VLSI DESIGN
2000, Vol. 00, No. 00, pp. 1–43
Reprints available directly from the publisher
Photocopying permitted by license only

Tutorial on VLSI Partitioning

SAO-JIE CHEN[a, †] and CHUNG-KUAN CHENG [b,*]

[a]*Dept. of Electrical Engineering, National Taiwan University, Taipei, Taiwan 10764;* [b]*Dept. of Computer Science and Engineering, University of California, San Diego, La Jolla, CA 92093-0114*

(Received 1 March 1999; In final form 10 February 2000)

The tutorial introduces the partitioning with applications to VLSI circuit designs. The problem formulations include two-way, multiway, and multi-level partitioning, partitioning with replication, and performance driven partitioning. We depict the models of multiple pin nets for the partitioning processes. To derive the optimum solutions, we describe the branch and bound method and the dynamic programming method for a special case of circuits. We also explain several heuristics including the group migration algorithms, network flow approaches, programming methods, Lagrange multiplier methods, and clustering methods. We conclude the tutorial with research directions.

Keywords: Partitioning, clustering, network flow, hierarchical partitioning, replication, performance driven partitioning

1. INTRODUCTION

Automatic partitioning [5, 61, 78, 72] is becoming an important topic with the advent of deep submicron technologies. An efficient and effective partitioning [12, 17, 19, 48, 69, 70, 81, 94, 105, 77] tool can drastically reduce the complexity of the design process and handle engineering change orders in a manageable scope. Moreover, the quality of the partitioning differentiates the final product in terms of production cost and system performance.

The size of VLSI designs has increased to systems of hundreds of millions of transistors. The complexity of the circuit has become so high that it is very difficult to design and simulate the whole system without decomposing it into sets of smaller subsystems. This divide and conquer strategy relies on partitioning to manipulate the whole system into hierarchical tree structure.

Partitioning is also needed to handle engineering change orders. For huge systems, design iterations require very fast turn around time. A hierarchical partitioning methodology can localize the modifications and reduce the complexity.

Furthermore, a good partitioning tool can decrease the production cost and improve the

[*] Corresponding author. Tel: (858)534-6184, Fax: (858)534-7029, e-mail: kuan@cs.ucsd.edu
[†] Tel: (8862)2363-5251 ext. 417, e-mail: csj@cc.ee.ntu.edu.tw

system performance. With the advance of fabrication technologies, the cost of a transistor drops while the cost of input/output pads remains fairly constant. Consequently, the size of the interface between partitions, *e.g*., between chips, determines a significant portion of the manufacturing expenses. And the quality of the partitioning has strong effect on production cost. Furthermore, in submicron designs, interconnection delays tend to dominate gate delays [8]; therefore system performance is greatly influenced by the partitions.

Partitioning has been applied to solve the various aspects of VLSI design problems [5, 36]:

- *Physical packaging* Partitioning decomposes the system in order to satisfy the physical packaging constraints. The partitioning conforms to a physical hierarchy ranging from cabinets, cases, boards, chips, to modular blocks.
- *Divide and conquer strategy* Partitioning is used to tackle the design complexity with a divide and conqure strategy [21]. This strategy is adopted to decompose the project between team members, to construct a logic hierarchy for logic synthesis, to transform the netlist into physical hierarchy for floorplanning, to allocate cells into regions for placement and RLC extraction, and manipulate hierarchies between logic and layout for simulation.
- *System emulation and rapid prototyping* One approach for system emulation and prototyping is to construct the hardware with field programmable gate arrays. Usually, the capacity of these field programmable gate arrays is smaller than current VLSI designs. Thus, these prototyping machines are composed of a hierarchical structure of field programmable gate arrays. A partitioning tool is needed to map the netlist into the hardware [110].
- *Hardware and software codesign* For hardware and software codesign, partitioning is used to decompose the designs into hardware and software.
- *Management of design reuse* For huge designs especially system-on-a-chip, we have to manage design reuse. Partitioning can identify clusters of the netlist and construct functional modules out of the clusters.

While partitioning is a tool required to manage huge systems in many fields such as efficient storage of large databases on disks, data mining, and *etc*., in this tutorial, we focus our efforts on partitioning with applications to VLSI circuit designs. In the next section, we describe the notations for the tutorial. In section three, the formulations of the partitioning problems are stated. Section four covers the models for multiple pin nets. Section five depicts the partitioning algorithms. The tutorial is concluded with research directions.

2. PRELIMINARIES

In this section, we establish notations used and formulate the partitioning problems addressed in our approaches. A circuit is represented by a hypergraph, $H(V, E)$, where the vertex set $V=\{v_i \mid i=1,2,\ldots,n\}$ denotes the set of modules and the hyperedge set $E=\{e_j \mid j=1,2,\ldots,m\}$ denotes the set of nets. Each net e_j is a subset of V with cardinality $|e_j| \geq 2$. The modules in e_j are called the *pins* of e_j.

The hypergraph representation for a circuit with 9 modules and 6 signal nets is shown in Figure 1, where nets e_1, e_3 and e_5 are two-pin nets, net e_6 is a three-pin net, and nets e_2 and e_4 are four-pin nets.

When the circuit has only two pin nets, we can simplify the representation to a graph $G(V, E)$. A net connecting modules v_i and v_j is represented by e_{ij} with a connectivity c_{ij}. We set $c_{ij}=0$ if there is no net connecting modules v_i and v_j. We shall show later that for certain formulations we replace multiple pin nets with models of two pin nets. The replacement is performed when the partitioning algorithm is devised for graph models.

(i) Module Size and Net Connectivity Each module v_i is attached with a size s_i in R^+, positive real numbers. We define $S(V_j)=\sum_{v_i\in V_j} s_i$ to be the size of a partition V_j. Each net e_i is attached with a

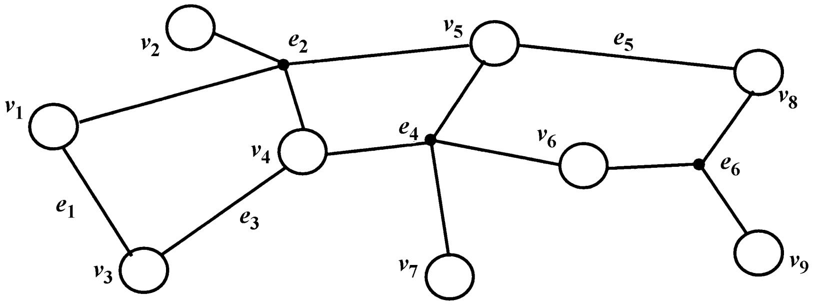


FIGURE 1 Hypergraph example.

connectivity c_i in R^+. By default, $c_i = 1$. For a bus of multiple signal lines, we can represent the bus with a net e_i of connectivity c_i equal to the number of lines. We can also assign higher weights for some important nets, this will enable us to keep the modules of these nets in the same partition.

In this tutorial, we will assume that circuits are represented as hypergraphs except when stated otherwise, hence, the terms *circuit*, *netlist*, and *hypergraph* are used interchangeably throughout the tuorial.

(ii) Partitions and Cuts The set of hyperedges connecting any two-way partition (V_1, V_2) of two disjoint vertex sets V_1 and V_2 is denoted by a cut $E(V_1, V_2) = \{e_j \in E \mid 0 < |e_j \cap V_1|$ and $0 < |e_j \cap V_2|\}$, *i.e.*, $e_j \in E(V_1, V_2)$ if there exist some pins of e_j in V_1 and some different pins of e_j in V_2. We define $C(V_1, V_2) = \sum_{e_i \in E(V_1,V_2)} c_i$ to be the cut count of the partition (V_1, V_2).

For a multiway partition $(V_1, V_2, \ldots, V_k)$ where $k > 2$, a cut $E(V_1, V_2, \ldots, V_k) = \{e_j \in E \mid \exists\, i$ s.t. $0 < |e_j \cap V_i| < |e_j|\}$. For each subset V_i, we denote its external cut set $E(V_i) = \{e_j \in E \mid 0 < |e_j \cap V_i| < |e_j|\}$. We denote its adjacent net set to be the nets with some pin contained in V_i, *i.e.*, $I(V_i) = \{e_i \mid |e_i \cap V_i| > 0\}$.

(iii) Replication Cuts and Directed Cuts For replication cuts and performance driven partitioning, the direction of the nets makes a difference in the process. We characterize the pins of each net into two types: source and sink. A directed net e_i is denoted by (a_i, b_i) where $a_i \subset V$ are the source pins of the net and $b_i \subset V$ are the sink pins of the net. We assume that $|a_i \cup b_i| \geq 2$, $|a_i| \geq 1$ and $|b_i| \geq 1$. Usually, each net has one source pin and multiple sink pins. However, some nets may have multiple sources which share the same interconnect line. Furthermore, one pin can be both a source pin and sink pin of the same net. Therefore, a_i and b_i may have a nonempty intersection.

For two disjoint vertex sets X and Y, we shall use $E(X \rightarrow Y)$ to denote the directed cut set from X to Y. Net set $E(X \rightarrow Y)$ contains all the nets $e_i = (a_i, b_i)$ such that X intersects the source pin set a_i and Y intersects the sink pin set b_i, *i.e.*, $E(X \rightarrow Y) = \{e_i \mid e_i = (a_i, b_i),\ a_i \cap X \neq \emptyset,\ b_i \cap Y \neq \emptyset\}$. We use the function $C(X \rightarrow Y)$ to denote the total cut count of the nets in $E(X \rightarrow Y)$, *i.e.*, $C(X \rightarrow Y) = \sum_{e_i \in E(X \rightarrow Y)} c_i$.

(iv) Performance Driven Partitioning In performance driven partitioning [106], modules are distinguished into two types: combinational elements and globally clocked registers. In illustration, we shall use circles to represent the combinational elements and rectangles to represent the registers in figures (Fig. 13). Each module v_i has an associated delay d_i.

A path of length k from a module v_i to a module v_j is a sequence $\langle v_{i_0}, v_{i_1}, \ldots, v_{i_k} \rangle$ of modules such that $v_i = v_{i_0}$, $v_j = v_{i_k}$ and for each $l \in \{1, 2, \ldots, k\}$, modules $v_{i_{l-1}}$ and v_{i_l} are a souce pin and a sink pin of a net in E, respectively.

(v) Clustering Given a hypergraph $H(V, E)$, highly connected modules in V can be grouped together to form some single supermodules called *clusters*. After this process, a *clustering* $\Gamma = \{V_1, V_2, \ldots, V_k\}$ of the *original* hypergraph H is obtained and a *contracted* (*i.e.*, coarser) hypergraph $H_\Gamma(V_\Gamma, E_\Gamma)$ is induced, where $V_\Gamma = \{v_1^\Gamma, v_2^\Gamma, \ldots, v_k^\Gamma\}$. For every $e_j \in E$, the contracted net $e_j^\Gamma \in E_\Gamma$ if $|e_j^\Gamma| \geq 2$, where $e_j^\Gamma = \{v_i^\Gamma | e_j \cap V_i \neq \emptyset\}$, that is, e_j^Γ spans the set of clusters containing modules of e_j. A contracted hypergraph, of course, can be used to induce another coarser contracted hypergraph based on the same clustering process. On the other hand, a contracted hypergraph $H_\Gamma(V_\Gamma, E_\Gamma)$ can be unclustered to return to a finer hypergraph $H(V, E)$.

3. PROBLEM FORMULATIONS

In this section, we describe different formulations of the partitioning problems addressed in this tutorial. We will cover two-way partitioning, multiway partitioning, multiple level partitioning, partitioning with replication, and performance driven partitioning.

3.1. Two-way Partitioning or Bipartitioning

We consider several possible variations on the size constraints and cost functions in the formulation. Additionally, in certain formulations, we fix two modules v_s and v_t to be on the opposite sides of the cut as two seeds.

3.1.1. Min-cut Separating Two Modules v_s and v_t

Given a hypergraph, we fix two modules denoted as v_s and v_t at two sides. A min-cut is a partition (V_1, V_2), $v_s \in V_1$ and $v_t \in V_2$ such that the cut count $C(V_1, V_2)$ is minimized, *i.e.*,

$$\min_{v_s \in V_1, v_t \in V_2} C(V_1, V_2) \tag{1}$$

where V_1 and V_2 are disjoint and the union of the two sets is equal to V.

This partitioning is strongly related to a linear placement problem. In a linear placement, we have $|V|$ equally spaced slots on a striaght line (Fig. 2). Modules v_s and v_t are fixed at the two extreme ends, *i.e.*, v_s on the first slot (left end) and v_t on the last slot (right end). The goal is to assign all modules to distinct slots to minimize the total wire length. Let us use x_i to denote the coordinate of module v_i after it is assigned to the slot. The length of a net e_i can be expressed as the difference of the maximum coordinate and the minimum coordinate of the modules in the net, *i.e.*, $\max_{v_j \in e_i} x_j - \min_{v_k \in e_i} x_k$. The total wire length can be expressed as follows.

$$\sum_{e_i \in E} (\max_{v_j \in e_i} x_j - \min_{v_j \in e_i} x_j) \tag{2}$$

The relation between partitioning and placement can be derived under the assumption that all nets are two pin nets [50].

THEOREM 3.1 *Given a graph $G(V, E)$ with modules v_s and v_t in V, let (V_1, V_2) be a min-cut partition separating modules v_s and v_t. Let v_s and v_t be the two modules locating at the two extreme ends of a linear placement. Then, there exists an optimal linear placement solution such that all modules in V_2 are on the slots right of all modules in V_1 (Fig. 2).*

Thus, we can use the min-cut to partition a linear

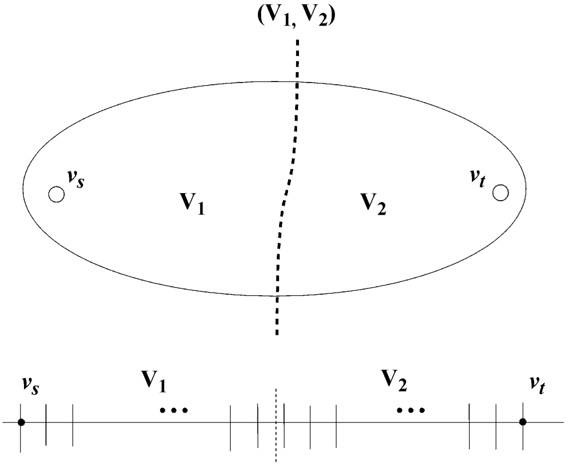


FIGURE 2 Suppose partition (V_1, V_2) is a min-cut separating modules v_s and v_t. There exists an optimal linear placement that modules in V_2 are at the right side of modules in V_1.

placement into two smaller problems and still maintain optimality. Conceptually, we can conceive that modules in V_1 or V_2 have stronger internal connection within the set than its mutual connection to the other set. Thus, if the span of modules in V_1 and in V_2 are mixed in a linear placement, we can slide all modules in V_1 to the left and all modules in V_2 to the right to reduce the total wire length. In fact, this is the procedure to prove the theorem.

The min-cut with no size constraints can be found in polynomial time using classical maximum flow techniques [1]. However, it may happen that the optimal solution separates only v_s or v_t from the rest of the modules, *i.e.*, $V_1 = \{v_s\}$ or $V_2 = \{v_t\}$. This result is very likely to happen because most VLSI basic modules have very small degrees of connecting nets (*e.g.*, the degree of a 3-input NAND gate = 4).

3.1.2. Minimum Cost Ratio Cut

The cost ratio cut formulation supplies a partition different from the min-cut that separates two fixed modules. Thus, if the min-cut cannot provide any nontrivial solution, we may adopt the cost ratio cut to perform another trial.

In cost ratio cut, we fix two modules v_s and v_t at two different sides. Our objective is to find a vertex set A to minimize a cost ratio function:

$$\frac{C(A, V - A - \{v_s\}) - C(A, \{v_s\})}{S(A)} \tag{3}$$

where vertex set A does not contain v_s and v_t. Vertex set A is non-empty, *i.e.*, $S(A) > 0$.

Cost ratio cut is also strongly related to a linear placement. Assuming that all nets are two pin nets, we can derive the following theorem [22]:

THEOREM 3.2 *Given a graph $G(V, E)$ with modules v_s and v_t in V, let (V_1, V_2) be an optimal cost ratio cut partition. There exists an optimal linear placement solution such that all modules in A are on the slots left of all modules in $V - A - \{v_s\}$.*

Conceptually, we can conceive that $C(A, V - A - \{v_s\})$ is the force to pull A to the right and $C(A, \{v_s\})$ is the force to push A to the left. The denominator $S(A)$ is the inertia of the set A. A set A with the minimum cost ratio moves with the fastest acceleration toward left end of the slots

Example In Figure 3, the circuit contains six modules. The optimum cost ratio cut solution has $A = \{v_1, v_2, v_3\}$ The cost ratio value is

$$\frac{C(A, V - A - \{v_s\}) - C(A, \{v_s\})}{S(A)} = \frac{4-3}{3} = \frac{1}{3}. \tag{4}$$

The cost ratio value of any other choice of set A is larger than expression 4.

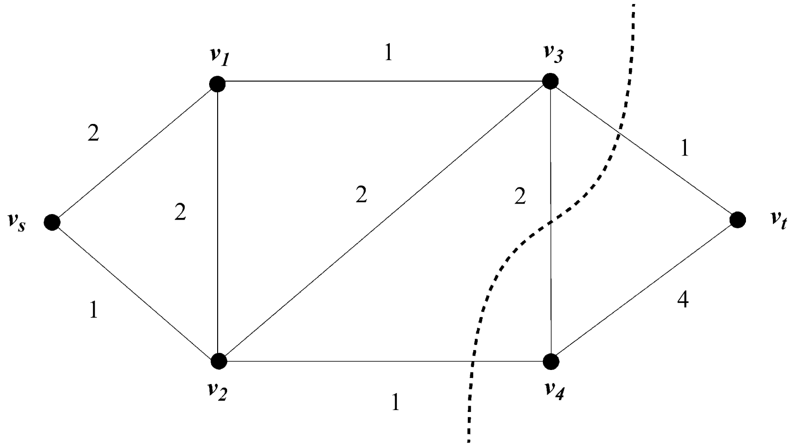


FIGURE 3 A six module circuit to illustrate the cost ratio cut.

The cost ratio cut solution can be found in polynomial time for a special case of serial parallel graphs [22]. We are unaware of algorithms for general cases. Note that, the solution may have $V - A - \{v_s\}$ equal to set $\{v_t\}$. In such case, the partitioning result is not useful for decomposing the circuit.

3.1.3. Min-cut with Size Constraints

For min-cut with size constraints, we have lower and upper bounds on the partition size S_l and S_u, where $0 < S_l \leq S_u < S(V)$ and $S_l + S_u = S(V)$. The bipartitioning problem is to divide vertex set V into two nonempty partitions V_1, V_2, where $V_1 \cap V_2 = \emptyset$ and $V_1 \cup V_2 = V$, with the objective of minimizing cut count $C(V_1, V_2)$ and subject to the following size constraints:

$$S_l \leq S(V_b) \leq S_u \quad \text{for } b = 1, 2 \tag{5}$$

The min-cut problem with size constraints is NP complete [43]. However, because of the importance of the problem in many applications, many heuristic algorithms have been developed.

Random Partitioning We use a random partition estimation of min-cut with size constraints to demonstrate that the quality variation of partitioning results can be significant. Let us simplify the case by assigning the modules with uniform size, *i.e.*, $s_i = 1$ for all v_i in V, and the nets with uniform connectivity, *i.e.*, $c_i = 1$ for all e_i in E.

Let us assume that the modules are partitioned into two sets V_1, V_2 with equal sizes: $S(V_1) = S(V_2)$. The partition is performed with an independent random process [10] so that each module has a 50% chance to go to either side. For a net e_i of two pins, we can derive that net e_i belongs to the cut set $E(V_1, V_2)$ with a 0.5 probability (Fig. 4). Similarly, we can derive that for a net e_i of k pins ($k > 2$), the probability that net e_i belongs to cut set $E(V_1, V_2)$ is $(2^k - 2)/2^k$. This probability is larger than 0.5 and approaches one as k increases. In other words, the expected cut count $C(V_1, V_2)$ is equal to or larger than half the number of nets. For example, a circuit of one million modules usually has an asymptotic number of nets, *i.e.*, $|E| = O(|V|) = 1{,}000{,}000$. The expected cut count would be $C(V_1, V_2) \geq 500{,}000$. This number is much worse than the results we can achieve. In practice, the cut counts on circuits of a million of modules are usually no more than several thousands [34, 36]. In other words, the probability that a net belongs to a cut set is small, below one percent for a circuit of one million gates.

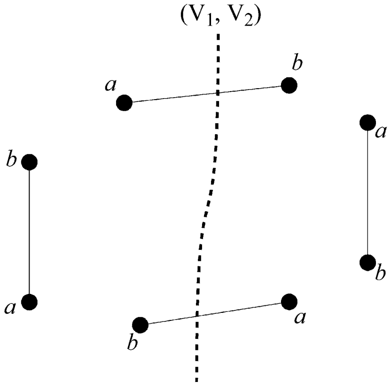


FIGURE 4 Four possible configurations of net $e_i = \{a, b\}$ in a random placement.

Suppose the two bounds of partitioned sizes are not equal, $S_l \neq S_u$. Using the proposed random graph model, the expected cut count $C(V_1, V_2)$ is proportional to the product of two sizes, *i.e.*, $S(V_1) \times S(V_2)$. Consequently, the expected cut count is smallest if the size of one partition approaches the upper bound $S(V_i) = S_u$ and the size of another partition approaches the lower bound $S(V_j) = S_l$. In practice, we do observe this behavior. One partition is fully loaded to its maximum capacity, while another partition is under utilized with a large capacity left unused. This phenomena is not desirable for certain applications.

3.1.4. Ratio Cut

Ratio cut formulation integrates the cut count and a partition size balance criterion into a single objective function [87, 109]. Given a partition (V_1, V_2) where V_1 and V_2 are disjoint and $V_1 \cup V_2 = V$, the objective funtion is defined as

$$\frac{C(V_1, V_2)}{S(V_1) \times S(V_2)} \tag{6}$$

The numerator of the objective function minimizes the cut count while the denominator avoids uneven partition sizes. Like many other partitioning problems, finding the ratio cut in a general network belongs to the class of *NP*-complete problems [87].

Example Figure 5 shows a seven module example. The modules are of unit size and the nets are of unit connectivity. Partition (V_1, V_2) has a cost $C(V_1, V_2)/(S(V_1) \times S(V_2)) = 2/(4 \times 3) = (1/6)$. Any other partition corresponds to a much larger cost.

The Clustering Property of the Ratio Cut The clustering property of the ratio cut can be illustrated by a random graph model. Let us assume that the circuit is a **uniformly distributed random graph**. with uniform module sizes, *i.e.*, $s_i = 1$. We construct the nets connecting each pair of modules with identical independent probability f. Consider a cut which partitions the circuit into two subsets V_1 and V_2 with comparable sizes $\alpha \times |V|$ and $(1-\alpha) \times |V|$ respectively, where $\alpha < 1$.

The expected cut count equals the probability f multiplied by the number of possible nets between V_1 and V_2.

$$\begin{aligned}\text{Expec}(C(V_1, V_2)) &= f \times |V_1| \times |V_2| \\ &= \alpha(1-\alpha)|V|^2 \times f. \end{aligned} \tag{7}$$

On the other hand, if another cut separates only one module v_s from the rest of the modules, the expected cut count is

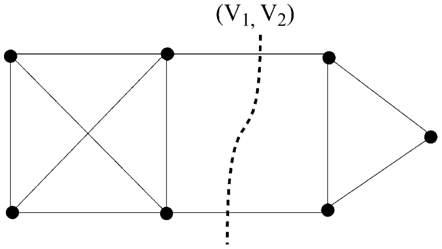


FIGURE 5 An example of seven modules, where partition (V_1, V_2) is a minimum ratio cut.

$$\text{Expec}(C(\{v_s\}, V - \{v_s\})) = (|V| - 1) \times f \tag{8}$$

As $|V|$ approaches infinity, the value of Eq. (7) becomes much larger than 8.

This derivation provides another explanation why the min-cut separating two fixed modules tends to generate very uneven sized subsets. The very uneven sized subsets naturally give the lowest cut value. Therefore, the ratio value $C(V_1, V_2)/(S(V_1) \times S(V_2))$ is proposed to alleviate the hidden size effect. As a consequence, the expected value of this ratio is a constant with respect to different cuts:

$$\text{Expec}\left(\frac{C(V_1, V_2)}{S(V_1) \times S(V_2)}\right) = \frac{f \times |V_1| \times |V_2|}{|V_1| \times |V_2|} = f \tag{9}$$

Thus, if the nets of the graph are uniformly distributed, all cuts have the same ratio value. In other words, the choice of the cuts and the partition sizes does not make difference in such a uniformly distributed random graph. In a general circuit different cuts generate different ratios. Cuts that go through weakly connected groups correspond to smaller ratio values. The minimum of all cuts according to their corresponding ratios defines the sparsest cut since this cut deviates the most from the expectation on a uniformly distributed graph.

3.2. Multi-way Partitioning

For multi-way partitioning, we discuss a *k*-way partitioning with fixed size constraints and a cluster ratio cut. These two problems are the extensions of the min-cut with fixed size constraints and the ratio cut from two-way to multi-way partitioning, respectively.

3.2.1. K-way Partitioning

For multi-way partitioning, we separate vertex set V into k disjoint subsets where $k > 2$, *i.e.*, $(V_1, V_2, \ldots, V_k)$. There is an upper bound S_u and a lower bound S_l on the size of each subset V_i, *i.e.*, $S_l \leq S(V_i) \leq S_u$.

There are different ways to formulate the cut cost because of the different criteria used to count the cost of multiple pin nets. In the following we list a few possible objective functions.

(i) Minimize the cut count,

$$C(V_1, V_2, \ldots, V_k) = \sum_{e_i \in E(V_1, V_2, \ldots, V_k)} c_i \tag{10}$$

(ii) Minimize the sum of cut counts of all vertex sets. Let us denote the cut count of vertex set V_i to be $C(V_i) = \sum_{e_i \in E(V_i)} c_i$. The sum of cut counts of all subsets can be expressed as

$$\sum_{i=1}^{k} C(V_i) = \sum_{i=1}^{k} \sum_{e_j \in E(V_i)} c_j \tag{11}$$

Thus, the cost of a net connecting three subsets is more expensive than the same net connecting two subsets.

(iii) Minimize the maximum cut count of all subsets, *i.e.*,

$$\max_{1 \le i \le k} C(V_i) \tag{12}$$

3.2.2. Cluster Ratio Cut

Cluster ratio cut is an extension of ratio cut from two-way partition to multiway partition. There is no bound on the size of each subset. Furthermore, the number of partitions, k, is not fixed, and instead is part of the objective function.

$$R_C = \min_{k>1} \frac{C(V_1, V_2, \ldots, V_k)}{\sum_{1 \le i \le k-1} \sum_{j \ge i} S(V_i) \times S(V_j)} \tag{13}$$

Note that we can rewrite the denominator to reduce complexity of the derivation.

$$R_C = \min_{k>1} \frac{C(V_1, V_2, \ldots, V_k)}{(1/2) \sum_{1 \le i \le k} S(V_i) \times [S(V) - S(V_i)]} \tag{14}$$

If the number of partitions is one, the denominator becomes zero. Thus, k is restricted to be larger than one.

Example Figure 6 shows a fifteen module circuit. The modules are of unit size and the nets are of unit connectivity. The square dot in the figure represents a hypernet. The partition shown by the dashed line is a minimum cluster ratio cut. The cost of the cut is

$$\frac{C(V_1,V_2,\ldots,V_4)}{(1/2)\sum_{1 \le i \le 4} S(V_i) \times [S(V) - S(V_i)]} = \frac{4}{(1/2)[4(15-4)+3(15-3)+4(15-4)+4(15-4)]} = \frac{1}{21} \tag{15}$$

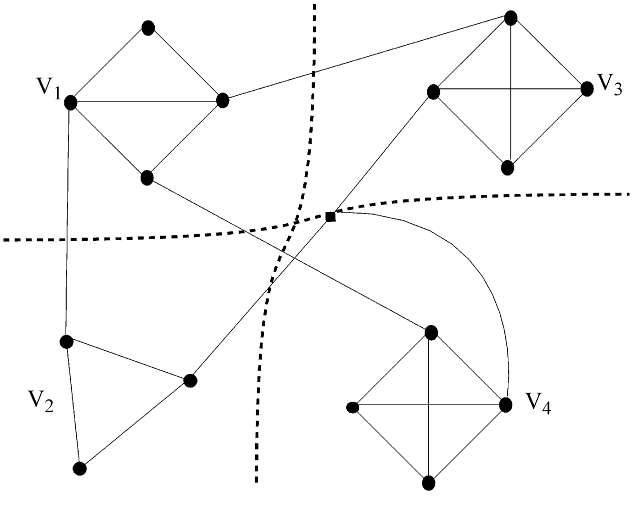


FIGURE 6 A fifteen module example to demonstrate cluster ratio cut.

The physical intuition of cluster ratio can be explained using a random graph model [10]. Let G be a uniformly distributed random graph. We construct the nets connecting each pair of modules with identical independent probability f. Since the nets are uniformly distributed, the probability of finding a subgraph which is significantly denser than the rest of the graph is very small, meaning that there is no distinct cluster structure in G.

Consider a cut $E(V_1, V_2, \dots, V_k)$, the expected value of $C(V_1, V_2, \dots, V_k)$ equals

$$\text{Expec}(C(V_1, V_2, \dots, V_k)) = f \times \sum_{i=j+1}^{k} \sum_{j=1}^{k-1} |V_i| \times |V_j| \tag{16}$$

and the expected value of cluster ratio equals

$$\begin{aligned}\text{Expec}(R_C) &= \text{Expec}\left(\frac{C(V_1, V_2, \dots, V_k)}{\sum_{i=j+1}^{k} \sum_{j=1}^{k-1} |V_i| \times |V_j|}\right) \\ &= \frac{f \times \sum_{i=j+1}^{k} \sum_{j=1}^{k-1} |V_i| \times |V_j|}{\sum_{i=j+1}^{k} \sum_{j=1}^{k-1} |V_i| \times |V_j|} = f\end{aligned} \tag{17}$$

Since f is a constant, all cuts have the same expected cluster ratio value. Therefore, if we use cluster ratio as the metric, all cuts would be equally favored, which is consistent with the fact that G has no distinct clusters. However, in a general circuit, different cuts generate different ratio values. Cuts that go through *weakly* connected groups correspond to *smaller* ratio values. The minimum of all cuts according to their cluster ratio values defines the cluster structure of the circuit since this cut deviates the most from the cuts of a uniformly distributed graph.

3.3. Multi-level Partitioning

In multi-level partitioning [4, 23, 47, 58, 67, 68, 109, 110], the final result is represented by a tree structure. All the modules are assigned to the leaves of the tree. The tree is directed from the root toward the leaves. The level of the nodes is defined to be the maximum number of nodes to traverse to reach the leaves. Thus, the leaves are ranked level zero. Each node is one level above the maximum level of its children. When the level of the root is only one, the problem is degenerated to two-way or multiway partitioning.

Each net e_i spans a set of leaves. Given a set of leaves, there is a unique lowest common ancestor. The level of the lowest ancestor is defined to be the level $l(e_i)$ of the net.

The cost of a net e_i is defined to be the multiplication of its connectivity c_i and the weight $w(l(e_i))$ of level $l(e_i)$ for net e_i to communicate, *i.e.*, $c_i \times w(l(e_i))$. The cost of the multi-level partition is the sum of the cost of all nets, *i.e.*, $\sum_{e_i \in E} c_i w(l(e_i))$.

3.3.1. J-level K-way Partitioning

When the root of the partitioning tree is level j and the number of branches of each node is no more than k, we say it a j-level k-way partition. We can set different communication weights for each level. Usually, the function is monotone, *i.e.*, $w(l)$ is larger when level l increases. The vertex set V_i of each leaf i has its size bounded by $S_l \le S(V_i) \le S_u$.

For electronic packaging, the tree is bounded by the number of external connections. We call a leaf is covered by a node if there is a directed path from the node to the leaf in the tree representation. For each node n_i, we define T_i to be the union of the modules in the leaves covered by node *ni*. Let $E(T_i)$ be the external nets of T_i, *i.e.*, $E(T_i) = \{e_i \,|\, 0 < |e_i \cap T_i| < |e_i|\}$. The cut count of each node should not exceed the capacity of the external connection of the packaging, *i.e.*,

$$C(T_i) = \sum_{e_j \in E(T_i)} c_j \le \text{Cap}(l(n_i)) \tag{18}$$

where $\text{Cap}(l(n_i))$ is the capacity of the external connection of level $l(n_i)$.

Example Figure 7 shows an example of a 3-level 5-way partitioning structure. The leaves are at level 0 and the root is at level 3. Each node has at most five children. Net $e_i = \{v_1, v_2, v_3\}$ is covered by node n_a at level $l(n_a) = 2$.

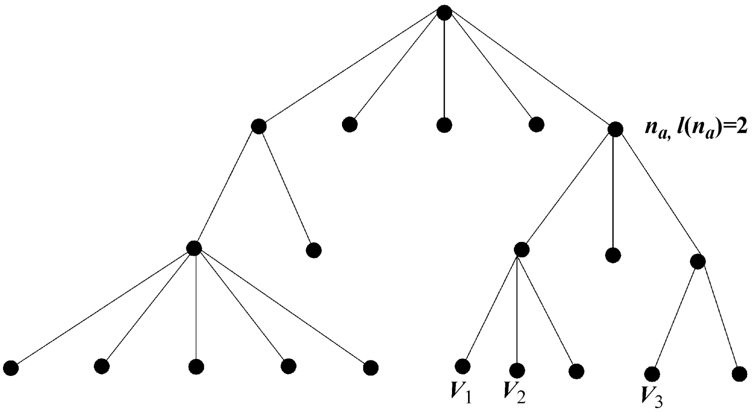


FIGURE 7 An example of a 3-level 5 way partitioning tree structure.

3.3.2. Generic Binary Tree

A generic binary tree structure [110] is proposed to simplify the multi-level partitioning. There is only one constant S_u to set in the binary tree. Thus, it is much easier to make a fair comparison between different algorithms.

In a generic binary tree, each internal node has exactly two children. The weight of each level is defined to be $w(l) = 2^l$. Thus, we have the objective function

$$\min \sum_{e_i \in E} c_i 2^{l(e_i)}$$

subject to the constraint on the capacity of the leaves, *i.e.*, $S(V_i) \leq S_u$ where V_i is the vertex set of leaf i. The level of the root is adjusted according to the minimization of the objective function.

Example Figure 8 illustrates a generic binary tree for partitioning. In this figure, the root is at level three. Each node has at most two children.

3.4. Replication Cut

In the replication cut problem, a subset of the circuit may be replicated to reduce the cut count of a partition [54, 64, 82]. In this section, we use a two-way partition to illusturate the problem. We fix two modules v_s and v_t at two sides of the cut. We use three vertex sets to represent the partition, V_1, V_2, and R, where V_1, V_2, and R are disjoint and $V_1 \cup V_2 \cup R = V$, $v_s \in V_1$, $v_t \in V_2$. Subsets V_1 and V_2 are separated by the cut and subset R is to be replicated at both sides (Fig. 9).

Each copy of R needs to collect a complete set of input signals in order to compute the function

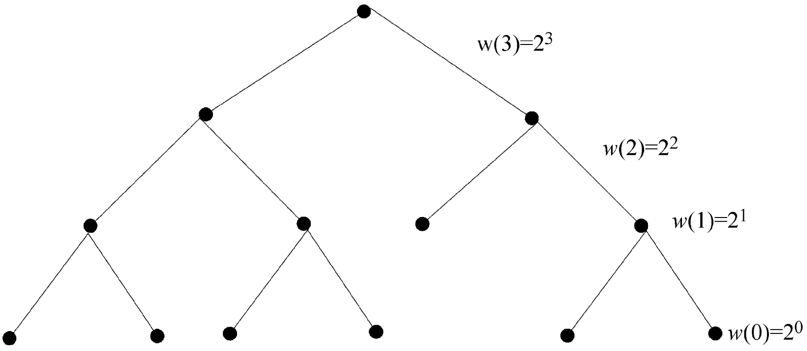
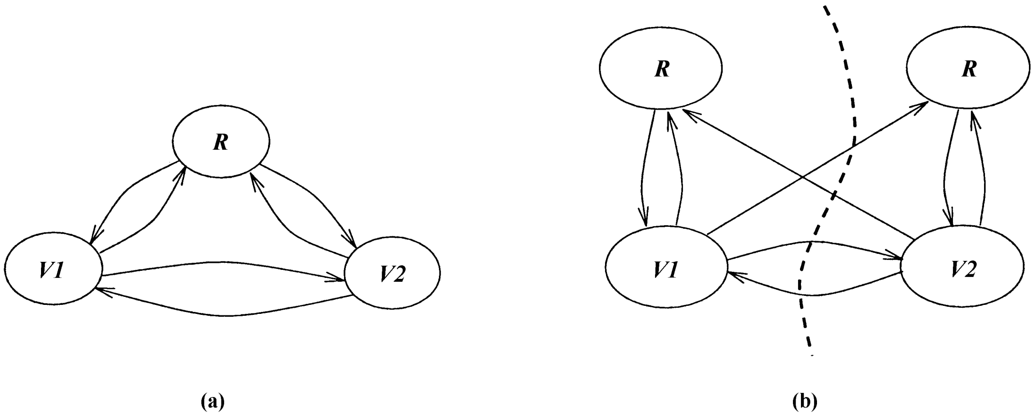


FIGURE 8 An example of a generic binary tree.



(a)

(b)

FIGURE 9 Replication cut problem: (a) the three sets of nodes V_1, R and V_2; (b) the duplicated circuit with R being replicated.

properly. Thus, the nets from V_1 to R and from V_2 to R are duplicated. However, the output signals of R can be obtained from either copy of R. For example, nets from the right side R to V_1 in Figure 9(b) are not duplicated because V_1 gets inputs from the left side R. For the same reason, we do not replicate the nets from the left side R to V_2. Given two disjoint sets V_1 and V_2, let a *replication cut* $R(V_1, V_2)$ denote the cut set of a partitioning with $R = V - V_1 - V_2$ being duplicated. From Figure 9(b), we can see that $R(V_1, V_2)$ is the union of four directed cuts, that is,

$$\begin{aligned} R(V_1, V_2) = {} & E(V_1 \to V_2) \cup E(V_2 \to V_1) \\ & \cup E(V_1 \to R) \cup E(V_2 \to R). \end{aligned}$$

Let S_l and S_u denote the size limits on the two partitioned subsets. We state the **Replication Cut Problem** as follows:

Given a directed circuit G, *we want to find a replication cut* $R(V_1, V_2)$ *with an objective*

$$\min C_R(V_1, V_2) = \sum_{e_i \in R(V_1, V_2)} c_i \qquad (19)$$

subject to the size constraints

$$S_l \le S(V_1 \cup R) \le S_u \textit{ and } S_l \le S(V_2 \cup R) \le S_u,$$

and the feasible condition

$$V_1 \cap V_2 = \emptyset, \quad R = V - V_1 - V_2.$$

Interpretation of the Replication Cut Suppose we rewrite the replication cut in the format:

$$\begin{aligned} R(V_1, V_2) &= E(V_1 \to R) \cup E(V_1 \to V_2) \\ &\quad \cup E(V_2 \to V_1) \cup E(V_2 \to R) \\ &= E(V_1 \to \bar{V}_1) \cup E(V_2 \to \bar{V}_2) \end{aligned}$$

where $\bar{V}_1$ and $\bar{V}_2$ denote the complementary sets of V_1 and V_2, *i.e.*, $\bar{V}_1 = V - V_1$ and $\bar{V}_2 = V - V_2$. The cut set becomes the union of $E(V_1 \to \bar{V}_1)$ and $E(V_2 \to \bar{V}_2)$. We can interpret the cut set of the replication cut $R(V_1, V_2)$ as two directed cuts on the original circuit G as shown in Figure 10.

3.5. Performance Driven Partitioning

The goal of performance driven partitioning is to generate a partition that satisfies some timing constraints. Due to the physical geometric distance and interface technology limitations, inter-partition delay contributes the dominant portion of signal propagation delay. Consequently, instead of minimizing the number of the crossing nets as the only objective during partitioning, we should take into account the interpartition delay to satisfy the timing constraints.

Clock period is a major measurement for circuit performance. It is determined by the longest signal propagation delay between registers. Each cross-

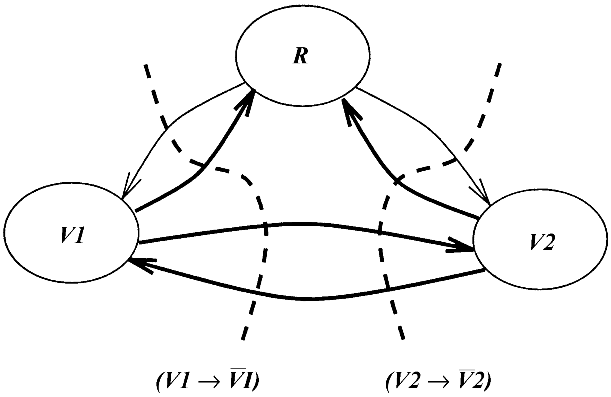


FIGURE 10 An interpretation of the replication cut, $R(V_1, V_2) = E(V_1 \to \bar{V}_1) \cup E(V_2 \to \bar{V}_2)$.

ing net is associated with an interpartition delay δ determined by VLSI technologies. Given a path p from one register to another register with no interleaving registers, let d_p be the sum of combinational block delays and $\widehat{d_p}$ be the sum of interpartition delays along path p. The longest delay $d_p + \widehat{d_p}$ among all paths p should be smaller than the clock period T, *i.e.*:

$$\max_p d_p + \widehat{d_p} \leq T. \tag{20}$$

Now we state the **performance-driven partitioning problem** as follows:
Given hypergraph $H(V, E)$, *clock period* T, *two bounds of sizes* S_l *and* S_u, *and interpartition delay* δ, *find a partition* (V_1, V_2) *with the minimum cut count, subject to* $S_l \leq S(V_1) \leq S_u$, $S_l \leq S(V_2) \leq S_u$, *and* $\max_p d_p + \widehat{d_p} \leq T$.

Example In Figure 11, path p starts at register v_i and ends at register v_j. The path crosses between the partition (V_1, V_2) three times. Thus, the interpartition delay $\widehat{d_p} = 3\delta$.

Replication can improve the performance of the partitioned results [83]. In Figure 12(a), vertex set R locates at the side of V_2. Path p crosses between the partition $(V_1, R \cup V_2)$ three times. By replicat-

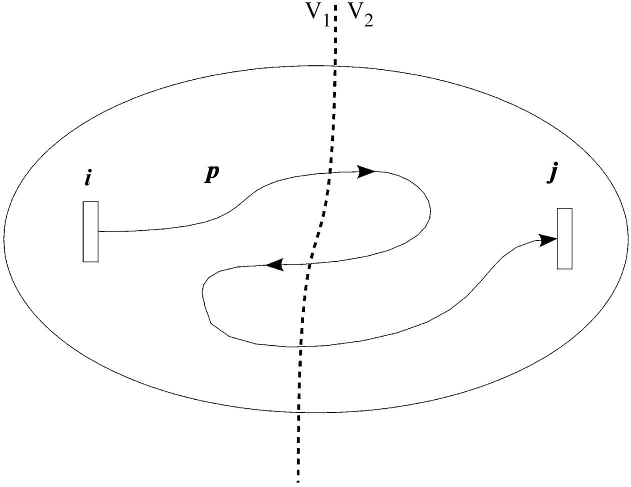


FIGURE 11 An illustration of performance driven partitioning.

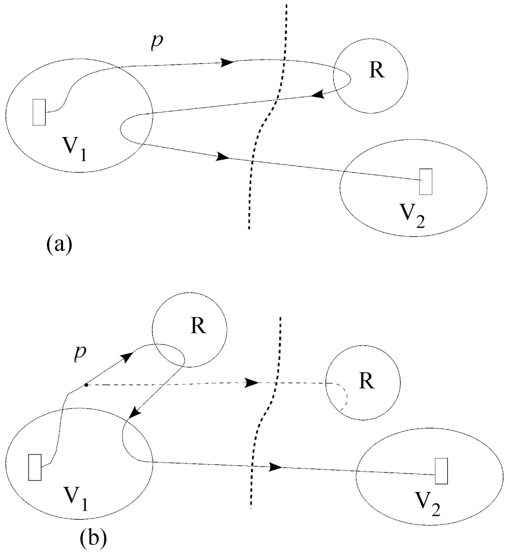


FIGURE 12 Illustration of replication and its effect on partitioning. The figure shows path p (a) before and (b) after vertex set R is replicated.

ing vertex set R (Fig. 12(b)), path p needs to cross the partition only once.

3.5.1. *Retiming*

Retiming shifts the locations of the registers to improve the system performance [76]. It is an effective approach to reduce the clock period. Moreover, the process also reduces the primary input to primary output latency which is another important measurement for circuit performance.

As in [85], we assume that the combinational blocks are fine-grained. A module is called fine-grained, if it can be split into several smaller modules. Alternatively, if a module cannot be split, it is called coarse-grained. The interpartition delay δ on crossing nets is inherently coarse-grained and cannot be split.

Given a path p, we use r_p to denote the number of registers on the path. Let $W(i,j)$ denote the minimum r_p among all possible paths p from i to j, *i.e.*,

$$W(i,j) = \min\ \{r_p | p \in P_{ij}\},$$

where P_{ij} is the set of all paths from module v_i to v_j. We define a path p from v_i to v_j as a *W-critical path* if r_p equals $W(i,j)$; W-critical path p is also called an *IO-W-critical* path if modules v_i and v_j are the primary input and output, respectively.

(i) Iteration Bound While retiming can reduce the clock period of a circuit, there is a lower bound imposed by the feedback loops in the hypergraph [92]. Given a loop l, let d_l, $\widehat{d_l}$ and r_l be the sum of combinational block delays, the sum of interpartition delays, and the number of registers in loop l, respectively. The delay-to-register ratio of a loop l is equal to $(d_l + \widehat{d_l})/r_l$. The *iteration bound* is defined as the maximum delay-to-register ratio, *i.e.*:

$$J(V_1, V_2) = \max\left\{\frac{d_l + \widehat{d_l}}{r_l} | l \in L\right\}, \qquad (21)$$

where L is the set of all loops. Note that the iteration bound of a given circuit yields a lower bound on the achieved clock period by retiming.

(ii) Latency Bound Let p denote the *IO-W*-critical path with maximum path delay among all *IO-W*-critical paths from v_i to v_j. Since the number of registers in path p is equal to $W(i,j)$, the *IO* latency (*i.e.* $(W(i,j) - 1) \times T$) between v_i and v_j is not less than $d_p + \widehat{d_p}$, where T denotes the clock period, and d_p and $\widehat{d_p}$ are the sum of combinational block delays and the sum of interpartition delays on path p, respectively. Thus, we define *latency bound M* as follows [85, 86]:

$$M(V_1, V_2) = \max\{d_p + \widehat{d_p} | p \in P_{IOW}\}, \qquad (22)$$

where P_{IOW} is the set of all *IO-W*-critical paths. Latency bound also imposes a lower bound on the system latency achieved by using retiming. An all-pair shortest-path algorithm can be used to calculate the latency bound.

We have two reasons to use the iteration and latency bounds. (i) It is faster to calculate these bounds. (ii) The iteration and latency bounds stand for the lower bounds of the clock period and system latency achieved by adopting retiming, respectively. The partition with lower iteration and latency bounds can achieve better clock period and system latency by using retiming. Therefore, we want to generate a partition with small iteration and latency bounds.

Statement of the Problem Now we state the performance-driven partitioning problem as follows:
Given hypergraph $H(V, E)$, two numbers $\tilde{J}$ and $\tilde{M}$, bounds of sizes S_l and S_u, and interpartition delay δ, find a partition (V_1, V_2) with the minimum number of cut count, subject to $S_l \leq S(V_1) \leq S_u$, $S_l \leq S(V_2) \leq S_u$, $J(V_1, V_2) \leq \tilde{J}$, and $M(V_1, V_2) \leq \tilde{M}$.

Example Figure 13 illustrates the effect of replication on the iteration bound. Let us assume that the interpartition delay is $\delta = 4$. Before replication, the iteration bound is dominated by loop l_1. The bound is equal to

$$\frac{d_{l_1} + \widehat{d_{l_1}}}{r_{l_1}} = \frac{8 + 2 \times 4}{4} = 4. \tag{23}$$

After replication [85], the bound contributed by loop l_1 is equal to

$$\frac{d_{l_1} + \widehat{d_{l_1}}}{r_{l_1}} = \frac{8}{4} = 2. \tag{24}$$

The iteration bound now is dominated by the union of loops l_1 and l_2,

$$\frac{d_{l_1+l_2} + \widehat{d_{l_1+l_2}}}{r_{l_1+l_2}} = \frac{18 + 2 \times 4}{8} = 3.25, \tag{25}$$

which is smaller than the iteration bound before replication.

3.6. Clustering

Clustering [6] is similar to multiway partitioning in that the process groups modules into k subsets. However, for clustering the number of subsets is usually much greater than for a typical multiway partitioning problem, *e.g.*, $k \geq 10$.

Often, a clustering process is used as part of a divide and conquer approach. Thus, it is important to choose an objective function that fits the target application. If the goal is to reduce problem complexity, we set the objective function to be:

$$\min \sum_{i=1}^{k} \frac{C(V_i)}{C_I(V_i)}, \tag{26}$$

where V_i's are disjoint vertex sets and their union is equal to V. Function $C(V_i)$ is the external cut count of cluster V_i and $C_I(V_i)$ is the count of nets connecting vertex set V_i, *i.e.*, $\sum_{e_i \in I(V_i)} c_i$.

For performance driven clustering, the objective function is to minimize the number of cuts between registers.

4. MULTIPLE PIN NET MODELS

The handling of multiple pin nets strongly depends on the partitioning approach [102]. A proper model is needed to reflect the correct cut count and improve the efficiency. In this section, we first introduce a shift model which is used for iterations of

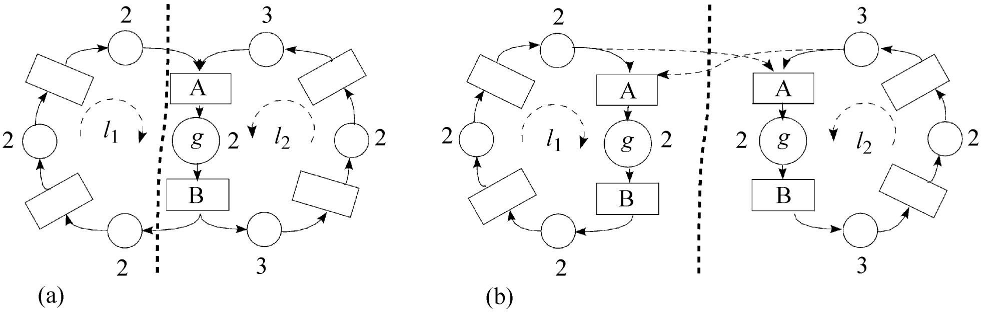


FIGURE 13 Illustration of replication and its effect on iteration bound.

shifting a module or swapping a pair of modules. We then describe a clique model which is used to replace a multiple pin net. The star and loop models are variations of two pin net models, however, with less complexity than the clique model. Finally, a flow model is introduced for network flow approaches.

4.1. Shift Model

The shift model [101] for multiple pin net is useful when we perturb the partition by shifting one module to a different vertex set or by swapping two modules between different vertex sets. Let us simplify the description by assuming only one module is shifted to a different vertex set. A swap of a pair of modules can be treated as two steps of module shifting.

For each shift, we want to update the cut count. We also want to update the potential change in cost for each module if it were to be shifted, so that we can rank the modules for the next move. Such cost revision can be expensive if the circuit has large nets which contain huge numbers of pins, *e.g.*, hundreds of thousand pins.

The shift model reduces the complexity of the cost revision by utilizing the property that for huge nets most shifts of its pins do not change the cost of the other pins in the net.

Let us simplify the description by considering a two way partitioning. The model can be extended to multiple way partitioning according to the choice of objective functions. Let module v_j be shifted from vertex set V_1 to V_2. The configuration of nets $e_i \in E(\{v_j\})$ connecting module v_j is revised. For each net e_i, we denote k_i to be the number of pins of e_i in V_1 and $|e_i| - k_i$ the number of pins of e_i in V_2 (Fig. 14). With respect to net e_i, we update the pin numbers k_i and $|e_i| - k_i$ after module v_j is shifted. We also update the cost of modules in nets e_i.

1. If the revised $k_i \geq 2$, the potential cost of pins due to net e_i is zero. For the case that $|e_i| - k_i = 1$, we increase the cut count by c_i and set the potential cost of pins in e_i. Otherwise, the move has no effect on the cut count and potential cost.
2. If the revised pin count $k_i = 1$, the shift of the last pin of e_i in V_1 will decrease the cut count by c_i. We then update the potential cost of this last pin.
3. If $k_i = 0$, the cut count reduces by c_i. However, the shift of any pin $v_k \in e_i$ from V_2 to V_1 will increase the cut count. Thus, in this case, we reflect the cost of potential shift on the pins of e_i, which takes $O(|e_i|)$ operations.

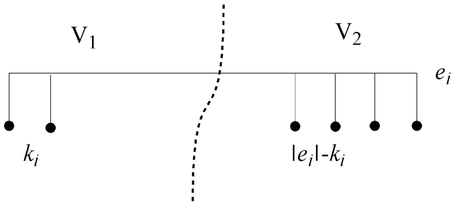


FIGURE 14 Multiple pin net model of shifting process.

4.2. Clique of Two Pin Nets

Some researchers use cliques of two pin nets to model multiple pin nets. Given a multiple pin net e_i, we construct a clique of $(1/2)|e_i|(|e_i| - 1)$ two pin nets to connect all pairs of pins in the net. The clique model maintains the symmetric relation of the modules of the same net in the sense that the order of the pins in the net has no effect on the cost.

The weight of two pin nets in the clique module is adjusted by some factor. One approach is to use $2/|e_i|$ to scale down the connectivity. The total weight of all the nets in the clique is $(2/|e_i|) \times (1/2)|e_i|(|e_i| - 1)c_i = (|e_i| - 1)c_i$. Note that it takes $|e_i| - 1$ two pin nets to form a spanning tree of $|e_i|$ modules.

Other factor has been proposed such as $1/(|e_i| - 1)$ which is based on a different probability model. However, no factor can exactly reflect the cost of a multiple pin net model.

Complexity of the Clique Model The complexity of the clique model is high. There are $O(|e_i|^2)$ two pin nets in a clique model. Suppose the process of each two pin net takes a constant time. It takes $O(|e_i|^2)$ operations to process a multiple pin net e_i. Therefore, in practice, if the pin number is larger than a threshold, the net is ignored in the process.

4.3. Star of Two Pin Nets

A star model introduces less complexity than a clique model. Given a net e_i, we create a dummy module $\tilde{v}_i$. The dummy module $\tilde{v}_i$ connects every pin in e_i with a two pin net. This module maintains the symmetry of the net. However, we need only $|e_i|$ two pin nets.

For the clique and star models, the cost of the partition depends on the number of pins on the two sides of the partition. The cost is higher when the pins are distributed more evenly on the two sides of the cut. Thus, these models discourage even partitioning of the pins in the nets.

4.4. Loop Model of Two Pin Nets

A loop model reflects the exact cut count [22], however, it is sensitive to the order of the pins. We can derive heuristic ordering of the pins using a linear placement. Modules are sequenced according to their x coordinates in the placement. We find the partition by collecting the modules according to the sequence.

Following the order of the modules in the x coordinates, we link the modules of a multiple pin net with two pin nets into a loop. We link the pins in a sequence (Fig. 15) alternating on every other module. The loop is formed by the two connections at the two ends.

A factor of (1/2) is assigned to the two pin nets so that the cut count separating modules according to the sequence is one. The model remains correct even if any two consecutive modules in the sequence swap their order.

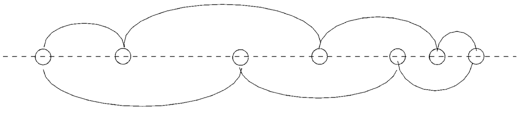


FIGURE 15 A loop model of multiple pin net where modules are placed on an x axis.

4.5. Flow Model

For the network flow approach, we consider each net e_i as a pipe. A set of saturated pipes forms a bottleneck of the flow. The union of the saturated pipes becomes the cut of the circuit. In such a model, we set the capacity of the pipe equal to the corresponding connectivity c_i [52].

Let x_{iu} be the amount of flow from pin v_i to net e_u and x_{uj} be the amount of flow from net e_u to pin v_j (Fig. 16). The total flow injected into the net should be smaller than or equal to its capacity and the incoming flow is equal to the outgoing flow, *i.e.*,

$$\sum_{v_i \in e_u} x_{iu} \leq c_u, \tag{27}$$

$$\sum_{v_i \in e_u} x_{iu} - \sum_{v_i \in e_u} x_{ui} = 0. \tag{28}$$

5. APPROACHES

In this section we introduce several approaches to partitioning. We first discuss two methods for optimal solutions: a branch and bound method and a dynamic programming algorithm. The branch and bound method is effective in searching exhaustively for the optimal solution for small circuits. The dynamic programming method presented runs in polynomial time and finds an optimal partition for a special class of circuits.

We then explain a few heuristic algorithms:

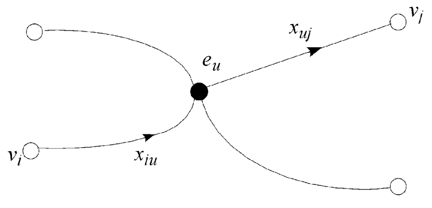


FIGURE 16 A flow model with respect to net e_u.

group migration, network flow, nonlinear programming, Lagrangian, and clustering methods. The group-migration approach is a popular method in practice due to its flexibility and effectiveness. The network flow method gives us a different view of the partitioning problem by transforming the minimization of the cut count into the maximization of the flow *via* a duality in linear programming. This approach derives excellent results with respect to certain objective functions. The nonlinear programming method provides a global view of the whole problem. The Lagrangian method is a useful approach for performance driven problems. Finally, we depict a clustering method for the partitioning.

In most cases, we illustrate the method in question using two-way partitioning as the target problem. However, many methods can be extended to other problems or different objective functions. For example, we can apply group migration to multiway [98, 99] or multiple level partitioning problems [68, 67] with modification to the cost of the moves. Furthermore, some methods may be combined to solve a problem. For example, we can use clustering to reduce the size of an input circuit and then use group migration to find a partition of the reduced circuit with much greater efficiency [24, 59]. In fact, this strategy derives the best results in terms of CPU time and cut count in recent benchmark [2].

5.1. Branch and Bound Method

The branch and bound method is an exhaustive search technique that may be effectively applied to the min-cut problem with size constraints for small cases. In the branch and bound process, the modules are first ordered in a sequence. For each module, we try placing it to either side of the cut.

The process can be represented by a complete binary tree with $|V|$ levels. The root of the tree is the first module in the sequence. The nodes in the kth level of the tree correspond to the kth module in the sequence. The two branches at each node represent the two trials where the kth module is placed on each of the two different sides. A path in the tree from the root to a leaf corresponds to one assignment for the partition.

We use a depth first search approach to traverse the binary tree. We prune the search space according to the size constraint and a partial cut count. In the binary tree, a node at level k along with the path from the root to the node represents a partition assignment of the first k modules. Let V_1 and V_2 be the two vertex sets of the partitions of the first k modules. If $S(V_i) > S_u$ for $i=1$ or 2, the size constraint is violated, and there is no need to proceed. Thus, we prune the branches below.

We also use a partial cut count to prune the binary tree. The cut of the partial partition is expressed as: $E(V_1, V_2)=\{e_i \,|\, |e_i \cap V_1| > 0$ and $|e_i \cap V_2| > 0\}$. The partial cut count is described as: $C(V_1, V_2) = \sum_{e_i \in E(V_1, V_2)} c_i$. If the partial cut count $C(V_1, V_2)$ is larger than the cut count of a known solution, the partition results below this node are going to be worse than the existing solution. We prune the branches of such a node.

Complexity of the Method Suppose the circuit has unit size $s_i=1$ on each module and the constraint requires an even size $S_l=S_u=|V|/2$ (assuming that $|V|$ is even). Applying Stirling's approximation [63], we have the number of possible partitions:

$$\frac{|V|!}{(|V|/2)!^2} \approx \sqrt{\frac{2}{\pi |V|}} 2^{|V|}. \tag{29}$$

Although the number of combinations is huge, we have found that the application to small circuits is practical. We improve the efficiency of the pruning by ordering the modules according to their degrees, *i.e.*, the number of nets connecting to the modules, in a descending order. With an elegant implementation, we can find optimal solutions when the number of modules is small, *e.g.*, $|V| \leq 60$.

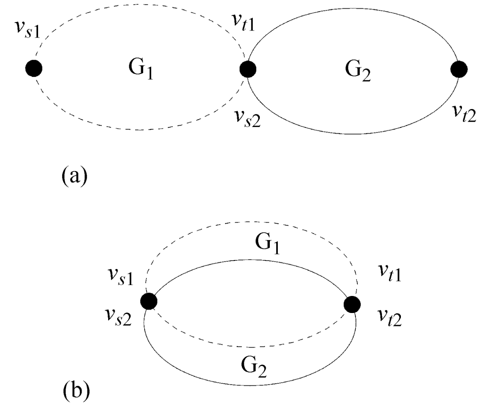
5.2. Dynamic Programming for a Serial and Parallel Graph

For the special case where the circuit can be represented by a serial and parallel graph of unit module size, we can find a minimum two way partition (V_1, V_2) with size constraints in polynomial time. In this section, we first describe the serial and parallel graph. We then depict a dynamic programming algorithm that solves the partitioning problem on this class of graphs. We assume that all modules are of unit size, *i.e.*, $s_i = 1$.

A serial and parallel graph can be constructed from smaller serial and parallel graphs by serial or parallel process. Each serial and parallel graph has a source module v_s and a sink module v_t. A graph $G(V, E)$ with two modules, $V = \{v_s, v_t\}$ and one edge $E = \{e\}$, $e = \{v_s, v_t\}$ is a basic serial and parallel graph. A serial and parallel graph is constructed from the basic graph by a series of serial and parallel processes.

Serial Process Given two serial and parallel graphs, $G_1(V_1, E_1)$ and $G_2(V_2, E_2)$, we construct a serial and parallel graph $G(V, E)$ by merging the sink module v_{t1} of G_1 and the source module v_{s2} of G_2 (Fig. 17(a)). The source module v_{s1} of graph G_1 becomes the source module of graph G, *i.e.*, $v_s = v_{s1}$. The sink module v_{t2} of graph G_2 becomes the sink module of graph G, *i.e.*, $v_t = v_{t2}$.

Parallel Process Given two serial and parallel graphs, $G_1(V_1, E_1)$ and $G_2(V_2, E_2)$, we construct a serial and parallel graph $G(V, E)$ by merging the source module v_{s1} of G_1 and the source module v_{s2} of G_2 and by merging the sink module v_{t1} of G_1 and the sink module v_{t2} of G_2 (Fig. 17(b)). The merged source module and merged sink module become the source module v_s and the sink module v_t of graph G, respectively.



(a)

(b)

FIGURE 17 Construction of serial and parallel graphs.

Dynamic Programming The dynamic programming algorithm performs a bottom up process according to the construction of the serial and parallel graph. It starts from the basic serial and parallel graph. For each graph $G(V, E)$, we derive two tables.

$a(i,j)$: the minimum cut count with i modules on the left hand side and j modules on the right hand side under the condition that source module v_s is on the left hand side and sink module v_t is on the right hand side.

$b(i,j)$: the minimum cut count with i modules on the left hand side and j modules on the right hand side under the condition that both source module v_s and sink module v_t are on the left hand side.

Let graph $G(V, E)$ be constructed with $G_1(V_1, E_1)$ and $G_2(V_2, E_2)$ by one of the serial and parallel processes. Let a_1, b_1 be the tables of graph G_1 and a_2, b_2 be the tables of graph G_2. We construct the tables a, b of graph $G(V, E)$ as follows.

Table Formulas for Parallel Process

$$a(i, j) = \min_{k+m=|V_2|} a_1(i + 1 - k, j + 1 - m) + a_2(k, m), \quad \forall i + j = |V|, \tag{30}$$

$$b(i, j) = \min_{k+m=|V_2|} b_1(i + 2 - k, j - m) + b_2(k, m), \quad \forall i + j = |V|. \tag{31}$$

For table $a(i,j)$, we try all combinations of tables a_1 and a_2 with the constraint that the number of modules on the left hand side is i and the number of modules on the right hand side is j. Note that the extra addition of 1 in the index is used to compensate the merging of the two source modules or the sink modules. For table $b(i,j)$, we try all combinations of tables b_1 and b_2 with the same size constraint.

Table Formula for Serial Process

$$a(i,j) = \min(\min_{k+m=|V_2|} a_1(i-k, j+1-m) + b_2(k,m),\ \min_{k+m=|V_2|} b_1(i+1-k, j-m) + a_2(k,m)), \quad \forall i+j=|V|, \tag{32}$$

$$b(i,j) = \min(\min_{k+m=|V_2|} a_1(i-k, j+1-m) + a_2(m,k),\ \min_{k+m=|V_2|} b_1(i+1-k, j-m) + b_2(k,m)), \quad \forall i+j=|V|. \tag{33}$$

For table $a(i,j)$, we try all combinations of tables a_1 and b_2 and all combinations of tables b_1 and a_2. For the combinations of tables a_1 and b_2, the merged module (by merging v_{t1} and v_{s2}) is on the right hand side. For the combinations of tables b_1 and a_2, the merged module is on the left hand side. For table $b(i,j)$, we try all combinations of tables a_1 and a_2 and all combinations of tables b_1 and b_2. For the combinations of tables a_1 and a_2, the merged module is on the right hand side. In terms of G_2, its source module v_{s2} is on the right hand side and its sink module v_{t2} is on the left hand side. Thus, the indices of table a_2 are reversed, *i.e.*, $a_2(m,k)$ instead of $a_2(k,m)$. For the combinations of tables b_1 and b_2, the merged module is on the left hand side.

5.3. Group Migration Algorithms

The group migration algorithm was first proposed by Kernighan and Lin [60] in 1970. Since then, many variations [15, 26, 27, 33, 39, 45, 49, 84, 97–99, 108, 111, 116] have been reported to improve the efficiency and effectiveness of the method. Today, it is still a popular method in practice.

The probability of finding the optimum solution in a single trial drops exponentially as the size of the circuit increases [60]. Using the original version, Kernighan and Lin showed that the probability of obtaining an optimal solution is a function of the problem size, $p(|V|) = 2^{-n/30}$. In other words, if the circuit size is large, then the heuristic Kernighan–Lin algorithm is unlikely to jump out of local minima, and so the optimum solution will not be found. The progress of the method has definitely pushed the envelope further.

In this section, we concentrate on two-way min-cut with size constraints. The method is flexible and can be extended to other partitioning problems with modifications of the moves and the cost function.

The algorithm performs a series of passes. At the beginning of a pass, each module is labeled *unlocked*. Once a module is shifted, it becomes *locked* in this pass. The group migration algorithm iteratively interchanges a pair of unlocked modules or shifts a single module to a different side with the largest reduction (*gain*) of the cost function. This continues until all modules are locked. The lowest cost along the whole sequence of swapping is recorded. The group migration takes the subsequence that produces the lowest cut count and undoes the moves after the point of the lowest cost. This partitioning result is then used as the initial solution for the next pass. The algorithm terminates when a pass fails to find a result with a cost lower than the cost of the previous pass.

Group Migration Algorithm Input: Hypergraph $H(V,E)$ and an initial partition. Cost function and size constraints.

1. One pass of moves.
   - 1.1. Choose and perform the best move.
   - 1.2. Lock the moved modules.
   - 1.3. Update the gain of unlocked modules.
   - 1.4. Repeat Steps 1.1–1.3 until all modules are locked or no move is feasible.
   - 1.5. Find and execute the best subsequence of the move. Undo the rest of the sequence.
2. Use the previous result as an initial partition.
3. Repeat the pass (Steps 1 and 2) until there is no more improvement.

Figure 18 illustrates the cost of a sequence of moves. This algorithm escapes from local optima by a whole sequence of the moves even when a single move may produce a negative gain.

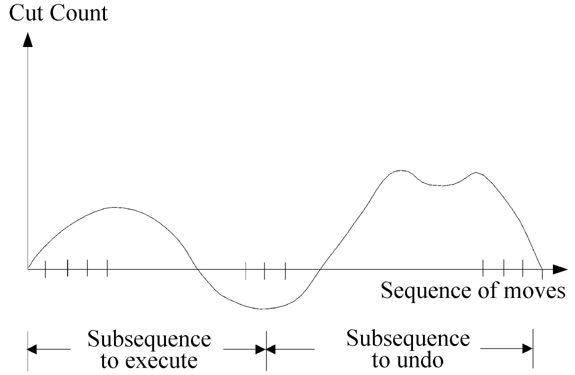


FIGURE 18 Cost of a sequence of moves and subsequence selection.

In the following, we discuss variations of several parts in the process: basic moves (Step 1.1), data structure, gains (Steps 1.1 and 1.3). At the end of this subsection, we introduce a net based move and a simulated annealing approach.

5.3.1. Basic Moves

Basic moves cover the shifting of a single module and the swapping of a pair of modules. A swapping can be conceived as two consecutive shifts, however, with consideration of the mutual effect between the two shifts.

(i) Module Shifting For each unlocked module, we check its *gain*: the cost function reduction by shifting the module to a different side assuming that the rest of the modules are fixed. To select the best module to shift, we order on each side the modules according to their shift gains. If the size constraints are violated after the shift, the move is not feasible. We search for the best feasible module to move [40].

(ii) Pairwise Swapping We exchange two modules in two vertex sets of the partition. Note that the gain of the swap is not equal to the sum of the gains of two shifts. The mutual effect between the two modules needs to be included when we derive the gain. Thus, the best pair may not be the two modules on the top of the two sides. The search of all pairs takes $O(|V_1||V_2|)$ operations. In practice, we order modules according to their shift gain. The search of the best pair is limited to the top k modules on each side, *e.g.*, $k=3$. Thus, the complexity is actually $O(k^2)$.

Pairwise swapping is a natural adoption when the size constraint is tight. When no single shift is feasible, we can use swapping to balance the size of the partition.

5.3.2. Data Structure

The choice of data structure strongly depends on the cost functions, gains, and the characteristic of VLSI circuitry. A sorting structure such as heap or AVL tree is a natural choice to sort for the top modules. However, for the case that the gain differs by a very limited quantities, an array structure can simplify the coding and the complexity.

(i) Heap or AVL Tree We can use a heap or AVL tree to sort the modules according to their shift gain. Each side of the partition keeps a heap. The top of the heap is the module of the maximum gain. The sorting of each module takes $O(|V|\log(|V|))$ operations.

(ii) Array (Bucket) of Link List Figure 19 illustrate a *bucket list data structure*. The gain is transformed to the index of the bucket [40]. Modules of the same gain are stored in the same bucket by a link list. A bucket is an effective data structure when the objective function is the cut count. The gain of cut count is limited by the maximum degrees of the modules, *i.e.*, $\deg_{\max} = \max_{v_i \in V} \sum_{e \in E(\{v_i\})} c_e$. Thus, the dimension of the bucket is set to be $2\deg_{\max}$.

For VLSI applications, the degree of modules is much smaller than the number of modules. Thus, the dimension of the bucket is small. It is very efficient to search and revise the module order in the bucket structure. In fact, it is proven that using the bucket structure and cut count as the objective

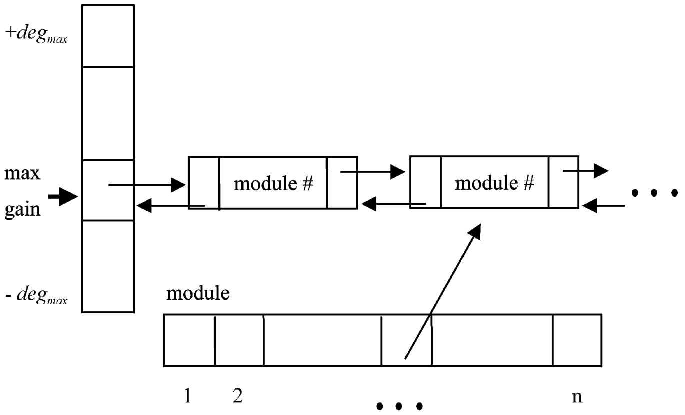


FIGURE 19 Bucket list.

function, it takes linear time proportional to the total number of pins to perform each pass [40].

5.3.3. Gains

In this subsection, we use cut count as the objective function. The extension to other cost functions is possible. However, we may loose efficiency.

(i) Shift Gain We use shift model for multiple pin nets. Given a module v_i, we check the set $E(\{v_i\})$ of nets connecting to this module. The contribution of each net $e \in E(\{v_i\})$ by shifting module v_i is the gain $g_e(v_i)$ of the net with respect to module v_i. The gain $g(v_i)$ of module v_i is the total gains of all its adjacent nets, *i.e.*, $g(v_i) = \sum_{e \in E(\{v_i\})} g_e(v_i))$

(ii) Swap Gain The swap gain is the sum of the gains of two modules v_i and v_j, deducting the effect on common nets, *i.e.*, $g(v_i) + g(v_j) - z \sum_{e \in E(\{v_i\}) \cap E(\{v_j\})} (g_e(v_i) + g_e(v_j))$.

(iii) Weights of Multipin Nets The sequence of the move depends much on the gain calculation. For a circuit of 1,000,000 modules, suppose the degree of most modules is less than 100 and each net is of unit weight. We have roughly 1,000,000 modules/200 gain levels = 5,000 modules per gain level. To differentiate these 5,000 modules, we have to adjust the weight of multiple pin nets.

(iii) (a) Levels with Priority The first level gain is identical to the shift gain of cut count. The second level gain is equal to the number of nets that have one more pins on the same side. Thus, the kth level gain is equal to the number of nets that have k more pins on the same side [65]. The pins on the other side will increase by one after the module is shifted. Thus, the negative gain of level k is contributed by the nets with $k-1$ pins on the other side.

Let us assume that module v_i is in vertex set V_1 to simplify the notation. For each net $e_j \in E(\{v_i\})$, we denote $k_j = |e_j \cap V_1|$ the number of pins in V_1. Let us define $E(+, i, k)$ to be the set of nets $e_j \in E(\{v_i\})$ with $k_j = k+1$ pins in V_1 (the extra one is used to count module v_i itself) and nonzero pins in V_2, *i.e.*, $|e_j| > k_j$. And $E(-, i, k)$ to be the set of nets $e_j \in E(\{v_i\})$ with no other pins in V_1 and $k-1$ pins in V_2, *i.e.*, $|e_j| = k$ and $k_j = 1$. Then, the kth level gain of module v_i, $g_i(k)$, is the weight difference of the two sets, $E(+, i, k)$ and $E(-, i, k)$.

$$g_i(k) = \sum_{e \in E(+,i,k)} c_e - \sum_{e \in E(-,i,k)} c_e \tag{34}$$

$$E(+, i, k) = \{e_j \mid e_j \in E(\{v_i\}), k_j = k+1, |e_j| > k_j\} \tag{35}$$

$$E(-, i, k) = \{e_j \mid e_j \in E(\{v_i\}), k_j = 1, |e_j| = k\} \tag{36}$$

We compare the modules with a priority on the lower level gain. In other words, we compare the first level first. If the modules are equal at the first level gain, we then compare the second level and so on. In practice, we limit the number of levels by a threshold, *e.g.*, $l \leq 3$.

(iii) (b) Probabilistic Gain In probabilistic gain model [37], each module v_i is assigned a weight $p(v_i)$. The weight $p(v_i)$ is a function of the gain $g(v_i)$ of module v_i to reflect the belief level (potential) that the shift of module v_i will be executed at the end of the pass. Thus, if module v_i is unlocked,

$$p(v_i) = f(g(v_i)). \tag{37}$$

Otherwise, $p(v_i) = 0$. Figure 20 illustrates function f, which increases monotonically. The slope within g_0 and g_{up} amplifies the difference of gains. The slope is clamped at two ends $p_{\max}$ and $p_{\min}$ ($0 \leq p_{\min} < p_{\max} \leq 1$) which represent the maximum potential that the module will shift or stay.

For each net $e \in E(\{v_i\})$, its contribution $g_e(v_i)$ to the gain of module v_i is the tendency that the whole net will shift with module v_i to the other side. To simplify the notation, let us assume that module v_i is in V_1. Thus, we have the following expression.

$$g_e(v_i) = c_e\left(\prod_{j \neq i, v_j \in e \cap V_1} p(v_j) - \prod_{v_j \in e \cap V_2} p(v_j)\right) \tag{38}$$

where $\prod_{v_j \in S} p(v_j) = 1$ if S is an empty set. The first term $\prod_{j \neq i, v_j \in e \cap V_1} p(v_j)$ in the parentheses is the potential that all the pins will shift with module v_i to V_2. Hence, $c_e \times \prod_{j \neq i, v_j \in e \cap V_1} p(v_j)$ is the expected gain if module v_i is shifted. The second term $\prod_{v_j \in e \cap V_2} p(v_j)$ is the potential that the pins in V_2 will shift to V_1. Thus, $c_e \times \prod_{v_j \in e \cap V_2} p(v_j)$ is the expected loss if module v_i is shifted.

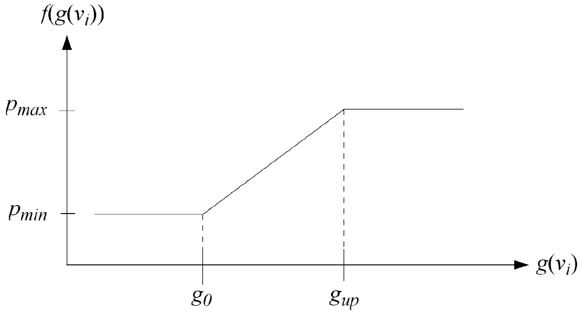


FIGURE 20 Function of probabilistic gain.

The gain of a module v_i is the total gains of the adjacent nets with respect to this module, *i.e.*,

$$g(v_i) = \sum_{e \in E(\{v_i\})} g_e(v_i). \tag{39}$$

Net gain $g_e(v_i)$ and module potential $p(v_i)$ are mutually dependent. We derive the values *via* iterations. Initially, we use the plain shift gain (by cut count) to derive the potential $p(v_i) = f(g(v_i))$. From these initial potentials, we derive the probabilistic net gain. The net gain is then used to derive the module gain. In practice, we stop after a limited number of cycles, *e.g.*, two iterations ([37]). Note that there is no guarantee that the iteration will converge.

After each move, the associated module potential and probabilistic net gains are updated and the plain cut count is recorded. Exact cut count is used when we select the subsequence of move to execute.

It has been shown *via* benchmarks released by ACM/SIGDA, the probabilistic gain model produces excellent partitioning results; it outperforms the other gain models by wide margins.

5.3.4. Net-based Move

The net based process [115, 32] is similar to the module based approach except that all operations are based on the concept of the critical and complementary critical sets. The main differences are (1) Instead of a single module, each move now shifts one critical or complementary critical set, depending on the type of objective function. For convenience, we say a move is *initiated* by a net e_u if this move is composed of shifting the critical or complementary critical set associated with e_u. (2) The locking mechanism is operated on a net, that is, if the critical or complementary critical set of a net has been moved then all the moves initiated by this net will be prohibited thereafter.

Given a net e_u and a vertex set V_b, let us define the *critical set* of net e_u with respect to set V_b as

$$s_{ub} = e_u \cap V_b, \tag{40}$$

and the *complementary critical set* of e_u with respect to set V_b as

$$s_{u\bar{b}} = e_u \cap \bar{V}_b \tag{41}$$

For a move associated with a net e_u, we can either place the critical set S_{ub} into a partition other than V_b, or the complementary critical set $S_{u\bar{b}}$ into the partition V_b. The gain of each move is then computed by evaluating the change of the cost due to the move of the critical or complementary critical set.

Usage of Basic Module Moves Although the net-based move model provides a different process to improve current partition, it is more expensive than the module-based move model because more modules are involved in each move.

We can mimic the net based move by adding weights to the connectivity of desired nets [38]. The basic move is still based on the modules. However, after module v_i is moved, we add more weights on the nets connecting to v_i, *i.e.*, $E(\{v_i\})$. These extra weights encourage the adjacent modules to go along with module v_i and thus achieves the effect of net based move. Empirical study finds improvement on the partitioning results.

5.3.5. Simulated Annealing Approach

For simulated annealing [20, 81, 62, 56], we can adopt the basic moves such as module shifting and pairwise swapping. There is no need of lock mechanism. To allow a larger searching space, we incorporate the size constraints into objective function, *e.g.*,

$$C(V_1, V_2) + \alpha(S(V_1) - S(V_2))^2. \tag{42}$$

where α is a coefficient. We can adjust it according to the annealing temperature. As temperature drops, we gradually increase α to enforce the size balance.

5.4. Flow Approaches

In this section, we assume that the circuit can be represented by a graph $G(V, E)$ with unit module size, *i.e.*, $s_i = 1$ and all nets are two pin nets. The flow approach can be extended to multiple pin nets using a flow model.

We first go through maximum flow minimum cut [1, 73] to introduce the duality [30] and the concept of shadow price. The derivation is then extended to a weighted cluster ratio cut and a replication cut. Finally, we introduce heuristic algorithms that accelerate the flow calculation. The flow approach can derive excellent results. Furthermore, exploiting its duality formulation, we can derive a tight bound of the optimal solutions.

5.4.1. Maximum Flow Minimum Cut

In maximum flow minimum cut formulation, the flow injects into module v_s and drains from module v_t. The flow is conservative at all other modules. The capacity of the nets e_{ij} is equal to its connectivity, c_{ij}. We set $c_{ij} = 0$ if there is no net connecting modules v_i and v_j. The notation x_{ij} denotes the amount of flow from module v_i to module v_j and x_{ji} denotes the amount of flow from module v_j to module v_i on net e_{ij}. The objective is to maximize the flow injection f into v_s.

$$\text{Obj: max} f \tag{43}$$

subject to the constraints,

$$x_{ij} + x_{ji} \leq c_{ij}, \quad \forall 1 \leq i, j \leq |V| \tag{44}$$

$$\sum_{j=1}^{|V|} x_{js} - \sum_{j=1}^{|V|} x_{sj} - f = 0 \tag{45}$$

$$\sum_{j=1}^{|V|} x_{jt} - \sum_{j=1}^{|V|} x_{tj} + f = 0 \tag{46}$$

$$\sum_{j=1}^{|V|} x_{ij} - \sum_{j=1}^{|V|} x_{ji} = 0, \quad \forall 1 \le i \le |V| \tag{47}$$

$$x_{ij} \ge 0, \quad \forall 1 \le i,j \le |V|. \tag{48}$$

To derive the duality, we use shadow prices: a bidirectional distance d_{ij} for each net e_{ij} Eq. (44), potential λ_i for each module v_i Eqs. (45)–(47) The dual problem can be expressed as follows [30].

$$\text{Obj}: \min \sum_{e_{ij} \in E} c_{ij} d_{ij} \tag{49}$$

subject to

$$d_{ij} \ge |\lambda_i - \lambda_j|, \quad \forall 1 \le i,j \le |V|, \tag{50}$$

$$\lambda_t - \lambda_s = 1. \tag{51}$$

Figure 21 illustrates the formulation. As we increase the flow, certain nets are going to saturate, *i.e.*, the two sides of inequality expression (44) become equal. Once the saturated nets become a bottleneck of the flow, the set of nets forms a cut $E(V_1, V_2)$ with $v_s \in V_1$ and $v_t \in V_2$. In duality, the potential of modules in V_2 increases to one, and the potential of modules in V_1 remains to be zero, *i.e.*, $\lambda_i = 1$, $\forall v_i \in V_2$ and $\lambda_i = 0$, $\forall v_i \in V_1$. The distance of nets in the cut is one, while the distance of nets outside the cut is zero, *i.e.*, $d_{ij} = 1$, $\forall c_{ij} \in E(V_1, V_2)$ and $d_{ij} = 0$, $\forall c_{ij} \notin E(V_1, V_2)$.

5.4.2. The Weighted Cluster Ratio Metric and a Uniform Multi-commodity Flow Problem

In a uniform multi-commodity flow problem [74, 75], the demand of flow between each pair of modules is equal to an identical value *f*. As we keep increasing *f*, some of the nets become saturated. These saturated nets form a bottleneck of communication and thus prescribes a potential clustering of the communication system [71].

We simplify the notation by assuming a graph model $G(V, E)$. From each module v_p, we inject flow $f/2$ to each of the rest modules. Summing up the flow in two directions, the flow between each pair of modules is *f*. We define the flow originated from module v_p as commodity *p*. Let $x_{ij}^{(p)}$ be the flow for commodity *p* on net e_{ij}. The objective is to maximize *f*:

$$\text{Obj}: \max f \tag{52}$$

subject to the flow demand from module v_p to the other modules v_i,

$$\sum_{j=1}^{|V|} x_{ij}^{(p)} - \sum_{j=1}^{|V|} x_{ji}^{(p)} = \begin{cases} -f/2 & \text{if } i \ne p, \quad \text{and} \quad 1 \le i,p \le |V|, \\ (|V|-1)f/2 & \text{if } i = p, \quad \text{and} \quad 1 \le i,p \le |V|, \end{cases} \tag{53}$$

and the net capacity constraint,

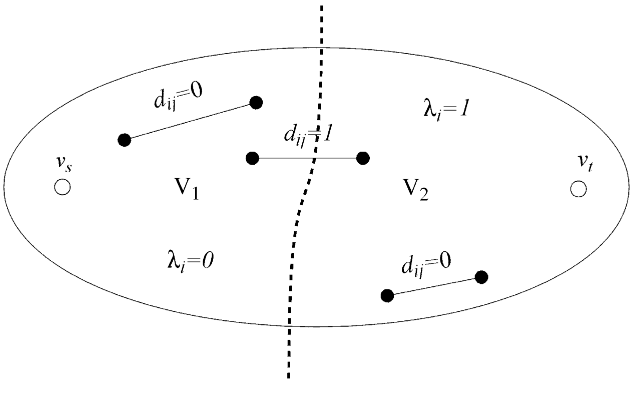


FIGURE 21 Illustration of maximum flow minimum cut formulation.

$$\sum_{p=1}^{|V|} x_{ij}^{(p)} + \sum_{p=1}^{|V|} x_{ji}^{(p)} \leq c_{ij}, \quad 1 \leq i,j \leq |V|. \tag{54}$$

We transform the above linear programming problem to its dual expression by assigning dual variables $\lambda_i^{(p)}$ to module v_i with respect to commodity p Eq. (53), and distance d_{ij} to net e_{ij} Eq. (54), then we have:

$$\text{Obj}: \min \sum_{e_{ij} \in E} c_{ij} d_{ij} \tag{55}$$

subject to

$$d_{ij} \geq \left|\lambda_i^{(p)} - \lambda_j^{(p)}\right|, \quad 1 \leq i,j,p \leq |V| \tag{56}$$

$$\frac{1}{2} \sum_{p=1}^{|V|} \sum_{i=1, i \neq p}^{|V|} \left(\lambda_i^{(p)} - \lambda_p^{(p)}\right) \geq 1 \tag{57}$$

The Properties of Shadow Prices The shadow price d_{ij} can be viewed as bidirectional, *i.e.*, $d_{ij} = d_{ji}$. It represents the distance of net e_{ij}, which corresponds to the cost to transmit flow through e_{ij}. Variable $\lambda_i^{(p)}$ is the potential of module v_i with respect to commodity p.

From constraints (56), (57), we can derive two properties for distance function d_{ij} and potential $\lambda_i^{(p)}$ [71].

Property I: Triangular Inequality The distance metric d_{ij} satisfies the triangular inequality:

$$d_{ij} + d_{jk} \geq d_{ik}, \quad \forall v_i, v_j, v_k \in V \tag{58}$$

Property II: Potential Function The term $\lambda_i^{(p)} - \lambda_p^{(p)}$ in expression (56) is equal to the shortest distance between modules v_i and v_p based on net distances d_{ij}. In fact, from triangular inequality, we obtain $\lambda_i^{(p)} - \lambda_p^{(p)} = d_{ip}$.

We normalize the objective function (55) with the left hand side terms of inequality (57). The objective function can be expressed as:

$$\text{Obj}: \min \frac{\sum_{e_{ij} \in E} c_{ij} d_{ij}}{(1/2) \sum_{p=1}^{|V|} \sum_{i=1, i \neq p}^{|V|} \left(\lambda_i^{(p)} - \lambda_p^{(p)}\right)} = \frac{\sum_{e_{ij} \in E} c_{ij} d_{ij}}{(1/2) \sum_{p=1}^{|V|} \sum_{i=1, i \neq p}^{|V|} d_{ip}} \tag{59}$$

In the solution of linear programming problem (52)–(56), the nets with positive d_{ij} values partition V into vertex sets $V_1, V_2, \ldots, V_k$. More specifically, nets connecting modules in different sets, V_i, V_j, $i \neq j$, have the same distance d_{ij} values (we use d_{ij} to denote the distance between vertex sets V_i and V_j when this does not cause confusion), while nets connecting only modules in the same subgraph have zero distance, $d_{ij} = 0$ (Fig. 22). We can rewrite the denominator of the objective function and state the problem as follows.

Statement of Weighted Cluster Ratio Cut [103] *Find the distance d_{ij} and the number of partition k with an objective function of weighted cluster ratio*:

$$\min_{d_{ij},k} W_C(V_1, V_2, \ldots, V_k) = \min_{d_{ij},k} \frac{\sum_{i=j+1}^{k} \sum_{j=1}^{k-1} d_{ij} C(V_i, V_j)}{\sum_{i=j+1}^{k} \sum_{j=1}^{k-1} d_{ij} S(V_i) \times S(V_j)} \tag{60}$$

where distance d_{ij} is subject to the property of triangular inequality.

According to the mechanism of the duality, the objective functions of the primal and dual formulations are equal when the solution is optimal [25].

THEOREM 5.1 *For feasible solutions, we have the inequality $f \leq W_C(V_1, V_2, \ldots, V_k)$. The equality holds when the solution is optimal, i.e., the maximum uniform multicommodity flow equals the*

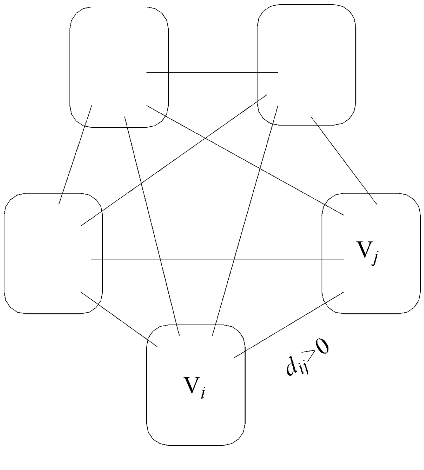


FIGURE 22 Distance between clusters.

minimum weighted cluster ratio of any cut, $max_{x_{ij}} f \leq min_{d_{ij},k} W_C(V_1, V_2, \ldots, V_k)$.

Expression (60), weighted cluster ratio [103], is similar to cluster ratio with a weighted metric d_{ij}. In general, the solution for the minimum weighted cluster ratio does not directly correspond to the partition of optimum cluster ratio. However, if distance d_{ij} is a constant value between all pairs of vertex sets V_i and V_j then the weighted cluster ratio provides the solution for cluster ratio.

When the nets with positive distance d_{ij} form a two-way partition, we can show that the partition defines the ratio cut. When the nets with positive distances form a k-way partition with $k \leq 4$, we also find that there exists a two-way partition that again defines the ratio cut [28].

THEOREM 5.2 *Let net set* $D = \{e_{ij} | d_{ij} > 0\}$ *define a cut that separates the circuit into k disconnected subsets. If* $k \leq 4$, *then there exists a ratio cut that is a subset of D.*

5.4.3. *A Replication Cut for Two-way Partitioning*

We adopt the linear programming formulation of network flow problem [1, 30], where each module is assigned a potential and a cut is represented by the difference of module potentials as shown in Figure 23. With respect to the directed cut $E(V_1 \rightarrow \bar{V}_1)$, we use w_{ij} to denote the potential difference between the cut from module $v_i \in V_1$ to module $v_j \notin V_1$. The potential of each module v_i is denoted by p_i. For module v_i in V_1, $p_i = 1$, and for modules v_i in $\bar{V}_1$, $p_i = 0$. Thus all nets $e_{ij} \in E(V_1 \rightarrow \bar{V}_1)$ have $w_{ij} = 1$. The remaining nets have $w_{ij} = 0$.

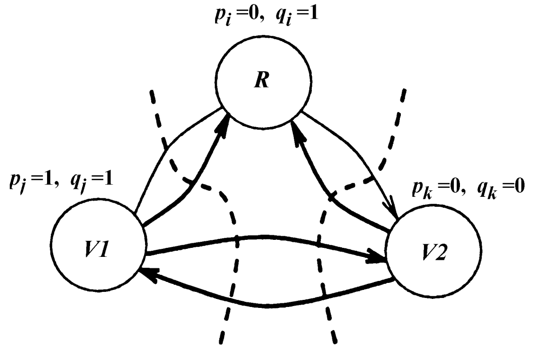


FIGURE 23 p potential and q potential of each module.

With respect to the directed cut $E(V_2 \rightarrow \bar{V}_2)$, we use u_{ji} with a reversed subscript ji to denote the potential difference between the cut from module $v_i \in V_2$ to module $v_j \notin V_2$ (Fig. 23). The potential of each module v_i is denoted by q_i. For modules v_i in $\bar{V}_2$, $q_i = 1$, and for modules v_i in V_2, $q_i = 0$. The potential difference u_{ji} has a reverse direction with net e_{ij} because we set the potential on $\bar{V}_2$ side high and the potential on V_2 side low. All nets $e_{ij} \in E(V_2 \rightarrow \bar{V}_2)$ have $u_{ji} = 1$. The remaining nets have $u_{ji} = 0$.

Primal Linear Programming Formulation The problem is to minimize the total weight of crossing nets:

$$\text{Obj: min} \sum_{e_{ij} \in E} c_{ij} w_{ij} + \sum_{e_{ij} \in E} c_{ji} u_{ij} \tag{61}$$

subject to

$$w_{ij} - p_i + p_j \geq 0 \quad \forall 1 \leq i,j \leq |V| \tag{62}$$

$$u_{ij} - q_i + q_j \geq 0 \quad \forall 1 \leq i,j \leq |V| \tag{63}$$

$$q_i - p_i \geq 0 \quad \forall v_i \in V, \ \ v_i \neq v_s, v_t \tag{64}$$

$$p_s = 1 \tag{65}$$

$$q_s = 1 \tag{66}$$

$$p_t = 0 \tag{67}$$

$$q_t = 0 \tag{68}$$

$$w_{ij}, u_{ij} \geq 0 \quad \forall 1 \leq i,j \leq |V| \tag{69}$$

To minimize objective function (61), the equality of constraint (62) holds, *i.e.*, $w_{ij} = p_i - p_j$, if $p_i \geq p_j$, otherwise, $w_{ij} = 0$. Similarly, constraint (63) requires $u_{ij} = q_i - q_j$ if $q_i \geq q_j$, otherwise $u_{ij} = 0$. Expression (64) demands potential q_i be not less than potential p_i for any module $v_i \in V$. Since high potential p_i corresponds to set V_1, and high potential q_i corresponds to set $\bar{V}_2$, inequality (64) enforces V_1 be a subset of $\bar{V}_2$. Consequently, the requirement that $V_1 \cap V_2 = \emptyset$ is satisfied.

Constraints (65)–(68) set the potentials of modules v_s and v_t. Constraint (69) requires potential difference w_{ij} and u_{ij} be nonnegative. Figure 23 shows one ideal potential configuration of the solution.

Dual Linear Programming Formulation If we assign dual variables (Lagrangian multiplier) x_{ij} to inequality (62) with respect to each net, x'_{ij} to inequality (63), λ_i to inequality (64) with respect to module v_i, and a_s, b_s, a_t, b_t to inequalities (65)–(68), respectively, then we have the dual formulation.

$$\text{Obj}: \max a_s + b_s \tag{70}$$

subject to

$$x_{ij} \le c_{ij} \quad \forall 1 \le i,j \le |V| \tag{71}$$

$$x'_{ij} \le c_{ji} \quad \forall 1 \le i,j \le |V| \tag{72}$$

$$\sum_{j=1}^{|V|} -x_{ij} + x_{ji} - \lambda_i = 0 \quad \forall v_i \in V,\ \ v_i \ne v_s, v_t \tag{73}$$

$$\sum_{j=1}^{|V|} -x'_{ij} + x'_{ji} + \lambda_i = 0 \quad \forall v_i \in V,\ \ v_i \ne v_s, v_t \tag{74}$$

$$\sum_{j=1}^{|V|} -x_{sj} + x_{js} + a_s = 0 \tag{75}$$

$$\sum_{j=1}^{|V|} -x_{tj} + x_{jt} + a_t = 0 \tag{76}$$

$$\sum_{j=1}^{|V|} -x'_{sj} + x'_{js} + b_s = 0 \tag{77}$$

$$\sum_{j=1}^{|V|} -x'_{tj} + x'_{jt} + b_t = 0 \tag{78}$$

$$\lambda_i, x_{ij}, x'_{ji} \ge 0 \quad \forall 1 \le i,j \le |V|,\ \ v_i \ne v_s, v_t \tag{79}$$

$$a_s, a_t, b_s, b_t \text{ unrestricted} \tag{80}$$

where inequalities (71), (72) are derived with respect to each w_{ij} and u_{ij} respectively. Similarly, Eqs. (73)–(78) are derived with respect to each p_i, q_i, p_s, p_t, q_s and q_t. The equality of Eqs. (73)–(78) holds because p_i, q_i, p_s, p_t, q_s and q_t are not restricted on sign in the primal formulation. Variables λ_i, x_{ij}, and x'_{ij} are positive in Eq. (79) because their corresponding expressions (62)–(64) are inequality constraints.

We can view $G(V, E)$ as a network flow problem and interpret c_{ij} as the flow capacity, x_{ij} as the flow of net e_{ij}. Constraint (71) requires that the flow x_{ij} be not larger than the flow capacity c_{ij} on each net e_{ij}. In constraint (72), the set of nets are in a reversed direction and flow x'_{ij} is not larger than the capacity of the capacity c_{ji} of net e_{ji} in E. Corresponding to $G(V, E)$, we use $G'(V', E')$ to denote the reversed graph.

Constraint (73) has the total flow x_{ij} injected from module v_i into G be equal to $-\lambda_i$. On the other hand, constraint (74) has the total flow x'_{ij} injected from module $v_{i'}$ into G' be equal to λ_i. Suppose we combine Eqs. (73) and (74), we have

$$\sum_j -x_{ij} + x_{ji} = \lambda_i = \sum_j x'_{ij} - x'_{ji}. \tag{81}$$

This means that the amount of flow λ_i which emanates from module v_i in G enters its corresponding module in $v_{i'}$ in G'.

Constraints (75)–(78) indicate that a_s and b_s are the flow injections to module v_s in G and its reversed circuit G'; a_t and b_t are the flow ejections from module v_t in G and its reversed circuit G', respectively. Combining circuit G and G' together, we have the maximum total flow, a_s+b_s, be the optimum solution of the minimum replication cut problem.

5.4.4. The Optimum Partition

In this subsection, we describe the construction of replication graph and take an example to describe it. We then apply the maximum flow algorithm on the constructed replication graph to derive an optimum replication cut. The optimality of the derived replication cut is proved by using a network flow approach.

Construction of Replication Graph Given a circuit $G(V, E)$ and modules v_s and v_t, we construct another circuit $G'(V', E')$ where $|V'| = |V|$ with each module v'_i in V' corresponding to a module v_i in V, and $|E'| = |E|$ with each directed net e_{ij} in E' in the reverse direction of net e_{ij} in E. We create super modules v^*_s and v^*_t and nets (v^*_s, v_s), (v^*_s, v'_s), (v_t, v^*_t), and (v'_t, v^*_t) with infinite capacity as shown in Figure 24. From every module v_i in V except v_s and v_t, we add a directed net of infinite capacity to the corresponding module v'_i in V'. We refer to the combined circuit as G^*.

Polynomial-time Algorithm The optimum replication cut problem with respect to module pair v_s and v_t and without size constraints can be solved by a maximum-flow minimum-cut solution of the circuit G^* with v^*_s as the source and v^*_t as the sink of the flow (Fig. 24). Suppose the maximum-flow minimum-cut finds partition $(X, \bar{X})$ of V with $v_s \in X$ and $v_t \in \bar{X}$ and partition $(X', \bar{X}')$ of V' with $v'_s \in X'$ and $v'_t \in \bar{X}'$. Then a replication cut (V_1, V_2) of the original circuit with $V_1 = X$, $V_2 = \{i | i' \in \bar{X}'\}$ and $R = V - V_1 - V_2$ is an optimum solution. Note that V_2 is derived from the cut in vertex set V'. To simplify the notation, we shall use $(X, \bar{X}')$ to denote the derived replication cut of G.

Example Given a circuit in Figure 25, its replication graph G^* is constructed as shown in Figure 26. The maximum-flow minimum-cut of G^* derives $(X, \bar{X}) = (\{v_s, v_a\}, \{v_b, v_c, v_t\})$ and $(X', \bar{X}') = (\{v'_s, v'_a, v'_b, v'_c\}, \{v'_t\})$ with a flow amount, 5 (Fig. 26). Thus the sets $V_1 = \{v_s, v_a\}$ and $V_2 = \{v_t\}$ define an optimum replication cut $R(V_1, V_2)$ with $R = \{v_b, v_c\}$ and a cut cost equal to 5 (Fig. 27).

The network flow approach leads to the optimality of the solution as stated in the following theorem.

THEOREM 5.3 *The replication cut $R(X, \bar{X}')$ derived from the transformed circuit G^* generates the minimum replication cut count $C_R(X, \bar{X}')$ (expression* (19)).

5.4.5. Heuristic Flow Algorithms

We introduce the heuristic approaches that accelerate the flow calculation and take advantage the optimality properties of the flow methods. We first introduce an approach that utilizes the maximum flow minimum cut method for the min cut with

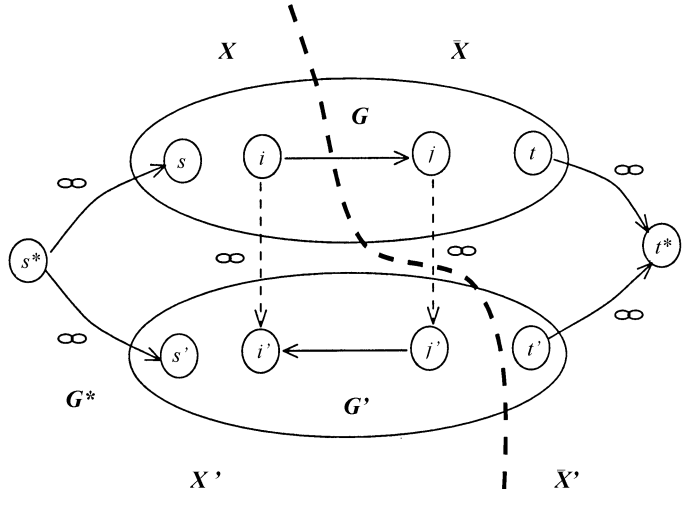


FIGURE 24 The replication graph G^*.

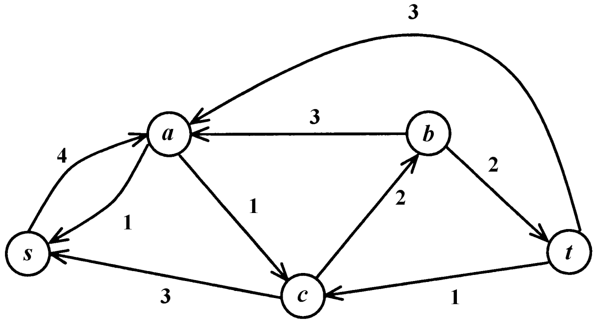


FIGURE 25 A five module circuit to demonstrate the replication cut.

size constraints. We then explain a shortest path method for multiple commodity flow calculation.

(i) Usage of Maximum Flow Minimum Cut We adopt a heuristic approach [113] to get around the unbalanced partition of the maximum flow and minimum cut method. First, we find two seeds as the source and the sink modules, v_s, v_t. We then use the maximum flow and minimum cut method to find partition (V_1, V_2) with $v_s \in V_1$ and $v_t \in V_2$. Suppose the size $S(V_1)$ of V_1 is larger than the size $S(V_2)$ of V_2, we find from V_1 a module v_i to merge with V_2 and shrink set V_2 as a new sink module. Otherwise, we find from V_2 a module v_i to merge with V_1 and shrink set V_1 as a new source module. We repeat the maximum flow minimum cut process on the graph with new source or sink module until the size of the partition fits the size constraint.

Two Way Partitioning using Maximum Flow Minimum Cut

1. Find two seeds as v_s and v_t.
2. Call Maximum Flow Minimum Cut to find partition (V_1, V_2).
3. If $S(V_1) > S(V_2)$, find a seed $v_i \in V_1$, merge $\{v_i\} \cup V_2$ into a new sink module v_t.
4. Else find a seed $v_i \in V_2$, merge $\{v_i\} \cup V_1$ into a new source module v_s.
5. Repeat Steps 1–4, until $S_l < S(V_1) < S_u$ and $S_l < S(V_2) < S_u$.

We can use parametric flow approach recursively to the maximum flow minimum cut problems recursively (Step 2). The total complexity is equivalent to a single maximum flow minimum cut.

The seeds are chosen according to its connectiv-

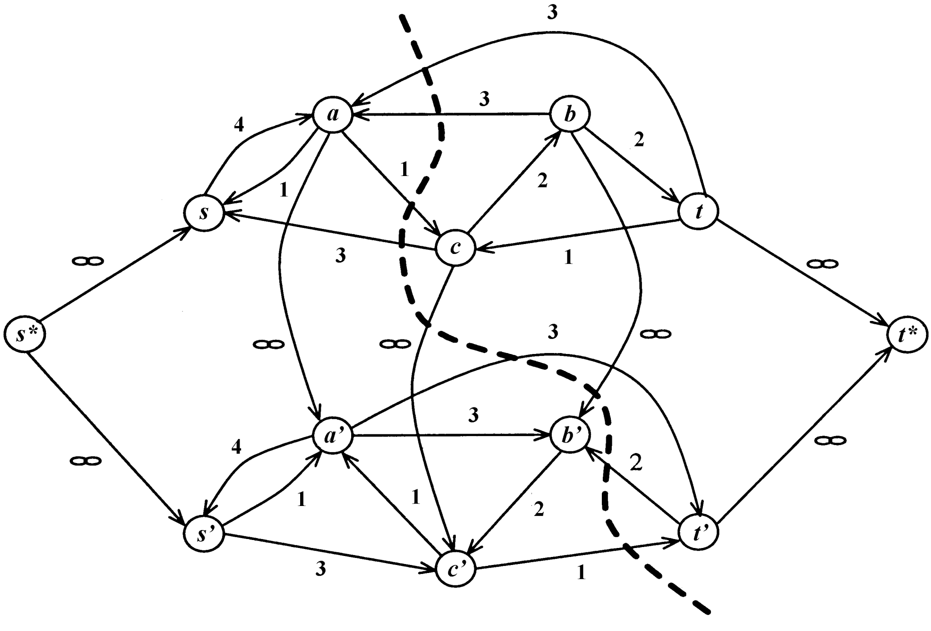


FIGURE 26 The constructed replication graph of the circuit shown in Figure 25.

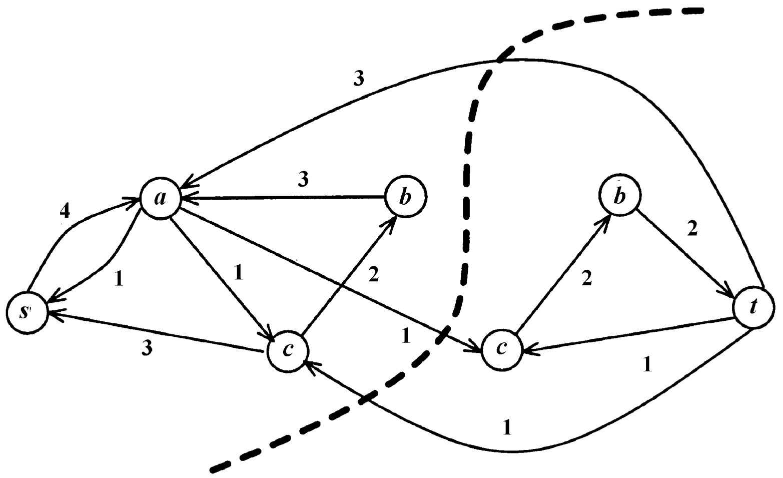


FIGURE 27 The duplicated circuit of the circuit shown in Figure 25.

ity to the vertex set in the other side. The result is sensitive to the choice of the seeds. We can make multiple trials and choose the best results. Other methods such as programming approach can serve as a guideline on the choice of the seeds [79, 80]. The method has shown to derive excellent results with reasonable running time.

(ii) Approximation of Multiple Commodity Flow Based on the multicommodity flow formulation [103], we try to solve a multiple way partitioning by deriving approximate multiple commodity flow with a stochastic process [13, 55, 114, 117].

Given a circuit $H(V, E)$, the flow increment Δ, and the distance coefficient α, the algorithm starts with procedure *Saturate-Network* to saturate the circuit with flows. A stochastic flow injection algorithm is adopted to reduce the computational complexity. Then, *Select-Cut* is activated to select a set of nets by the flow values to constitute a cut. The conversion from weighted ratio cut to cluster ratio cut is performed by a *Select-Cut* routine which selects the subset of the cut derived from *Saturate-Network* with a greedy approach.

Multiple Commodity Flow Approximation (H, Δ, α)

```
1. Iterate the following procedures
    1.1. Saturate-Network $(H, \Delta, \alpha)$.
    1.2. Select-Cut $(H)$ until the clustering result
         are satisfactory
2. Output clustering result.
```

Procedure Saturate-Network (H, Δ, α)

```
1. Set the distance of each net $e$ to be one.
2. While ($H$ is connected) do 2.1 to 2.3.
    2.1. Randomly pick two distinct modules $v_s$ and $v_t$.
    2.2. Find the shortest path between $v_s$ and $v_t$.
    2.3. For each net $e$ on the shortest path, let $f(e)$
         and $d_e$ be the flow and distance of net $e$.
        2.3.1. If $n$ is not saturated, increase $f(e)$ by $\Delta$
               and set $d_e = \exp((\alpha \times f(e))/c_e)$.
        2.3.2. If $e$ is saturated, set $d_e$ to be $\infty$.
3. Output $E$ with flow informations.
```

The initial distance of each net is one since there is no flow being injected (see the distance formulation in Step 2.3.1). Step 2.1 uses a random process with even distribution over all modules to pick two distinct modules, and Steps 2.2–2.3 inject Δ amount of flows along the shortest path between the modules. In Steps 2.3.1–2.3.2, the distances of the nets whose flow has been increased are recomputed using an exponential function $d_e = \exp$

$((\alpha \times f(e))/c_e)$ to penalize the congested nets, where d_e and $f(e)$ are the distance and flow of net e, respectively. Steps 2.1 – 2.3 are iteratively executed until a pair of modules are chosen where all possible paths between them are saturated by flows. These saturated nets identify a partition of the circuit.

Figure 28 shows a sample circuit saturated by flows after executing *Saturate-Network* with $\Delta = 0.01$ and $\alpha = 10$. The flow values are shown by the numbers right beside each net. The dashed lines indicate the cut lines along the set of saturated nets to form the three clusters. These saturated nets define an approximate weighted cluster ratio cut which are *potential* set of nets for a selection of cluster ratio cut.

5.5. Programming Approaches

For programming approaches [7, 18, 35, 41, 46, 44], we adopt two way minimum cut with size constraints as the target problem. We assume that the nets are two pin nets and thus, the circuit can be described as a graph $G(V, E)$. We also assume the modules are of unit size, *i.e.*, $s_i = 1$.

The two way partition (V_1, V_2) is represented by a linear placement with only two slots at coordinates -1 and 1. For an even sized partition, half of the modules are assigned to each slot. Let x_i denote the coordinate of module v_i. If $v_i \in V_1$, $x_i = 1$, else $x_i = -1$ for $v_i \in V_2$. The cut count can be expressed as follows.

$$C(V_1, V_2) = \frac{1}{4} c_{ij}(x_i - x_j)^2 = \frac{1}{4} X^\top B X \qquad (82)$$

where X is a vector of x_i, and $X^\top$ is the transpose of vector X. Matrix B has its entry $b_{ij} = -c_{ij}$ if $i \neq j$, else $b_{ii} = \sum_{1 \leq j \leq |V|} c_{ij}$. Suppose we relax the slot constraint by enforcing only the rules of the gravity center and the norm. The constraint of vector X can be expressed as:

$$1^\top X = 0, \qquad (83)$$

$$X^\top X = |V| \qquad (84)$$

Matrix B is symmetric and diagonally semidominant. Thus, it is semipositive definite, *i.e.*, all eigenvalues are nonnegative. And its eigenvectors are orthogonal. Let us order its eigenvalues from

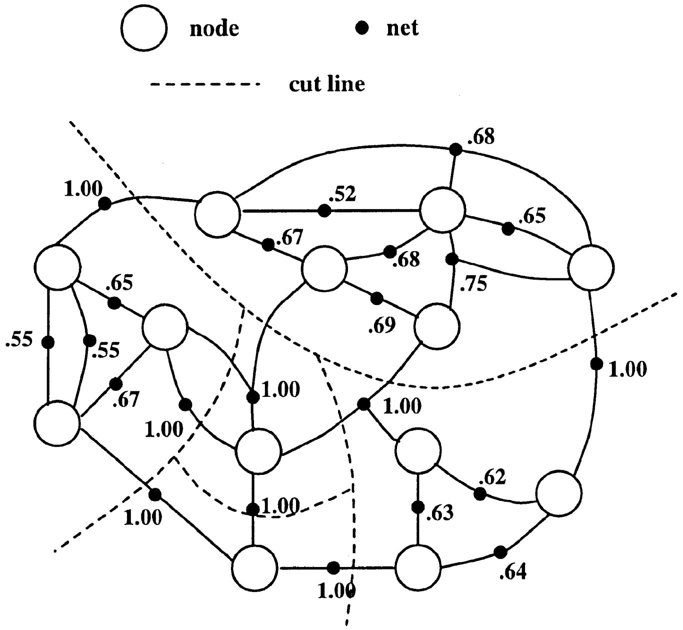


FIGURE 28 The flow and partition generated by saturate-network.

small to large, *i.e.*, $\lambda_0 \leq \lambda_1 \cdots \leq \lambda_{|V|-1}$. The smallest eigenvalue $\lambda_0 = 0$ with its eigenvector $X_0 = 1$. The second eigenvalue λ_1 is nonnegative with its eigenvector orthogonal to the first eigenvector, *i.e.*, $X_0^\top X_1 = 1^\top X_1 = 0$. Therefore, the second eigenvector X_1 is an optimal solution to objective function (82) with constraints (83) [46]. Since $X^\top X = |V|$ Eq. (84) the solution

$$\frac{1}{4} X_1^\top B X_1 = \frac{1}{4}\lambda_1 \times X_1^\top X_1 = \frac{1}{4}\lambda_1 \times |V|, \qquad (85)$$

which is a lower bound of the min-cut problem.

To push for a higher lower bound, we can adjust the diagonal term of matrix B by adding constants d_i. Let

$$\begin{aligned}\tilde{C}(V_1, V_2) &= C(V_1, V_2) + \frac{1}{4}\sum_{1 \leq i \leq |V|} d_i \times x_i^2 \\ &\quad - \frac{1}{4}\sum_{1 \leq i \leq |V|} d_i \qquad (86)\\ &= \frac{1}{4}\left(X^\top \tilde{B} X - \sum_{1 \leq i \leq |V|} d_i\right),\end{aligned}$$

where matrix $\tilde{B}$ has its entry $\tilde{b}_{ij} = b_{ij}$ if $i \neq j$, else $\tilde{b}_{ii} = b_{ii} + d_i$. Either $x_i = 1$ or $x_i = -1$, the last two terms cancel each other. The modification thus does not alter the optimal partition solution.

The new nonlinear programming problem is to find the assignment of d_i to maximize the objective function [11]:

$$\frac{1}{4}\left(\tilde{\lambda}_1 \times |V| - \sum_{1 \leq i \leq |V|} d_i\right) \qquad (87)$$

where $\tilde{\lambda}_1$ is the second smallest eigenvalue of matrix $\tilde{B}$. The solution is an upper bound of the partition. It is larger than λ_1 in the sense that λ_1 can serve as an initial feasible solution to maximize expression (87).

Remarks The programming approach finds a global view of the problem [9, 79, 80, 118]. However, the formulation is very restricted. The extension to multiple pin nets and the incorporation of fixed modules will destroy the nice structure based on which we have the eigenvalue and eigenvector as optimal solutions. Therefore, it is difficult to utilize the approach recursively.

For a general case, we can view the problem as nonlinear programming with Boolean quadratic objective function. Nonlinear programming techniques are adopted to derive the results [16, 107].

5.6. A Lagrange Multiplier Approach for Performance Driven Partitioning

Lagrange multiplier is one useful tool for performance optimization. In this section, we demonstrate the usage of Lagrange multiplier for performance driven partitioning. The problem is to optimize the performance of a two-way partition (V_1, V_2) with retiming [86].

We first introduce a vector of binary variables to represent a partition. The performance-driven partitioning problem is thus represented by a Boolean quadratic programming formulation with nonlinear constraints. We then absorb the nonlinear constraints into the objective function as a Lagrangian. We use primal and dual subproblems to decompose the Lagrangian and derive the partitions. Lagrange multiplier is adjusted in each iteration *via* a subgradient method to monitor the timing criticality and improve the performance.

5.6.1. Programming Formulation with Lagrange Multiplier

We assume that the circuit can be represented by a graph $G(V, E)$ with two pin nets and unit module size. The two-way partition is described by a vector $x = (x_{1,1}, \ldots, x_{1,n}, x_{2,1}, \ldots, x_{2,n})$, where $x_{b,i}$ is 1 if module v_i is assigned to vertex set V_b, otherwise $x_{b,i}$ is 0. If modules v_i and v_j are in different vertex set, the value of the term $x_{1,i}x_{2,j} + x_{2,i}x_{1,j}$ is equal to 1. This contributes one interpartition delay δ into the delay of the net e_{ij}. Let $g_l(x)$ denote the delay to register ratio of loop l. Delay ratio $g_l(x)$ can be written as the following formula:

$$g_l(x) = \frac{d_\ell + \sum_{e_{ij} \in l} \delta \times (x_{1,i}x_{2,j} + x_{2,i}x_{1,j})}{r_l} \tag{88}$$

Given a path p, the total delays $h_p(x)$ of p is as follows:

$$h_p(x) = d_p + \sum_{e_{ij} \in p} \delta \times (x_{1,i}x_{2,j} + x_{2,i}x_{1,j}) \tag{89}$$

To formulate the problem, we use an objective function of cut count:

$$\min \sum_{e_{ij} \in E} c_{ij}(x_{1,i}x_{2,j} + x_{2,i}x_{1,j}), \tag{90}$$

subject to the following constraints:

C1 (Size Constraints)

$$\sum_{i=1}^{|V|} x_{b,i}s_i \leq S_u \quad \forall\, b \in \{1, 2\}. \tag{91}$$

C2 (Variable Assignment Constraints)

$$\sum_{b=1}^{2} x_{b,i} = 1 \quad \forall\, v_i \in V. \tag{92}$$

C3 (Iteration Bound Constraints)

$$g_l(x) \leq \tilde{J} \quad \forall \text{ loop } l. \tag{93}$$

C4 (Latency Bound Constraints)

$$h_p(x) \leq \tilde{M} \quad \forall\, IO\text{-critical path } p. \tag{94}$$

Actually, we don't need to consider all loops in C3. Because all loops are composed of simple loops, we have the following lemma:

Lemma 1 *Given a number $\tilde{J}$, if $g_l(x)$ is less than or equal to $\tilde{J}$ for any simple loop l, then $g_l(x)$ is less than or equal to $\tilde{J}$ for all loops l.*

Let π_c and π_p represent the number of the simple loops and the number of *IO*-critical paths, respectively. Let λ denote the vector $(\lambda_{g_1}, \ldots, \lambda_{g_{\pi_c}}, \lambda_{h_1}, \ldots, \lambda_{h_{\pi_p}})$. Using Lagrangian Relaxation [104], we absorb the constraints (93) and (94) into the objective function (90). The Lagrangian-relaxed problem is as follows.

$$\max_{\lambda \geq 0} \min_{x} L(x, \lambda) \tag{95}$$

subject to constraints C1 and C2, where

$$\begin{aligned} L(x,\lambda) = {} & \sum_{e_{ij} \in E} c_{ij}(x_{1,i}x_{2,j} + x_{2,i}x_{1,j}) \\ & + \sum_{\forall \text{ simple loop } l} \lambda_{g_l}(g_l(x) - \tilde{J}) \\ & + \sum_{\forall\, IO\text{-critical path } p} \lambda_{h_p}(h_p(x) - \tilde{M}) \end{aligned} \tag{96}$$

(i) The Dual Problem Given vector x, we can represent (96) as a function of variable λ, *i.e.*, $L_x(\lambda)$. Thus, the dual problem can be written as:

$$\max_{\lambda \geq 0} L_x(\lambda) \tag{97}$$

(ii) The Primal Problem Let F_{ij} and Q_{ij} denote the sets of the simple loops and *IO*-critical paths passing the net e_{ij}. The cost a_{ij} of net e_{ij} is composed of connectivity c_{ij} and the penalty of the timing constraints.

$$a_{ij} = c_{ij} + \sum_{l \in F_{ij}} \frac{\delta}{r_l} \lambda_{g_l} + \sum_{p \in Q_{ij}} \delta \lambda_{h_p} \tag{98}$$

Given vector λ, we can represent (96) as a function of vector x, *i.e.*, $L_\lambda(x)$. Thus, the primal problem can be rewritten as:

$$\min\ L_\lambda(x) = \min \sum_{e_{ij} \in E} a_{ij}(x_{1,i}x_{2,j} + x_{2,i}x_{1,j}) + \beta \tag{99}$$

subject to C1 and C2, where β represents the constant contributed by λ.

5.6.2. *Subgradient Method using Cycle Mean Method*

We solve the partitioning problem through primal and dual iterations on the Lagrangian. A Quadratic Boolean Programming, QBP, [16] is used to solve the primal problem and generate a solution x (Step 2).

For the dual problem based on x, we select the set of loops and paths that violates the timing constraints as active loops and paths. The nets contained in the active loops or paths are termed active nets.

Active Loops and Paths Given a solution x, a loop l is called active, if $g_l(x)$ is not less than $\tilde{J}$. A path p is called active, if $h_p(x)$ is not less than $\tilde{M}$.

Active Nets Given a net e, we define e to be an active net, if net e is covered by an active loop or an active path.

We call a minimum cycle mean algorithm [57] and an all-pairs shortest-paths algorithm to mark all the nets on active loops and paths, respectively (Step 3). For every net e_{ij} on active paths, we record q_{ij}: the maximum path delay among all paths passing through e_{ij}. For every net e_{ij} on active loops, we record p_{ij}: the maximum delay-to-register ratio among all loops passing through e_{ij}. We then calculate the subgradient on the marked nets and update the constants a_{ij} for the next primal dual iteration (Steps 4–5). We increase the costs of active nets using subgradient approach [104]. The iteration proceeds until the bound of all loops and paths are within the given limits.

Algorithm using Lagrange Multiplier Input: Constants $\tilde{J}, \tilde{M}, \alpha = 1.3$ and an initial partition $\left(V_1^{(0)}, V_2^{(0)}\right)$.

1. Initialize $k \leftarrow 1$; $a_{ij}^{(0)} = c_{ij}$.
2. Run QBP [16] to find a partition $\left(V_1^{(k)}, V_2^{(k)}\right)$ with an object to minimize cut count $C\left(V_1^{(k)}, V_2^{(k)}\right) = \sum_{e \in E(V_1^{(k)}, V_2^{(k)})} a_{ij}^{(k)}$.
3. Calculate the iteration and latency bounds of the partition $\left(V_1^{(k)}, V_2^{(k)}\right)$, respectively. Stop if timing constraints are satisfied. Otherwise, revise p_{ij} and q_{ij} for all nets e_{ij}.
4. Compute
$$t^{(k)} = \frac{\alpha \left| C\left(V_1^{(k)}, V_2^{(k)}\right) - C\left(V_1^{(0)}, V_2^{(0)}\right) \right|}{\sum_{e_{ij} \in E} (p_{ij} - \tilde{J})^2 + \sum_{e_{ij} \in E} (q_{ij} - \tilde{M})^2}$$
5. Revise shadow price a_{ij} for all nets $e_{ij} \in E$:
   $a_{ij}^{(k+1)} = a_{ij}^{(k)}$;
   if net e_{ij} is in active loop, then $a_{ij}^{(k+1)} = a_{ij}^{(k)} + t^{(k)}(p_{ij} - \tilde{J})$;
   if net e_{ij} is in active path, then $a_{ij}^{(k+1)} = a_{ij}^{(k)} + t^{(k)}(q_{ij} - \tilde{M})$.
6. While $k \leq$ MaxNumIter, set $k \leftarrow k+1$ and goto 2.

5.7. Clustering Heuristics

We first discuss the usage of clustering heuristics. We then discuss top down clustering and bottom up clustering approaches. At the last, we discuss some variations of clustering metrics.

5.7.1. *Usage of Clustering Heuristics*

The usage of clustering heuristics plays an important role in determining the quality of the final results. In the following, we discuss the issue in different topics. We use a two-way partitioning with size constraints as the target problem.

1. Top Down Clustering *versus* Bottom Up Clustering: Top down clustering approach provides a global view of the solution. The operations are consistent with the target problem. However, it is more time consuming because the clustering operates on the whole circuit [29]. Bottom up clustering is efficient. However, because the process operates locally, the target solution is sensitive to the clustering heuristics [59].
2. The Level of the Clustering: Suppose we represent the clustering results with a hierarchical tree structure. Let the root correspond to the whole circuit, the leaves correspond to the smallest clusters, and the internal nodes correspond to the intermediate clusters. Hence, the size of the clusters grows with the level of the nodes. Top down clustering creates clusters corresponding to nodes in high levels, while bottom up clustering creates clustering corresponding to nodes in low levels.

For example, in [60], Kernighan and Lin proposed a top down clustering approach, which divides the whole circuit into four clusters only. In [59], Karypis *et al.*, used a bottom up clustering which starts with clusters of two modules or a net. If we continue the application of bottom up clustering on intermediate clusters, the quality of the clusters degenerates as the size of the clusters grows bigger.

3. Iteration of Clustering and Unclustering: We go through the iterations of clustering and unclustering to improve the quality of the results. At each level of the hierarchical tree, we derive an intermediate target solution, *e.g.*, a two-way partition. In unclustering, we go down the level of tree hierarchy to find an expanded circuit with more modules. In clustering, we go up the level of tree hierarchy with a circuit of a smaller number of modules. The previous partitioning result becomes the initial of the new partitioning problem. Note that the hierarchical tree is constructed dynamically. For each clustering, the modules can be grouped based on the current partitioning configuration.
4. The Clustering Operations and the Target Solution: The clustering operation has to be consistent with the target solution. For example, suppose the target is finding a two-way min-cut with size constraints. Then, it is natural to cluster modules based on net connectivity because the probability that a net is in an optimal cut set is small (see the subsection of min-cut with size constraints in problem formulations). Moreover, it is important that the clustering follows the current partitioning results, *i.e.*, only modules in the same partition are clustered.

5.7.2. *Top Down Clustering Approach for Partitioning*

We use an application to two-way cut with size constraints to illustrate the top down clustering approach [24, 29]. The partitioning of huge designs is complicated and the results can be erratic. Our strategy (Fig. 29) is to reduce the circuit complexity by constructing a contracted hypergraph. The clusters for the contracted hypergraph are searched *via* a recursive top down partitioning method. The number of modules is much reduced after we contract the clusters. Hence, a group

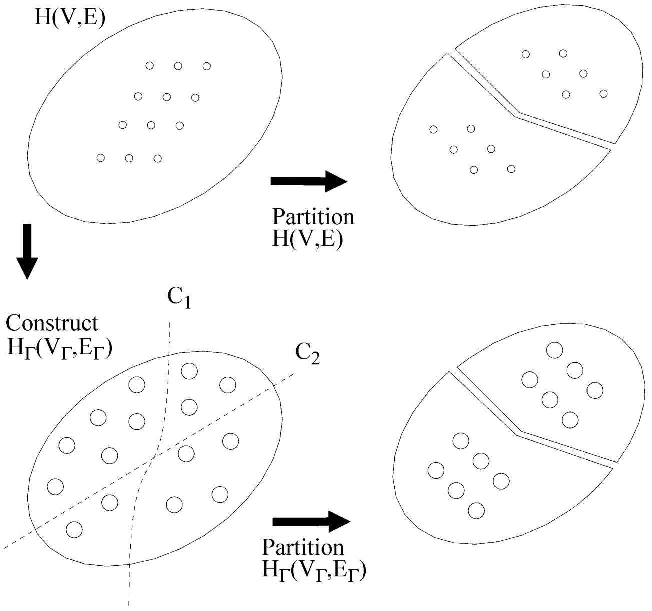


FIGURE 29 Strategy of top down clustering.

migration approach can derive excellent two way cut results on the contracted hypergraph with much efficiency. Furthermore, since the clusters are grouped *via* a top down partitioning, conceptually a minimum cut on the hypergraph can take advantage of the previous results and generate better solutions.

In this section, we describe a top down clustering algorithm. A ratio cut is adopted to perform the top down clustering process. Other partition approaches can also be used to replace the ratio cut. A group migration method is used to find a minimum cut of the contracted hypergraph with size constraint. Finally, we apply a last run of the group migration algorithm to the original circuit to fine tune the result.

Input a hypergraph $H(V, E)$, an integer k for the number of expected clusters, an integer *num_of_reps* for repetition, and S_l, S_u for the size constraints of two resultant subsets.

1. Initialize $\Psi=\{V\}$ and $V^*=V$.
2. Apply ratio cut [109] to obtain a partition (A, A') of $V^*=A\cup A'$.
3. Set $\Psi=(\Psi-V^*\})\cup\{A, A'\}$. Set V^* to be a vertex set in Ψ such that $S(V^*)=\max_{V_i\in\Psi} S(V_i)$.
4. While $S(V^*) > ((S(V))/k)$, repeat Steps 2, 3.
5. Construct a contracted hypergraph $H_\Gamma(V_\Gamma, E_\Gamma)$.
6. Apply *num_of_reps* times of a group migration algorithm to H_Γ with the size constraints S_l, S_u.
7. Use the best result from Step 6 to the circuit H as an initial partition. Apply a group migration algorithm once to H with the size constraints S_l, S_u.

The choice of cluster number k It was shown [24] that the cut count *versus* cluster number k is a concave curve. When k is small, the quality is not as good because the cluster is too coarse. When k is large, there are too many clusters. We lose the benefit of the clustering.

For the case that the circuit is large, we may need to adopt multiple levels of clustering to push for the performance and efficiency [58, 66].

5.7.3. Bottom Up Clustering Approaches

In this section, we discuss bottom up clustering [90] with two applications: linear placement and performance driven designs. We then show two strategies to perform the clustering: maximum matching and maximum pairing. We will demonstrate *via* examples the advantage of maximum pairing over maximum matching.

(i) Linear Placement For linear placement, we reduce the complexity of the problem by a bottom up clustering approach [96, 100, 53]. The clustering is based on the result of a tentative placement. We adopt a heuristic approach to generate tentative placements throughout iterations. In each iteration, we cluster modules only when they are in consecutive order of the placement. We then construct a contracted hypergraph. In the next iteration, the heuristic approach generates the placement of the contracted hypergraph. For each iteration, we either grow the size of the clusters or construct new clusters adaptively.

Inspired by the property of the minimum cut separating two modules (Theorem 3.1), we use a density as a measure to find the cluster. A density $d(i)$ at a slot i of a linear placement is the total connectivity of nets connecting modules on the different sides of the slot. The following algorithm describes the clustering using a given placement. Each cluster size is between L and U.

Input placement P, two parameters L and U.

1. Initialize cluster boundary at slot $p=1$.
2. Scan placement P from slot p toward the right end. Find slot i such that $p+L\le i\le p+U$ and density $d(i)$ is minimum among $d(p+L)\cdots d(p+U)$.
3. Cluster modules between slots p and i. Set $p=i+1$
4. Repeat Steps 2, 3 until the scan reaches the right end.

Remark The proposed clustering process and the criteria are consistent with the target linear placement application. The whole process depends on an efficient and effective linear placement.

(ii) Performance Driven Clustering For performance driven clustering [31, 112], nets which contribute to the longest delay are termed critical nets. Pins of the critical net are merged to form clusters.

For a special case that the circuit is a directed tree, we can find optimal solution in polynomial time. Let us assume the tree has its leaves at the input and its root at the output. We use a dynamic programming approach to trace from the leaves toward the root. Each module is not traced until all its input modules are processed. For each module, we treat it as a root of a subtree and find the optimal clustering of the subtree. Since all the modules in the subtree except its root have been processed, we can derive an optimal solution of the root in polynomial time.

(iii) Maximum Matching The maximum matching pairs all modules into $|V|/2$ groups simultaneously. Given a measurement of pairing modules, we can find a matching that maximizes the total pairing measurement in polynomial time.

We can call maximum matching recursively to create clusters of equal sizes. However, this strategy may enforce unrelated pairs to merge. The enforcement will sacrifice the quality of final clustering results.

Example Figure 30 illustrates the clustering behavior of maximum matching. The circuit contains twelve modules of equal size. The first level maximum matching pairs modules (a, b), (d, e), (g, h), (j, k), (c, l), and (f, i). Modules in the first four pairs are strongly connected with their partners. However, the last two are not. Module c and l have no common nets but are merged because their choices are taken by others.

Furthermore, as we proceed to the next level maximum matching, the merge of pairs (c, l) and (f, i) will enforce grouping modules into cluster $\{a, b, c, j, k, l\}$ and cluster $\{d, e, f, g, h, i\}$. If we measure the quality of the results with cluster cost (expression (26)), the cost of the two clusters is $\sum_i((C(V_i))/(C_I(V_i))) = 4/12+4/12 = 2/3$. For this case, we can find a better solution of clusters $\{a, b, c, d, e, f\}$ and $\{g, h, i, j, k, l\}$ of which the cluster cost is equal to zero.

Figure 31 shows another example of twelve modules with connectivities attached to the nets. The connectivity is 1 if not specified. Figure 31(a) shows an optimum cut with cut count 6.6. If a *maximum matching* [61] criterion is adopted in the bottom up clustering approach, then modules with a net of weight 1.1 between them will be merged. A minimum cut on the merged modules yields a cut count of 18 (Fig. 31(b)). In general, a $2n$ module circuit having a symmetric configuration as in Figure 31 will have a cut count of $n^2/2$ if the maximum matching criterion is applied to perform the clustering; while the optimum solution will have a cut weight of $1.1 \times n$. From this extreme case, we can claim the following theorem:

THEOREM 5.4 *There is no constant factor of error bound of the cut count generated by the maximum matching approach, from the cut count of a minimum cut.*

Proof As shown in the above example, the factor of error bound is $(n^2/2)/(1.1 \times n) = n/2.2$, which is not a constant. Q.E.D.

(iv) Maximum Pairing The maximum pairing is

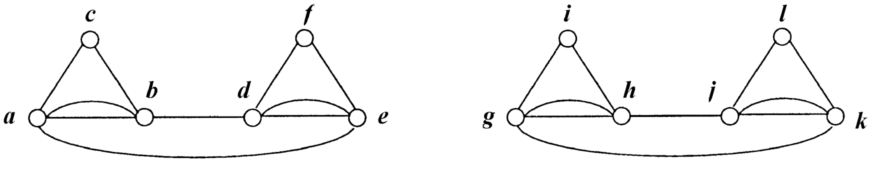


FIGURE 30 Clustering of two module circuit.

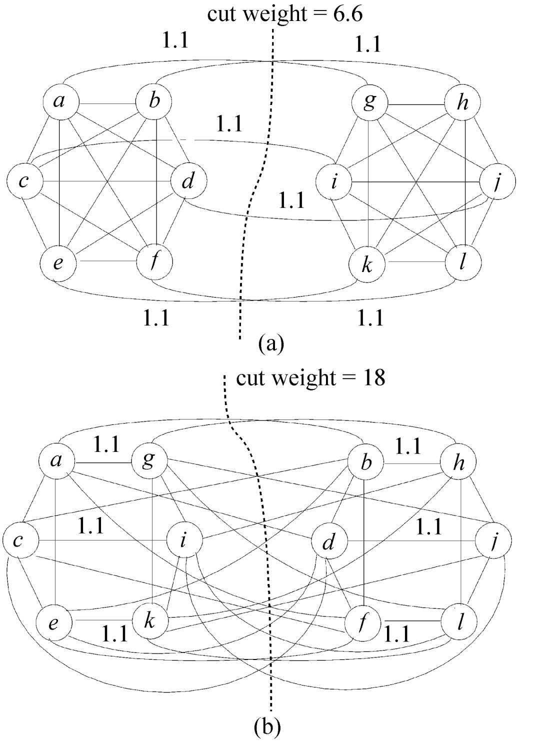


FIGURE 31 A twelve module example to demonstrate maximum matching.

similar to maximum matching, except that it does not enforce the matching of all modules. Only the top q percent of the modules are paired. Thus, we can avoid the enforced pairing of unrelated modules.

However, this strategy may cause certain modules to keep on growing and produce very uneven cluster results. Thus, we need to choose a proper cost function that discourages unlimited growth of the cluster size, *e.g.*, cost function (26).

5.7.4. Variations of Clustering Metric

In order to identify good clusters, we need to look beyond the direct adjacency between modules. It is useful if we can also extract the relation between the neighbors' neighbors, or even several levels of neighbors' neighbors. The probabilistic gain model of group migration approach is one good example of such approach [37, 42].

In this section, we will discuss a few different clustering metrics. For the case of k connectivity, we count the number of k-hop paths between two modules. Or, we use an analogy of a resistive network to check the conductance between the modules. Furthermore, we check beyond the hypergraph and use other information such as the module functions, pin locations, and control signals.

(i) *k*th Connectivity The number of k-hop paths between two modules provides a different aspect of information on the adjacency. Suppose the circuit has only two-pin nets. We can derive the kth connectivity with sparse matrix multiplication. Let C be the connectivity matrix with connectivity c_{ij} as its elements at row i column j, and at row j column i, and its diagonal entry $c_{ii}=0$. Note that we set $c_{ij}=0$ if there is no net connecting modules v_i and v_j.

Let $c_{ij}^{(2)}$ be the element of the square of matrix C (C^2), and $c_{ij}^{(k)}$ be the element of the kth order of matrix C (C^k). Then we have $c_{ij}^{(k)}$ representing the number of distinct k-hop paths connecting modules v_i and v_j.

(ii) Conductivity We use a resistive network analogy [21, 93] to derive the relation between modules. Suppose the circuit has only two pin nets. We replace each net e_{ij} with a resistor of conductance c_{ij}. Hence, we can view the whole system as a resistive network and derive the conductance between modules. The system conductance between two modules v_i and v_j reveals the adjacency relation between the two modules.

The network conductance can be derived using circuit analysis. We can also approximate the conductance with a random walk approach. In a random network model, we start walking from a module v_i. At each module v_k, the probability to walk *via* net e_{kl} to module v_l is proportional to the connectivity, *i.e.*, ($c_{kl}/\sum_m c_{km}$). We can derive the relation between the random walk and the conductivity [89]:

$$h_{ij} + h_{ji} = \frac{2\sum_{e \in |E|} c_e}{\sigma_{ij}}, \qquad (100)$$

where h_{ij} denotes the expected number of hops to walk from modules v_i and v_j, and σ_{ij} denotes the conductance between v_i and v_j.

(iii) Similarity of Signatures We can use certain features beyond connectivity for the clustering metric [88, 91]. For example, the index of data bits, sequence of the pins, function of logic, and relation with common control signals can serve as signatures of function blocks in data path designs. All these features form the first level adjacency. We can extend the relation to multiple levels. For example, two modules connecting a set of modules with strong similarity makes these two modules similar.

Example As shown in Figure 32, modules A and B are similar in signature because they are of the same OR function, connected to consecutive bit number at the same pin location, and controlled by the same control signal at the same pin location.

Modules C and D become similar because module C obtains signal from A, module D obtains signal from B, and modules A and B are similar.

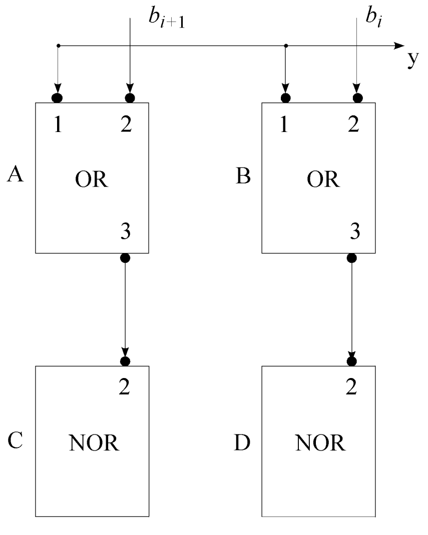


FIGURE 32 Signature identifies data structure.

6. RESEARCH DIRECTIONS

Partitioning remains to be an important research problem. Many applications such as floorplanning, engineering change orders, and performance driven emulation demand effective and efficient partitioning solutions.

Recent efforts released benchmarks with reasonable complexity [3]. However, more design cases are still needed to represent the class of huge circuitry with details of functions and timing.

In this section, we touch on a few interesting research problems regarding the correlation between the partition of logic and physical designs, the manipulation of hierarchical tree structure, and the performance driven partitioning.

6.1. Correlation of Hierarchical Partitioning Structure Between Logic Synthesis and Physical Layout

It is desired to correlate the logic hierarchy with the physical design hierarchy. The main reason is the control of timing for huge designs. Currently, the design turnaround takes 2–8 months for ASIC and much longer for custom designs. Throughout the design process, designs keep on changing. We don't want to lose control of timing as design changes. A tight correlation of logic and physical hierarchies makes timing predictable. Without this kind of mechanism, the timing characteristics of a floorplan may become erratic after iterations of design changes.

6.2. Manipulation of Hierarchical Partitioning Structure

One main issue in mapping a huge hierarchical circuit is the utilization of the hierarchy to reduce the mapping complexity. We can drastically improve the efficiency of the mapping process, if we properly exploit the structure of the design hierarchy. The generic binary tree is a good formulation to start with.

The handling of a hierarchy tree gives rise to many fundamental research problems. For example, finding k shortest-paths or exploring the maximum-flow minimum-cut of the whole circuit [51] embedded in a hierarchical tree can be useful for interconnect analysis and optimization. Such research can also benefit many different fields which have to handle huge hierarchical systems.

6.3. Performance Driven Partitioning

For performance driven partitioning, we need a fast evaluation on the hierarchical tree structure. The analysis needs to be incremental with incorporation of signal integrity.

The network flow method is a potential approach for the partitioning with timing constraints. More efforts are needed to improve the speed and derive desired results.

Acknowledgements

The authors thank the editor for the encouragement of preparing this manuscript. The authors would also like to thank Ted Carson, Lung-Tien Liu, and John Lillis for helpful discussions.

References

[1] Ahuja, R. K., Magnanti, T. L. and Orlin, J. B., *Network Flows*, Prentice Hall, 1993.

[2] Alpert, C. J., "The ISPD98 circuit benchmark suite", *Int. Symp. on Physical Design*, pp. 80 – 85, April, 1998.

[3] Alpert, C. J., Caldwell, A. E., Kahng, A. B. and Markov, I. L., "Partitioning with Terminals: a "New" Problem and New Benchmarks", *Int. Symp. on Physical Design*, pp. 151 – 157, April, 1999.

[4] Alpert, C. J., Huang, J. H. and Kahng, A. B., "Multilevel circuit partitioning", In: *Proc. ACM/IEEE Design Automation Conf.*, June, 1997, pp. 530 – 533.

[5] Alpert, C. J. and Kahng, A. B., "Recent directions in netlist partitioning: a survey", *Integration: The VLSI J.*, **19**(1), 1 – 81, August, 1995.

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

[7] Alpert, C. J. and Yao, S. Z., "Spectral partitioning: the more eigenvectors, the better", In: *Proc. ACM/IEEE Design Automation Conf.*, June, 1995, pp. 195 – 200.

[8] Bakoglu, H. B., *Circuits, Interconnections, and Packaging for VLSI*, MA: Addison-Wesley, 1990.

[9] Blanks, J. (1989). "Partitioning by Probability Condensation", *ACM/IEEE 26th Design Automation Conf.*, pp. 758 – 761.

[10] Bollobas, B. (1985). *Random Graphs*, Academic Press Inc., pp. 31 – 53.

[11] Boppana, R. B. (1987). "Eigenvalues and Graph Bisection: An Average Case Analysis", *Annual Symp. on Foundations in Computer Science*, pp. 280 – 285.

[12] Breuer, M. A., *Design Automation of Digital Systems*, Prentice-Hall, NY, 1972.

[13] Bui, T., Chaudhuri, S., Jones, C., Leighton, T. and Sipser, M. (1987). "Graph bisection algorithms with good average case behavior", *Combinatorica*, **7**(2), 171 – 191.

[14] Bui, T., Heigham, C., Jones, C. and Leighton, T., "Improving the performance of the Kernighan-Lin and simulated annealing graph bisection algorithms", In: *Proc. ACM/IEEE Design Automation Conf.*, June, 1989, pp. 775 – 778.

[15] Buntine, W. L., Su, L., Newton, A. R. and Mayer, A., "Adaptive methods for netlist partitioning", In: *Proc. IEEE Int. Conf. Computer-Aided Design*, November, 1997, pp. 356 – 363.

[16] Burkard, R. E. and Bonniger, T. (1983). "A Heuristic for Quadratic Boolean Programs with Applications to Quadratic Assignment Problems", *European Journal of Operational Research*, **13**, 372 – 386.

[17] Camposano, R. and Brayton, R. K. (1987). "Partitioning Before Logic Synthesis", *Int. Conf. on Computer-Aided Design*, pp. 324 – 326.

[18] Chan, P. K., Schlag, D. F. and Zien, J. Y., "Spectral *k*-way ratio-cut partitioning and clustering", *IEEE Trans. Computer-Aided Design*, **13**(9), 1088 – 1096, September, 1994.

[19] Charney, H. R. and Plato, D. L., "Efficient Partitioning of Components", *IEEE Design Automation*, July, 1968, pp. 16.0 – 16.21.

[20] Chatterjee, A. C. and Hartley, R., "A new Simultaneous Circuit Partitioning and Chip Placement Approach based on Simulated Annealing", In: *Proc. ACM/IEEE Design Automation Conf.*, June, 1990, pp. 36 – 39.

[21] Cheng, C. K. and Kuh, E. S., "Module Placement Based on Resistive Network Optimization", *IEEE Trans. on Computer-Aided Design*, **CAD-3**, 218 – 225, July, 1984.

[22] Cheng, C. K., "Linear Placement Algorithms and Applications to VLSI Design", *Networks*, **17**, 439 – 464, Winter, 1987.

[23] Cheng, C. K. and Hu, T. C., "Ancestor Tree for Arbitrary Multi-Terminal Cut Functions", *Porc. Integer Programming/Combinatorial Optimization Conf.*, Univ. of Waterloo, May, 1990, pp. 115 – 127.

[24] Cheng, C. K. and Wei, Y. C. (1991). "An Improved Two-Way Partitioning Algorithm with Stable Performance", *IEEE Trans. on Computer Aided Design*, **10**(12), 1502 – 1511.

[25] Cheng, C. K. (1992). "The Optimal Partitioning of Networks", *Networks*, **22**, 297 – 315.

[26] Cherng, J. S. and Chen, S. J., "A Stable Partitioning Algorithm for VLSI Circuits", In: *Proc. IEEE Custom Integrated Circuits Conf.*, May, 1996, pp. 9.1.1 – 9.1.4.

[27] Cherng, J. S., Chen, S. J. and Ho, J. M., "Efficient Bipartitioning Algorithm for Size-Constrained Circuits", *IEEE Proceedings-Computers and Digital Techniques*, **145**(1), 37 – 45, January, 1998.

[28] Cheng, C. K. and Hu, T. C. (1992). "Maximum Concurrent Flow and Minimum Ratio Cut", *Algorithmica*, **8**, 233 – 249.

[29] Chou, N. C., Liu, L. T., Cheng, C. K., Dai, W. J. and Lindelof, R., "Local Ratio Cut and Set Covering Partitioning for Huge Logic Emulation Systems", *IEEE Trans. Computer-Aided Design*, pp. 1085 – 1092, September, 1995.

[30] Chvatal, V. (1983). *Linear Programming*, W. H. Freeman and Company.

[31] Cong, J. and Ding, Y., "FlowMap: An Optimal Technology Mapping Algorithm for Delay Optimization in Lookup-Table Based FPGA Designs", *IEEE Trans. Computer-Aided Design*, January, 1994, **13**, 1 – 12.

[32] Cong, J., Labio, W. and Shivakumar, N., "Multi-way VLSI circuit partitioning based on dual net representation", In: *Proc. IEEE Int. Conf. Computer-Aided Design*, November, 1994, pp. 56 – 62.

[33] Cong, J., Li, H. P., Lim, S. K., Shibuya, T. and Xu, D., "Large scale circuit partitioning with loose/stable net removal and signal flow based clustering", In: *Proc. IEEE Int. Conf. Computer-Aided Design*, November, 1997, pp. 441 – 446.

[34] Donath, W. E. and Hoffman, A. J. (1973). "Lower Bounds for the Partitioning of Graphs", *IBM J. Res. Dev.*, pp. 420 – 425.

[35] Donath, W. E. and Hoffman, A. J. (1972). "Algorithms for partitioning of graphs and computer logic based on eigenvectors of connection matrices", *IBM Technical Disclosure Bulletin 15*, pp. 938 – 944.

[36] Donath, W. E. (1988). "Logic partitioning", In: *Physical Design Automation of VLSI Systems*, Preas, B. and Lorenzetti, M. (Eds.) Menlo Park, CA: Benjamin/ Cummings, pp. 65 – 86.

[37] Dutt, S. and Deng, W., "A Probability-based Approach to VLSI Circuit Partitioning", In: *Proc. ACM/IEEE Design Automation Conf.*, June, 1996, pp. 100 – 105.

[38] Dutt, S. and Deng, W., "VLSI Circuit Partitioning by Cluster-Removal Using Iterative Improvement Techniques", In: *Proc. IEEE Int. Conf. Computer-Aided Design*, November, 1996, pp. 194 – 200.

[39] Enos, M., Hauck, S. and Sarrafzadeh, M., "Evaluation and optimization of Replication Algorithms for logic Bipartitioning", *IEEE Trans. on Computer-Aided Design*, September, 1999, **18**, 1237 – 48.

[40] Fiduccia, C. M. and Mattheyses, R. M., "A Linear-Time Heuristic for Improving Network Partitions", In: *Proc. ACM/IEEE Design Automation Conf.*, June, 1982, pp. 175 – 181.

[41] Frankle, J. and Karp, R. M. (1986). "Circuit Placement and Cost Bounds by Eigenvector Decomposition", *Proc. Int. Conf. on Computer-Aided Design*, pp. 414 – 417.

[42] Garbers, J., Promel, H. J. and Steger, A. (1990). "Finding clusters in VLSI circuits", In: *Proc. IEEE Int. Conf. Computer-Aided Design*, pp. 520 – 523.

[43] Garey, M. R. and Johnson, D. S., *Computers and Instractability: A Guide to the Theory of NP-Completeness*, W.H. Freeman, San Francisco, CA, 1979.

[44] Hagen, L. and Kahng, A. B., "New spectral methods for ratio cut partitioning and clustering", *IEEE Trans. Computer-Aided Design*, **11**(9), 1074 – 1085, September, 1992.

[45] Hagen, L. and Kahng, A. B., "Combining problem reduction and adaptive multistart: a new technique for superior iterative partitioning", *IEEE Trans. Computer-Aided Design*, **16**(7), 709 – 717, July, 1997.

[46] Hall, K. M., "An *r*-dimensional Quadratic Placement Algorithm", *Management Science*, **17**(3), 219 – 229, November, 1970.

[47] Hamada, T., Cheng, C. K. and Chau, P., "An Efficient Multi-Level Placement Technique Using Hierarchical Partitioning", *IEEE Trans. Circuits and Systems*, **39**, 432 – 439, June, 1992.

[48] Hennessy, J. (1983). "Partitioning Programmable Logic Arrays Summary", *Int. Conf. on Computer-Aided Design*, pp. 180 – 181.

[49] Hoffmann, A. G., "The Dynamic Locking Heuristic – A New Graph Partitioning Algorithm", In: *Proc. IEEE Int. Symp. Circuits and Systems*, May, 1994, pp. 173 – 176.

[50] Adolphson, D. and Hu, T. C., "Optimal Linear Ordering", *SIAM J. Appl. Math.*, **25**(3), 403 – 423, November, 1973.

[51] Hu, T. C., "Decomposition Algorithm", pp. 17 – 22, In: *Combinatorial Algorithms*, Addison Wesley, 1982.

[52] Hu, T. C. and Moerder, K., "Multiterminal flows in a hypergraph", In: *VLSI Circuit Layout*: *Theory and Design*, Hu, T. C. and Kuh, E. (Eds.) NY: IEEE Press, 1985, pp. 87 – 93.

[53] Hur, S. W. and Lillis, J. (1999). "Relaxation and Clustering in a Local Search Framework: Application to Linear Placement", *Design Automation Conference*, pp. 360 – 366.

[54] Hwang, J. and Gamal, A. E., "Optimal Replication for Min-Cut Partitioning", *Proc. IEEE/ACM Intl. Conf. Computer-Aided Design*, November, 1992, pp. 432 – 435.

[55] Iman, S., Pedram, M., Fabian, C. and Cong, J., "Finding uni-directional cuts based on physical partitioning and logic restructuring", In: *Proc. ACM/SIGDA Physical Design Workshop*, May, 1993, pp. 187 – 198.

[56] Johnson, D. S., Aragon, C. R., McGeoch, L. A. and Schevon, C. (1989). "Optimization by Simulated Annealing: an Experimental Evaluation, Part I, Graph Partitioning", *Operations Research*, **37**(5), 865 – 892.

[57] Karp, R. M. (1978). "A Characterization of The Minimum Cycle Mean in A Digraph", *Discrete Mathematics*, **23**, 309 – 311.

[58] Karypis, G., Aggarwal, R., Kumar, V. and Shekhar, S., "Multilevel Hypergraph Partitioning: Application in VLSI Domain", In: *Proc. ACM/IEEE Design Automation Conf.*, June, 1997, pp. 526 – 529.

[59] Karypis, G., Aggarwal, R., Kumar, V. and Shekhar, S. (1998). "*Multilevel Hypergraph Partitioning*: *Application in VLSI Domain*", Manuscript of CS Dept., Univ. of Minnesota, pp. 1 – 25 (http://www.users.cs.umn.edu/karypis/metis/publications/).

[60] Kernighan, B. W. and Lin, S., "An Efficient Heuristic Procedure for Partitioning Graphs", *Bell Syst. Tech. J.*, **49**(2), 291 – 307, February, 1970.

[61] Khellaf, M., "On The Partitioning of Graphs and Hypergraphs", *Ph.D. Dissertation*, Indus. Engineering and Operations Research, Univ. of California, Berkeley, 1987.

[62] Kirkpatrick, S., Gelatt, C. and Vechi, M., "Optimization by Simulated Annealing", *Science*, **220**(4598), 671 – 680, May, 1983.

[63] Knuth, D. E., *The Art of Computer Programming*,

Addison Wesley, 1997.

[64] Kring, C. and Newton, A. R. (1991). "A Cell-Replicating Approach to Mincut Based Circuit Partitioning", *Proc. IEEE Int. Conf. on Computer-Aided Design*, pp. 2–5.

[65] Krishnamurthy, B., "An Improved Min-Cut Algorithm for Partitioning VLSI Networks", *IEEE Trans. Computers*, **C-33**(5), 438–446, May, 1984.

[66] Krupnova, H., Abbara, A. and Saucier, G. (1997). "A Hierarchy-Driven FPGA Partitioning Method", *Design Automation Conf.*, pp. 522–525.

[67] Kuo, M. T. and Cheng, C. K., "A New Network Flow Approach for Hierarchical Tree Partitioning", In: *Proc. ACM/IEEE Design Automation Conf.*, June, 1997, pp. 512–517.

[68] Kuo, M. T., Liu, L. T. and Cheng, C. K., "Network Partitioning into Tree Hierarchies", In: *Proc. ACM/IEEE Design Automation Conf.*, June, 1996, pp. 477–482.

[69] Kuo, M. T., Liu, L. T. and Cheng, C. K., "Finite State Machine Decomposition for I/O Minimization", In: *Proc. IEEE Int. Symp. on Circuits and Systems*, May, 1995, pp. 1061–1064.

[70] Kuo, M. T., Wang, Y., Cheng, C. K. and Fujita, M., "BDD-Based Logic Partitioning for Sequential Circuits", In: *Proc. ASP/DAC*, Chiba, Japan, January, 1997, pp. 607–612.

[71] Lomonosov, M. V. (1985). "Combinatorial Approaches to Multiflow Problems", *Discrete Applied Mathematics*, **11**(1), 1–94.

[72] Landman, B. S. and Russo, R. L., "On a Pin *Versus* Block Relationship for Partitioning of Logic Graphs", *IEEE Trans. on Computers*, **C-20**, 1469–1479, December, 1971.

[73] Lawler, E. L., *Combinatorial Optimization: Networks and Matroids*, Holt, Rinehart and Winston, New York, 1976.

[74] Leighton, T. and Rao, S. (1988). "An Approximate Max-Flow Min-cut Theorem for Uniform Multicommodity Flow Problems with Applications to Approximation Algorithms", *IEEE Symp. on Foundations of Computer Science*, pp. 422–431.

[75] Leighton, T., Makedon, F., Plotkin, S., Stein, C., Tardos, E. and Tragoudas, S., "Fast Approximation Algorithms for Multicommodity Flow Problems", *Tech. report no. STAN-CS-91-1375*, Dept. of Computer Science, Stanford University.

[76] Leiserson, C. E. and Saxe, J. B. (1991). "Retiming Synchronous Circuitry", *Algorithmica*, **6**(1), 5–35.

[77] Lengauer, T. and Muller, R. (1988). "Linear Arrangement Problems on Recursively Partitioned Graphs", *Zeitschrift fur Operations Research*, **32**, 213–230.

[78] Lengauer, T., *Combinatorial Algorithms for Integrated Circuit Layout*, Wiley, 1990.

[79] Li, J., Lillis, J. and Cheng, C. K., "Linear decomposition algorithm for VLSI design applications", In: *Proc. IEEE Int. Conf. Computer-Aided Design*, November, 1995, pp. 223–228.

[80] Li, J., Lillis, J., Liu, L. T. and Cheng, C. K., "New Spectral Linear Placement and Clustering Approach", In: *Proc. ACM/IEEE Design Automation Conf.*, June, 1996, pp. 88–93.

[81] Liou, H. Y., Lin, T. T., Liu, L. T. and Cheng, C. K., "Circuit Partitioning for Pipelined Pseudo-Exhaustive Testing Using Simulated Annealing", In: *Proc. IEEE Custom Integrated Circuits Con.*, May, 1994, pp. 417–420.

[82] Liu, L. T., Kuo, M. T., Cheng, C. K. and Hu, T. C., "A Replication Cut for Two-Way Partitioning", *IEEE Trans. Computer-Aided Design*, May, 1995, pp. 623–630.

[83] Liu, L. T., Kuo, M. T., Cheng, C. K. and Hu, T. C., "Performance-Driven Partitioning Using a Replication Graph Approach", In: *Proc. ACM/IEEE Design Automation Conf.*, June, 1995, pp. 206–210.

[84] Liu, L. T., Kuo, M. T., Huang, S. C. and Cheng, C. K., "A gradient method on the initial partition of Fiduccia-Mattheyses algorithm", In: *Proc. IEEE Int. Conf. Computer-Aided Design*, November, 1993, pp. 229–234.

[85] Liu, L. T., Shih, M., Chou, N. C., Cheng, C. K. and Ku, W., "Performance-Driven Partitioning Using Retiming and Replication", In: *Proc. IEEE Int. Conf. Computer-Aided Design*, November, 1993 pp. 296–299.

[86] Liu, L. T., Shih, M. and Cheng, C. K., "Data Flow Partitioning for Clock Period and Latency Minimization", In: *Proc. ACM/IEEE Design Automation Conf.*, June, 1994, pp. 658–663.

[87] Matula, D. W. and Shahrokhi, F., "The Maximum Concurrent Flow Problem and Sparsest Cuts", *Tech. Report*, southern Methodist Univ., 1986.

[88] McFarland, M. C., S.J.,"Computer-aided partitioning of behavioral hardware descriptions", In: *Proc. ACM/IEEE Design Automation Conf.*, June, 1983, pp. 472–478.

[89] Motwani, R. and Raghavan, P. (1995). *Randomized Algorithms*, Cambridge University Press.

[90] Ng, T. K., Oldfield, J. and Pitchumani, V., "Improvements of a mincut partition algorithms", In: *Proc. IEEE Int. Conf. Computer-Aided Design*, November, 1987, pp. 470–473.

[91] Nijssen, R. X. T., Jess, J. A. G. and Eindhoven, T. U., "Two-Dimensional Datapath Regularity Extraction", *Physical Design Workshop*, April, 1996, pp. 111–117.

[92] Parhi, K. K. and Messerschmitt, D. G. (1991). "Static Rate-Optimal Scheduling of Iterative Data-Flow Programs *via* Optimum Unfolding", *IEEE Trans. on Computers*, **40**(2), 178–195.

[93] Riess, B. M., Doll, K. and Johannes, F. M., "Partitioning very large circuits using analytical placement techniques", In: *Proc. ACM/IEEE Design Automation Conf.*, June, 1994, pp. 646–651.

[94] Roy, K. and Sechen, C., "A Timing Driven N-Way Chip and Multi-Chin Partitioner", *Proc. IEEE/ACM Int. Conf. on Computer-Aided Design*, pp. 240–247, November, 1993.

[95] Russo, R. L., Oden, P. H. and Wolff, P. K. Sr., "A heuristic procedure for the partitioning and mapping of computer logic graphs", *IEEE Trans. on Computers*, **C-20**, 1455–1462, December, 1971.

[96] Saab, Y., "A fast and robust network bisection algorithm", *IEEE Trans. Computers*, **44**(7), 903–913, July, 1995.

[97] Saab, Y. and Rao, V. (1989). "An Evolution-Based Approach to Partitioning ASIC Systems", *ACM/IEEE 26th Design Automation Conf.*, pp. 767–770.

[98] Sanchis, L. A., "Multiple-Way Network Partitioning", *IEEE Trans. Computers*, **38**(1), 62–81, January, 1989.

[99] Sanchis, L. A., "Multiple-Way Network Partitioning with Different Cost Functions", *IEEE Trans. on Computers*, pp. 1500–1504, December, 1993.

[100] Schuler, D. M. and Ulrich, E. G. (1972). "Clustering and Linear Placement", *Proc. 9th Design Automation Workshop*, pp. 50–56.

[101] Schweikert, D. G. and Kernighan, B. W. (1972). "A Proper Model for the Partitioning of Electrical Circuits", *Proc. 9th Design Automation Workshop*, pp. 57–62.

[102] Sechen, C. and Chen, D. (1988). "An Improved Objective Function for Mincut Circuit Partitioning", *Proc. Int. Conf. on Computer-Aided Design*, pp. 502–505.

[103] Shahrokhi, F. and Matula, D. W., "The Maximum Concurrent Flow Problem", *Journal of the ACM*, **37**(2), 318–334, April, 1990.

[104] Shapiro, J. F. (1979). *Mathematical Programming: Structures and Algorithms*, Wiley, New York.

[105] Sherwani, N. A. (1999). *Algorithms for VLSI Physical Design Automation*, 3rd edn., Kluwer Academic.

[106] Shih, M., Kuh, E. S. and Tsay, R.-S. (1992). "Performance-Driven System Partitioning on Multi-Chip Modules", *Proc. 29th ACM/IEEE Design Automation Conf.*, pp. 53–56.

[107] Shih, M. and Kuh, E. S. (1993). "Quadratic Boolean Programming for Performance-Driven System Partitioning", *Proc. 30th ACM/IEEE Design Automation Conf.*, pp. 761–765.

[108] Shin, H. and Kim, C., "A Simple Yet Effective Technique for Partitioning", *IEEE Trans. on Very Large Scale Integration Systems*, pp. 380–386, September, 1993.

[109] Wei, Y. C. and Cheng, C. K. (1991). "Ratio Cut Partitioning for Hierarchical Designs", *IEEE Trans. on Computer-Aided Design*, **10**(7), 911–921.

[110] Wei, Y. C., Cheng, C. K. and Wurman, Z., "Multiple Level Partitioning: An Application to the Very Large Scale Hardware Simulators", *IEEE Journal of Solid State Circuits*, **26**, 706–716, May, 1991.

[111] Woo, N. S. and Kim, J. (1993). "An Efficient Method of Partitioning Circuits for Multiple-FPGA Implementation", *Proc. ACM/IEEE Design Automation Conf.*, pp. 202–207.

[112] Yang, H. and Wong, D. F. (1994). "Edge-Map: Optimal Performance Driven Technology Mapping for Iterative LUT Based FPGA Designs", *Int. Conf. on Computer- A Aided Design*, pp. 150–155.

[113] Yang, H. and Wong, D. F., "Efficient Network Flow based Min-Cut Balanced Partitioning", In: *Proc. IEEE Int. Conf. Computer-Aided Design*, November, 1994, pp. 50–55.

[114] Yeh, C. W., "On the Acceleration of Flow-Oriented Circuit Clustering", *IEEE Trans. Computer-Aided Design*, **14**(10), 1305–1308, October, 1995.

[115] Yeh, C. W., Cheng, C. K. and Lin, T. T. Y., "A general purpose, multiple-way partitioning algorithm", *IEEE Trans. Computer-Aided Design*, **13**(12), 1480–1488, December, 1994.

[116] Yeh, C. W., Cheng, C. K. and Lin, T. T. Y., "Optimization by iterative improvement: an experimental evaluation on two-way partitioning", *IEEE Trans. Computer-Aided Design*, **14**(2), 145–153, February, 1995.

[117] Yeh, C. W., Cheng, C. K. and Lin, T. T. Y., "Circuit clustering using a stochastic flow injection method", *IEEE Trans. Computer-Aided Design*, **14**(2), 154–162, February, 1995.

[118] Zien, J. Y., Chan, P. K. and Schlag, M., "Hybrid spectral/iterative partitioning", In: *Proc. IEEE Int. Conf. Computer-Aided Design*, November, 1997 pp. 436–440.

Authors' Biographies

Sao-Jie Chen has been a member of the faculty in the Department of Electrical Engineering, National Taiwan University since 1982, where he is currently a full professor. During the fall of 1999, he held a visiting appointment at the Department of Computer Science and Engineering, University of California, San Diego. His current research interests include: VLSI circuits design, VLSI physical design automation, and object-oriented software engineering. Dr. Chen is a member of the Association for Computing Machinery, the IEEE, and the IEEE Computer Society.

Chung-Kuan Cheng received the B.S. and M.S. degrees in electrical engineering from National Taiwan University, and the Ph.D. degree in electrical engineering and computer sciences from University of California, Berkeley in 1984. From 1984 to 1986 he was a senior CAD engineer at Advanced Micro Devices Inc. In 1986, he joined the University of California, San Diego, where he is a Professor in the Computer Science and Engineering Department, an Adjunct Professor in the Electrical and Computer Engineering Department. He served as a chief scientist at Mentor Graphics in 1999. He is an associate editor of IEEE Trans. on Computer Aided Design since 1994. He is a recipient of the best paper award, IEEE Trans. on Computer-Aided Design 1997, the NCR excellence in teaching award, School of Engineering, UCSD, 1991. His research interests include network optimization and design automation on microelectronic circuits.