, which satisfies

$$\sum_{j=1}^{n} a_{ij} x_j \geq 1$$

for all i, then

$$\begin{aligned} \sum_{j \in J^*} c_j &= \sum_{i=1}^{m} y_i \\ &\leq \sum_{i=1}^{m} (\sum_{j=1}^{n} a_{ij} x_j) y_i \\ &= \sum_{j=1}^{n} (\sum_{i=1}^{m} a_{ij} y_i) x_j \\ &\leq \sum_{j=1}^{n} H(\sum_{i=1}^{m} a_{ij}) c_j x_j. \end{aligned}$$

Since $H(x) = \log x$, if we take x to be the incidence vector of an optimal cover, then the theorem is proved. □

In view of a very recent result by Lund and Yannakakis [99], namely that, for any $0 < c < 1/4$, the set-cover problem cannot be approximated within ratio of $c \log p$ in polynomial time unless $NP \subset DTIME(p^{poly \log p})$—a very unlikely event—the greedy peeling heuristic is likely the best-possible approximation algorithm for the optimization problems of the cluster-cover structure.

6.5 Theoretical Justification of Previous Empirical Results

In this section, we examine some experimental results reported in the literature on the use of greedy peeling in the two VLSI design applications. In particular, we show how our theoretical analysis is confirmed by empirical evidence.

Table 6.2 shows experimental results reported by Iyengar and Vijayan for application timing assignment [67]. Iyengar and Vijayan made the following observation: *For some examples, there are relatively large difference in the number of assignments required for $\alpha = 1.1$ and $\alpha = 1$. In other words, while the slack lower bound of a huge fraction of the fault sites is achieved by the first few assignments, a large number of new assignments may be needed to achieve the lower bound for the remaining few sites.*

Another remark made by Iyengar and Vijayan is that the number of assignments does not seem to grow with the size of combinational circuits. This is true from our analysis, since the number of assignments only relates to the size of the maximum independent set of the compatibility graph.

At the end of their paper, Iyengar and Vijayan posed the following two open questions: *Can the TAT_multiple heuristic be improved to bring the number of assignments for $\alpha = 1$ closer to that for $\alpha = 1.1$? Is there a good lower bound for*

Table 6.2: Experimental results of application timing assignments.

			#assignments to achieve		
circuit	#edges	#fault sites	$\alpha = 1.1$	$\alpha = 1.05$	$\alpha = 1$
c432	225	432	1	1	1
c499	1312	499	3	3	11
c880	419	880	4	5	5
c1355	1312	1355	5	7	9
c1908	807	1908	1	1	1
c2670	1143	2746	4	5	5
c3540	724	3540	3	5	15
c5315	2978	5315	5	5	8
c6288	784	6283	5	5	8
c7522	3544	7553	4	5	10

the number of assignments necessary to achieve $\alpha = 1$. We basically settle the two open questions. For the first question, because greedy peeling has been proved to be almost optimum, and because its error cannot be less than the logarithm of its input size, it is very unlikely that the heuristic can be improved to bring the number of assignments for $\alpha = 1$ closer to that for $\alpha = 1.1$. For the second question, our prime covering paradigm indeed provides a way of obtaining a good lower bound, even an optimum solution, for the number of assignments necessary to achieve $\alpha = 1$.

Table 6.3 shows experimental results reported by Cong, Hossain and Sherwani on multi-layer topological planar routing [31]. They made the following interesting observations. *First, we can have a planar routing for the majority of nets. ... Given a relatively large number of routing layers (say, more than four layers), we can route most of the nets without vias. Second, insisting on planar routing for all the nets is very costly, i.e., it requires a large number of routing layers. Although we can have* *planar routing for over 60% of the nets in the first five layers, we need 4-13 layers to route the remaining 20-40% of the nets. Therefore, it is unrealistic to insist on planar routing for all the nets.*

Table 6.3: Experimental results of multi-layer topological planar routing.

circuit	1L	2L	3L	4L	5L	total layer
bus	41%	58%	70%	79%	83%	9
ex1	38%	52%	66%	76%	80%	9
ex3a	45%	59%	68%	75%	79%	11
ex3b	31%	53%	68%	75%	79%	10
ex3c	37%	55%	62%	70%	75%	13
ex4b	41%	57%	68%	75%	81%	13
ex5	31%	48%	59%	70%	83%	9
ex5b	31%	48%	60%	71%	79%	11
deut	23%	38%	50%	58%	63%	18

We note that the second observation made by Cong, Hossain and Sherwani is not exact. The cost was caused by their greedy peeling algorithm. It may be the case that the number of layers really needed for planar routing of all the nets is not that large. It will be interesting to apply prime covering to their problem for optimum solutions.

6.6 Summary

In this chapter, a framework called cluster-cover has been established to capture a class of NP-hard optimization problems in VLSI design, including combinational logic minimization, constrained encoding for sequential logic synthesis, multilayer topological channel routing, application timing for delay fault testing, and monitoring circuit design for BIST enhancement. Two paradigms, called prime covering and greedy peeling, were described as abstractions of various previous cluster-covering algorithms. A theoretical analysis of the performance of greedy peeling has been provided. This analysis is applicable to a class of published heuristics which previously could only be evaluated with respect to benchmarks.

Chapter 7

Mathematical Programming Formulation of Maximum Balance

Mathematical programming is a generic approach to a certain type of optimization problem that can be explicitly expressed as a task of maximizing or minimizing an objective function subject to a set of mathematical inequality constraints. It was originally developed in the area of operations research, and has been a useful tool for VLSI optimization, but was traditionally restricted to problems, such as circuit optimization, that do not have any explicit structural properties to exploit in deriving efficient solution.

Recently, mathematical programming techniques have attracted an increasing interest from the VLSI design automation community. This is mainly due to the ever increasing demand to reduce the "time-to-market" of a VLSI design and to improve the performance. We have to consider various constraints—due to timing, performance, power consumption, and testability—simultaneously when we synthesize the circuits. Generally, it is impossible to characterize those constraints by combinatorial structures. For this type of constrained optimization problems, traditional combinatorial optimization techniques alone are no longer sufficient, and mathematical programming is a natural, maybe indispensable, alternative.

At the same time, significant progress has been made in in the theory of computation. A major result is Karmarkar's polynomial-time algorithm for linear programming [75]. Inspired by this discovery, researchers have put considerable effort into formulations of combinatorial optimization problems in the framework of linear programming*. Since most such problems are NP-hard, in the worst case, the number of linear inequalities required by a linear programming characterization is likely exponential. Nevertheless, a whole new branch of combinatorial mathematics— polyhedral combinatorics—has been created, with the generation of as "compact" as possible linear programming characterizations as the central theme. For recent results of this discipline, see the work of Pulleyblank [119]. In another direction, Karmarkar's algorithm has been extended directly to handle integer programming [76, 138].

In this chapter, we investigate how to solve the maximum balance problem by mathematical programming. With the practical motivation for performance-driven layer assignment as described in Chapter 8, our emphasis here is to search for a good linear programming characterization of the maximum balance problem in planar signed graphs. We first give in Section 7.1 an integer linear programming formulation for maximum balance in general signed hypergraphs. We then derive in Section 7.2 a linear programming formulation for planar signed graphs, by using a fundamental theorem of polyhedral combinatorics. Furthermore, we present several reduction techniques in Section 7.3; these lead to a linear programming formulation consisting of only a polynomial number of inequality constraints.

*A precise historical review of this subject can be found in the preface of the book by Lovász and Plummer [97].

7.1 Integer Linear Programming Formulation

In this section, we introduce an integer linear programming formulation for the maximum balance problem in signed hypergraphs. Consider a signed hypergraph $H = (V, E, \psi)$, consisting of n vertices and m edges; we are looking for a bipartition $\pi = (V^+, V^-)$ of V such that the number of balance edges is maximized.

Given a signed hypergraph H and a bipartition π, let us introduce a *bipartition vector* $\mathbf{x} \in \{0,1\}^n$ such that

$$x_i = \begin{cases} 1 & \text{if } v_i \in V^+, \\ 0 & \text{if } v_i \in V^-, \end{cases}$$

and a *balance vector* $\mathbf{y} \in \{0,1\}^m$ such that

$$y_e = \begin{cases} 1 & \text{if edge } e \text{ is balanced by } \pi, \\ 0 & \text{otherwise.} \end{cases}$$

Let V_e^+ represent the set of vertices positively incident with edge e, and let V_e^- represent the set of vertices negatively incident with edge e. Then edge e is balanced by the partition π if and only if $V_e^+ \subseteq V^+$ and $V_e^- \subseteq V^-$, or $V_e^+ \subseteq V^-$ and $V_e^- \subseteq V^+$. This observation can be expressed as

$$y_e = \prod_{v_i \in V_e^+, v_j \in V_e^-} x_i(1 - x_j) + \prod_{v_i \in V_e^+, v_j \in V_e^-} (1 - x_i)x_j. \tag{7.1}$$

Then the maximum balance problem is to find $\mathbf{x} \in \{0,1\}^n$ and $\mathbf{y} \in \{0,1\}^n$ so as to maximize $\sum_e y_e$ subject to (7.1).

To convert the integer nonlinear programming formulation above into an integer linear programming problem, we define

$$y_e^+ = \prod_{v_i \in V_e^+, v_j \in V_e^-} x_i(1 - x_j), \tag{7.2}$$

$$y_e^- = \prod_{v_i \in V_e^+, v_j \in V_e^-} (1 - x_i)x_j. \tag{7.3}$$

There two conditions are equivalent to the following four set of conditions:

$$y_e^+ \leq \quad x_i \quad \forall v_i \in V_e^+, \tag{7.4}$$

$$y_e^+ \leq \; 1 - x_j \quad \forall v_j \in V_e^-, \tag{7.5}$$

$$y_e^- \leq \; 1 - x_i \quad \forall v_i \in V_e^+, \tag{7.6}$$

$$y_e^- \leq \quad x_j \quad \forall v_j \in V_e^-, \tag{7.7}$$

for each $e \in E$. These inequalities are called *vertex-edge incidence constraints.*

The maximum balance problem can now be formulated as the following linear integer programming problem:

$$\mathit{maximize} \qquad \sum_e (y_e^+ + y_e^-), \tag{7.8}$$

subject to the vertex-edge incidence constraints (7.4)-(7.7). It consists of $n + 2m$ integer variables and $2p$ constraints.

7.2 Linear Program for Planar Signed Graphs

In this section, we derive a linear programming formulation for the maximum balance problem in planar signed graphs. We first set up an integer linear program for signed graphs and then reduce it to a linear program (without the integrality restriction) by applying both a fundamental theory of polyhedral combinatorics and the planarity property of planar signed graphs.

Consider a signed graph $G = (V, E, \psi)$. Suppose that $\pi = (V^+, V^-)$ is a bipartition, and $C \subseteq E$ is the corresponding cut, i.e., a set of edges such that each edge joins a vertex in V^+ with a vertex in V^-. We associate a *cut* variable z_e with each edge $e \in E$ as follows:

$$z_e = \begin{cases} 1 & \text{if } e \in C, \\ 0 & \text{otherwise.} \end{cases} \tag{7.9}$$

A feasible solution to the maximum balance problem corresponds to a vector $\mathbf{z}$ in $R^{|E|}$ that defines a cut, where R is the set of real numbers. Let $\mathcal{Q}$ be the set of all the cycles in G; then vector $\mathbf{z} \in R^{|E|}$ defines a cut, if and only if

$$z_e \in \{0, 1\}, \tag{7.10}$$

for all $e \in E$, and

$$\sum_{e \in Q} z_e \equiv \ 0 \ (\text{mod } 2). \tag{7.11}$$

for all $Q \in \mathcal{Q}$. Here (7.11) states that each cycle must be cut by a bipartition an even number of times.

In a signed graph, a positive edge is an edge with either two positive vertex-edge incidences, or two negative incidences; a negative edge is an edge with one positive vertex-edge incidence and one negative incidence. Let E^+ (E^-) denote the set of positive (negative) edges in E; then edge e is balanced by a bipartition if and only if

$$z_e = 0 \qquad \text{for } e \in E^+,$$
$$z_e = 1 \qquad \text{for } e \in E^-.$$

The maximum balance problem is to find $\mathbf{z} \in R^{|E|}$ for an integer linear program defined by

$$\mathit{maximize} \qquad \sum_{e \in E^+} w_e(1 - z_e) + \sum_{e \in E^-} w_e z_e,$$

subject to (7.10) and (7.11), where w_e is a real weight associated with edge e.

We know from Chapter 3 that the maximum balance problem for a planar signed graph reduces to the minimum T-join problem in its planar dual. Then, according to Edmonds and Johnson [42], and noticing that cycles and cuts are exchangeable under planar duality, (7.10)-(7.11) can be replaced by the following set of linear inequalities:

$$\sum_{e \in Q-U} z_e + \sum_{e \in U} (1 - z_e) \geq 1, \tag{7.12}$$

where Q is a cycle in $\mathcal{Q}$, $U \subseteq Q$ and $|U|$ is odd, and $Q - U$ denotes edges in Q but not in U. The inequalities in (7.12) are called *blossom inequalities.*

In summary, the maximum balance problem in a planar signed graph is formulated as the following linear program:

$$\textit{maximize} \qquad \sum_{e \in E^+} w_e(1 - z_e) + \sum_{e \in E^-} w_e z_e,$$

subject to

$$z_e \geq 0, \tag{7.13}$$

and

$$\sum_{e \in Q-U} z_e + \sum_{e \in U}(1 - z_e) \geq 1, \tag{7.14}$$

where $Q \in \mathcal{Q}$, $U \subseteq Q$ and $|U|$ is odd.

Any linear program can be solved in polynomial time in the size of the program [75]. However, the number of blossom inequalities in our formulation above may be exponential in terms of the graph size, as seen from the following proposition.

Proposition 7.2.1 *For a cycle of length k, the number of blossom inequalities in (7.14) is 2^{k-1}.*

7.3 Reduction of Blossom Inequalities

In this section, we show that the number of blossom inequalities in (7.14) can be reduced to be polynomial in the size of G. This result, together with the fact that a linear program can be solved in polynomial time, leads to another polynomial-time algorithm for solving the maximum balance problem in planar signed graphs[†]. Note

[†]One such polynomial-time algorithm for maximum balance in planar signed graphs was given in Chapter 3 via the graph T-join concept.

that planar maximum balance includes planar maximum cut, which was solved in polynomial time by Hadlock [57], and Lipton and Tarjan [94].

The reduction of blossom inequalities is accomplished by using three techniques, namely graph simplification, graph triangulation, and redundance elimination. We first describe the graph simplification technique. A signed graph that arises from the optimum layer assignment problem may contain the following graph structures:

- *Loops*: Edges that connect to only one vertex;
- *Parallel edges*: Two or more edges that join the same two vertices;
- *Series edges*: Two or more edges connected in series, i.e., that form a single path between two vertices;
- *Cut edges*: Edges with the following property: The removal of such an edge leads to a graph with more connected components. For example, the two edges indicated by dashed lines in Fig. 7.1 are cut edges.

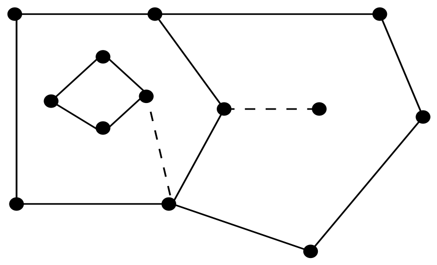


Figure 7.1: Illustration of cut edges.

For the maximum balance problem, such a signed graph can be simplified as follows:

- Loops: A loop can be positive or negative. A positive loop will always be balanced by any bipartition; it thus can be ignored. A negative loop will never be balanced by any bipartition; it thus can be removed from the consideration with an adjustment to the objective function.

- Parallel edges: A set E_{ij} of parallel edges joining vertices i and j can be replaced by a positive edge joining i and j with a weight equal to the sum of the weights of positive edges minus the sum of the weights of negative edges. The reason for this is as follows: All the edges e in E_{ij} are either cut ($z_e = 1$), or uncut ($z_e = 0$); thus all the cut variables z_e, $e \in E_{ij}$, can be represented by a single variable z_{ij}. This leads to $|E_{ij}|$ identical equations obtained from (7.11); these equations can be replaced by a single one. The part of the objective function contributed by edges in E_{ij} is

$$\sum_{e\in E_{ij}^+} w_e(1 - z_e) + \sum_{e\in E_{ij}^-} w_e z_e = (\sum_{e\in E_{ij}^+} w_e - \sum_{e\in E_{ij}^-} w_e)(1 - z_{ij}) + \sum_{e\in E_{ij}^-} w_e.$$

  This means that all the edges in E_{ij} can be replaced by a positive edge weighted $(\sum_{e\in E_{ij}^+} w_e - \sum_{e\in E_{ij}^-} w_e)$ and an adjustment (addition) of $\sum_{e\in E_{ij}^-} w_e$ to the objective function.
- Series edges: A set E_{ij} of series edges connecting vertices i and j can be replaced by a single edge joining i and j with a weight equal to the smallest of the weights of all series edges, and with a sign equal to the product of the signs of all the series edges. The reason for this is as follows: The contribution of all the edges in E_{ij} to the sign of the related cycles is the product of the signs of all the edges in E_{ij}. Further, we can remove the edge with the smallest weight to break the cycles involving E_{ij}.
- Cut edges: It is trivial to see that cut edges can be ignored for the maximum balance problem.

In practice, the formulation of the optimum layer assignment problem gives rise to a signed multigraph, especially in the case of large clusters (See Chapter 8). Therefore the use of the simplification technique above usually results in a substantial reduction.

The *graph triangulation* technique is motivated by Proposition 7.2.1—the number of blossom inequalities induced by a cycle is exponential in the length of the cycle. This suggests triangulation of a given planar signed graph by adding certain zero-weighted edges so that each face is enclosed by exactly three edges. Figure 7.2 illustrates an example where dashed lines represent the added zero-weighted edges.

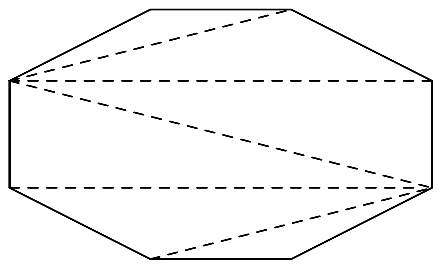


Figure 7.2: Graph triangulation for blossom inequality reduction.

We now turn to the *redundance elimination* technique. We need to introduce the following concepts. Given a set of linear inequalities, its solution set is called a *polyhedron*. If the addition of an inequality c into a set C of linear inequalities does not change the polyhedron defined by the set C, then the inequality c is *redundant*. We also say that inequality c is implied by the set C. Redundance elimination is based on the following observation:

Lemma 7.3.1 *For a connected planar signed graph, all the blossom inequalities are implied by blossom inequalities for bounded faces.*

Proof. We first consider a cycle enclosing two faces, and then extend the result to a cycle enclosing an arbitrary number of faces.

Consider two faces f_1 and f_2 that are adjacent with respect to an edge e, as illustrated in Fig. 7.3(a)[‡]. We also use f_1 (f_2) to denote the cycle that forms the

[‡]Here we make use of a property of triangulated graphs to have a conceptually simple proof. In general, this lemma is applicable to non-triangulated signed graphs. For such graphs, however, two faces may be adjacent with respect to two or more edges as illustrated in Fig. 7.3(b); this complicates the proof.

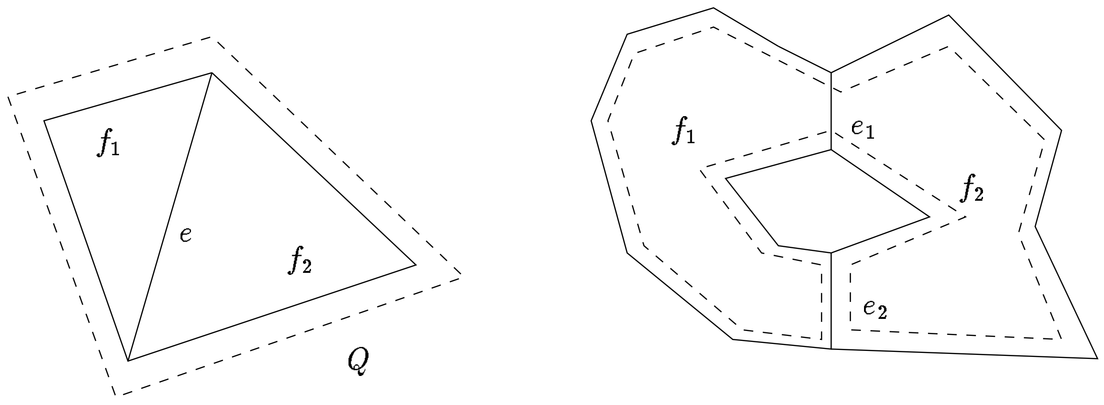


Figure 7.3: Illustration in the proof of Lemma 7.3.1.

boundary of f_1 (f_2). Let Q be the cycle enclosing f_1 and f_2, i.e., $Q = f_1 + f_2 - e$. Let $U_1 \subseteq f_1 - e$ and $U_2 \subseteq f_2 - e$. Then $U_1 + U_2 \subseteq Q$.

We write blossom inequalities associated with face f_1 as follows:

$$\sum_{i \in f_1 - U_1 - e} z_i + \sum_{i \in U_1} (1 - z_i) - z_e \geq 0, \tag{7.15}$$

for all U_1 such that $|U_1|$ is even, and

$$\sum_{i \in f_1 - U_1 - e} z_i + \sum_{i \in U_1} (1 - z_i) + z_e \geq 1, \tag{7.16}$$

for all U_1 such that $|U_1|$ is odd.

Similarly, for face f_2, we have:

$$\sum_{i \in f_2 - U_2 - e} z_i + \sum_{i \in U_2} (1 - z_i) - z_e \geq 0, \tag{7.17}$$

for all U_2 such that $|U_2|$ is even, and

$$\sum_{i \in f_2 - U_2 - e} z_i + \sum_{i \in U_2} (1 - z_i) + z_e \geq 1, \tag{7.18}$$

for all U_2 such that $|U_2|$ is odd.

Combining (7.15) with (7.18), and (7.16) with (7.17), we have

$$\sum_{i \in f_1+f_2-U_1-U_2-e} z_i + \sum_{i \in U_1+U_2} (1-z_i) \geq 1, \tag{7.19}$$

for all such $U_1 + U_2$ that $|U_1 + U_2|$ is odd. This is precisely the set of blossom inequalities for cycle Q.

Now we consider a cycle enclosing an arbitrary number of faces. Given the set of blossom inequalities for G, if we replace the set of blossom inequalities associated with the two faces f_1 and f_2 by the blossom inequalities for the cycle Q, this leads to the set of blossom inequalities for $G(E-e)$. This is equivalent to merging two faces f_1 and f_2 into one face Q. In this sense any cycle can be solved by repeatly considering the case of combining two faces. Therefore the lemma is proved. □

Lemma 7.3.1 states that it is necessary to construct blossom inequalities only for the bounded faces, and not for all the cycles. Since the number of faces is linear in terms of the graph size, whereas the number of cycles is exponential, the use of redundance elimination leads to an exponential reduction.

In summary, we have the following main result:

Theorem 7.3.1 *For a connected planar signed graph with n vertices and an exterior of length k, there exists a linear programming formulation of the maximum balance problem that consists of $3n-k-3$ variables, $8n-4k+8$ blossom inequalities, and $32n-16k+32$ nonzero elements.*

Proof. For a connected planar signed graph with n vertices, we can apply the graph simplification technique to yield a simple signed graph. Then, by graph triangulation, we obtain a triangulated graph with n vertices and exterior of length k. The triangulated graph has $3n-k-3$ edges with $2n-k+2$ bounded faces. Therefore we have $3n-k-3$ variables. By Lemma 7.3.1 (redundance elimination), we only need blossom inequalities for bounded faces. From Euler's formula, for a planar graph with n vertices and m edges, there exist $m - n + 1$ bounded faces. Since there are $2^{3-1} = 4$ blossom inequalities for each face, we have $8n - 4k + 8$ blossom inequalities in total. Because each inequality has 4 nonzero elements, the total number of nonzero elements is $32n - 16k + 32$. $\Box$

Graph triangulation makes a graph dense; however, the linear programming formulation turns out be even sparser, as indicated by the following lemma.

Proposition 7.3.1 *The linear program for a triangulated planar signed graph is always sparser than that of the original planar signed graph.*

Proof. For the triangulated graph, each blossom inequality gives rise to 4 nonzero elements. There are $3n - k - 3$ variables. Thus the sparsity of the linear program is $4/(3n - k - 3)$. For the original graph, the number of variables is the number of edges in the original graph, which is smaller than $3n - k - 3$. Each blossom inequality induces at least 4 nonzero elements. Thus the linear program for the original graph is at least as dense as the linear program for the triangulated graph. $\Box$

For example, consider a cycle of $k = 10$ edges. The formulation for the original graph requires $k = 10$ variables and $2^{k-1} = 512$ constraints. The total number of nonzero elements is $2^{k-1}k = 5120$. The sparsity is 1. The new formulation yields $2k - 3 = 17$ variables and $4(k - 2) = 32$ constraints. The total number of nonzero elements is $4(k - 2)2^{3-1} = 128$. Thus the sparsity is $\frac{128}{32 \times 17} = \frac{4}{17} < 0.25$.

7.4 Summary

In this chapter, an integer linear programming formulation has been presented for the exact solution of the maximum balance problem in signed hypergraphs. For planar signed graphs, a polynomial-size linear programming formulation was derived, based on the theory of polyhedral combinatorics, and the planarity property. Combining this with any polynomial-time algorithm (such as Karmarkar's algorithm [75]) for linear programs, we have a polynomial-time algorithm for the maximum balance problem in planar signed graphs.

The mathematical programming approach offers two advantages. First, it can take into account additional constraints naturally, and thus provides a basis to performance-driven layer assignment as described in Chapter 8. Second, it has the same complexity for the weighted version of the problem. We can extend our polynomial-size linear programming formulation for planar signed hypergraphs, by applying the graph estimation technique described in Section 4.4 to approximate a planar signed hypergraph by a planar weighted signed graph. The solution provides a tight lower bound, which can be used to evaluate the quality of various heuristics.

Chapter 8

Application to Layout Wiring

The optimum layer assignment problem was first addressed for Manhattan-type channel routing in [60] and has been studied extensively in the past two decades. For a simple-split routing—where each potential via connects at most three wire segments, Pinter [117], and also Chen, Kajitani and Chan [23], have shown that the problem can be reduced to a maximum cut problem in a planar graph. An $O(n^{\frac{3}{2}} \log n)$ algorithm (where n is the number of potential vias) was developed later [6, 87]. Very few results were known about the general problem, perhaps because of the fact that standard graph theory is not a convenient framework. Some ad hoc efforts towards extending the graph notation were made [21, 23], but none of them leads to a proper formulation. The complexity issue was open for a long time, until Choi, Nakajima and Rim [24] showed that the problem is NP-complete.

Another concern, especially for very large scale integrated (VLSI) circuits, is the delay introduced by interconnect wiring. Wiring delay is a dominating factor limiting the performance of VLSI digital circuits. An important factor that determines the wiring delay is the layer assignment of wire segments. Because the sheet resistance of the polysilicon layer is greater than that of the metal layer ($0.03\Omega/\Box$ for metal versus 20-50$\Omega/\Box$ for polysilicon for 2-um CMOS process) [155], there exists a significant difference in delay performance between a wire assigned to the metal layer and the same wire assigned to the polysilicon layer. This gives rise to layer preference constraints in practice. For example, some nets, such as power supply and clock lines, or some terminals of the circuit modules, may be preassigned to the metal layer [155]. Even if all wiring layers have the same conductivity, layer assignment still affects the delay performance. This is because pure via minimization may lead to the congestion of vias at a critical net, thus degrading the system performance. Therefore, it is desirable to consider timing performance during the layer assignment process; this is called performance-driven layer assignment.

We show in this chapter that the optimum layer assignment problem is equivalent to the maximum balance problem in a planar signed hypergraph. Therefore all the results obtained in this thesis for planar signed hypergraphs are applicable to optimum layer assignment. This implies a simple proof of the NP-completeness of the optimum layer assignment problem (Chapter 2), a more efficient polynomial-time algorithm for the restricted case (Chapter 3), a pseudo-polynomial-time algorithm for the general case (Chapter 3), and an approximation algorithm for the general case (Chapter 4). Our emphasis in this chapter is to formulate performance-driven layer assignment and to describe our experimental program *POLAR2* for *P*erformance-*O*riented *L*ayer *A*ssignment of two-layer *R*outings.

This chapter is structured as follows: In Section 8.1, we show formally that the optimum layer assignment problem is equivalent to the maximum balance problem in a planar signed hypergraph. In Section 8.2, we present a linear inequality formulation of timing constraints for performance-driven layer assignment. In Sections 8.3 and 8.4, we describe our program *POLAR2* and some experimental results.

8.1 Signed Hypergraph Model of Layer Assignment

We describe the optimum layer assignment problem and its formulation by the example of Fig. 8.1(a). We are given a set P of *points* ($t_1, \ldots, t_{11}$ and $p_1, \ldots, p_5$ in the figure) in a plane, which are either *terminals* (t_i) or *potential vias* (p_i). We are also given a set N of *(wire) nets*, each net being a wire in the plane with multiple end-points and connecting several points from P. For example, $\{t_7, t_9, p_4\}$ represents a net. Two nets may *cross* each other (e.g., $\{t_7, t_9, p_4\}$ and $\{t_8, p_1\}$); this defines a *crossing relation* X on N. The *optimum (two-) layer assignment* problem* is to assign nets to the two layers in such a way that (1) no crossing nets appear in the same layer, and (2) the number of vias that connect nets assigned to different layers is minimized. As is usually done in practice, *we assume that there does exist a solution that satisfies condition (1).*

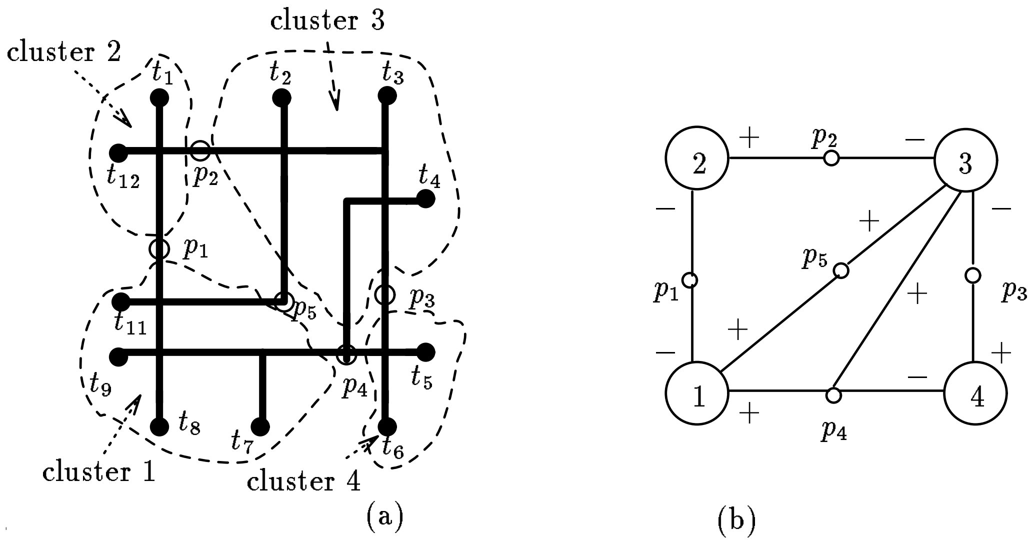


Figure 8.1: (a) A routing (terminals — •; potential vias — ∘). (b) Signed hypergraph.

Clearly, two nets that cross each other should be assigned to different layers.

*The routing model implied by our mathematical framework is simple, concise and general. It contains the Manhattan routing, the knock-knee routing, and the more general gridless routing.

Similarly, two nets that both cross a common third net should be assigned to the same layer. Such rules can be captured conveniently by the *cluster relation* C on N, which is defined to be the reflexive-and-transitive closure of the crossing relation X. Since the crossing relation is symmetric, the cluster relation is an equivalence relation on the set N of nets. Each equivalence class of C is called a *cluster* of nets. Once the layer assignment of any one net in a cluster is made, the layer assignment of all the other nets within the cluster is also determined. There are four clusters in Fig. 8.1(a).

By our assumption, in any cluster, it is possible to assign nets to the two layers so that no two nets in the same layer cross. Therefore, for each cluster, we arbitrarily choose one set of nets as *reference* nets and assign to them the positive polarity $+$, and the other set as *non-reference* nets, with negative polarity $-$.

A cluster can have electrical connections to other clusters only through potential vias. (For example, cluster 1 is connected to clusters 2, 3, and 4 through p_1, p_5, and p_4, respectively.) If we represent each cluster by a vertex, then each potential via corresponds to an edge. Since more than two clusters can meet at a potential via, an edge may connect more than two vertices, i.e., the natural concept required here is that of a hypergraph rather than just a graph. Furthermore, a potential via may be connected to a cluster through a positive or negative net; thus the hypergraph is naturally a signed hypergraph. For our example, Fig. 8.1(b) shows a signed hypergraph for the routing problem of Fig. 8.1(a). The ordinary graph obtained by treating edge nodes in the same way as the vertices of H is called the *underlying graph* of H.

We arbitrarily assign layer 1 to the all the positive nets and layer 2 to all the negative nets. It is clear that vias p_2, p_3, and p_4 are required for this choice of reference nets. The condition for not requiring a real via at the position of a potential via is that all the nets connected to that potential via have the same polarity. To find an optimal layer assignment we may have to interchange the reference nets with the non-reference nets in some of the clusters. For example, if we perform this interchange for clusters 2 and 4, only via p_1 is required in the resulting signed hypergraph.

In terms of the signed hypergraph of a routing, a solution assigning nets to the two layers corresponds to a partition of the vertices into two blocks. All vertices (clusters) for which the reference net assignment is the initial one are in one block of the partition, whereas those for which the complementary assignment is used are in the other block. If a bipartition balances an edge e, then all the vertices incident with e with one of the two polarities will have their reference net assignments interchanged. Consequently, no via will be required for that edge after the interchange. In other words, we want to partition all the vertices into two blocks so as to maximize the number of balanced edges.

By construction, the underlying graph for a given routing is planar. More precisely, it is a plane embedding of a planar graph. On the other hand, given a *planar* signed hypergraph H, one can always construct a two-layer routing in such a way that the maximum balance problem of H can be solved by the optimum layer assignment problem of that routing. Therefore, we have the following result.

Proposition 8.1.1 *The optimum layer assignment problem in a two-layer routing is equivalent to the maximum balance problem in a planar signed hypergraph.*

8.2 Formulation of Timing Constraints

Synchronous digital integrated circuits exhibit three types of timing problems: the long path problem, where the delay associated with a path is too long; the short path problem, where the path delay is too short; and the time skew problem, where the delays of the two paths are not matched. In this section, we show how to consider these timing problems within the allowable margin of layer assignment.

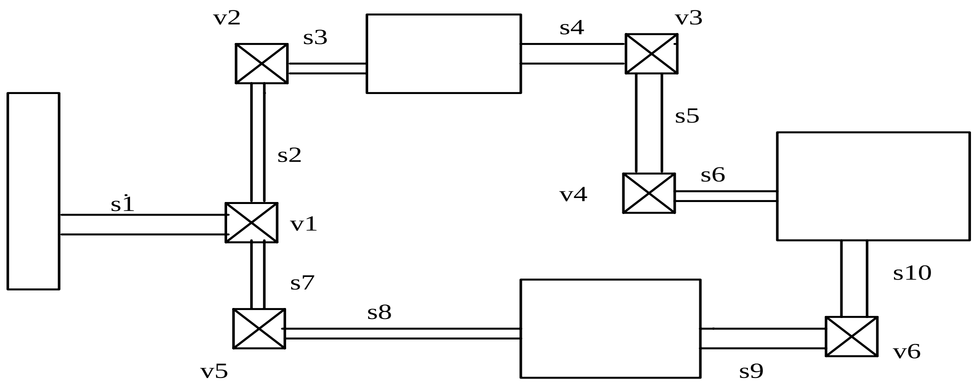


Figure 8.2: Part of a circuit layout.

8.2.1 The RC Network Model

We use the Resistor-Capacitor (RC) network model for delay characterization. Consider a circuit layout composed of a set of modules and wire segments, as shown in Fig. 8.2. Ignoring the fringing capacitance between wire segments, a well-accepted model for the delay characteristic of the circuit is a collection of RC trees, as shown in Fig. 8.3. Each RC tree corresponds to a routing net. It models delays contributed by modules, wire segments, and potential vias. In the following, we consider each separately.

First, the delay contribution of a module is characterized by a resistor R and a capacitor C at each terminal (or port). The values of such resistors and capacitors are independent of layer assignment.

Second, associated with each wire segment s_k is the lumped resistance, R^s_k, and stray capacitance, C^s_k. The value of R^s_k (C^s_k) depends upon actual layer assignment. Let z_k be defined by

$$z_k = \begin{cases} 1 & \text{if } s_k \text{ is assigned to layer } A, \\ 0 & \text{otherwise,} \end{cases}$$

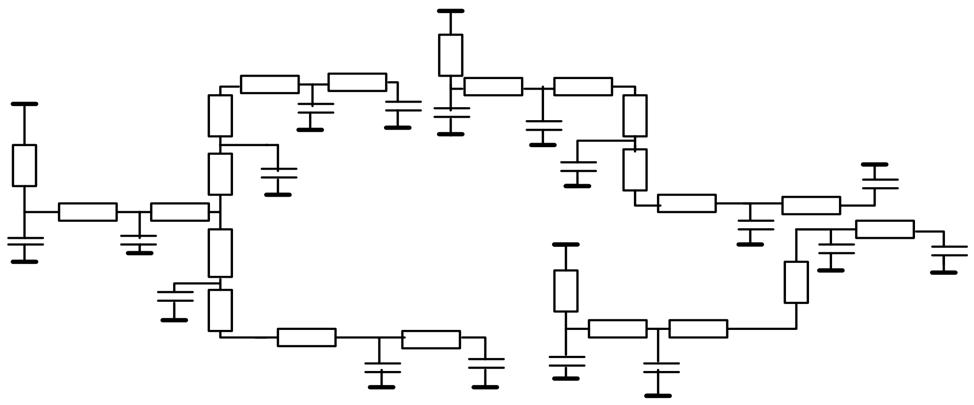


Figure 8.3: The RC network of the layout.

and let the cluster of s_k be $c(k)$. Then we have

$$z_k = \begin{cases} x_{c(k)} & \text{if } s_k \text{ belongs to a reference wire net of } c(k), \\ 1 - x_{c(k)} & \text{otherwise.} \end{cases} \tag{8.1}$$

The resistance and the capacitance of s_k can now be expressed as:

$$R_k^s = R_k^A z_k + R_i^B (1 - z_k) \tag{8.2}$$

$$C_k^s = C_k^A z_k + C_i^B (1 - z_k), \tag{8.3}$$

where R_k^A (C_k^A) is the resistance (capacitance) value if s_k is assigned to layer A, and R_k^B (C_k^B) if assigned to layer B.

Finally, we consider potential vias. Suppose that a signal passes from wire segment s_i to s_j through potential via v_k; then the resistance contributed by v_k to the signal propagation can be expressed as

$$R_k^v = r_k(z_i(1 - z_j) + (1 - z_i)z_j), \tag{8.4}$$

where r^k is the resistance when potential via v_k becomes a real via.

8.2.2 An Analytical Expression of Interconnect Delay

In this subsection, we consider how to calculate the delay for each RC tree. We use the Elmore delay as a delay measure [43, 140]. Since each block is an RC tree, the Elmore delay associated with each node in the RC tree can be computed by the TREE algorithm [120, 140]. Analytically, the delay at output node o can be expressed as

$$T_{Do} = \sum_{k(o)} R_k^s \sum_{j(k)} C_j^s + \sum_{k(o)} R_k^v \sum_{j(k)} C_j^s, \tag{8.5}$$

where the summations $\sum_{k(o)} R_k^s$ and $\sum_{k(o)} R_k^v$ extend over the resistances of those wire segments and potential vias that lie along the path from input pin to output pin o. The indices $k(o)$ refer to the corresponding segments and potential vias along the path. Indices $j(k)$ refer to those wire segments and potential vias that can be reached by the signal passing through wire segment s_k or potential via v_k.

Note that R_k^s and C_j^s are linear functions of $\mathbf{z}$, and R_k^v is a quadratic function of $\mathbf{z}$, and the delay at output node o yields a nonlinear expression of $\mathbf{z}$. In the following we show how it can be linearized. First, we observe that, for most MOS processes, capacitance per unit length of interconnect wire is relatively independent of the actual layer assignment. A typical value of capacitance per unit area, $C_\square$, in a 2-um CMOS or nMOS process, is $0.3 \times 10^{-4} pF/um^2$ for metal, and $0.5 \times 10^{-4} pF/um^2$ for polysilicon [155]. Thus, the term $\sum_{j(k)} C_j^s$ becomes constant for a given routing, regardless of the actual layer assignment. For the sake of notational simplicity, we denote the capacitance that can be reached by wire segment s_k by $C(s_k)$, and the capacitance that can be reached by potential via v_k by $C(v_k)$. Second, we introduce dummy variables y_{ij}^+ and y_{ij}^- as follows:

$$y_{ij}^+ = z_i(1 - z_j), \tag{8.6}$$

$$y_{ij}^- = (1 - z_i)z_j. \tag{8.7}$$

Thus we have

$$T_{Do} = \sum_{k(o)} R_k^A z_k + R_k^B (1 - z_k) C(s_k) + \sum_{ij(o)} (y_{ij}^+ + y_{ij}^-) C(v_i), \tag{8.8}$$

where indices $ij(o)$ refer to the corresponding potential vias along the path. This is a linear expression of interconnect delay in terms of layer assignment variables $\mathbf{z}$ and $\mathbf{y}$. The relation between $\mathbf{y}$ and $\mathbf{z}$, as specified by (8.6) and (8.7), can be characterized by the following four linear inequalities.

$$y_{ij}^+ \leq z_i, \tag{8.9}$$

$$y_{ij}^+ \leq 1 - z_j, \tag{8.10}$$

$$y_{ij}^- \leq 1 - z_i, \tag{8.11}$$

$$y_{ij}^- \leq z_j. \tag{8.12}$$

Here z is used as an intermediate variable for x defined by (8.1).

8.2.3 Formulation of Path-Delay Constraints

The collection of RC trees used for modeling the delay of a circuit layout can be further abstracted to a directed weighted acyclic graph, where each vertex represents a module, each edge represents a signal flow from one module to the other, and the weight associated with an edge is the delay computed above, which is a linear function of layer assignment variables. Using this model, we consider the three timing problems in synchronous circuits.

There are four basic types of propagation paths in synchronous circuits: primary input to sequential element, sequential element to sequential element, sequential element to primary output, and primary input to primary output. For each path endpoint, we can calculate the latest signal arrival time, T_{Dk}, by the viable critical path algorithms [102, 161]. Let T_{rl} be the required arrival time T_{rk}, which is determined by the chip frequency, clocking methodology, etc. Then the long path and the short path problem can be solved by setting

$$min_k \leq T_{rk} - T_{Dk} \leq max_k,$$

where min_k and max_k are two parameters for the path k. The time skew problem can be resolved by requiring that the delays of the two paths, say i and j, be matched,

$$-\epsilon \leq T_{ri} - T_{rj} \leq \epsilon,$$

where ϵ is a given margin.

8.2.4 Special Path-Delay Constraints

Two types of degenerate path-delay constraints deserve a special remark. One is the *prefixed layer assignment*, which specifies that some wire nets must be assigned to a specific layer. This can be expressed by

$$x_k = 0 \text{ or } x_k = 1. \tag{8.13}$$

The other is the *capacity constraint*, which specifies the maximum number of vias that can be allowed in a signal net. Let S be a set of wire nets that forms a signal net, and let C_S represent the maximal number of vias allowable at S, which is called the *via capacity*; then the capacity constraint is as follows:

$$\sum_{e \in S} y_e \leq C_S. \tag{8.14}$$

8.3 The *POLAR2* Layer Assigner

POLAR2 is our experimental program for *P*erformace-*O*riented *L*ayer *A*ssignment of two-layer *R*outings. *POLAR2* accepts a signed hypergraph description of the optimal layer assignment problem, which is extracted from a layout. It is used to find a better layer assignment that uses fewer vias. It can handle the following performance constraints:

- *Fixed layer assignment constraints*—certain wire nets must be assigned to a fixed layer.
- *Capacity constraints*—the number of vias per signal net is bounded.

POLAR2 employs two solution methods described in this thesis — integer linear programming (*POLAR2-exact*) and local search heuristic (*POLAR2-heuristic*). It invokes *CPLEXMIP* [19] to solve the integer linear programming formulation. *POLAR2* is written in C with 6,000 lines of code. It runs interactively and provides the users with 15 commands.

Note that the local search heuristic described in Chapter 5 naturally handles fixed layer assignment. Such constraints are imposed by pre-locking the fixed vertices and keeping them locked during the entire local search process.

Now we describe a heuristic (depicted in Fig. 8.4) for handling the capacity constraints. The basic idea is as follows. Given a bipartition, a path (net) is segmented into several pieces by unbalanced edges (vias). If we switch the vertices in a piece from their current group to the other group, then we decrease the number of unbalanced edges by two (for the two-end segment, we decrease the number of unbalanced edges by one). Meanwhile we may increase the number of unbalanced edges in the "adjacent" paths because of such switchings. So we lock all the vertices in the path, and invoke the local search algorithm to find a best possible adjustment. An adjustment that leads to paths in which the number of vias exceeds the capacity is discarded. The process is repeated for all the segments in the path. The one with the minimum number of unbalanced edges (vias) is used. This exhaustive local adjustment procedure is performed for all those paths exceeding their corresponding capacities. As will be demonstrated in the next section, this simple heuristic works extremely well for handling the capacity constraints. Moreover, it provides a way to improve the quality of the local search heuristic.

```
Algorithm local-adjustment:
    1. enqueue all paths according to the number of unbalanced edges.
    2. while (not emptyqueue)
       2.1  pop the first one.
       2.2  for each path segment starting from one end
            2.2.1 if any vertex in the segment is locked, then continue
            2.2.2 move all vertices in the segment
                  and set all vertices in the path to ``locked"
            2.2.3 do local search for maximum balance
            2.2.4 if (one path exceeds its capacity) then continue
                    else
                        save this solution if it is better than previous
      2.3  if no success in 2.2, then no local feasible adjustment
             else
                 output the best one (accept it as a solution)
                 and set the vertices in this path to be locked.
end.
```

Figure 8.4: A local adjustment heuristic for handling capacity constraints.

8.4 Implementation and Experimental Results

POLAR2 has been tested on several routings described in [52, 162]; these routings have been used by Chang and Du in evaluating their via minimization algorithm [21]. In our experiments, we first do not allow vias between routing tracks and choose only turning points of wires as potential vias, as did by Chang and Du [21]. By using our model of signed hypergraph, the resulting problem is so small that optimum solutions can be trivially seen. The obtained solutions are the same as those of Chang and Du by running sophisticated via minimization algorithms. In fact, most potential vias in these routings are essential.

In order to test larger problem instances, we then chose potential vias in such a way that each crossing of wires belonging to different nets is surrounded by either potential vias or terminals. This give rises to the maximal set of potential vias. This is theoretically very interesting, since the optimality of via minimization with respect to this selection of potential vias determines the *global* optimality of via minimization. In other words, unless we change the topology of routing—which leads to the problem of topological via minimization [65]—the result is the minimal number of vias that we can use to achieve a feasible layer assignment. On the other hand, we note that, this selection scheme may violate physical design rules, which, however, can be re-enforced by using techniques similar to symbolic layout compaction [86].

Under the selection of the maximal set of potential vias, each crossing determines a cluster. In each cluster, we chose the horizontal wire segment as the reference wire segment, marked with "+".

In Table 8.1, we give statistics of these examples under the columns *problem instance*. The name of each routing, (*name*), the number of potential vias, (*p-via*), and the number of clusters, (*cluster*), are indicated. Under the columns *previous* are the number of vias, (*o-via*), used in the original routing [52, 162] and the minimal number of vias needed, (*via*), if only vias in the original routing are chosen as potential vias [21]. We summarize the results of pure via minimization under the columns *POLAR2-exact* and *POLAR2-heuristic*, where (*via*) is the number of vias needed and all the CPU times are on a Sun Sparc2 workstation. For *POLAR2-exact*, the size of integer linear programming formulation; i.e., the number of variables, (*var*), and the number of constraints, (*const*), is also reported.

Table 8.1: Experimental results without timing constraints.

problem instance			*previous*		*POLAR2-exact*				*POLAR2-heuristic*			Reference
name	p-via	clu	o-via	via	var	const	via	CPU (s)	via	pass	CPU (s)	
yk2	11	11	20	1	33	46	1	0.15	1	1	0.02	[162], Fig.2
gcw23	7	6	6	2	20	28	2	0.12	2	1	0.00	[52], Fig.23
gcw22	67	40	16	9	174	268	7	27.17	7	1	0.02	[52], Fig.22
gcw17	87	50	26	16	224	348	12	2152	12	275	0.58	[52], Fig.17
yk25	352	200	57	40	904	1471	≤ 36	24h	36	1	0.03	[162], Fig.25

For the first four examples, both *POLAR2-exact* and *POLAR2-heuristic* obtained optimal solutions. *POLAR2-heuristic* is more than 1000 times faster than *POLAR2-exact*. Note that the mixed integer programming formulation is inherently slow. For *yk25*, *POLAR2-exact* found a solution using 36 vias in 36 CPU hours, whereas *POLAR2-heuristic* obtained a solution with the same quality in 0.03 seconds. For *gcw17*, an optimal solution is achieved with 275 passes of the local search algorithm, each starting with a randomly selected initial bipartition. Optimum layer assignments for those routings (except for *yk25*, we do not know whether it is an optimum or not) are given in Fig. 8.5 to Fig. 8.9.

We attribute the high quality results generated by the local search heuristic to the fact that it uses a good initial bipartition to start with. The initial bipartition is chosen so that all the vertices are in one group. Because we labeled all the vertical wire segments by positive signs and all the horizontal wire segments by negative signs, the chosen initial bipartition actually corresponds to an initial layer assignment where all the vertical segments are assigned to one layer and all the horizontal wire segments

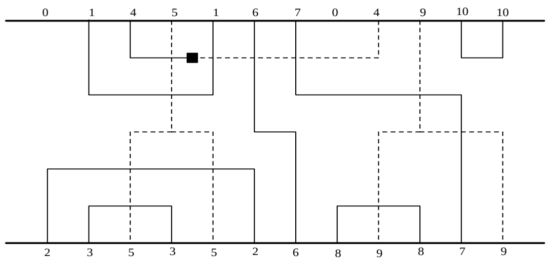


Figure 8.5: An optimum layer assignment for *yk2*.

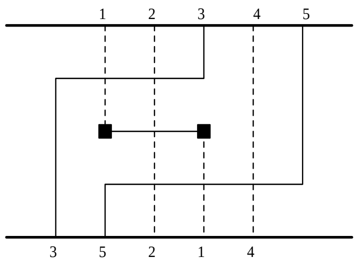


Figure 8.6: An optimum layer assignment for *gcw23*.

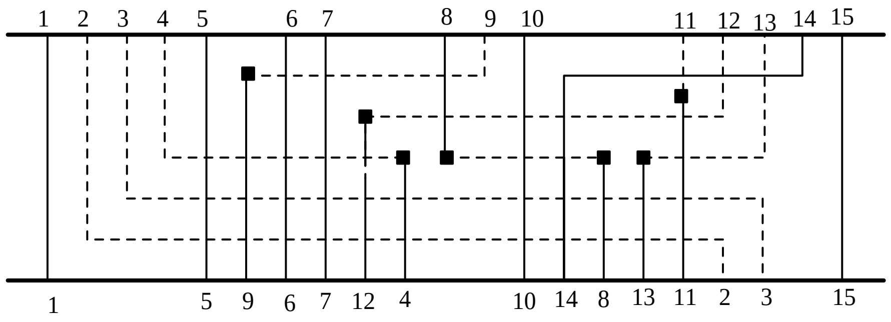


Figure 8.7: An optimum layer assignment for *gcw22*.

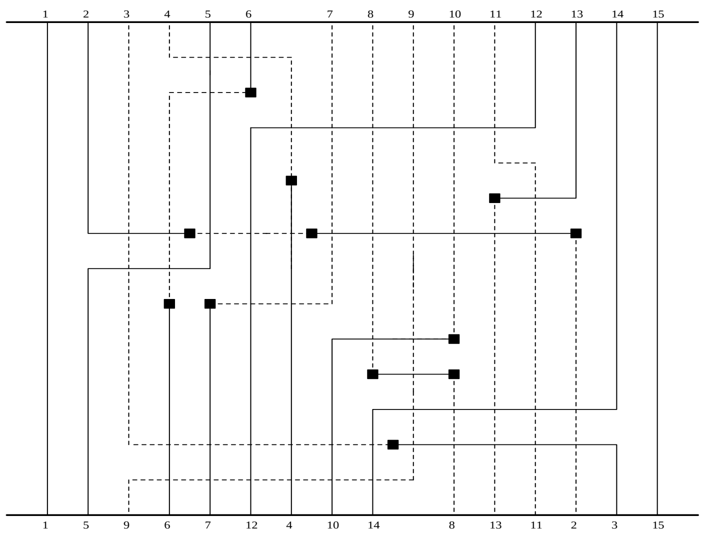


Figure 8.8: An optimum layer assignment for *gcw17*.

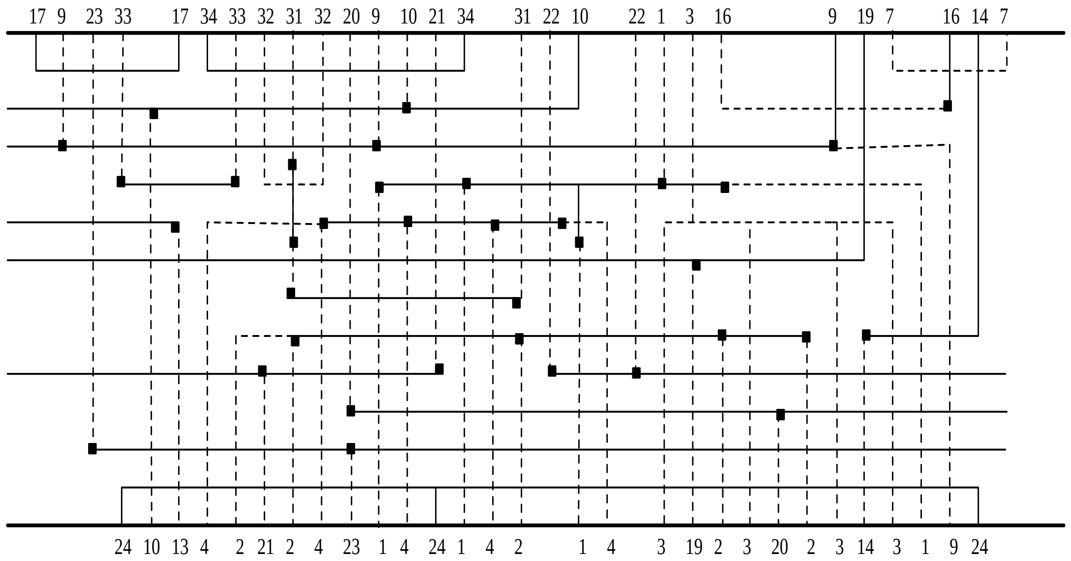


Figure 8.9: A layer assignment for *yk25*.

are assigned to the other layer. This is the layer assignment used in the original routings [52, 162].

Next we evaluate the performance of *POLAR2* in handling capacity constraints. Results with three examples *gcw23*, *gcw22* and *gcw17* are summarized in Table 8.2. Each row reports the results for a particular capacity constraint. For example, in the third row, we wish to find an optimum layer assignment for *gcw23* under the capacity constraint that the number of vias in each net does not exceed 1. In addition to the number of vias (*via*) and the CPU time, we also report the number of iterations (*iter*) of ILP branch and bound under *POLAR2-exact*, the number of local adjustment times (*adj*), and the distribution of vias. For example, the distribution {5,4,0,1} stands for 5 nets having 0 vias, 4 nets having 1 via, 0 net having 2 vias, and 1 net having 3 vias. For *gcw17*, there is no layer assignment such that each net uses at most one via.

Table 8.2: Experimental results with capacity constraints.

capacity		*POLAR2-exact*				*POLAR2-heuristic*			
		via	iter	CPU (s)	distribution	via	CPU (s)	adj	distribution
gcw23	1	2	46	0.15	{3,2,0}	2	0.02	1	{3,2,0}
	2	2	54	0.17	{4,0,1}	2	0.02	0	{4,0,1}
gcw22	1	7	1565	11.87	{8,7,0}	7	0.01	2	{8,7,0}
	2	7	1871	17.12	{9,5,1}	7	0.01	0	{9,5,1}
gcw17	1	/	1515	9.05	/	/	/	/	/
	2	13	37563	454.57	{6,5,4,0}	13	0.73	1	{6,5,4,0}
	3	12	97949	1354.35	{7,5,2,1}	12	0.58	5	{7,5,2,1}

We can make some observations from Table 8.2. First, for all the test cases, *POLAR2-heuristic* (using local adjustment) not only succeeded in meeting the capacity constraints, but also achieved optimal solutions. Again the CPU time taken is considerably smaller than that of the branch and bound approach (*POLAR2-exact*).

Second, vias distribute to nets in a reasonably uniform way. It is unlikely that via minimization causes a heavy congestion of vias at a single net. This is also seen from experiments in Table 8.3. This observation contrasts with some previous beliefs. It is demonstrated that nonuniform capacity constraints, where each net has its own capacity, are more practically relevant. Our local adjustment heuristic successfully handles this situation. Moreover, for these cases, solutions by both *POLAR2-exact* and *POLAR2-heuristic* have the same distribution, although most of them are different bipartitions. Third, the smaller the capacity constraints, the faster *POLAR2-exact* obtains the solution. This is because the small capacity cuts the solution space needed to search by the branch and bound approach.

Table 8.3: Experimental results with fixed layer assignment.

name	nets	*POLAR2-exact*				*POLAR2-heuristic*			
		via	iter	CPU (s)	distribution	via	CPU (s)	pass	distribution
yk2	4	1	50	0.02	{9,1}	1	0.01	0	{9,1}
gcw23	1	2	16	0.00	{3,2}	2	0.00	0	{3,2}
gcw22	8	8	219	1.57	{9,4,2}	8	0.00	1	{9,4,2}
	12	8	208	0.73	{9,4,2}	8	0.00	1	{9,4,2}
	8,12	9	175	0.52	{7,7,1}	9	0.02	3	{7,7,1}
gcw17	2	15	768	8.55	{5,5,5}	15	0.00	1^*	{4,7,4}
	3	13	21311	346.63	{7,3,5}	14	0.02	1	{5,6,4}
	6	13	10935	175.32	{8,3,3,0,1}	13	0.02	5	{7,5,2,0,1}
	2,3	16	223	0.80	{4,6,5}	16	0.00	1	{4,6,5}
	2,6	16	546	5.53	{5,4,6}	16	0.03	3^*	{5,5,4,1}

*: with local adjustment.

Finally, we experimented with another particular class of timing constraints – fixed layer assignment. Note that the capability of handling fixed layer assignment is a basic feature of the local search heuristic. Actually it has been used in our local adjustment heuristic for handling the capacity constraint. Here we use fixed layer assignment to have critical nets free of vias. This is an alternative to using capacity constraints directly. Results are summarized in Table 8.3. The second column (*nets*) indicates the critical nets which we want to be free of vias. Again, we observed that, for most cases, *POLAR2-heuristic* achieved optimal solutions. *POLAR2-heuristic* is significantly faster than *POLAR2-exact*. For two cases of *gcw17*, where critical nets are chosen as 2, and {2,3}, respectively, optimal solutions are obtained by invoking the local adjustment heuristic after local search. Since vertices in one path segment are moved simultaneously, the local adjustment heuristic may help to jump out of a local minimum, where each time only *one* vertex can be moved.

8.5 Summary

We have presented a signed hypergraph model and a CAD tool called *POLAR2*, for *P*erformace-*O*riented *L*ayer *A*ssignment of two-layer *R*outings. *POLAR2* employs an *exact* method and a *fast* heuristic. The exact method is based on a new compact linear integer programming formulation developed in Chapter 7. The fast heuristic is based on local search described in Chapter 5.

POLAR2 has been tested on several practical routing examples and demonstrated its ability in handling various practical constraints. In particular, our comprehensive experiments with pure via minimization (without timing constraints), and with performance-oriented via minimization (with capacity constraints and fixed layer assignment), have shown that *POLAR2* heuristic achieved optimum solutions for most test cases, but used less than one thousandth of the CPU time taken by the branch-and-bound based integer programming approach. Therefore, *POLAR2* heuristic is a promising improvement tool in the performance-driven layout design.

Chapter 9

Applications to Logic Synthesis

In this chapter, we apply the theoretical results developed in this thesis to VLSI sequential logic synthesis. Although the formulation of constrained encoding in the state assignment of asynchronous sequential machines was discovered in the 1960s [146], its relation with the optimal state assignment for synchronous sequential machines was understood only very recently [104, 158]. Indeed, despite a huge volume of literature on logic optimization, the problem is still not fully understood. In this chapter, we first review briefly some previous work in Section 9.1. We then formally define in Section 9.2 the model of dichotomy-based constrained encoding and present a unified framework for logic encoding in sequential logic synthesis. In the subsequent four sections (Sections 9.3-9.6), we show how dichotomy-based constrained encoding relates to various sequential logic design problems. In Sections 9.7-9.8, we describe our experimental tool, called *ENCORE*, for sequential logic synthesis, and report various application results.

9.1 Introduction

Constrained encoding is an important problem arising in many aspects of the synthesis of combinational and sequential logic circuits. Given a set $S = \{s_1, ..., s_m\}$ of m *states*, the *(complete) constrained encoding* problem is to find an encoding $\boldsymbol{\alpha}$ of S into a set $\{\boldsymbol{\alpha}(s_1), ..., \boldsymbol{\alpha}(s_m)\}$ of m binary k-tuples (k-bit vectors), in such a way that all the constraints (defined below) are satisfied and k is minimized. A *binary constraint*, also known as a *(partial) dichotomy constraint*, requires that a subset P of S be distinguished from a disjoint subset Q of S by at least one bit, i.e., that bit must have the value 0 for all the states in P and 1 for all the states in Q, or vice versa. If Q is empty, we have the special case of *unary constraint* requiring that a subset P of S must be identified by at least one bit b of the k-tuples, in the sense that the value of b should be the same for all the states in P. A variation, called the *partial constrained encoding* problem, aims at maximizing the number of constraints that are satisfied using a fixed number of bits.

For example, consider a unary constraint $(\{s_1, s_2, s_4\})$, and four binary constraints $(\{s_1, s_3\}, \{s_2\})$, $(\{s_1, s_3\}, \{s_4\})$, $(\{s_3, s_4\}, \{s_1\})$, and $(\{s_3, s_4\}, \{s_2\})$, on set $S = \{s_1, ..., s_4\}$. Table 9.1(a) shows a minimum-length encoding satisfying all the constraints. Table 9.1(b) gives a two-bit encoding satisfying the largest number (4) of constraints.

Table 9.1: Examples of encodings.

(a)

s	$\boldsymbol{\alpha}(s)$		
s_1	1	0	0
s_2	1	0	1
s_3	0	1	0
s_4	1	1	1

(b)

s	$\boldsymbol{\alpha}'(s)$	
s_1	1	0
s_2	1	1
s_3	0	0
s_4	0	1

The constrained encoding problem was first formulated by Tracey [146] for critical-race-free state assignment of asynchronous machines. Unger [148] pointed out, for certain kinds of finite state machines (FSMs), the problem of obtaining an asynchronous implementation such that the correctness is independent of the presence of arbitrary gate and wire delays can be reduced to the problem of constrained encoding. Recent studies indicate that the problem of encoding states of FSMs to have a minimum PLA implementation is related to the partial constrained encoding problem [152, 158].

The search for efficient solutions for the constrained encoding problem was pioneered by Tracey [146]. He proposed a process similar to Boolean logic minimization, which consists of two basic steps: First, construct all maximal compatible sets of dichotomy constraints; each such set can be satisfied by a one bit assignment, which is called a *prime bit assignment*. Second, find a minimal number of prime bit assignments to cover all the given dichotomy constraints; this problem is known as the covering problem. This prime-covering method gives an exact solution to the complete constrained encoding problem. However, the number of prime bit assignments may be exponential in the number of states, and the covering problem is NP-complete [48]. In practice, the process described above has been approximated using various heuristic [149, 158]. In addition, it is not clear how to apply the prime-covering approach to partial constrained encoding. As a consequence, for the optimum state assignment for synchronous sequential machines, the dichotomy-based approaches (such as *DIET* [158]) have not achieved the success of classical approaches (such as *KISS* [103], *CREAM* [104], *NOVA* [152], and [126]), which are based on group constraints. A group constraint specifies that a set of states must be encoded in a neighbourhood or a face in the Boolean space.

9.2 Constrained Encoding Model of Logic Synthesis

In this section, we give a mathematical formulation of constrained encoding. We then show how our model of constrained encoding provides a unified framework for logic encoding in sequential logic synthesis.

A *constraint* c on a set $S = \{s_1, ..., s_m\}$ is a pair $c = (c^+, c^-)$ of disjoint subsets (called *blocks*) of S. We may distinguish two types of constraints: A *binary constraint* consists of two nonempty blocks. A *unary constraint* is a constraint with one empty block. Binary constraints are traditionally called *(partial) dichotomies* [146, 149, 158].

Let $B = \{-1, 1\}$ for Boolean values. Conventionally, $B = \{0, 1\}$ is used. We choose to use $\{-1, 1\}$ instead, as it will considerably simplify our notation. Given a set $S = \{s_1, ..., s_m\}$ of $m > 0$ *states* and an integer $k > 0$, a *binary encoding* (or simply an *encoding*) $\boldsymbol{\alpha}$ of S is a mapping $\boldsymbol{\alpha}$: $S \rightarrow B^k$. Note that $\boldsymbol{\alpha}$ need not be one to one. We may think of the encoding as a matrix $\mathbf{A}$: The *ith* row of the matrix represents the *word* assigned by $\boldsymbol{\alpha}$ to state s_i, and the *jth* column represents *bit* j of the encoding. We use α to refer to a particular column of $\mathbf{A}$; such a column is called a *bit assignment*, and can be interpreted as a mapping α: $S \rightarrow B$. We denote by $\alpha_1, \ldots, \alpha_m$ the components of bit assignment α.

With a slight abuse of notation, we say that state s_i is *contained* in constraint c and write $s_i \in c$ if $s_i \in c^+ \cup c^-$. We use $|c|$ to denote the number of states contained in constraint c. A set $C = \{c_1, ..., c_n\}$ of n constraints can be described by the *constraint matrix* $\mathbf{C} = (c_{ij})_{m \times n}$, where

$$c_{ij} = \begin{cases} 1 & \text{if } s_i \in c_j^+, \\ -1 & \text{if } s_i \in c_j^-, \\ 0 & \text{if } s_i \notin c_j. \end{cases}$$

We use C_i to denote the subset of constraints in C containing s_i, and $|C_i|$ be its cardinality.

A bit assignment α: $S \rightarrow B$ is said to *satisfy* a constraint $c = (c^+, c^-)$ if there exists a value $b \in B$ such that for all $s \in c^+$, $\alpha(s) = b$, and for all $s \in c^-$, $\alpha(s) = \overline{b}$, where $\overline{b}$ is the complement of b. An encoding $\boldsymbol{\alpha}$:$S \rightarrow B^k$ is said to *satisfy* a constraint $c = (c^+, c^-)$ if at least one bit assignment of $\boldsymbol{\alpha}$ satisfies c.

Table 9.2: Examples of assignments.

s	$\alpha(s)$	$\beta(s)$	$\gamma(s)$	$\epsilon(s)$
1	-1	1	-1	1
2	-1	1	1	1
3	-1	-1	1	1
4	1	-1	-1	1
5	-1	1	1	1
6	1	-1	-1	1

To illustrate these definitions, let $S = \{1, ..., 6\}$ and consider bit assignments α, β, γ and ϵ shown in Table 9.2 and constraints c_1, c_2, c_3 and c_4 defined below.

- $c_1 = (\emptyset, \emptyset)$ and $c_2 = (\{3\}, \emptyset)$. Any bit assignment satisfies c_1 and c_2. Therefore constraint $(\emptyset, \emptyset)$ and constraints with one empty block and one one-state block are *trivial* constraints, and will be excluded.
- $c_3 = (\{2, 3, 5\}, \emptyset)$. Bit assignments α, γ and ϵ satisfy constraint c_3, but β does not.
- $c_4 = (\{1, 2, 5\}, \{4, 6\})$. Bit assignments α and β satisfy constraint c_4, but γ and ϵ do not.

The encoding composed of bit assignments α, β, γ and ϵ satisfies the above four constraints.

The *(complete) constrained encoding problem* is defined as follows: Given a set S of m states, and a set C of n constraints on S, find an encoding $\boldsymbol{\alpha}$ of S with minimum k such that $\boldsymbol{\alpha}$ satisfies each constraint $c \in C$.

A variation of the above problem, the *partial constrained encoding problem*, is as follows: Given a set S of m states, a set C of n constraints on S, and an integer h, find an encoding $\boldsymbol{\alpha}$ of S with $k = h$ such that $\boldsymbol{\alpha}$ satisfies as many constraints of C as possible. If $h = 1$, this problem is called the *optimal bit generation* problem.

If we represent each state by a vertex and each constraint by an edge, then S and C form a signed hypergraph. For example, for $S = \{s_1, \ldots, s_5\}$, the constraint $(\{s_1, s_2\}, \{s_5\})$ corresponds to an edge with positive incidences with s_1 and s_2 and a negative incidence with s_5. A bit assignment corresponds to a bipartition of the set of vertices. A constraint is satisfied by a bit assignment if and only if the corresponding edge is balanced by the corresponding bipartition. Therefore the constrained encoding problem is to find the minimum number of bipartitions such that each edge is balanced by at least one bipartitions. Therefore we have the following result.

Proposition 9.2.1 *The constrained encoding problem is equivalent to the minimum covering problem of a signed hypergraph. The partial constrained encoding problem is equivalent to the partial covering problem of a signed hypergraph.*

With our model of constrained encoding, a unified framework of logic encoding for sequential logic synthesis is illustrated in Fig. 9.1. Depending on different design styles, a set of dichotomy constraints is generated which guarantees that its satisfaction will lead to a correct implementation. This is the set of *correctness constraints.* Another set of dichotomy constraints relates to an economic implementation. This constitutes the set of *optimization constraints.* The optimum logic encoding problem for sequential logic synthesis is to find an encoding that satisfies as many optimization constraints as possible, while satisfying all the correctness constraints.

Now we show how our framework can handle the output encoding problem. As shown in [123], modeling of the output encoding problem requires the *dominance* and *disjunctive* constraints, in additional to dichotomy constraints. A word (say, row 1 of $\mathbf{A}$) is said to dominate another word (say, row 2 of $\mathbf{A}$), if for each bit position in the second row that contains a 1, the corresponding bit position in the first row also

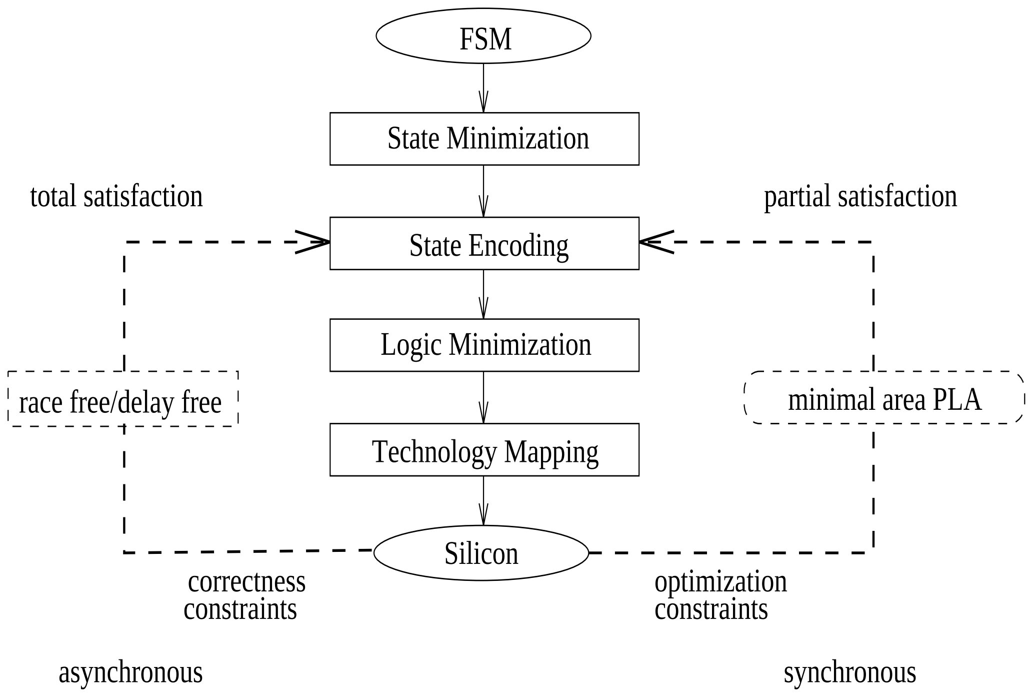


Figure 9.1: A unified framework of constrained encoding for logic synthesis.

contains a 1. A row, say row 1 of **A**, is said to be a *disjunction* of rows 2 and 3, if $\alpha_1 = \alpha_2 \vee \alpha_3$, for each bit. A formulation of constrained encoding problem resulting from output encoding is as follows [123]: Given a set of dichotomy constraints, a set of dominance constraints, and a set of disjunction constraints, find an encoding of minimum number of bits such that it satisfies all the dominance, disjunction, and dichotomy constraints.

We can use the same algorithm of constrained encoding, but each bit generated must satisfied all the dominance constraints and all the disjunction constraints. The dominance constraints can be imposed as follows. Initially $\alpha_1 = \alpha_2 = 1$. When the component of maximum gain is α_1, we check to see the value of α_2. If $\alpha_2 = 1$, then we do not change α_1; or say it is an *infeasible* move prohibited by the dominance requirement. So we select the component of the second largest gain. And so on. The disjunction constraints can be handled similarly. Initially $\alpha_1 = \alpha_2 = \alpha_3 = 1$. When the component of maximum gain is α_1, we check the values of α_2 and α_3. If either α_2 or α_3 is equal to 1, then we do not change α_1; this is an *infeasible* move prohibited by the disjunctive constraint. When we have changed α_2 and α_3, we select α_1 as a component to change in the next move, no matter what γ_1 is.

The above extension provides a simple way to handle the output encoding problem, while maintains the same time complexity as the basic bit generation algorithm. The problem size p now shall include those due to dominance constraints and the disjunction constraints. We note that, in order to satisfy these dominance constraints and disjunction constraints, a framework of ordered dichotomies was introduced, which led to even complicated prime generation and prime covering [123].

9.3 Race-Free State Assignment for Asynchronous Machines

The design of sequential logic circuits begins with a behavior specification, which is often *state table*, where the columns corresponds to inputs, the rows to present states, and the entries to transitions, which are ordered pairs representing the next state and the current output, respectively. An example of a state table is given in Table 9.3. To find a logic implementation, states are encoded by binary k-tuples. For example, an encoding of (s_1, s_2, s_3, s_4) is $(00, 01, 11, 10)$. This can be viewed as an assignments of two binary *state variables* y_1 and y_2. With such encoding, the transition functions can be described by Boolean logic functions in terms of state variables.

A circuit is said to be *asynchronous*, if it has no clocks. Such a circuit can be constructed directly from the transition functions and uses feedback lines to relate the current state variables and the next state variables. If more than one state variable must change in the course of a transition, the subsequent state of the circuit may depend on which state variable changes first, that is, which one wins the *race*. In the worst case, race can be *critical*, which leads to a malfunction of the circuit.

Table 9.3: A flow table.

	x_1x_2 00	01	11	10
s_1	$s_1, 0$	$s_2, 0$	$s_1, 0$	$s_1, 0$
s_2	$s_3, 0$	$s_2, 0$	$s_2, 0$	$s_4, 0$
s_3	$s_3, 0$	$s_2, 1$	$s_1, 0$	$s_3, 0$
s_4	$s_3, 0$	$s_4, 1$	$s_2, 0$	$s_4, 0$

Critical races can be avoided by choosing state encoding carefully. It is assumed that only one binary input variable changes at a time, and that the delays in the feedback lines are sufficiently large to let all the circuit changes complete before any state variable can affect the gate inputs. Suppose that, under a given input, if the machine starts in state s_i it should change to state s_j, while if it starts in state s_k it should remain in s_k. If the transition $s_i \rightarrow s_j$ involves a critical race, the circuit may end in state s_k. To avoid this, it is sufficient that one state variable be assigned one value in states s_i and s_j and the opposite value in state s_k. This constraint is represented by a dichotomy $(\{s_i, s_j\}, \{s_k\})$. Two transitions are *disjoint* if all the states involved are distinct. In general, the encoding of states should be such that all the states "spanned" by a transition occurring within one column must have one bit differing from the encoding assigned to the states spanned by any disjoint transition in this column. These conditions are known as *Tracey's conditions* [146]. For example, Tracey's conditions for a race-free implementation of Table 9.3 are as below.

- column 00: $(\{s_1\}, \{s_2, s_3, s_4\})$
- column 01: $(\{s_1, s_2, s_3\}, \{s_4\})$
- column 11: $(\{s_1, s_3\}, \{s_2, s_4\})$
- column 10: $(\{s_1\}, \{s_3\})$ $(\{s_1\}, \{s_2, s_4\})$ $(\{s_3\}, \{s_2, s_4\})$

9.4 Delay-Free State Assignment for Asynchronous Machines

In general, to avoid critical race and other delay-related timing problems, one has to insert certain delays in the feedback lines. The question arises whether the states of a given FSM can be encoded in such a way that its correctness is independent of the stray delays in the circuit, without the insertion of any delays. Such an encoding is called a *delay-free* assignment. It turns out that a delay-free assignment exists if an FSM satisfies certain conditions discovered by Unger, namely it has no "essential hazards" [149].

Table 9.3 has no essential hazards. Single input changes are assumed. In order to produce a delay-free realization, the encoding of states should be such that all the states involved in every possible transition occurring among any two adjacent columns have at least one state variable differing from all the states involved in any other disjoint transition beginning or ending in one of the two adjacent columns. These are called *Unger's conditions* [148]. From Table 9.3, we have:

- column 00 and column 01: $(\{s_1, s_2, s_3\}, \{s_4\})$ $(\{s_1\}, \{s_2, s_3, s_4\})$
- column 01 and column 11: $(\{s_2, s_4\}, \{s_1\})$ $(\{s_1, s_2\}, \{s_4\})$
- column 11 and column 10: $(\{s_2, s_4\}, \{s_1, s_3\})$
- column 10 and column 00: $(\{s_3, s_4\}, \{s_1)\}$

9.5 Optimal State Assignment for Synchronous Machines

In the *synchronous design*, clocks are used to control each transition so as to avoid critical races and hazards. The major concern for the state assignment of synchronous FSM's is to find a state encoding so as to minimize the cost of implementation. If a PLA is used to implement combinational logic blocks, then the PLA area, which is the main portion of the chip area, is the objective to minimize. The optimal state assignment problem here is to find a state encoding that has a minimum-area two-level logic implementation.

Table 9.4: A synchronous state table.

	x_1x_2			
	00	01	11	10
s_1	s_3	s_4	s_4	—
s_2	s_3	s_2	s_2	—
s_3	s_1	s_4	s_2	—
s_4	s_1	s_2	s_2	—

To show why the state assignment problem here can be solved using dichotomy constraints, we consider the FSM of Table 9.4. We can group together those entries in the state table that have the same next state, and express the next state function as follows.

$$s_1 = x_2'(s_3 + s_4) \tag{9.1}$$

$$s_2 = x_1s_3 + x_2(s_2 + s_4) \tag{9.2}$$

$$s_3 = x_2'(s_1 + s_2) \tag{9.3}$$

$$s_4 = x_2s_1 + x_1'x_2s_3 \tag{9.4}$$

There exist many such groupings; we select the one with the minimal number of "groups". This is known as symbolic logic minimization. A good tool for this purpose is *ESPRESSO-MV* [17].

Note that the area of a PLA is determined by the number of binary variables times the number of distinct product terms. If we encode the states in such a way that each group is represented by one Boolean product of the encoding variables $\alpha_1, ..., \alpha_k$, then the number of products in the final logic is no more than the number of "groups". This can be achieved by using dichotomy constraints, as explained below.

States of S appear in each group either as a singleton or as a sum. A singleton can be expressed directly as a product of the encoding variables $\alpha_1, ..., \alpha_k$. For example, if $\alpha(s_1) = (101)$, then s_1 is represented as $\alpha_1 \alpha_2' \alpha_3$. If only two states appear in a sum and the code words assigned to those two states are adjacent, it is still straightforward to represent a sum as one product of $\alpha_1, ..., \alpha_k$. Consider $s_1 + s_3$ for an example, if $\alpha(s_1) = (100)$ and $\alpha(s_3) = (000)$; then $\alpha(s_1) + \alpha(s_3) = (-00)$, i.e., $s_1 + s_3$ can be represented as $\alpha_2' \alpha_3'$. This is the smallest "subcube" that contains the code words assigned to every state in $\{s_1, s_3\}$. Now, suppose $\alpha(s_1) = (101)$ and $\alpha(s_3) = (110)$, i.e., the two code words are not adjacent. If we still take the smallest subcube containing $\alpha(s_1)$ and $\alpha(s_3)$, that is $(1 - -)$, to represent the sum $s_1 + s_3$, it will include not only code words (101) and (110) assigned to s_1 and s_3, but also two additional code words (100) and (111). Such an encoding would be invalid, if (100) and (111) are assigned to states s_2 and s_4. This can be avoided by setting up constraints $(\{s_1, s_3\}, \{s_2\})$ and $(\{s_1, s_3\}, \{s_4\})$. In general, for every sum $s_{i_1} + s_{i_2} + \cdots + s_{i_j}$ we introduce constraints $(\{s_{i_1}, s_{i_2}, ..., s_{i_j}\}, \{s_l\})$ for all $l \in I$ but $l \neq i_1, i_2, ..., i_j$, where I denotes the integer set ranging from 1 to n. In our example, we thus have constraints $(\{s_3, s_4\}, \{s_1\})$ $(\{s_3, s_4\}, \{s_2\})$ $(\{s_2, s_4\}, \{s_1\})$ $(\{s_2, s_4\}, \{s_3\})$ $(\{s_1, s_2\}, \{s_3\})$ and $(\{s_1, s_2\}, \{s_4\})$.

It should be noted that partial constrained encoding, i.e, bounded-length encoding may be more relevant than complete constrained encoding. Partial constrained encoding may result in more product terms, but it uses fewer encoding variables. Since the PLA area is related to the product of these two parameters, it is possible that partial encoding yields less PLA area.

9.6 PLA Decomposition

Another problem that surprisingly resembles optimum state assignment is PLA decomposition [38, 39]. To illustrate why PLA decomposition can be solved within the framework of constrained encoding, we consider a PLA with seven primary inputs and two primary outputs, described by the following expressions:

$$y_1 = x_1x_3x_4'x_6 + x_2x_5x_6'x_7 + x_1x_4x_7' + x_1'x_3'x_4x_7' + x_3'x_5'x_6'x_7' \qquad (9.5)$$

$$y_2 = x_2'x_4'x_6 + x_3x_4'x_6 + x_2'x_5x_6'x_7 + x_1'x_3'x_4x_7' + x_1'x_3'x_5'x_6'x_7' + x_2'x_4'x_5'x_7' \qquad (9.6)$$

This PLA has 10 distinct product terms and cannot be further simplified by using logic minimizers such as *ESPRESSO* [17].

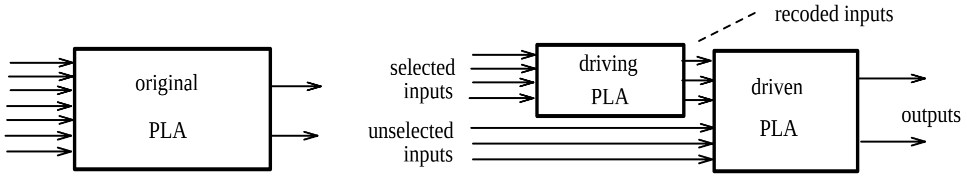


Figure 9.2: PLA Architecture.

We would like to decompose the given PLA into the configuration of Fig. 9.2. We assume that the selected subset of inputs is $SI = \{x_4, x_5, x_6, x_7\}$. Five product terms of the selected inputs appear in (9.5) and (9.6): $x_4'x_6$, $x_5x_6'x_7$, x_4x_7', $x_5'x_6'x_7'$, and $x_4'x_5'x_7'$. In order to re-encode SI, we first need to make all involved product terms of the selected inputs *disjoint*. Products x_4x_7' and $x_5'x_6'x_7'$ are not disjoint. Either are $x_5'x_6'x_7'$ and $x_4'x_5'x_7'$. So we expand those terms into minterms. By removing some redundant product terms, the above expressions reduce to

$$y_1 = x_1x_3x_4'x_6 + x_2x_5x_6'x_7 + x_1x_4x_7' + x_1'x_3'x_4x_7' + x_3'x_4'x_5'x_6'x_7' \qquad (9.7)$$

$$y_2 = x_2'x_4'x_6 + x_3x_4'x_6 + x_2'x_5x_6'x_7 + x_1'x_3'x_4x_7' + x_1'x_3'x_4'x_5'x_6'x_7' + x_2'x_4'x_5'x_6'x_7' \qquad (9.8)$$

Now all product terms of the selected inputs, $x_4'x_6$, $x_5x_6'x_7$, x_4x_7', and $x_4'x_5'x_6'x_7'$, are disjoint. We may view them as four values of a multiple-valued symbolic input variable s, denoted respectively by s_1, s_2, s_3 and s_4. Then the above logic expressions with three binary-valued inputs $\{x_1, x_2, x_3\}$, one four-valued input $\{s\}$, and two binary-valued outputs $\{y_1, y_2\}$, can be simplified by multiple-valued symbolic logic minimization. For this example, by using *ESPRESSO-MV* [121], we have

$$y_1 = x_2 s_2 + x_1 x_3 (s_1 + s_3) + x_3^{'}(s_3 + s_4) \tag{9.9}$$

$$y_2 = x_2^{'}(s_1 + s_2 + s_4) + x_3 s_1 + x_1^{'} x_3^{'}(s_3 + s_4) \tag{9.10}$$

There are six *symbolic product terms* in the above expressions.

Similar to the optimal state assignment problem, we reduce it to a constrained encoding problem. For this example, totally we have constraints $(\{s_1, s_3\}, \{s_2\})$ $(\{s_1, s_3\}, \{s_4\})$ $(\{s_3, s_4\}, \{s_1\})$ $(\{s_3, s_4\}, \{s_2\})$ and $(\{s_1, s_2, s_4\}, \{s_3\})$. It is easily verified that the encoding $\alpha(s_1, s_2, s_3, s_4) = (001, 011, 100, 111)$ is a minimum-length binary encoding satisfying all the constraints derived above. Therefore, we need three binary variables, denoted as $(x_8 x_9 x_{10})$, to encode the symbolic input variable s. Substituting into (9.9) and (9.10), we obtain the re-encoded PLA:

$$y_1 = x_2 x_8^{'} x_9 x_{10} + x_1 x_3 x_9^{'} + x_3^{'} x_8 \tag{9.11}$$

$$y_2 = x_2^{'} x_{10} + x_3 x_8^{'} x_9^{'} x_{10} + x_1^{'} x_3^{'} x_8 \tag{9.12}$$

The driving PLA is expressed as

$$x_8 = x_4 x_7^{'} + x_4^{'} x_5^{'} x_6^{'} x_7^{'} \tag{9.13}$$

$$x_9 = x_5 x_6^{'} x_7 + x_4^{'} x_5^{'} x_6^{'} x_7^{'} \tag{9.14}$$

$$x_{10} = x_4^{'} x_6 + x_5 x_6^{'} x_7 + x_4^{'} x_5^{'} x_6^{'} x_7^{'} \tag{9.15}$$

Comparison: Note that PLA area is calculated by (2*inps+out)*cubes. Thus the original PLA takes (2*7+2)*10=160. The decomposed PLA, which is the sum of the driving PLA and the driven PLA, takes (2*4 + 3) *4 + (2*(3+3) + 2)*6 = 44 + 84 = 128. Both the original PLA and the decomposed PLA has 10 product terms.

9.7 The *ENCORE* Synthesis Package

We have developed a package called *ENCORE* for constrained encoding, based on the greedy peeling heuristic described in Chapter 6. The algorithm used inside greedy peeling for finding maximum balance is the local search heuristic presented in Chapter 5. *ENCORE* is written using the C programming language.

Since greedy peeling does not guarantee optimality, we have implemented two techniques in *ENCORE*, which have been demonstrated effectiveness in improving the quality of constrained encoding. The first technique is the iterative greedy peeling as described in Section 6.2. This strategy can be accomplished by assigning a sufficiently large weight to each edge in E'. Suppose that each edge e_j in a given set E is associated with an integer weight, w_j. Our algorithm and data structures can be used directly, except that the range of the bucket list is now from $-q$ to q, where $q = max\{\sum_{e_j \in E_i} w_j, 1 \leq i \leq m\}$. This enhancement does not increase the time complexity. Experiments have shown that only a few runs are sufficient to improve the solution quality.

The second technique is to impose a "balance criterion" on bit generation. A bit assignment used to satisfy a constraint $c = (c^+, c^-)$ distinguishes states in c^+ from those in c^-. If there are no more constraints, the number of additional bits needed to distinguish states in c^+ (c^-) from each other is $\lceil log_2|c^+|\rceil$ $(\lceil log_2|c^-|\rceil)$; thus, the minimal number of additional bits is $max\{\lceil log_2|c^+|\rceil, \lceil log_2|c^-|\rceil\}$. Hence, in order to minimize the number of encoding bits, it is desirable to have the number of -1s and 1s in a bit assignment balanced. The balance criterion implemented in *ENCORE* is as follows: Given an integer p, $0 < p \leq m$, as the desirable number of 1s in a bit assignment, and a tolerance $0 \leq r \leq min\{p, m-p\}$, a bit assignment is said to be *balanced* if $-2r \leq m - 2p + \sum_{i=1}^{m} \alpha_i \leq 2r$. When $p = m/2$, we need to have $-2r \leq \sum_{i=1}^{m} \alpha_i \leq 2r$; this is the scheme implemented in *DIET* [158].

In *ENCORE*, the balance strategy is accomplished by first generating an initial balanced assignment and then maintaining the balance during the process of bit generation. Starting from the initial bit assignment with all components being 1, ENCORE selects a component with maximum gain to change until the balance criterion is satisfied. In the rest of the first pass and also in all the following passes, a component with maximum gain is selected to move only if changing it would not cause imbalance. Otherwise another component with maximum gain or even the second largest gain is selected and checked for the balance criterion. If there are several components having the same largest gain, we select the one which gives the minimum absolute value of $m - 2p + \sum_{i=1}^{m} \alpha_i$.

There is a special case for which optimality is guaranteed by greedy peeling. The problem is to find a minimum-length encoding for a set S of m states such that each state is assigned a distinct code word. It can be described in our framework of constrained encoding, by a set of $n = \frac{1}{2}m(m-1)$ dichotomies with one state in each block. We need to add this set of *distinct-state* constraints when we handle partial constrained encoding arising from the optimum state assignment problem of synchronous FSMs.

9.8 Experimental Results

The first set of small examples comes from the early literature on the synthesis of asynchronous FSMs. Here the aim is either a race-free [146] or a delay-free implementation [148]. We have written a program to derive the dichotomy constraints from the original flow table specification; *ENCORE* is then used to find the minimum-length encoding that satisfies all these dichotomy constraints. As summarized in Table 9.5, *ENCORE* generated encodings with the same lengths as those given by *exact methods* in the literature for all these examples.

The second set of tests consists of 40 industrial examples available from the MCNC benchmark set representing a wide range of finite state automata. The raw data can

Table 9.5: Synthesis of asynchronous FSM's.

FSM	#states	#constraints	#bits	References
fsm1	5	7	3	[148], p.84
fsm2	6	16	4	[148], p.97
fsm3	9	19	4	[148], p.108
fsm4	7	94	4	[148], p.144
fsm5	5	10	3	[146], Fig.3
fsm6	6	10	3	[146], Fig.4

be found in [152]. We have incorporated *ENCORE* into Berkeley *octtools* to produce PLA realizations from given FSM specifications.

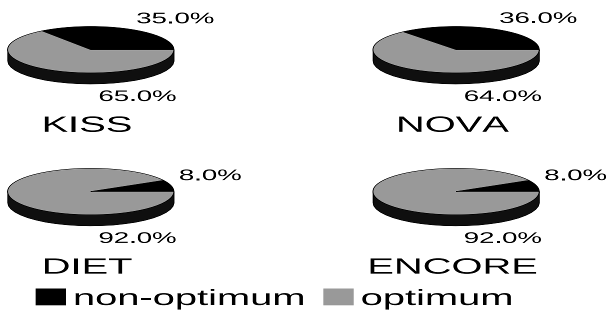


Figure 9.3: Comparison of several encoding programs.

We have conducted two groups of experiments. In the first group, we solve the constrained encoding completely, i.e., we look for a minimum-length encoding that satisfies all the constraints. We have written a program to generate the dichotomy constraints from the face-embedding constraints, where face-embedding constraints are obtained by running *ESPRESSO-MV* [17]. We compared *ENCORE* with several available state assignment programs, *KISS* [103], *NOVA* [152], and *DIET* [158]. The results are summarized in Table 9.6. For each example tested, the table reports the minimum number #*bits* of bits by unconstrained encoding, the minimum number #*cbits* of bits by constrained encoding, the encoding lengths obtained by *KISS*, *NOVA*, *DIET*, and *ENCORE* (with and without a balance criterion), and the CPU time used. For all the test examples (with the exception of the *keyb* FSM), *ENCORE* using the balance criterion obtained the shortest length encodings, with only one tenth to one thousandth of the CPU-time used by *NOVA* and *DIET*. Note that both *ENCORE* and *DIET* work on binary constraints, but *DIET* uses the prime-covering approach. *KISS* and *NOVA* work on the face-embedding constraints and are based on different theoretical foundations. The time complexities of the encoding algorithms used in these programs are *at best* quadratic in the size of the problem.

The second group of experiments gave very interesting results. Here we solve the partial constrained encoding problem: Given a bound on the encoding length, maximize the number of satisfied binary constraints. The lengths chosen are the minimum ones needed to distinguish all the states. *ENCORE* produces better overall results than *NOVA* (cf. Table 9.7). For most of the FSMs, especially the *large* ones, the final PLA implementations occupy less area than those given by *NOVA*. Note that *ENCORE* aims at maximizing the number of satisfied binary constraints, where *NOVA* aims at maximizing the number of satisfied face-embedding constraints.

9.9 Summary

There are two major results in this chapter. First, we presented a CAD tool called *ENCORE* for solving various constrained encoding problem in sequential logic synthesis. It generates better results than the existing programs developed specifically in each application field. Since *ENCORE* is orders of magnitude faster, it is a promising alternative to existing techniques for solving large-size VLSI-CAD problems. We note that the lack of efficient methods for finding state assignments in asynchronous sequential synthesis has once been considered as a major obstacle to the use of the

Table 9.6: Encoding with all the input constraints satisfied.

	the encoding length							CPU-time(s)*		
FSM	#bits	#cbits	*KISS*	*NOVA*	*DIET*	*ENCORE*^{nb}	*ENCORE*^{b}	*NOVA*	*DIET*	*ENCORE*^{b}
dk15	2	4	4	4	4	4	4	3.2	0.1	0.00**
lion	2	-	-	-	2	2	2	-	0.1	0.00
mc	2	-	-	-	-	2	2	-	0.1	0.00
tav	2	2	-	-	2	2	2	-	0.1	0.01
train4	2	2	-	-	2	2	2	-	0.1	0.00
s8	3	3	-	-	3	3	3	-	0.1	0.00
bbtas	3	3	3	-	-	3	3	0.0	0.1	0.00
beecount	3	4		4	-	4	4	0.1	0.2	0.01
dk14	3	4	4	5	4	5	4	1.9	0.4	0.05
dk27	3	3	-	3	-	3	3	0.28	0.3	0.01
dk17	3	4	4	4	4	4	4	0.5	0.6	0.01
ex6	4	4	5	5	4	5	4	0.9	0.3	0.01
shiftreg	3	3	-	3	3	3	3	0.35	0.2	0.01
ex5	4	5	5	6	5	5	5	3.95	0.6	0.01
lion9	4	4	4	-	4	4	4	-	0.6	0.02
bbara	5	5	-	5	-	5	5	120.6	0.5	0.01
ex3	4	5	5	6	7	6	6	0.53	1.2	0.03
ex7	4	-	-	-	6	6	6	-	0.8	0.02
opus	4	4	-	-	-	4	4	-	0.5	0.00
train11	4	5	6	5	5	5	5	3.46	1.7	0.00
modulo12	4	4	-	-	-	4	4	-	1.4	0.00
ex4	4	-	-	-	4	4	4	-	2.5	0.02
dk512	4	5	6	6	5	5	5	35	4.0	0.03
mark1	5	-	5	-	4	4	4	29	2.5	0.01
bbsse	6	6	6	6	6	6	6	0.3	2.1	0.04
cse	4	5	6	5	5	6	5	0.2	3.4	0.03
kirkman	4	-	-	-	6	6	6	-	6.7	0.10
sse	4	-	-	-	6	6	6	-	2.1	0.02
ex2	5	6	6	6	6	7	6	2.46	8.6	0.05
keyb	5	7	8	7	8	9	8	28	7.7	0.10
ex1	5	7	7	7	7	7	7	35	11.2	0.12
s1	5	5	5	5	5	5	5	2.33	12.1	0.01
s1a	5	5	5	-	5	5	5	-	8.7	0.01
donfile	5	≤ 11	12	15	7	9	6	555	26.6	0.01
dk16	5	7	10	10	8	8	8	311	87.6	0.09
styr	5	6	6	9	6	8	6	52	26.4	0.15
sand	5	6	6	6	6	6	6	39.71	62.1	0.03
tbk	5	-	-	?	*f*	23	23	1301	*f*	0.14
planet	6	6	-	6	*f*	7	7	37.6	*f*	1.07
scf	7	≤ 8	8	-	*f*	8	8	-	*f*	0.53

*	VAX-11/8650
**	less than 0.01
-	not applicable
?	solution not found within the used CPU time
f	failed
b	with balance requirement
nb	without balance requirement

Table 9.7: Comparisons of *NOVA/ih*, *NOVA/ig* and *ENCORE*.

	1-hot		*NOVA/ih*[1]		*NOVA/ig*[1]		*ENCORE*		random		
FSM	#cubes	#bits	#cubes	area	#cubes	area	#cubes	area	#cubes[2]	area[2]	area[3]
dk15	17	2	19	323	18	306	18	306	20	340	368
lion	8	2	6	66	6	66	7	77	7	77	96
mc	10	2	9	153	9	153	8	136	9	153	157
tav	12	2	11	198	11	198	11	198	11	198	198
train4	7	2	6	66	6	66	6	66	7	77	80
s8	14	3	10	180	10	180	9	162	9	162	198
bbtas	16	3	9	135	12	180	10	150	12	180	206
beecount	12	3	13	247	12	228	10	190	14	266	299
dk14	25	3	29	580	27	540	27	540	33	660	743
dk27	10	3	9	117	8	104	8	104	9	117	141
dk17	20	3	19	304	19	304	17	272	19	304	352
ex6	23	3	25	675	25	675	25	675	30	810	842
shiftreg	9	4	4	48	8	96	6	72	11	132	132
ex5	19	4	14	252	18	324	16	288	18	324	352
lion9	10	4	8	136	9	153	8	136	11	187	250
bbara	34	4	25	550	25	550	25	550	28	616	662
ex3	21	4	18	324	18	324	17	306	19	342	405
ex7	20	4	17	306	17	306	18	324	17	306	387
opus	19	4	16	448	16	448	18	504	20	560	581
train11	11	4	9	153	12	204	10	170	12	204	243
modulo12	24	4	12	180	12	180	14	210	12	180	195
ex4	21	4	19	627	19	627	21	693	19	627	683
dk512	21	4	18	306	19	323	19	323	22	374	419
mark1	19	4	21	798	19	722	18	684	19	722	786
bbsse	30	4	30	990	30	990	30	990	32	1056	1173
cse	57	4	46	1518	45	1485	45	1485	51	1683	2083
kirkman	61	4	79	3318	77	3234	61	2745	87	3654	4641
sse	30	4	30	990	30	990	30	990	32	1056	1176
ex2	38	5	29	609	36	756	32	672	38	798	905
keyb	77	5	48	1488	55	1705	51	1581	58	1798	3154
ex1	44	5	48	2496	51	2652	45	2475	60	3120	3281
s1	92	5	80	2960	87	3219	86	3182	96	3553	3703
s1a	92	5	76	2812	80	2960	73	2701	84	3108	3468
donfile	24	5	35	700	48	960	**18**	**360**	60	1200	1382
dk16	55	5	59	1298	72	1584	58	1276	87	1914	1981
styr	111	5	94	4042	103	4429	93	3999	126	5418	5691
sand	114	5	101	4646	102	4692	100	4600	93	4278	4972
tbk	173	5	154	4620	176	5280	**128**	**3840**	183	5490	6090
planet	92	6	91	4641	89	4539	90	4590	96	4896	5260
scf	151	7	148	19388	146	19126	140	18340	152	19912	21294
TOTAL				63688		65858		60979		70852	79029
%				90		93		86		100	112

[1]: To enable a fair comparison, we do not use the -r option.
[2]: best random solution
[3]: average of random solutions
#bits : code-length
#cubes: number of product-terms after *ESPRESSO* logic minimization
area: (2*(#inputs + #bits) +#bits +#outputs)*#cubes

asynchronous design methodology [25].

Second, it is demonstrated for the first time that synthesis results obtained using dichotomy constraints are comparable with the conventional face-embedding constraints in terms of PLA area used. We note that, while the reason for maximizing the number of satisfied face-embedding constraints is intuitively clear, the reason why maximizing the number of satisfied dichotomy constraints still yields the same result is not obvious. A theoretical analysis is needed as to improve our understanding of this aspect of sequential logic synthesis.

Chapter 10

Conclusions and Future Research

Layer assignment and logic encoding are two problems that have presented a challenge to both theoreticians and CAD practitioners for more than two decades. Except for some special cases, these problems were formulated and solved previously in an ad hoc manner. Such an approach cannot keep pace with the ever-increasing complexity of VLSI design, which demands an efficient and effective solution of large instances of problems. In this thesis, a new graph-theoretic framework was established, which, for the first time, captures precisely the underlying combinatorial structure of the two problems.

In this chapter, the main results of the thesis are first summarized in terms of this newly-established graph-theoretic framework. The practical impact of these results on the two VLSI design problems is then reviewed. Finally, some research topics in VLSI design that appear to be promising in view of the results of the thesis are pointed out.

10.1 Contributions to Mathematical Theory

A notion of signed hypergraph was formally introduced in the thesis. A signed hypergraph is a hypergraph in which each vertex is incident with an edge either positively or negatively. This notion captures the underlying structures of the two practical VLSI problems, generalizes Harary's notion of signed graph, Berge's notion of hypergraph, and formalizes Yannakakis' notion of bipartite weighted graphs.

With the motivation from VLSI applications, we studied one specific property, called balance, of signed hypergraphs. An edge e is said to be balanced by a bipartition if all the vertices positively incident with e are in one block of the bipartition, and all the vertices negatively incident with e are in the other block. A signed hypergraph is balanced if there exists a bipartition that balance all the edges. We established a structural characterization of balanced signed hypergraphs: such signed hypergraphs are free of negative cycles, where a negative cycle is one with an odd number of negative vertex-edge incidences. Furthermore, a linear-time algorithm for balance testing was described. Our characterization is a generalization of Harary's theorem on signed graphs. We also noted that a slightly different definition of the sign of the cycle leads to the balance concept related to matrix unimodularity, a fundamental problem in mathematical programming.

We studied two balance-related optimization problems in signed hypergraphs. The maximum balance problem is to find a bipartition that maximizes the number of balanced edges in a signed hypergraph. The minimum covering problem is to find a minimum number of bipartitions such that each edge is balanced by at least one bipartition. On the basis of our structural theorem, we proved that both the maximum balance problem and the minimum covering problem are NP-complete. Further, we proved that the maximum balance problem for planar signed hypergraphs is also NP-complete.

An integer linear programming formulation was presented for the exact solution of the maximum balance problem. By using polyhedral combinatorics, we derived a polynomial-size linear programming formulation for planar signed graphs.

We introduced a new concept of hypergraph T-join, as a generalization of graph T-join. We defined the minimum hypergraph T-join problem as finding the hypergraph T-join of maximum cardinality. We showed that the maximum balance problem in a planar signed hypergraph reduces to the minimum hypergraph T-join problem in its planar dual. Although the minimum hypergraph T-join problem is NP-complete even for planar hypergraphs, its special case—the minimum graph T-join problem—can be solved very efficiently.

We addressed the problem of modeling signed hypergraphs by real-weighted hypergraphs or graphs and established a hierarchy in terms of a cut property. We settled a conjecture of Lengauer which states that a clique is the best approximate model for a hyperedge even if dummy vertices are allowed. This modeling gives rise to good approximation algorithms for the maximum balance problem.

We presented an implementation of local search for the maximum balance problem in general signed hypergraphs. With suitable data structures, one pass of local search takes linear time. It may take an exponential number of passes for local search to converge; however, in practice, only a small number of passes is needed. We also pointed out our implementation belongs to the complete class PLS (Polynomial-time Local Search) studied by theoreticians.

We described a simple greedy peeling heuristic for solving the minimum covering problem. We proved that greedy peeling has a guaranteed performance bound for solving a class of VLSI optimization problems of the so-called cluster-cover structure. Our theoretical result is applicable to a variety of published heuristics, which previously could be evaluated only with respect to benchmarks.

10.2 Contributions to Optimum Layer Assignment

The optimum layer assignment problem in two-layer routing is, for the first time, captured precisely by the graph-theoretic framework established in this thesis. It is formulated as the maximum balance problem in planar signed hypergraphs. Therefore all the results obtained in this thesis for planar signed hypergraphs are applicable to optimum layer assignment. This implies a simple proof of the NP-completeness of the optimum layer assignment problem (Chapter 2), a more efficient polynomial algorithm for the restricted case (Chapter 3), a compact linear programming formulation for the restricted case (Chapter 7), a pseudo-polynomial algorithm for the general case (Chapter 3), and a near-optimum approximation algorithm for the general case (Chapter 4). On the basis of theoretical properties of signed hypergraphs, our work reveals the inherent difficulty of the formulation of the problem by existing graph concepts (Chapter 4). An experimental program has also been written for optimum layer assignment with timing constraints (Chapter 8).

10.3 Contributions to Logic Synthesis

Our model of constrained encoding provides a unified framework for encoding problems arising from optimum state assignment for synchronous circuits, race-free state assignment for asynchronous circuits, delay-free state assignment for asynchronous circuits without essential hazards, and combinational logic decomposition. The combinatorial structure of constrained encoding was described by a graph-theoretic notion for the first time. The constrained encoding problem was formulated as the minimum covering problem in signed hypergraphs. The problem has been proved to be NP-complete, on the basis of theoretical properties of signed hypergraphs.

A simple greedy heuristic was presented and implemented in an experimental program *ENCORE*. *ENCORE* has been applied to a variety of practical problem instances. For a number of examples found in the literature on the synthesis of asynchronous sequential machines, *ENCORE* consistently obtains optimal or near-optimal results. For the optimum state assignment of the MCNC FSM benchmarks, *ENCORE* generates the same or even shorter encoding lengths than the programs *KISS*, *NOVA* and *DIET*, but takes much less CPU time. It was demonstrated for the first time that PLA implementations of synchronous FSMs using dichotomy constraints compare very favorably with respect to area with those based on traditional group constraints.

10.4 Incompletely Constrained Via Minimization

We consider a slightly relaxed version of constrained via minimization. Illustrated in Fig 10.1(a) is an instance of the constrained via minization problem and its optimum solution. It uses two vias at a single net (net 1). Now we assume that the geometry position of the wire of net 1 can be changed. We further assume that wires do not take any space. Then, only one via is needed, as shown in Fig. 10.1(b). This routing model is called *incompletely constrained via minimization*. In the extreme case, if all the routing wires are allowed to change their geometry positions (only terminals are fixed), this gives rise to *topological (unconstrained) via minimization*, a problem that was first addressed by Hsu [65], and has been well studied by Sarrafzadeh and Lee [124, 125].

In order to have incompletely constrained via minimization practically useful, geometrical mapping is required in the same way as for topological via minimization [65]. Nevertheless, the incompletely constrained model of via minimization posed here is a generalized, perhaps more realistic, version of the constrained and topological models of via minimization. There are two basic situations where this model may be useful. First, constrained via minimization may result in many vias in critical nets; this usually degrades the circuit performance. A possible remedy to this problem is to reroute these critical nets to achieve a further via reduction. This interesting question also arises in real layout design: It is often the case that all the nets except a few (less than 5%) have been routed successfully. Then an engineering practice, known

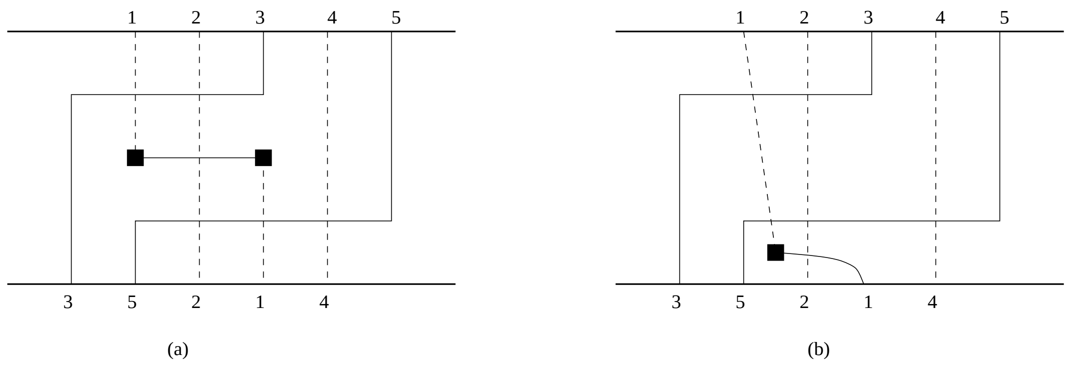


Figure 10.1: An example of incompletely constrained via minimization.

as rip-up and re-routing, as discussed in the survey paper by Kuh and Ohtsuki [86], is invoked. Second, if we use the topological routing approach, some critical nets may have to be routed first due to performance consideration. This also gives rise to incompletely constrained via minimization. In addition, if geometrical mapping for incompletely constrained via minimization can be easier than that for unconstrained via minimization—this may be the case because some routings have been mapped already—then incompletely constrained via minimization may be useful in bringing topological routing to become useful in real layout design [86].

Now we show how the incompletely constrained via minimization problem can be formulated using the framework developed in this thesis. For simplicity, we assume that we use the maximal set of potential vias: each crossing forms a cluster. Then the planar sign graphs for routings in Fig. 10.1(a) and Fig. 10.1(b) are shown in Fig. 10.2(a) and Fig. 10.2(c), respectively. Figure 10.2(b) gives the planar signed graph for the routing without net 1.

It will be more convenient if we consider the planar dual of a planar signed graph (Chapter 3). In order to take into account the channel routing restriction, i.e., the area outside the channel is not allowed for routing, we need to consider each face

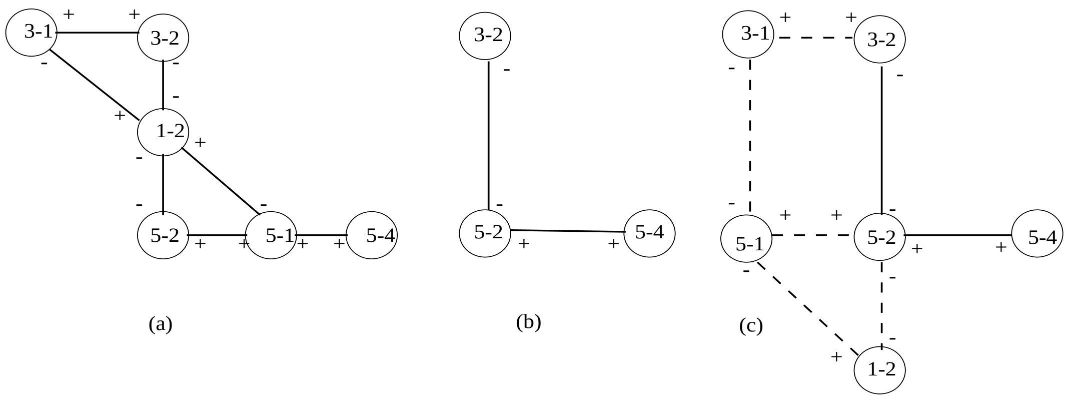


Figure 10.2: Signed graphs for incompletely constrained via minimization.

adjacent to the channel separately (If we allow the "over-the-cell" routing, then all these faces will be merged into one exterior face). Note that, as far as constrained via minimization is concerned, all these faces must be merged to form the exterior face (see Chapter 3). With the selection of a maximal set of potential vias, we can draw the planar dual directly from the routing. Figure 10.3(a) shows the planar dual for our routing example without net 1. It is easily verified that as far as constrained via minimization is concerned, this planar dual degenerates to a single-vertex graph, its T-join is empty, and no via is required.

Now we consider how to represent a topological routing of a net in the planar dual. We mark by s and t the two vertices corresponding to the two *terminal* faces where the terminals of the net are located (See Fig. 10.3(a)). (We consider here only two-terminal nets.) A routing of the net goes through a set of faces between the two terminal faces; each such face will be separated into two faces by the routing. In terms of the planar dual, this process is equivalent to replacing an $s-t$ path by a corresponding *ladder*. An example of a ladder is shown in Fig. 10.3 for the routing of net 1 in Fig 10.1(a)). An $s-t$ path determines a unique ladder as follows: each vertex in the path is split into two new vertices, and the edges connected to the path from one side of the path are connected to the new vertex in that side. For example, the replacement of the path in Fig. 10.3(a) by its corresponding ladder in Fig. 10.3(b) leads to the planar dual in Fig. 10.4(c) for Fig. 10.4(a). The planar dual for Fig. 10.4(b) is shown in Fig. 10.4(d). It is easily verified that, in Fig. 10.4(c), there exist two negative vertices (pointed to by arrows). A minimum T-join that connects the two vertices uses two edges (two vias). For Fig. 10.4(d), there exist two negative vertices (pointed to by arrows) one being the exterior vertex. The minimum T-join here uses only one edge (one via).

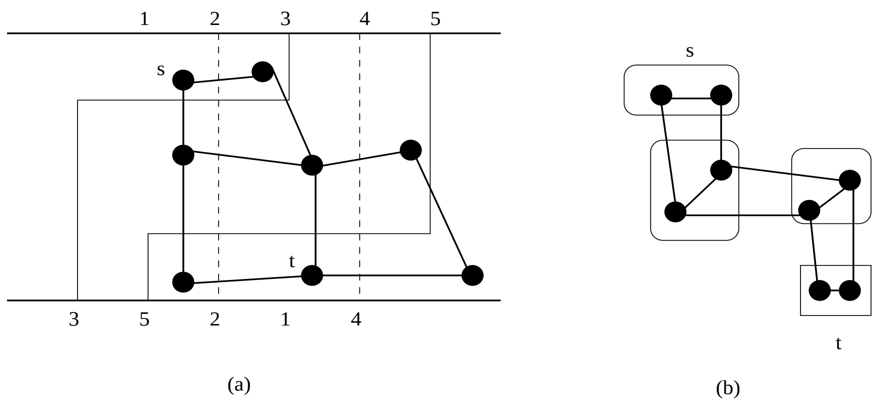


Figure 10.3: Planar dual graphs for incompletely constrained via minimization.

We have the following graph-theoretic formulation of the incompletely constrained via minimization problem for a given routing with additional k unconstrained nets. We are given a planar marked graph corresponding to the given routing, and k pairs of $s-t$ vertices corresponding to terminals of k nets. The problem is to find k $(s-t)$-paths such that the replacement by their corresponding ladders will result in a graph with the smallest minimum T-join.

The problem formulated above is a new graph-theoretic problem. It will be interesting to investigate its complexity, since we know that the constrained via minimization for two-layer routing with simple splits is polynomially solvable, as is the

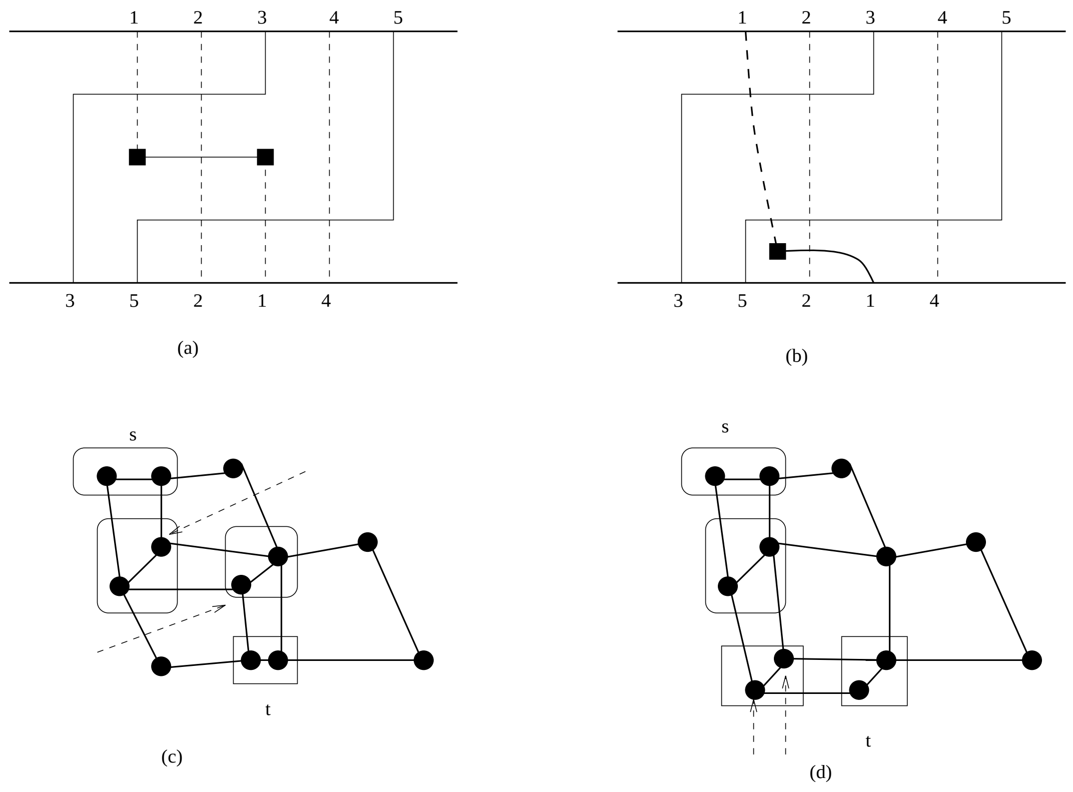


Figure 10.4: Planar dual examples for incompletely constrained via minimization.

topological via minimization for two-layer channel routing [124]. We note that it is difficult to formulate incompletely constrained via minimization using existing topological via minimization approaches.

10.5 Other Problems

In addition to the specific topic described in the last section, several other research topics in VLSI design appear to be promising in view of the results of this thesis. Among them, an application of our model of dichotomy-based constrained encoding to various non-conventional logic synthesis problems is of primary interest to logic designers. These problems include FPGA-targeted, timing-driven, and testability-driven logic synthesis. We feel that the objectives and conditions of these problems can be modeled by dichotomy constraints, which in turn can be formulated in our framework of signed hypergraph. Two additional problems along this line are state minimization and state splitting [148]. Our experiments suggest that state minimization and state splitting can be combined together with state encoding. Several other layer assignment problems are of interest to layout designers. These include multi-layer via minimization, multichip-module via minimization [61], and layout wirability [144].

On the theoretical side, because our notion of signed hypergraph is a generalization of several known graph-theoretic concepts, future research may also include the study of other questions, that have been studied before for graphs, for signed hypergraphs. In particular, we are investigating the possibility of applying the notion of signed hypergraph to derive a polynomial-time local search algorithm for the *1-opt* MAX-CUT problem.

Bibliography

[1] J. I. Acha and J. Calvo, "On the implementation of sequential circuits with PLA modules", *IEE Proceedings*, vol. 132, pt. E., no. 5, pp. 246-250, Sept. 1985.

[2] I. Adler, N. Karmarkar, M. G. C. Resende and G. Veiga, *An implementation of Karmarkar's algorithm for linear programming*, Technical Report, Department of Industrial Engineering and Operations Research, University of California, Berkeley, CA 94720, June 1987.

[3] A. V. Aho, J. E. Hopcroft, and J. D. Ullman, *The Design and Analysis of Computer Algorithms*, Reading, MA: Addison-Wesley, 1976.

[4] K. Aoshima and M. Iri, "Comments on F. Hadlock's paper: Finding a maximum cut of a planar graph in polynomial time", *SIAM J. Comput.*, vol. 6, no. 1, pp. 86-89, Mar. 1977.

[5] S. Arona, C. Lund, R. Motwani, M. Sudan and M. Sezgedy, "Proof verification and intractability of approximation problems", pp. 14-23 in *Proc. 33rd Annual IEEE Symposium on Foundations of Computer Science*, Pittsburgh, U.S.A., Oct. 24-27, 1992.

[6] F. Barahona, "On via minimization", *IEEE Trans. Circuits Syst.*, vol. CAS-37, no. 4, pp. 527-530, April 1990.

[7] F. Barahona, “Planar multicommodity flows, max cut, and the Chinese postman problem”, *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, vol. 1, pp. 189-202, 1990.

[8] F. Barahona, M. Grötschel, M. Jünger and G. Reinelt, “An application of combinatorial optimization to statistical physics and circuit layout problem”, *Operations Research*, vol. 36, no. 3, pp. 493-513, 1988.

[9] L. W. Beuneke and F. Harary, “Consistent graphs with signed points”, *Riv. di Mat. per le Sci. Econ. e Soc.*, vol. 1, pp. 81-88, 1978.

[10] J. L. Bentley, “Experiments on traveling salesman heuristics”, pp. 91-99 in *Proc. First SIAM Symposium on Discrete Algorithms*, 1990.

[11] C. Berge, “Balanced matrices”, *Math. Programming*, vol. 2, pp. 19-31, 1972.

[12] C. Berge, *Graphs and Hypergraphs*, Amsterdam: North-Holland, 1973.

[13] J. A. Bondy and U. S. R. Murty, *Graph Theory with Applications*, New York: American Elsevier, 1976.

[14] E. Boros, Y. Crama, and P. L. Hammer, “Chvátal cuts and odd cycle inequalities in quadratic 0-1 optimization”, *SIAM J. Disc. Math.*, vol. 5, no. 2, pp. 163-177, May 1992.

[15] M. L. Brady and D. J. Brown, “VLSI routing: four layers suffice”, *Advances in Computing Research*, vol. 2, pp. 245-257, ed. F. P. Preparata, JAI Press Inc, 1984.

[16] D. Braun, J. L. Burns, F. Romeo, A. L. Sangiovanni-Vincentelli, K. Mayaram, S. Devadas, and H.-K. T. Ma, “Techniques for multilayer channel routing”, *IEEE Trans. Computer-Aided Design*, vol. CAD-7, no. 6, pp. 713-722, June 1988.

[17] R. K. Brayton, G. D. Hachtel, C. T. McMullen and A. L. Sangiovanni-Vincentelli, *Logic Minimization Algorithms for VLSI synthesis*, New York: Kluwer Academic Publishers, 1984.

[18] J. A. Brzozowski and C.-J. Seger, *Asynchronous Networks*, Research Monograph, in preparation.

[19] *Using the CPLEX Callable Library and CPLEX Mixed Integer Library*, CPLEX Optimization, Inc., 1992.

[20] K. Cameron and J. Edmonds, "Existentially polytime theorems", *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, vol. 1, pp. 83-100, 1990.

[21] K. C. Chang and D. H.-C. Du, "Efficient algorithms for layer assignment problem", *IEEE Trans. Computer-Aided Design*, vol. CAD-6, no. 1, pp. 67-78, Jan. 1987.

[22] K. C. Chang and D. H.-C. Du, "Layer assignment problem for three-layer routing", *IEEE Trans. Computer*, vol. C-37, pp. 625-632, May 1988.

[23] R. W. Chen, Y. Kajitani, and S. P. Chan, "A graph-theoretic via minimization algorithm for two-layer printed boards", *IEEE Trans. Circuits Syst.*, vol. CAS-30, no. 5, pp. 284-299, May 1983.

[24] H.-A. Choi, K. Nakajima, and C. S. Rim, "Graph bipartition and via minimization", *SIAM J. Disc. Math.*, vol. 2, pp. 38-47, Feb. 1989.

[25] T. A. Chu, *Synthesis of Self-Timed VLSI Circuits from Graph-Theoretic Specifications*, Ph.D. Dissertation, Department of Electrical Engineering and Computer Science, Massachusetts Institute of Technology, June 1987.

[26] V. Chvátal, "A greedy heuristic for the set covering problem", *Mathematics of Operations Research*, vol. 4, pp. 233-235, 1979.

[27] M. J. Ciesielski and E. Kinnen, "An optimum layer assignment for routing in ICs and PCBs", pp. 733-737 in *Proc. 18th IEEE/ACM Design Automat. Conf.*, 1981.

[28] M. J. Ciesielski, "Layer assignment for VLSI interconnect delay minimization", *IEEE Trans. Computer-Aided Design*, vol. CAD-8, no. 6, pp. 702-707, June 1989.

[29] M. Conforti and M. R. Rao, "Structural properties and recognition of restricted and strongly unimodular matrices", *Math. Programming*, vol. 38, pp. 17-27, 1987.

[30] M. Conforti and G. Cornuéjols, "A class of logic problems solvable by linear programming", pp. 670-675 In *Proc. 33rd Annual IEEE Symposium on Foundations of Computer Science*, IEEE Computer Society, 1992.

[31] J. Cong, M. Hossain, and N. A. Sherwani, "A provably good multilayer topological planar routing algorithm in IC layout designs", *IEEE Trans. Computer-Aided Design*, vol. 12, no. 1, pp. 70-78, Jan. 1993.

[32] S. A. Cook, "The complexity of theorem-proving procedures", pp. 151-158 in *Proc. 3rd Annual ACM Symposium on Theory of Computing*, 1971.

[33] T. H. Cormen, C. E. Leiserson, and R. L. Rivest, *Introduction to Algorithms*, Cambridge, MA: The MIT Press, 1990.

[34] H. A. Curtis, *A New Approach to the Design of Switching Circuits*, Princeton NJ: D. Van Nostrand, 1962.

[35] D. Cvetkovic, M. Doob, I. Gutman, and A. Torgasev, *Recent Results in the Theory of Graph Spectra*, New York: North-Holland, 1988.

[36] G. B. Dantzig, *Linear Programming and Extensions*, Princeton NJ: Princeton University Press, 1962.

[37] M. Davis and H. Putnam, "A computing procedure for quantification theory", *J. ACM*, pp. 201-215, 1960.

[38] S. Devadas and A. R. Newton, "Exact algorithms for output encoding, state assignment, and four-level Boolean minimization", *IEEE Trans. Computer-Aided Design*, vol. CAD-10, no. 1, pp. 13-27, Jan. 1991.

[39] S. Devadas, A. R. Wang, A. R. Newton, and A. L. Sangiovanni-Vincentelli, "Boolean decomposition of programmable logic arrays", pp. 2.5.1-2.5.5 in *Proc. IEEE Custom Integrated Circuits Conf.*, 1988.

[40] W. E. Donath and A. J. Hoffman, "Lower bounds for the partitioning of graphs", *IBM J. Res. Develop.* pp. 420-425, 1973.

[41] T. A. Dolotta and E. J. McCluskey, "The coding of internal states of sequential machines", *IEEE Trans. Elect. Comput.*, vol. EC-13, pp. 549-562, Oct. 1964.

[42] J. Edmonds and E. L. Johnson, "Matching, Euler tours and the Chinese postman", *Math. Programming*, vol. 5, pp. 88-124, 1973.

[43] W. Elmore, "The transient response of damped linear networks with particular regard to wideband amplifiers", *J. of Applied Physics*, vol. 19, pp. 55-63, Jan. 1948.

[44] U. Faigle, "The greedy algorithm for partially ordered sets", *Discrete Mathematics*, vol. 28, pp. 153-159, 1979.

[45] C. M. Fiduccia and R. M. Mattheyses, "A linear-time heuristic for improving network partitions", pp. 175-181 in *Proc. 19th IEEE/ACM Design Automat. Conf.*, 1982.

[46] L. R. Ford, Jr. and D. R. Fulkerson, *Flows in Networks*, Princeton, NJ: Princeton University Press, 1962.

[47] Z. Galil, S. Micali, and H. Gabow, "An $O(EV \log V)$ algorithm for finding a maximal weighted matching in general graphs", *SIAM J. Comput.*, vol. 15, no. 1, pp. 120-130, 1986.

[48] M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*, San Francisco, CA: Freeman, 1979.

[49] M. R. Garey, D. S. Johnson, and L. Stockmeyer, "Some simplified NP-complete graph problems", *Theoretical Computer Science*, no. 1, pp. 237-267, 1976.

[50] R. S. Garfinkel and G. L. Nemhauser, *Integer Programming*, New York: Wiley, 1972.

[51] M. Gondran and M. Minoux, *Graphs and Algorithms*, New York: Wiley, 1984.

[52] I. S. Gopal, D. Coppersmith, and C. K. Wong, "Optimal wiring of movable terminals", *IEEE Trans. Computers*, vol. C-32, no. 9, pp. 845-858, Sept. 1983.

[53] M. Gössel and H. Jürgensen, "Monitoring BIST by covers", pp. 208-213 in *Proc. Euro-DAC'93 — European Design Automation Conf.*, Hamburg, Germany, Sept. 1993, IEEE Computer Society Press.

[54] J. Gu, "Efficient local search for very large scale satisfiability problems", *SIGART Bulletin*, vol. 3, no. 1, pp. 8-12, Jan. 1992.

[55] A. Haken and M. Luby, "Steepest descent can take exponential time for symmetric connection networks", *Complex Systems*, vol. 2, pp. 191-196, 1988.

[56] S. W. Hadley, B. L. Mark, and A. Vannelli, "An efficient eigenvector approach for finding netlist partitions", *IEEE Trans. Computer-Aided Design*, vol. CAD-11, no. 7, pp. 885-892, July 1992.

[57] F. Hadlock, "Finding a maximum cut of a planar graph in polynomial time", *SIAM J. Compu.*, vol. 4, no. 3, pp. 221-225, Sept. 1975.

[58] L. Hagen and A. B. Kahng, "New spectral methods for ratio cut partitioning and clustering", *IEEE Trans. Computer-Aided Design*, vol. CAD-11, no. 9, pp. 1074-1085, Sept. 1992.

[59] F. Harary, "On the notation of balance of a signed graph", *Michigan Math. J.*, vol. 2, pp. 143-146, 1953-54.

[60] A. Hashimoto and J. Stevens, "Wire routing optimizing channel assignment within large apertures", pp. 155-169 in *Proc. 8th Design Automat. Workshop*, June 1971.

[61] J. M. Ho, M. Sarrafzadeh, G. Vijayan, and C. K. Wong, "Layer assignment for multichip modules", *IEEE Trans. Computer-Aided Design*, vol. 9, no. 12, pp. 1272-1277, Dec. 1990.

[62] C. Hoede, "A characterization of consistent marked graphs", *J. of Graph Theory*, vol. 16, no. 1, pp. 17-23, 1992.

[63] J. E. Hopcroft and J. D. Ullman, *Introduction to Automata Theory, Languages, and Computation*, Reading, MA: Addison-Wesley, 1979.

[64] T. C. Hu, *Integer Programming and Network Flows*, Reading, MA: Addison-Wesley, 1970.

[65] C. P. Hsu, "Minimum-via topological routing", *IEEE Trans. Computer-Aided Design*, vol. CAD-2, no. 4, pp. 235-246, Oct. 1983.

[66] E. Ihler, D. Wagner and F. Wagner, *Modeling hypergraphs with graphs with the same mincut properties*, B92-22, Series B - Informatik, Fachbereich Mathematik, Freie Universität Berlin, Oct. 1992.

[67] V. S. Iyengar and G. Vijayan, "Optimized test application timing for AC test", *IEEE Trans. Computer-Aided Design* vol. 11, no. 11, pp. 1439-1449, Nov. 1992.

[68] M. A .B. Jackson and E. S. Kuh, "Performance-driven placement of cell based IC's", pp. 370-375 in *Proc. 26th IEEE/ACM Design Automat. Conf.*, 1989.

[69] D. S. Johnson, "Approximating algorithms for combinatorial problems", *J. of Computer and System Sciences*, vol. 9, pp. 256-278, 1974.

[70] D. S. Johnson, C. H. Papadimitriou, and M. Yannakakis, "How easy is local search?", pp. 39-42 in *Proc. 26th Annual IEEE Symposium on Foundations of Computer Science*, IEEE Computer Society, 1985; *J. of Computer and System Sciences*, vol. 37, pp. 79-100, 1988.

[71] D. A. Joy and M. J. Ciesielski, "Layer assignment for printed circuit boards and integrated circuits", *Proceedings of the IEEE*, vol. 80, no. 2, Feb. 1992.

[72] A. B. Kahng, "Fast hypergraph partition", pp. 762-766 in *Proc. 26th IEEE/ACM Design Automat. Conf.*, 1989.

[73] Y. Kajitani, "On via hole minimization of routing on a 2-layer board", pp. 295-298 in *Proc. IEEE International Conference on Circuits and Computers*, June 1980.

[74] T. Kameda and I. Munro, "A $O(|V||E|)$ algorithm for maximum matching of graphs", *Computing*, vol. 12, pp. 91-98, 1974.

[75] N. Karmarkar, "A new polynomial-time algorithm for linear programming", *Combinatorica*, vol. 4, no. 4, pp. 373-395, 1984.

[76] A. P. Kamath, N. K. Karmarkar, K. G. Ramarkrishnan and M. G. C. Resende, *Computational experience with an interior point algorithm on the Satisfiability*

problem, October 1989, Working Paper, Mathematical Sciences Research Center, AT&T Bell Laboratories, Murray Hill, NJ 07974 USA.

[77] B. W. Kernighan and S. Lin, "An efficient heuristic procedure for partitioning graphs", *Bell Systems Technical J.*, vol. 49, pp. 291-307, 1970.

[78] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi, "Optimization by simulated annealing", *Science*, vol. 220, pp. 671-680, May 1983.

[79] P. Klein and R. Ravi, "A nearly best-possible approximation algorithm for node-weighted Steiner trees", manuscript, 1993.

[80] B. Korte and L. Lovász, "Greedoids – A structural framework for the greedy algorithm", *Progress in Combinatorial Optimization*, W. Pulleyblank (ed.), pp. 221-243, 1984.

[81] L. T. Kou, L. J. Stockmeyer, and C. K. Wong, "Covering edges by cliques with regard to keyword conflicts and intersection graphs", *Comm. ACM*, vol. 21, pp. 135-138, 1978.

[82] M. W. Krentel, "On finding locally optimal solutions" pp. 132-137 in *Proc. 4th Annual IEEE Conf. on Structure in Complexity*, Eugene, OR, 1989; *SIAM J. Comput*, vol. 19, pp. 742-749, 1990.

[83] M. S. Krishnamoorthy and N. Deo, "Node-deletion NP-complete problems", *SIAM J. Comput.*, pp. 619-625, no. 8, 1979.

[84] B. Krishnamurthy, "An improved min-cut algorithm for partitioning VLSI networks", *IEEE Trans. Computers*, vol. C-33, no. 5, pp. 438-446, May 1984.

[85] B. Krishnamurthy, "Constructing test cases for partitioning heuristics", *IEEE Trans. Computers*, vol. C-36, no. 9, pp. 1112-1114, Sep. 1987.

[86] E. S. Kuh and T. Ohtsuki, "Recent advances in VLSI layout", *Proceedings of the IEEE*, vol. 78, No.2, pp. 237-263, Feb. 1990.

[87] Y. S. Kuo, T. C. Chern, and W. K. Shih, "Fast algorithm for optimal layer assignment", pp. 554-559 in *Proc. 25th IEEE/ACM Design Automat. Conf.*, 1988.

[88] E. L. Lawler, *Combinatorial Optimization: Networks and Matroids*, New York: Holt, Rinehart and Winston, 1976.

[89] D. T. Lee, S. J. Hong, and C. K. Wong, "Number of vias: A control parameter for global wiring of high-density chips", *IBM J. Res. Develop.*, vol. 25, no. 4, pp. 261-271, July 1981.

[90] T. Leighton and S. Rao, "An approximate max-flow min-cut theorem for uniform multicommodity flow problems with applications to approximation algorithms", pp. 422-431 in *Proc. 28th Annual IEEE Symposium on Foundations of Computer Science*, 1988.

[91] T. Lengauer, *Combinatorial Algorithms for Integrated Circuit Layout*, Chichester, U.K.: Wiley-Teubner, 1990.

[92] D. Lichtenstein, "Planar formulae and their uses", *SIAM J. Comput*, vol. 11 no. 2, pp. 329-336, 1982.

[93] S. Lin and B. W. Kernighan, "An efficient heuristic for the traveling salesman problem", *Operations Research*, vol. 21, pp. 498-516, 1973.

[94] R. J. Lipton and R. E. Tarjan, "Applications of a planar separator theorem", *SIAM J. Comput.*, vol. 9, pp. 615-627, 1980.

[95] W. Lipski, Jr, "On the structure of three-layer wirable layouts", pp. 231-243 in *Advances in Computing Research*, vol. 2, ed. F. P. Preparata, JAI Press Inc, 1984.

[96] L. Lovász, "On the ratio of optimal integral and fractional covers", *Discrete Mathematics*, vol. 13, pp. 383-390, 1975.

[97] L. Lovász and M. D. Plummer, *Matching Theory*, Budapest: Akadémiai Kiadó, 1986.

[98] D. G. Luenberger, *Linear and Nonlinear Programming*, 2nd edition, Reading, MA: Addison-Wesley, 1984.

[99] C. Lund and M. Yannakakis, "On the hardness of approximating minimization problems", Manuscript, AT&T Bell Lab., Murray Hill, NJ 07974.

[100] M. Marek-Sadowska, "An unconstrained topological via minimization problem for two-layer routing", *IEEE Trans. Computer-Aided Design*, vol. CAD-3, pp. 184-190, July 1984.

[101] D. W. Marquardt, "An algorithm for least-squares estimation of nonlinear parameters", *SIAM J. Appl. Math.*, vol. 11, pp. 431-441, 1963.

[102] P. C. McGeer and R. K. Brayton, "Efficient algorithms for computing the longest viable path in a combinational network", pp. 561-567 in *Proc. 26th IEEE/ACM Design Automat. Conf.*, 1989.

[103] G. D. Micheli, R. K. Brayton, and A. L. Sangiovanni-Vincentelli, "Optimal state assignment for finite state machines", *IEEE Trans. Computer-Aided Design*, vol. CAD-4, no. 3, pp. 269-285, July 1985.

[104] G. D. Micheli, "Symbolic design of combinational and sequential logic circuits implemented by two-level logic macros" *IEEE Trans. Computer-Aided Design*, vol. CAD-5, no. 1, pp. 597-616, Oct. 1986.

[105] B. A. Murtagh and M. A. Saunders, *MINOS 5.0 User's Guide*, Technical Report SOL 83-20, Department of Operations Research, Stanford University, Stanford, CA, 1983.

[106] N. Munksgaard, "Solving sparse symmetric sets of linear equations by preconditioned conjugate gradients", *TOMS*, vol. 6, pp. 206-219, 1980.

[107] P. Molitor, "On the contact minimization problem" pp. 420-431 in *Proc. 4th Annu. Symp. on Theoretical Aspects of Computer Science*, Feb. 1987.

[108] P. Molitor, "Constrained via minimization for systolic arrays", *IEEE Trans. Computer-Aided Design*, vol. CAD-9, no. 5, pp. 537-542, May 1990.

[109] F. Motika, N. N. Tendolkar, C. C. Beh, W. R. Heller, C. E. Radke, and P. J. Nigh, "A logic chip delay test method based on system timing", *IBM J. Res. Develop.*, vol. 34, nos.2/3. pp. 299-313, Mar./May 1990.

[110] N. J. Naclerio, S. Masuda, and K. Nakajima, "Via minimization for gridless layouts", pp. 159-165 in *Proc. 24th IEEE/ACM Design Automat. Conf.*, Miami Beach, FL, June 1987.

[111] N. J. Naclerio, S. Masuda, and K. Nakajima, "The via minimization problem is NP-complete", *IEEE Trans. Computers*, vol. C-38, no. 11, pp. 1604-1608, Nov. 1989.

[112] R. Nair, C. L. Berman, P. S. Hauge, and E. J. Yoffa, "Generation of performance constraints for layout", *IEEE Trans. Computer-Aided Design*, vol. CAD-8, no. 8, pp. 860-874, Aug. 1989.

[113] C. H. Papadimitriou and K. Steiglitz, *Combinatorial Optimization: Algorithms and Complexity*, Englewood Cliffs, NJ: Prentice-Hall, 1982.

[114] C. H. Papadimitriou, A. A. Schäffer, and M. Yannakakis, "On the complexity of local search", pp. 438-445 in *Proc. 22rd Annual ACM Symposium on Theory of Computing*, 1990.

[115] C. H. Papadimitriou and M. Yannakakis, "Optimization, approximation, and complexity classes", *J. of Computer and System Sciences*, vol. 43, pp. 425-440, 1991.

[116] C. H. Papadimitriou, "The complexity of the Lin-Kernighan heuristic for the traveling salesman problem", *SIAM J. Compt.*, vol. 21, no. 3, pp. 450-465, June 1992.

[117] R. Y. Pinter, "Optimal layer assignment for interconnect", pp. 398-401 in *Proc. IEEE International Conference on Circuits and Computers*, 1982.

[118] S. Prasitjutrakul and W. J. Kubtiz, "Path-delay constrained floorplanning: a mathematical programming approach for initial placement", pp. 364-369 in *Proc. 26th IEEE/ACM Design Automat. Conf.*, 1989.

[119] W. Pulleyblank, *Polyhedral Combinatorics*, pp. 312-345 in *Mathematical Programming, the State of the Art: Bonn 2982*, Eds. A. Backem, M. Gróschel and B. Korte, Berlin: Springer-Verlag, 1983.

[120] J. Rubinstein, P. Penfield, and M. A. Horowitz, "Signal delay in RC networks", *IEEE Trans. Computer-Aided Design*, pp. 202-210, July 1982.

[121] R. L. Rudell and A. L. Sangiovanni-Vincentelli, "Multiple-valued minimization for PLA optimization", *IEEE Trans. Computer-Aided Design*, vol. CAD-6, no. 5, pp. 727-750, Sept. 1987.

[122] Y. Saab and V. Rao, "An evolution-based approach to partitioning ASIC circuits", pp. 767-770 in *Proc. 26th IEEE/ACM Design Automat. Conf.*, 1989.

[123] A. Saldanha, T. Villa, R. K. Brayton, and A. L. Sangiovanni-Vincentelli, "A framework for satisfying input and output encoding constraints", pp. 170-175 in *Proc. 28th IEEE/ACM Design Automat. Conf.*, 1991.

[124] M. Sarrafzadeh and D. T. Lee, "A new approach to topological via minimization", *IEEE Trans. Computer-Aided Design*, vol. 8, pp. 890-900, 1989.

[125] M. Sarrafzadeh and D. T. Lee, "Topological via minimization revisited", *IEEE Trans. Computer*, vol. 40, no. 11, pp. 1307-1312, Nov. 1991.

[126] G. Saucier, C. Duff and F. Poirot, "State assignment using a new embedding method based on an intersecting cube theory", pp. 321-326 in *Proc. 26th IEEE/ACM Design Automat. Conf.*, Las Vegas, June 1989.

[127] A. A. Schäffer and M. Yannakakis, "Simple local search problems that are hard to solve", *SIAM J. Comput.*, vol. 20, no. 1, pp. 56-87, Feb. 1991.

[128] A. Sebö, "Finding the t-join structure of graphs", *Math. Programming*, vol. 36, pp. 123-134, 1986.

[129] M. Servit, "Minimizing the number of feedthroughs in two-layer printed boards", *Digital Processes*, vol. 3, pp. 177-183, 1977.

[130] C.-J. Shi, *Constrained Via Minimization for VLSI and PCB Layouts*, M.A.Sc. Thesis, Department of Electrical and Computer Engineering, University of Waterloo, 1990.

[131] C.-J. Shi, "Modeling k-way splits of physical routings for constrained via minimization", pp. 4B.5.1-4B.5.8 in *Proc. Canadian Conf. on VLSI*, Aug. 1991.

[132] C.-J. Shi, "A signed hypergraph model of constrained via minimization", *Microelectronics J.*, vol. 23, no. 7, pp. 533-542, Nov. 1992. Also pp. 159-166 in *Proc. Second Great Lakes Symp. on VLSI*, Kalamazoo, MI, Feb. 1992.

[133] C.-J. Shi, "Constrained via minimization and signed hypergraph partitioning", *Algorithmic Aspects of VLSI Layouts*, eds. D. T. Lee and M. Sarrafzadeh, River Edge, NJ: World Scientific, 1993.

[134] C.-J. Shi, "Analysis, sensitivity, and macromodeling of the Elmore delay in linear networks for performance-driven VLSI design", *Int. J. of Electronics*, vol. 75, no. 3, pp. 467-484, Sep. 1993.

[135] C.-J. Shi and J. A. Brzozowski, "Efficient constrained encoding for VLSI sequential logic synthesis", pp. 266-271 in *Proc. First European Design Automation Conf.*, Hamburg, Germany, Sept. 1992.

[136] C.-J. Shi and J. A. Brzozowski, "An efficient algorithm for constrained encoding and its applications", *IEEE Trans. Computer-Aided Design*, pp. 1813-1826, vol. 13, no. 12, Dec. 1993; Also Technical Report, CS-92-20, Department of Computer Science, University of Waterloo, April 1992.

[137] C-J. Shi, A. Vannelli, and J. Vlach, "A hypergraph partitioning approach to the via minimization problem", pp. 2.7.1-2.7.8 in *Proc. Canadian Conf. on VLSI*, Oct. 1990.

[138] C.-J. Shi, A. Vannelli and J. Vlach, "An improvement on Karmarkar's algorithm for integer programming", *COAL Bulletin of Mathematical Programming Society*, vol. 21, pp. 23-28, 1992.

[139] C.-J. Shi, A. Vannelli, and J. Vlach, "Optimum layer assignment of two-layer routing with timing constraints", *IEEE Trans. Computer-Aided Design*, (to appear).

[140] C.-J. Shi and K. Zhang, "A robust approach to timing verification", pp. 56-59 in *Proc. IEEE Int. Conf. Computer-Aided Design*, 1987.

[141] W. K. Shih, S. Wu, and Y. S. Kuo, "Unifying maximum cut and minimum cut of a planar graph", *IEEE Trans. Compt.* vol. C-39, no. 5, pp. 694-697, May 1990.

[142] K. R. Stevens and W. M. vanCleemput, "Global via minimization in generalized routing environment", pp. 689-69 in *Proc. IEEE Int. Symp. Circuits Syst.*, 1979.

[143] C.-J. Tan, "State assignment for asynchronous sequential machines", *IEEE Trans. Computer*, vol. C-20, no. 4, April 1971, pp. 382-391.

[144] I. G. Tollis, "A new approach to wiring layouts", *IEEE Trans. Computer-Aided Design*, vol. CAD-10, no. 10, pp. 1392-1400, Oct. 1991.

[145] I. G. Tollis and A. V. Vaguine, "Improved techniques for wiring layouts in the square grid", pp. 1875-1878 in *Proc. IEEE Int. Symp. Circuits Syst.*, 1989.

[146] J. H. Tracey, "Internal state assignment for asynchronous sequential machines" *IEEE Trans. Electron. Comput.*, pp. 551-560, Aug. 1966.

[147] K. Truemper, "Alpha-balanced graphs and matrices and *GF(3)*-representability of matroids", *J. of Combinatorial Theory*, Series B 32, pp. 112-139, 1982.

[148] S. H. Unger, "A row assignment for delay-free realizations of flow tables without essential hazards", *IEEE Trans. Electron. Comput.*, vol. C-17, no. 2, pp. 145-158, Feb. 1968.

[149] S. H. Unger, *Asynchronous Sequential Switching Circuits*, New York: Wiley, 1969.

[150] A. Vannelli and S. W. Hadley, "A Gomory-Hu cut tree representation of a netlist partitioning problem", *IEEE Trans. Circuits Syst.*, vol. CAS-37, no. 9, pp. 1133-1139, Sept. 1990.

[151] D. Varma and E. A. Trachtenberg, "A fast algorithm for the optimal state assignment of large finite state machines", pp. 152-155 in *Proc. Int. Conf. Computer-Aided Design*, 1988.

[152] T. Villa and A. L. Sangiovanni-Vincentelli, "NOVA: State assignment of finite state machines for optimal two-level logic implementation", *IEEE Trans. Computer-Aided Design*, vol. CAD-9, no. 9, pp. 905-924, Sep. 1990.

[153] R.-S. Wei and A. L. Sangiovanni-Vincentelli, "ROTEUS: a logic verification system for combinational circuits", pp. 350-359 in *Proc. IEEE Int. Test Conf.*, 1986.

[154] Y. C. Wei and C. K. Cheng, "Towards efficient hierarchical designs by ratio cut partitioning", pp. 298-301 in *Proc. IEEE Int. Conf. Computer-Aided Design*, 1989.

[155] N. Weste and K. Eshraghian, *Principles of CMOS VLSI Design: A System Perspective*, Reading, MA: Addison-Wesley, 1985.

[156] S. Wu and U. Manber, "Path-matching problems", *Algorithmica*, vol. 8, pp. 89-101, 1992.

[157] X. M. Xiong and E. S. Kuh, "A unified approach to the via minimization problem", *IEEE Trans. Circuits Syst.*, vol. CAS-36, no. 2, pp. 190-204, Feb. 1989.

[158] S. Yang and M. J. Ciesielski, "Optimum and suboptimum algorithms for input encoding and its relationship to logic minimization", *IEEE Trans. Computer-Aided Design*, vol. CAD-10, no. 1, pp. 4-12, Jan. 1991.

[159] M. Yannakakis, "Node- and edge-deletion NP-complete problems, pp. 253-264 in *Proc. 10th Annual ACM Symposium on Theory of Computing*, San Diego, CA, Oct. 1979.

[160] M. Yannakakis, "On a class of totally unimodular matrices", *Mathematics of Operations Research*, vol. 10, no. 2, pp. 280-304, 1985.

[161] S. H. C. Yen, D. H.-C. Du, and S. Ghanta, "Efficient algorithms for extracting the K most critical paths in timing analysis", pp. 649-654 in *Proc. IEEE/ACM Design Automat. Conf.*, 1989.

[162] T. Yoshimura and E. S. Kuh, "Efficient algorithms for channel routing", *IEEE Trans. Computer-Aided Design*, vol. CAD-1, no. 1, pp. 25-35, Jan. 1982.

[163] T. Zaslavsky, "Characterizations of signed graphs", *J. of Graph Theory*, vol. 5, pp. 401-406, 1981.

[164] T. Zaslavsky, "Orientation embedding of signed graphs", *J. of Graph Theory*, vol. 16, no. 5, pp. 399-422, 1992.